RL in the Real World: From Chip Design to LLMs

Anna Goldie
Anthropic & Stanford
(Includes work done at Google Brain)

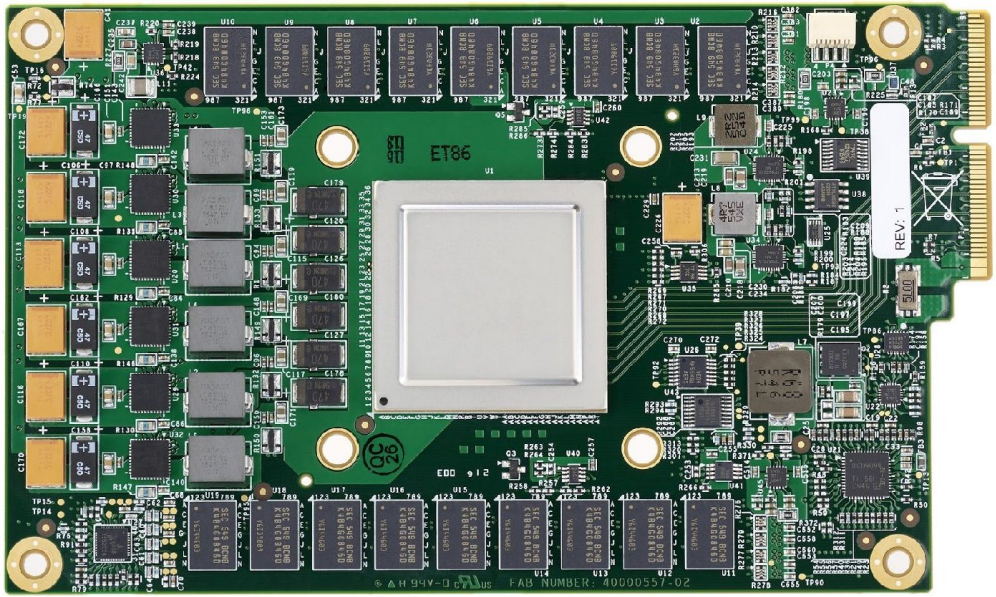
Structure of this Talk

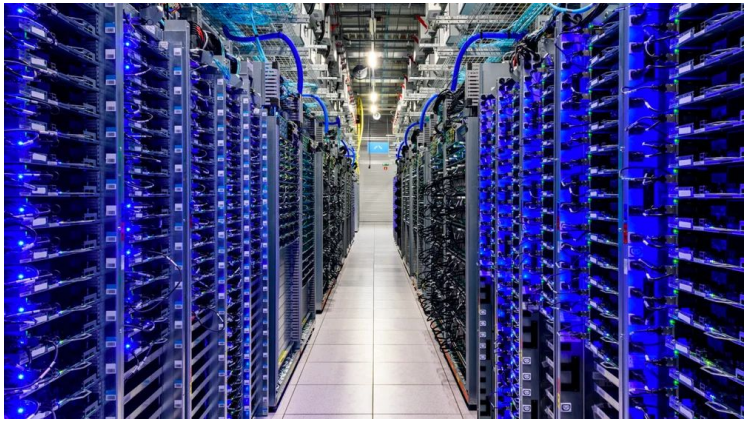
- RL for Chip Design
  - RL for AI Accelerators
- RL for Large Language Models
  - RL from Human Feedback
  - RL from AI Feedback

Structure of this Talk

- **RL for Chip Design**
  - **RL for AI Accelerators**
- RL for Large Language Models
  - RL from Human Feedback
  - RL from AI Feedback

RL-Generated Chip Floorplans used in TPU!
(Now for multiple generations, including the latest)





Published in *Nature*

Article | Published: 09 June 2021

A graph placement methodology for fast chip design

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 & Jeff Dean

Nature **594**, 207–212 (2021) | Cite this article

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

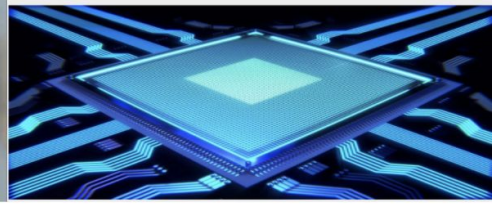
PC GAMER

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

By Paul Lilly 10 days ago

Chips making chips.

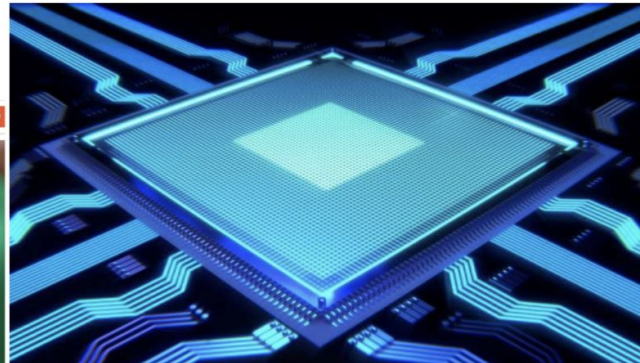
COMMENTS



Google Proposes AI as Solution for Speedier AI Chip Design

Google trains chips to design themselves

by Peter Grad , Tech Xplore



Google uses artificial intelligence to optimize AI chip production

By Mario McKellop · April 2, 2020

Google Hoping The Next AI Chips Will Be Designed By AI

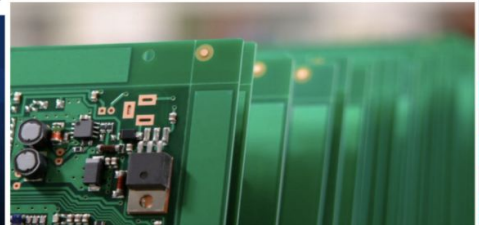
Company researchers have come up with an AI system that can design other AI chips. The goal is to help improve AI with the help of AI.

BY ASWIN NARAYANAN
MAR 31, 2020

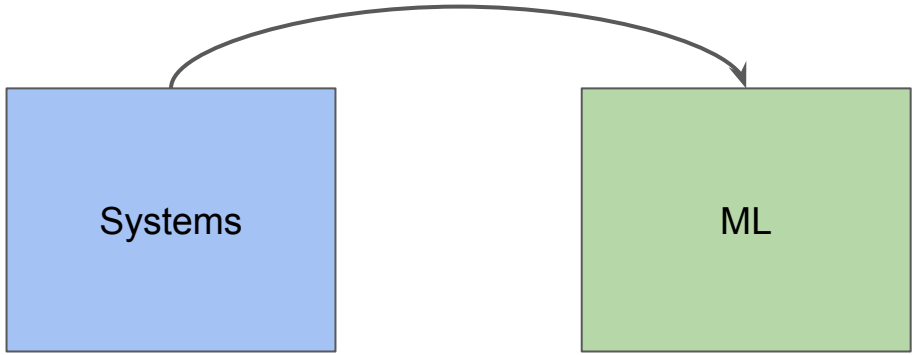
Google Researchers Create AI-ception with an AI Chip That Speeds Up AI

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

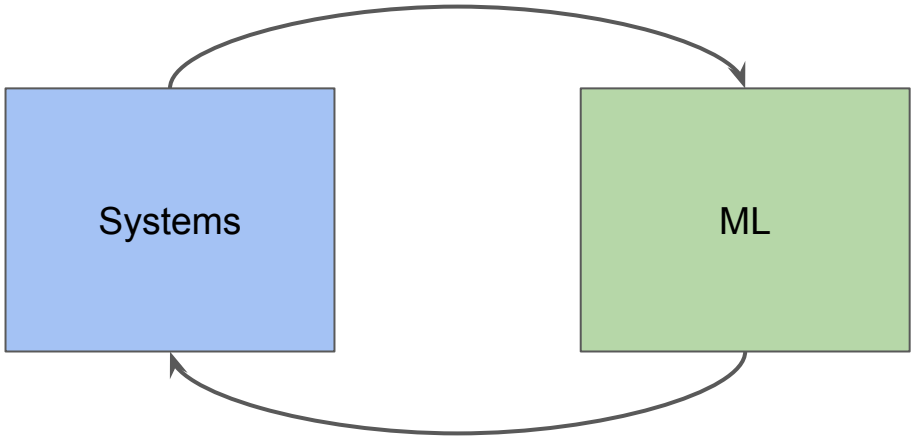


In the past decade, systems and hardware have transformed ML.



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

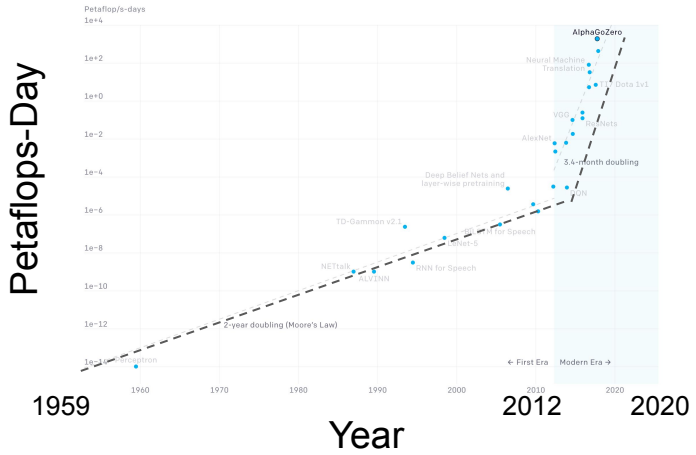
Now, it's time for ML to transform systems and hardware.



Demand for Compute Outpacing Supply (Moore's Law)

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*

Scaling Laws: Compute Fuels Progress in ML

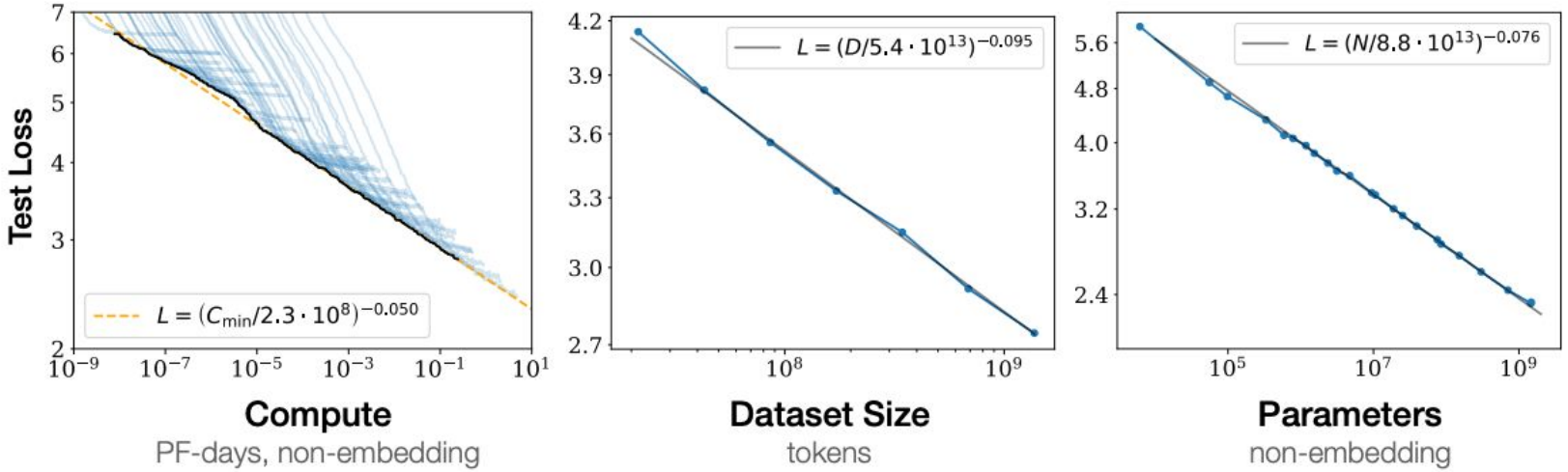


Figure 1 Language modeling performance improves smoothly as we increase the model size, datasetset size, and amount of compute[2] used for training. For optimal performance all three factors must be scaled up in tandem. Empirical performance has a power-law relationship with each individual factor when not bottlenecked by the other two.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, Dario Amodei. Scaling Laws for Neural Language Models. OpenAI 2020.

Value of Machine Learning for Chip Design

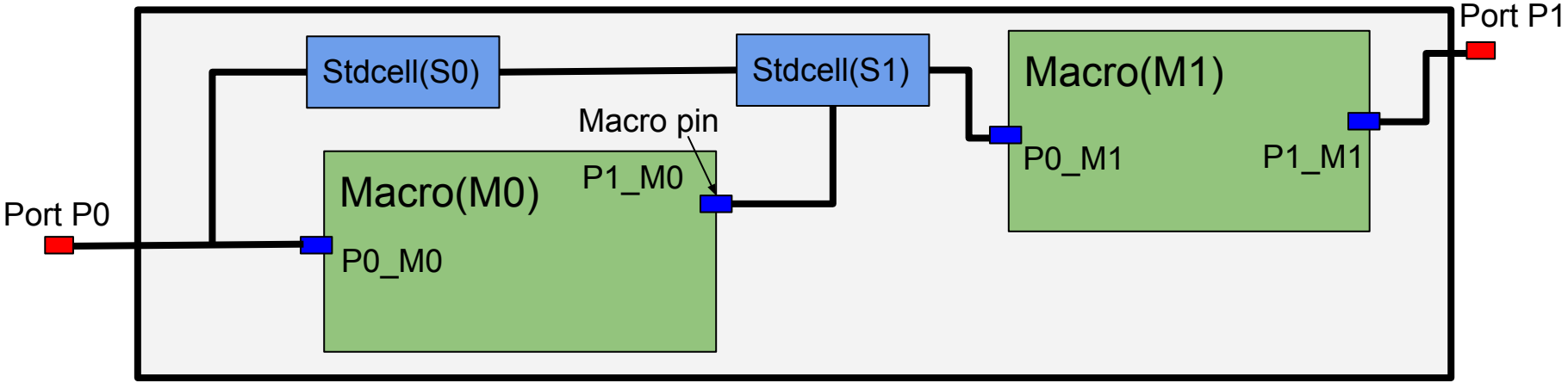
- Enabling cheaper, faster, and more environmentally friendly chips

- Potential to reduce 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)

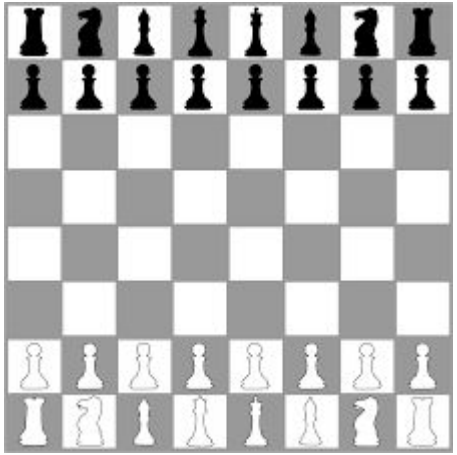
Chip Floorplanning Problem

- 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.



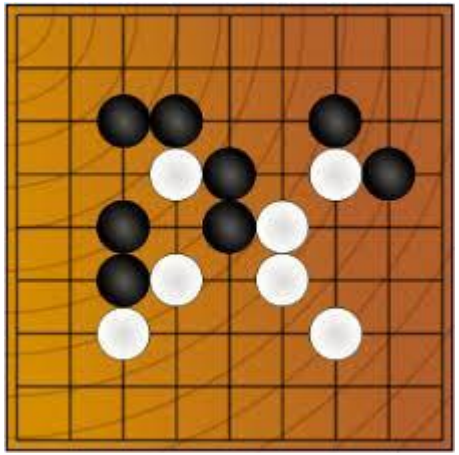
Complexity of Chip Placement Problem

Chess



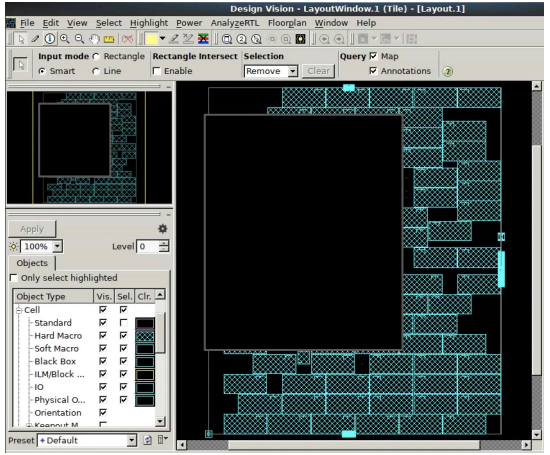
Number of states ~ 10^{123}

Go



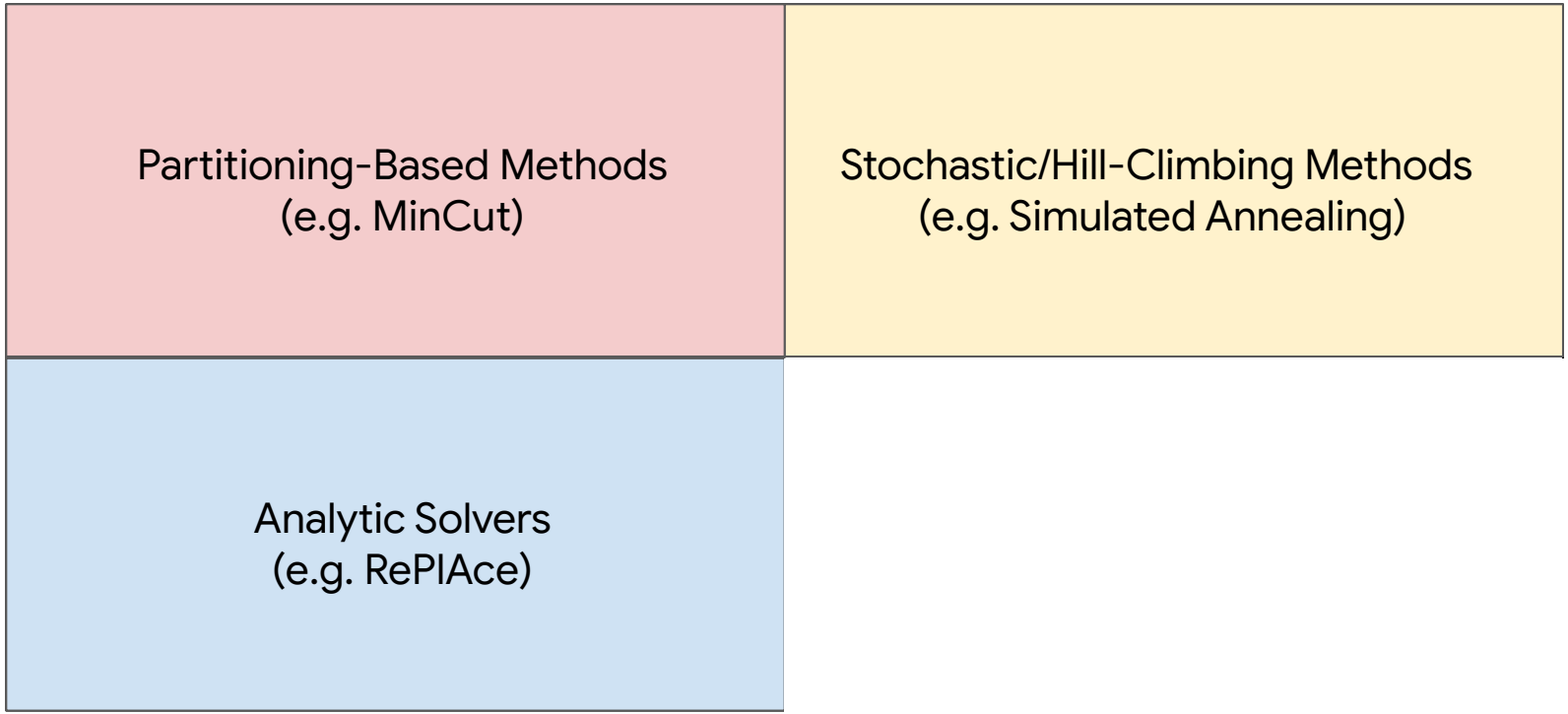
Number of states ~ 10^{360}

Chip Placement

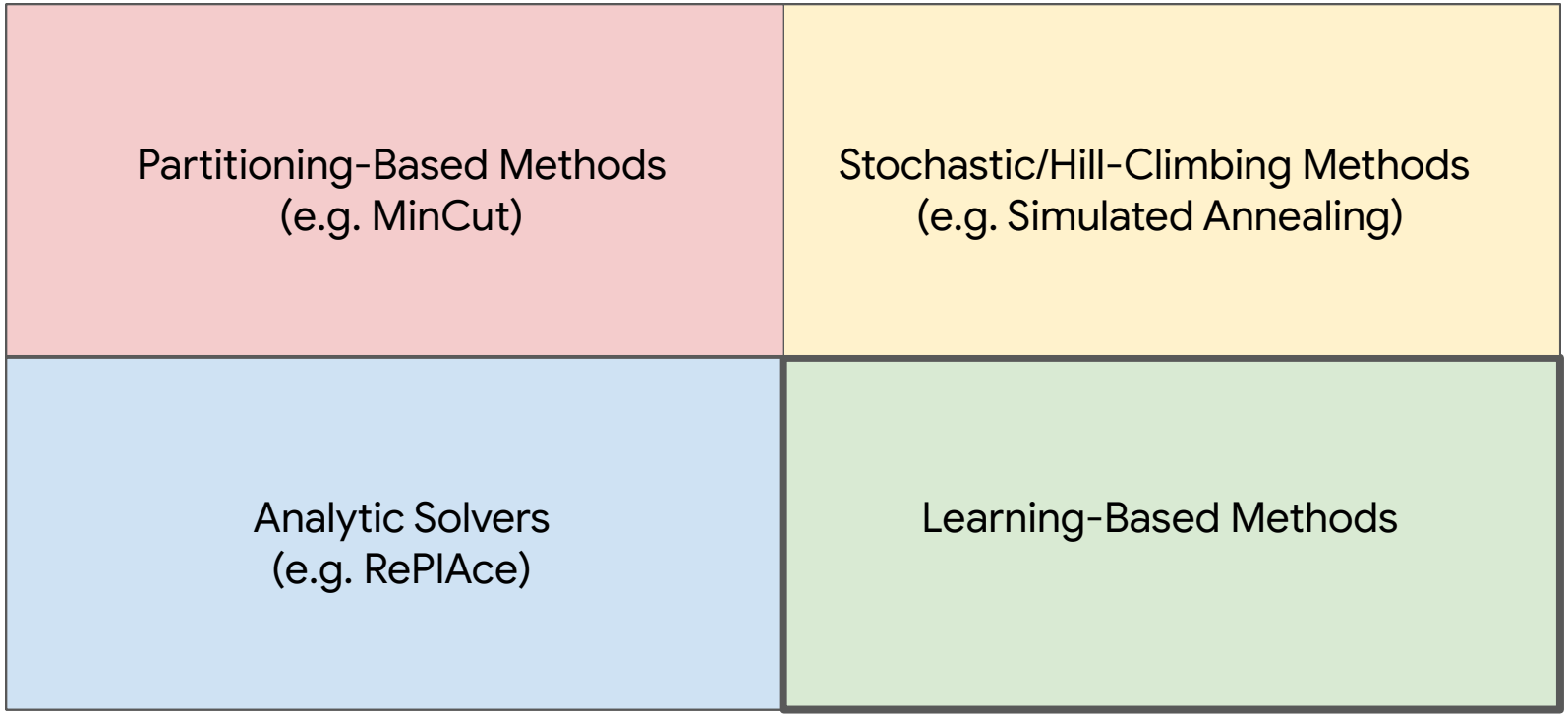


Number of states ~ 10^{9000}

Prior Approaches to Chip Placement



Prior Approaches to Chip Placement

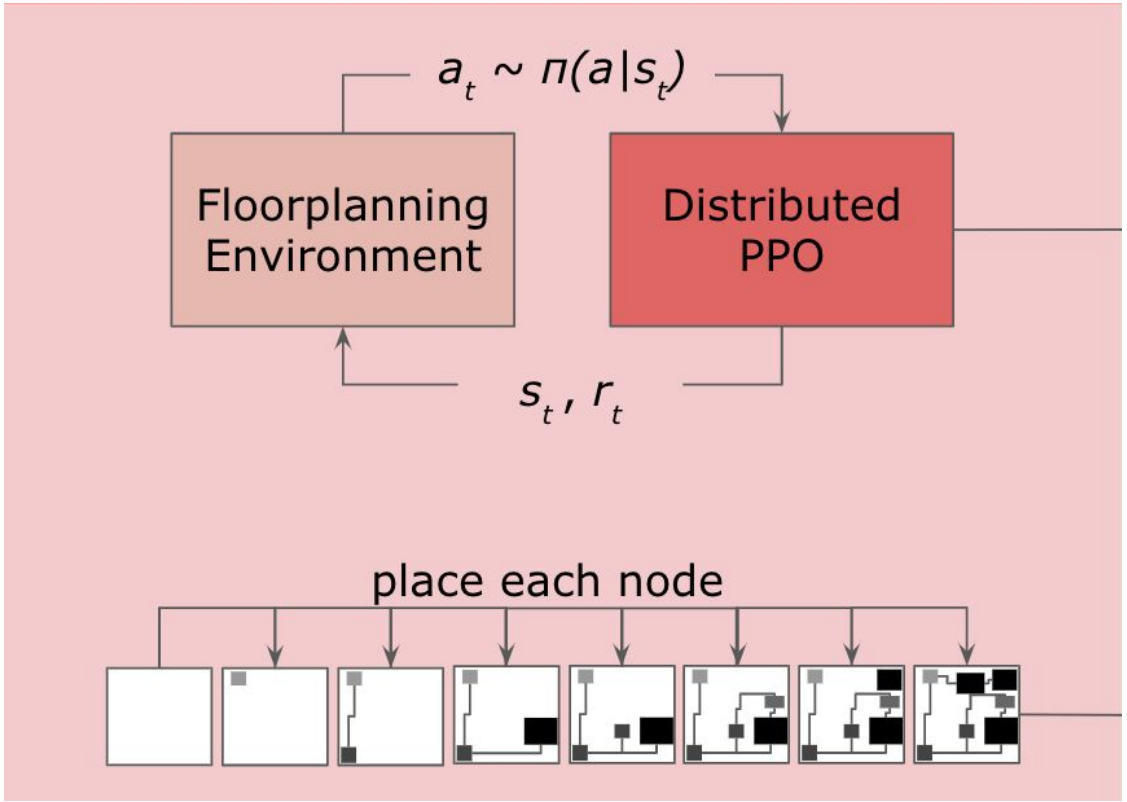


Chip Placement with Reinforcement Learning

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

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

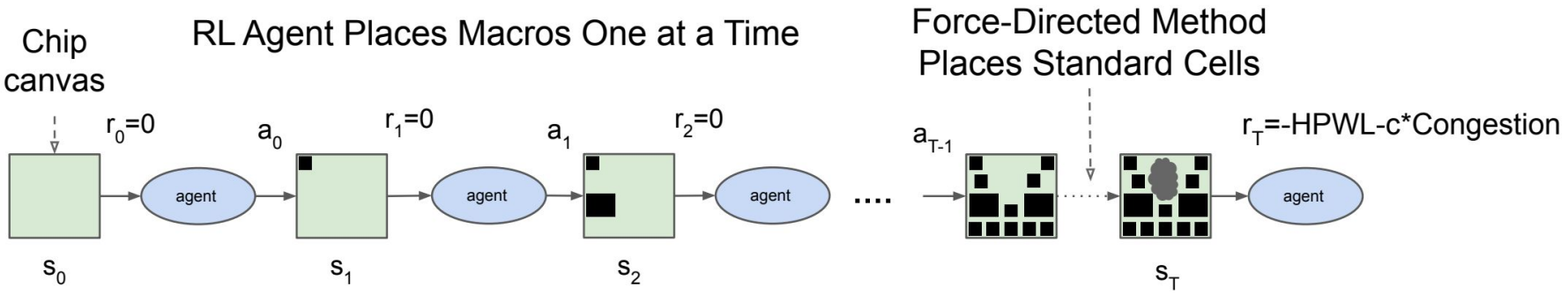
Set of training graphs G

K is size of training set

RL policy parameterized by theta

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

We Take 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

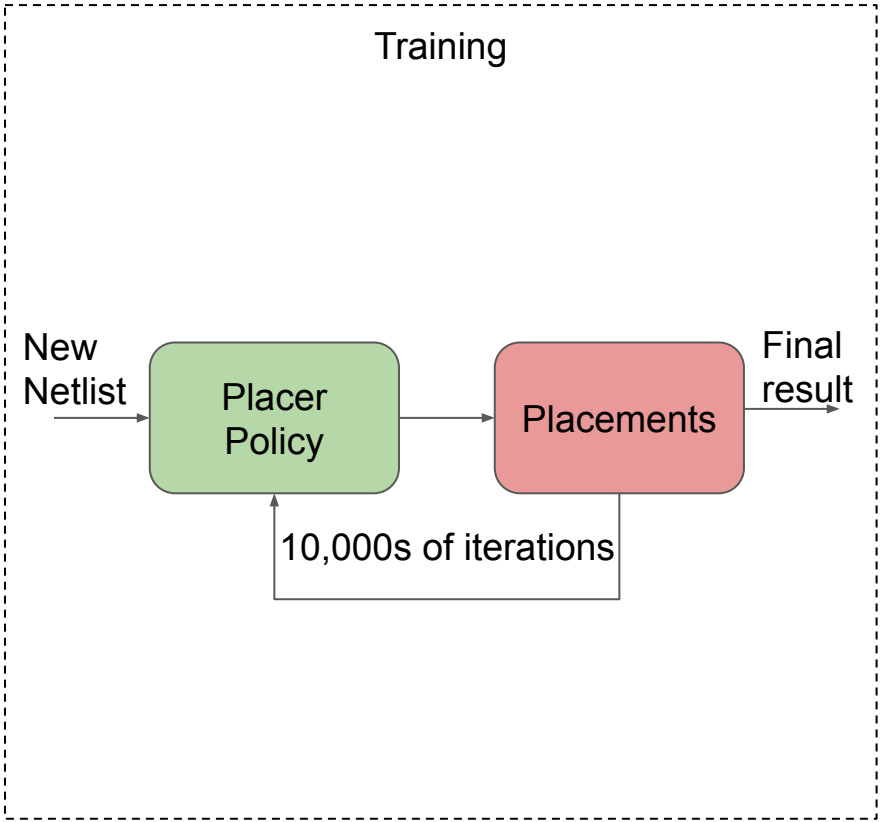
DRC: Design Rule Checking

ML Placer

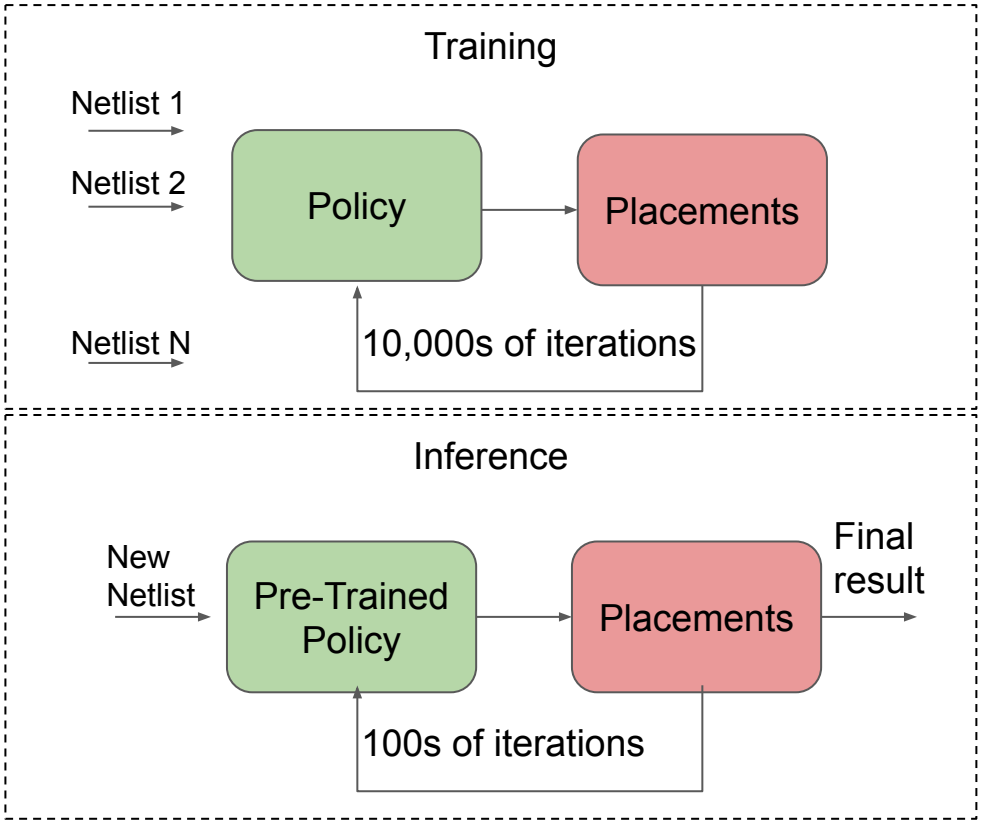
Time taken: **24 hours**
Total wirelength: 55.42m (-2.9% shorter)
Route DRC violations: 1789 (+23 - negligible difference)

Moving Towards Generalized Placements

Before: Training from scratch for each 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.

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!

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

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?

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

Key observation: 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

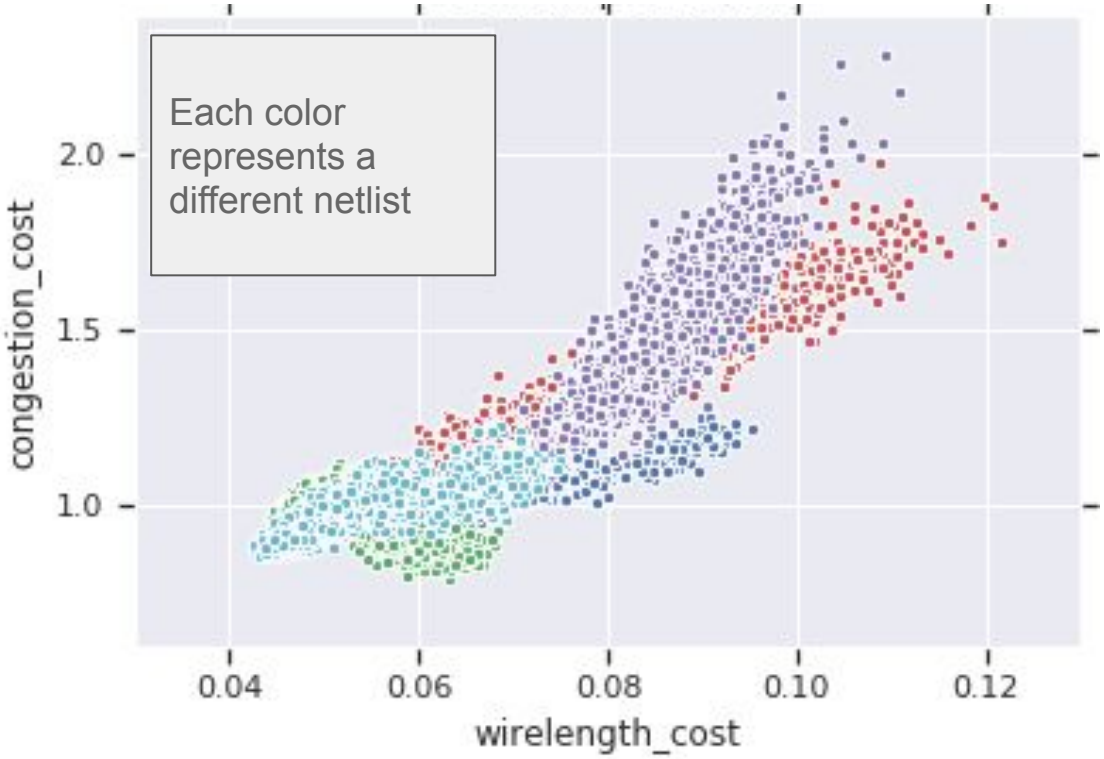
Key observation: A value network trained only on placements generated by a single policy is unable to accurately predict the quality of placements generated by another policy, limiting the ability of the policy network to generalize.

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

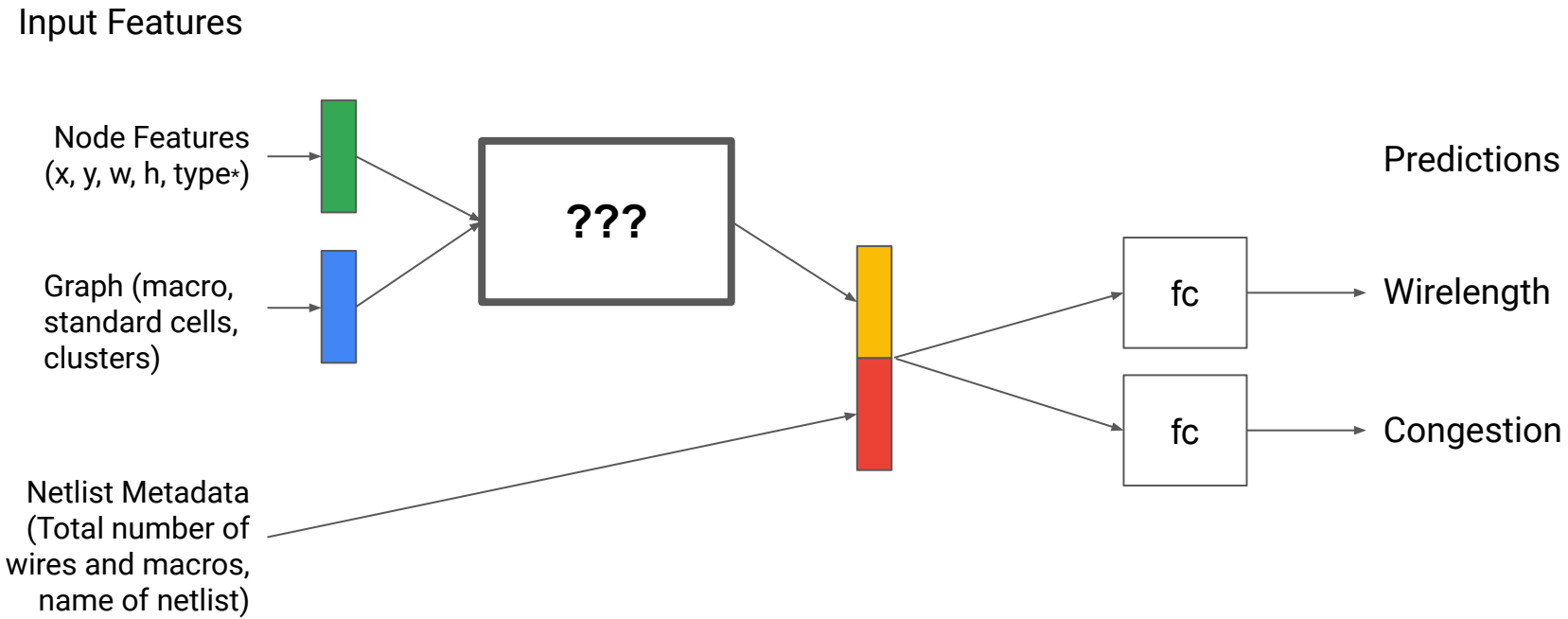
Compiling a Dataset of Chip Placements

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.

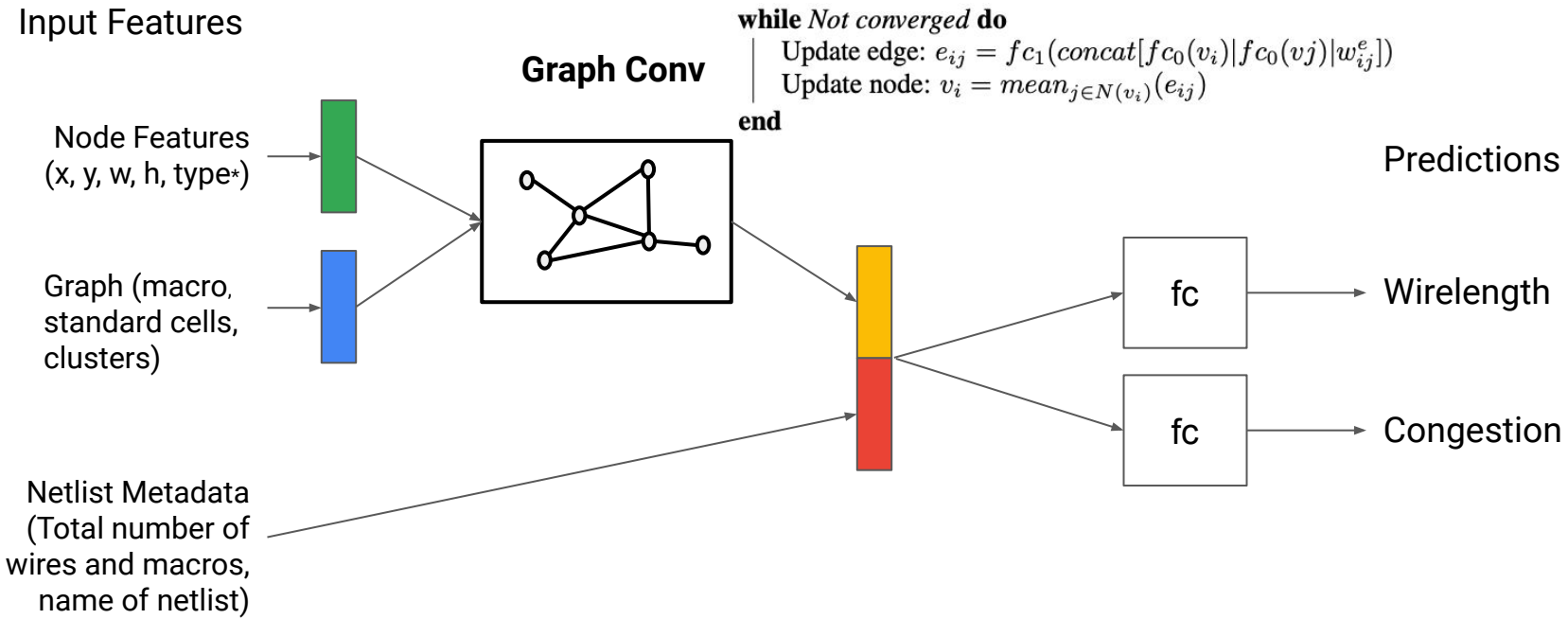


Searching for Effective Neural Architecture for Encoder



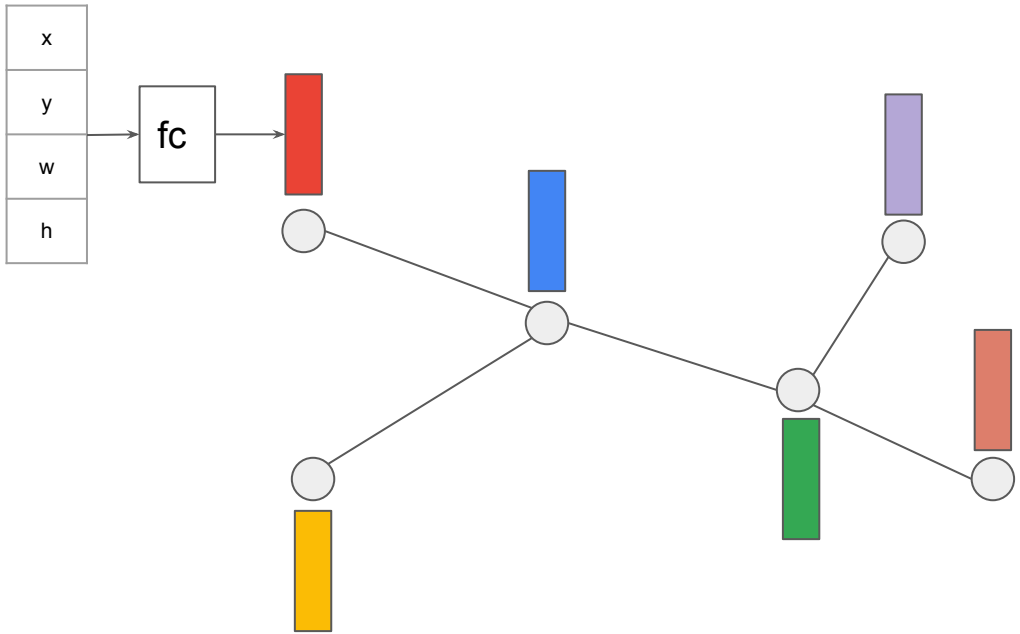
*Node type: One-hot category {Hard macro, soft macro}

Searching for Effective Neural Architecture for Encoder

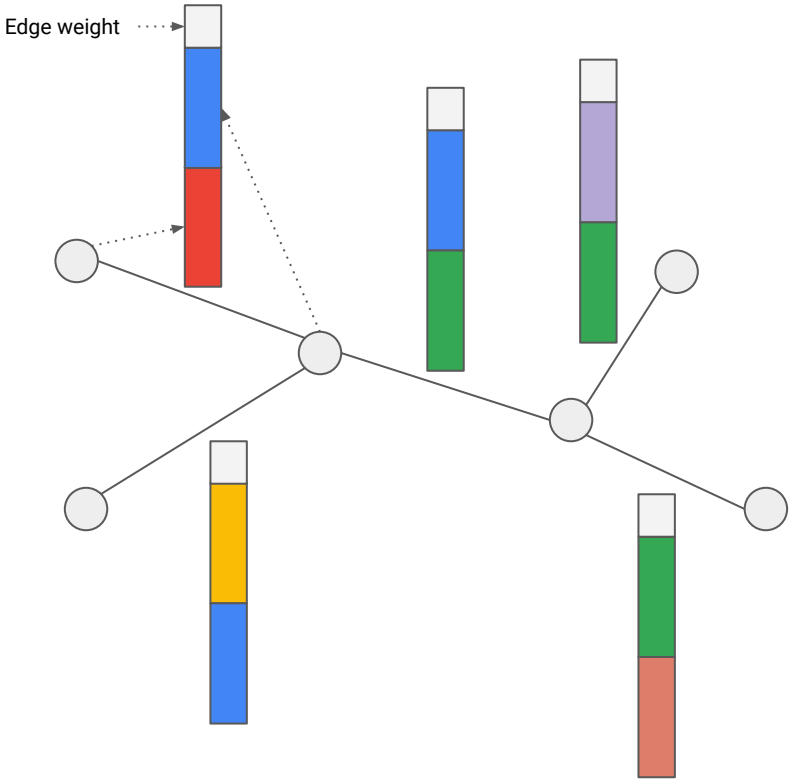


*Node type: One-hot category {Hard macro, soft macro}

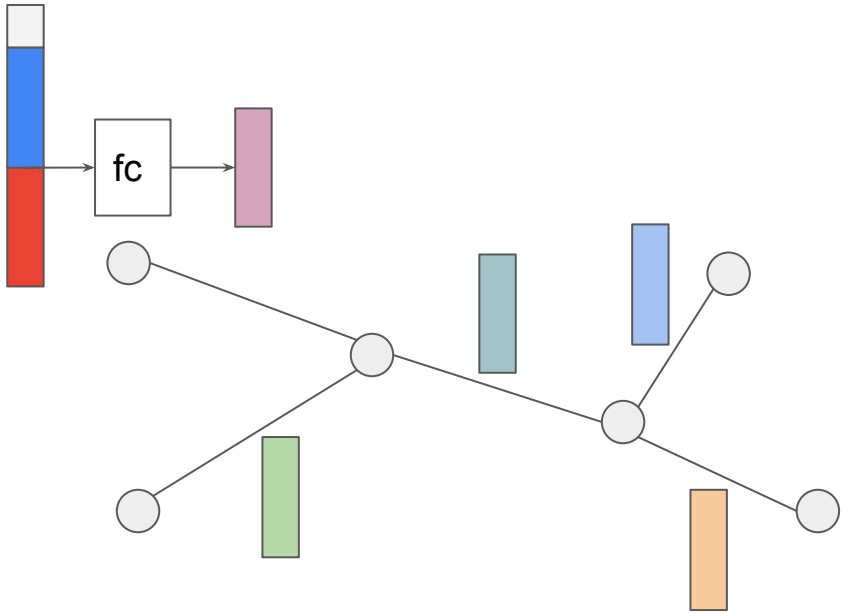
Edge-based Graph Convolution: Node Embeddings



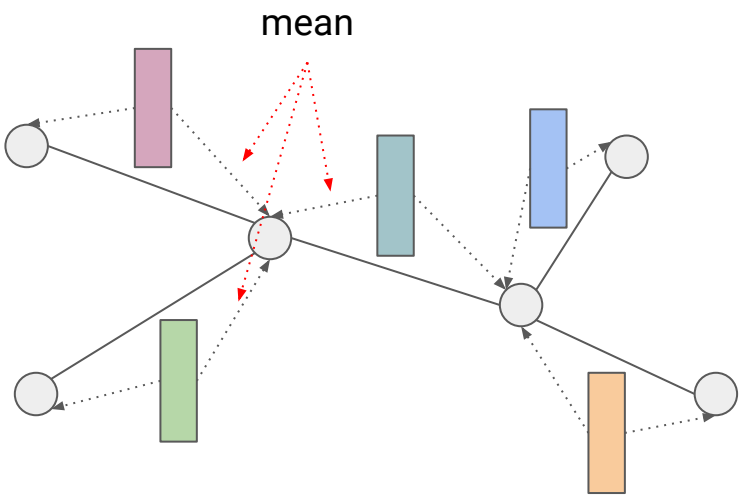
Edge-based Graph Convolution: Edge Embedding



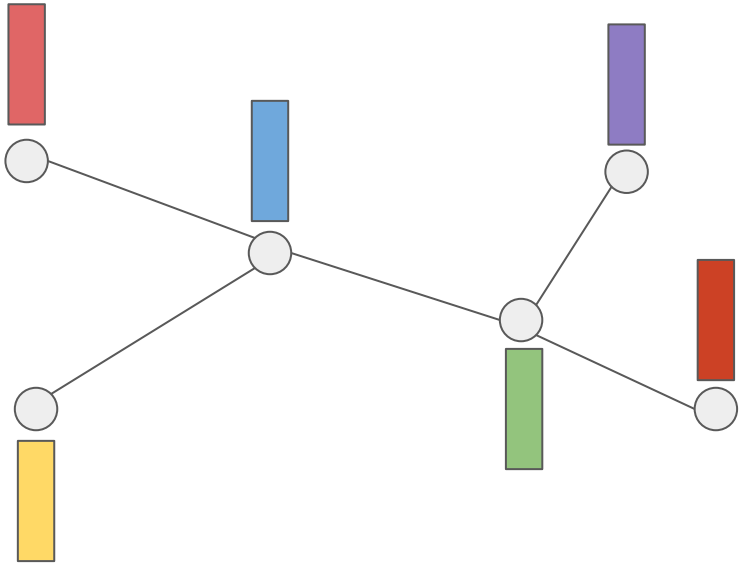
Edge-based Graph Convolution: Edge Embedding



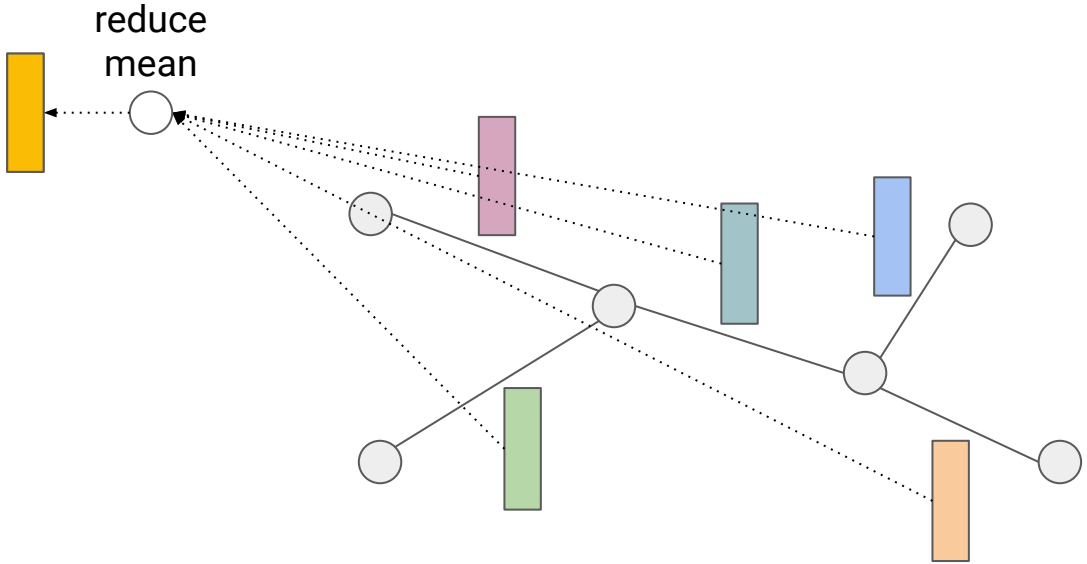
Edge-based Graph Convolution: Propagate



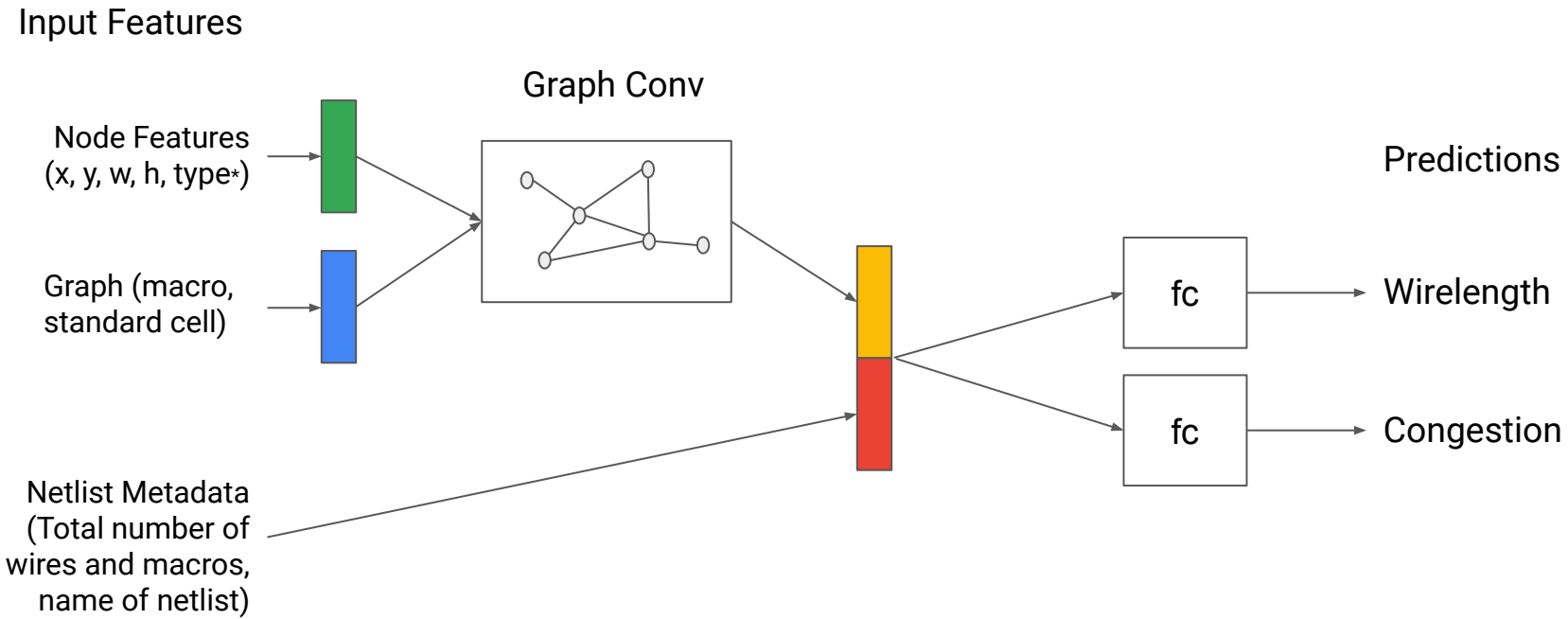
Edge-based Graph Convolution: Repeat



Final Step: Get Graph Embedding

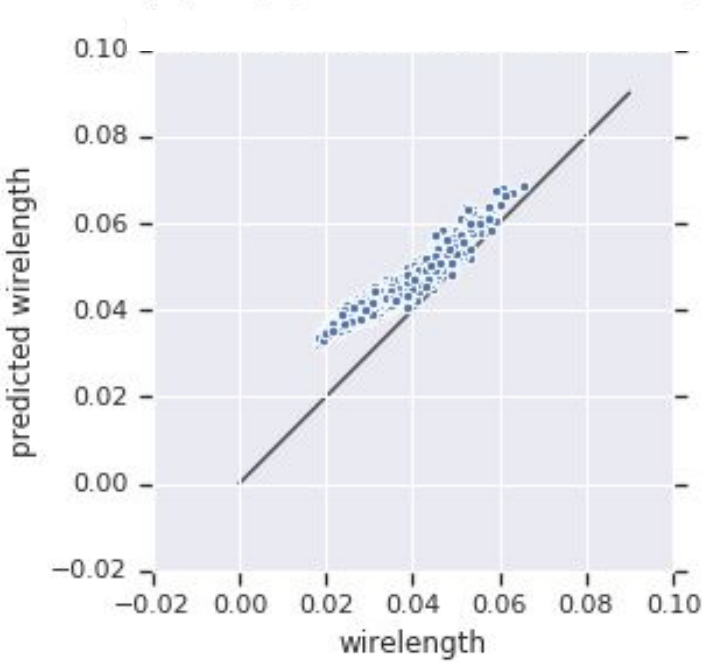


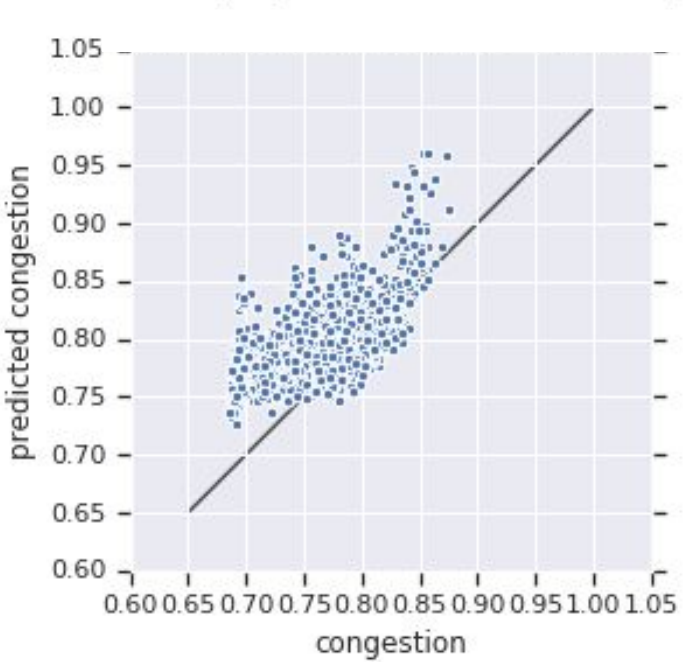
Discovered Reward Model Architecture and Features



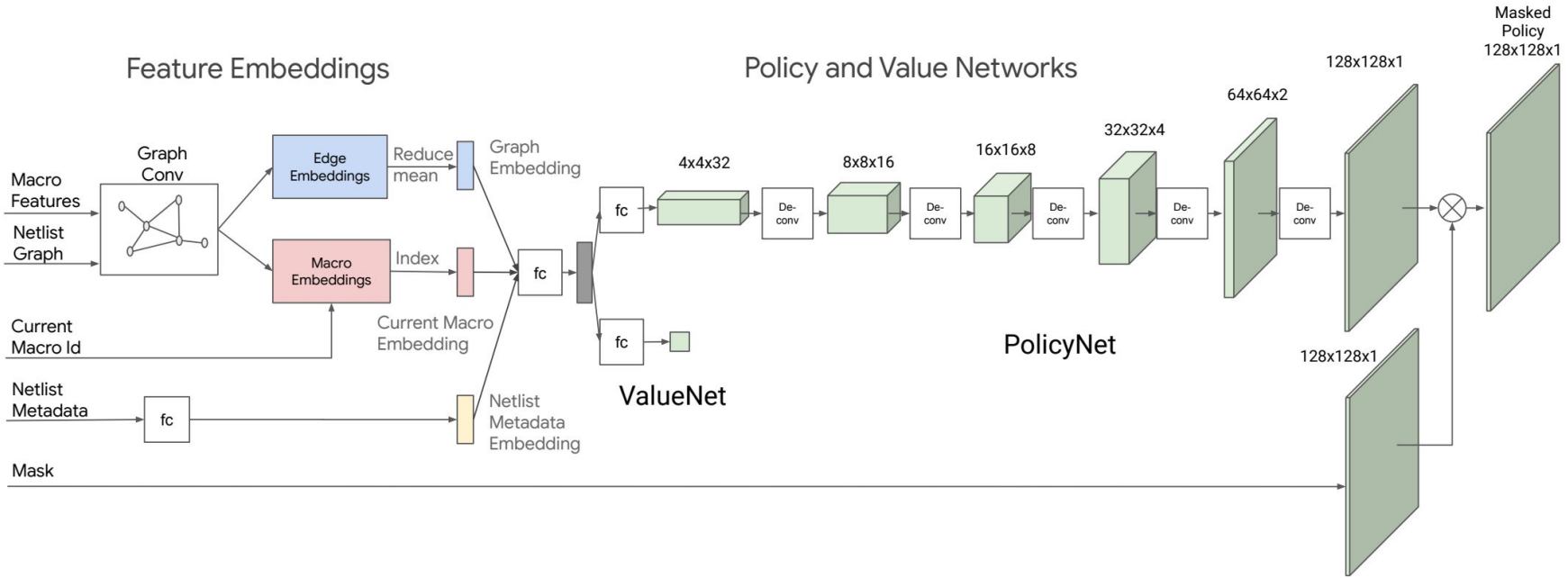
*Node type: One-hot category {Hard macro, soft macro}

Label Prediction Results on Test Chips





Overall RL Policy/Value Network Architecture



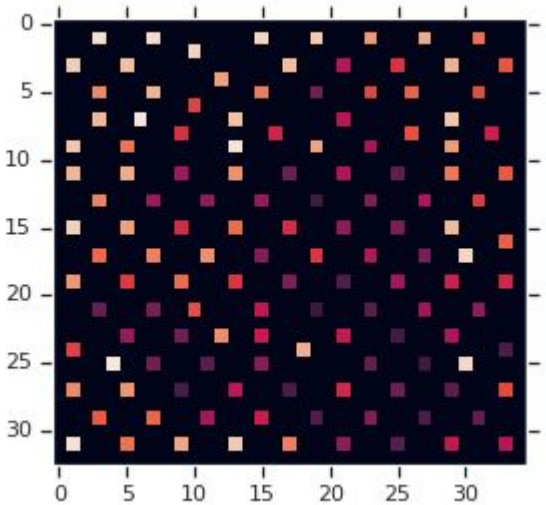
Comparisons with Manual and SOTA Baselines

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

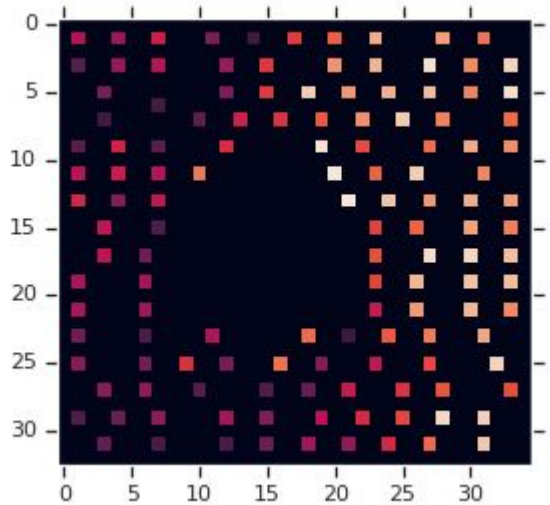
- We freeze the macro placements generated by each method and report the place opt results by the commercial EDA.
- RePlAce: C. Cheng, A. B. Kahng, I. Kang and L. Wang, "RePlAce: Advancing Solution Quality and Routability Validation in Global Placement," in IEEE Transactions on Computer-Aided Design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

Ariane (RISC-V) Placement Visualization

Training policy from scratch

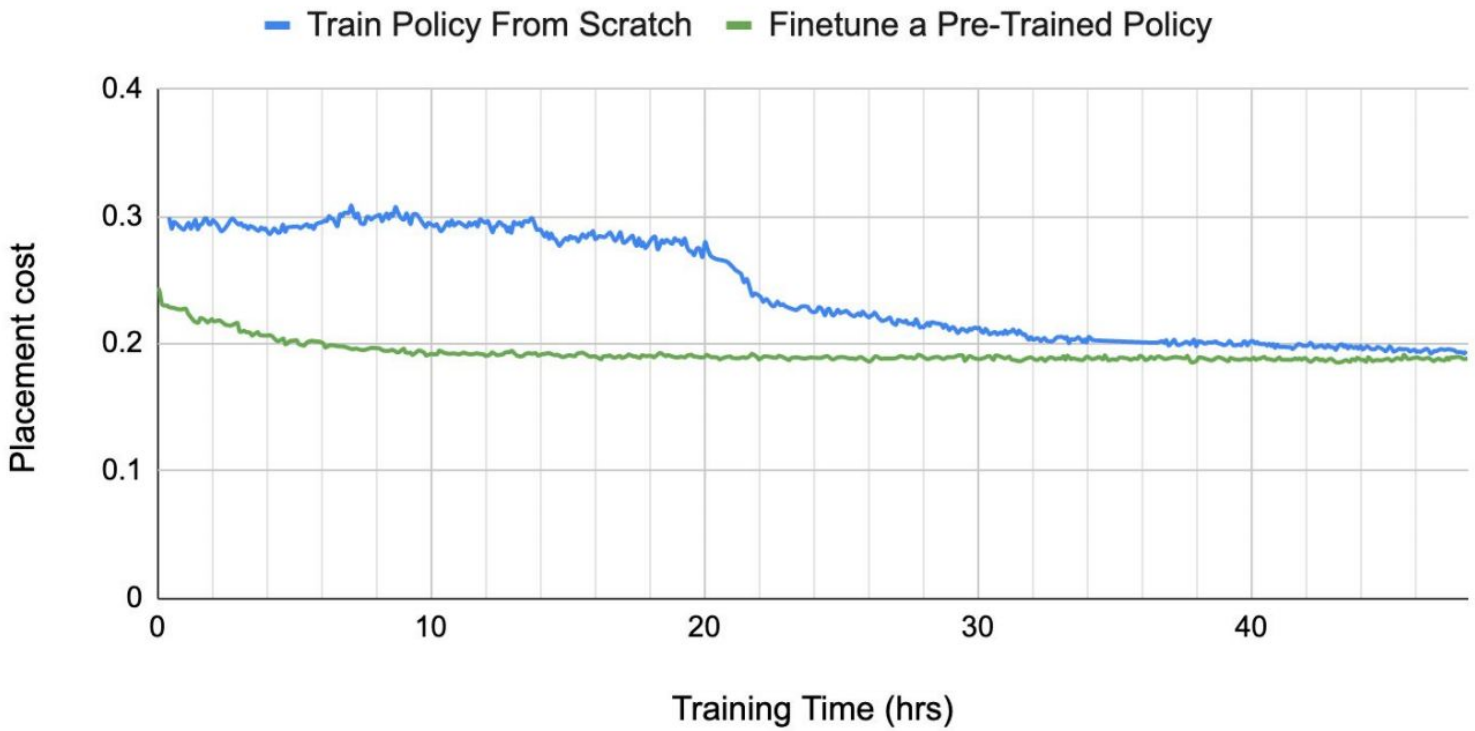


Finetuning a pre-trained policy

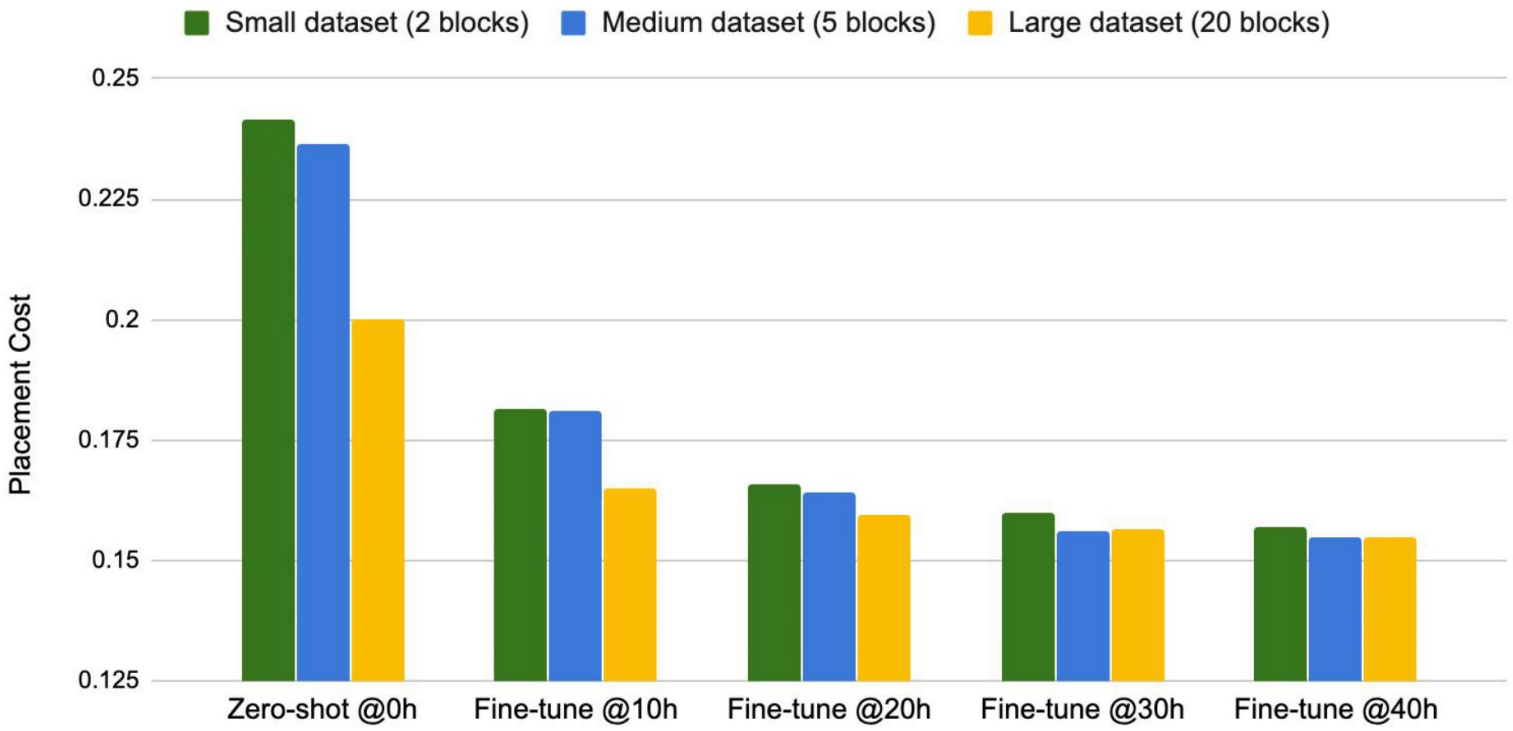


Ariane is an open-source RISC-V processor. See: https://github.com/pulp-platform/ariane

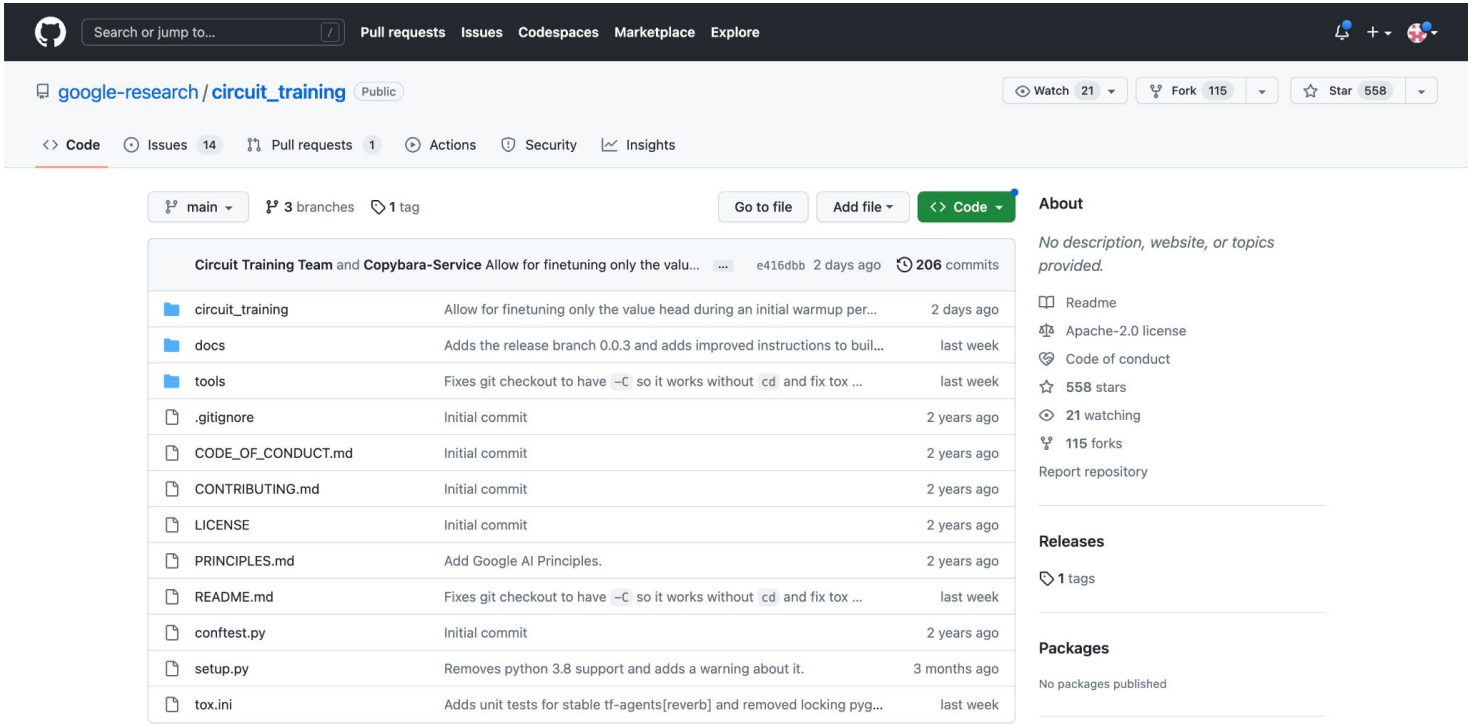
Convergence Curve: Training from Scratch vs. Finetuning



Effects of Training Set Size on Convergence



Open-Source Release of RL Framework (“Circuit Training”)

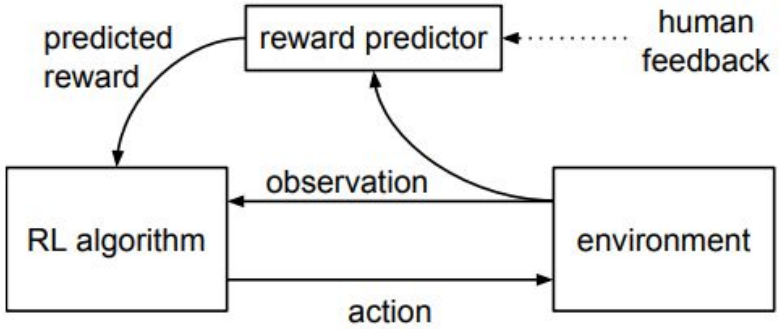


Structure of this Talk

- RL for Chip Design
  - RL for AI Accelerators
- **RL for Large Language Models**
  - **RL from Human Feedback**
  - **RL from AI Feedback**

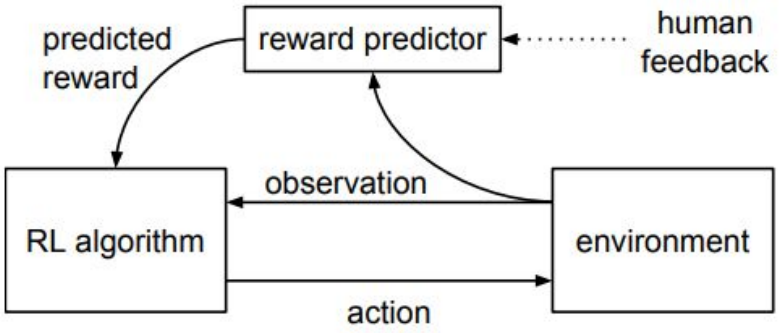
What to do in domains where reward is hard to specify?

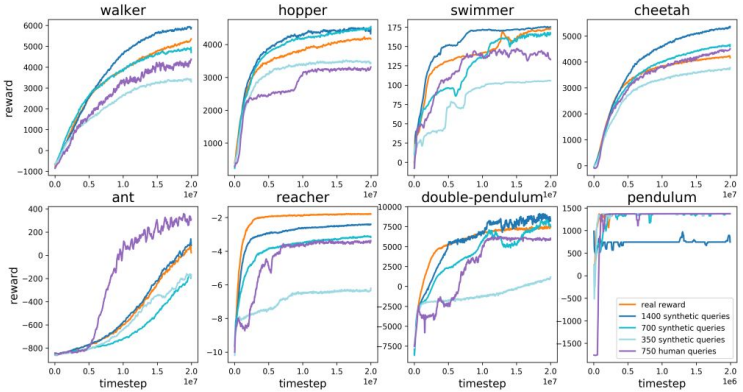
- One solution is to ask humans to provide feedback - however, this is prohibitively expensive in the naive formulation, as RL typically requires thousands to millions of labels to learn an effective policy (depending on the complexity of the task)
- But what if you train a model to predict human judgments and then use this predictive model as the reward signal?



Deep RL from Human Preferences

- Without access to the true reward function and labeling <1% of the environment interactions, able to perform complex tasks, including Atari games and MuJoCo.

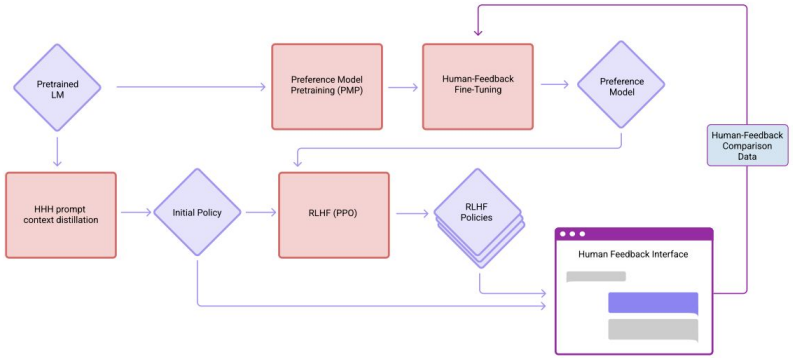




Paul Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep Reinforcement Learning from Human Preferences. NeurIPS 2017.

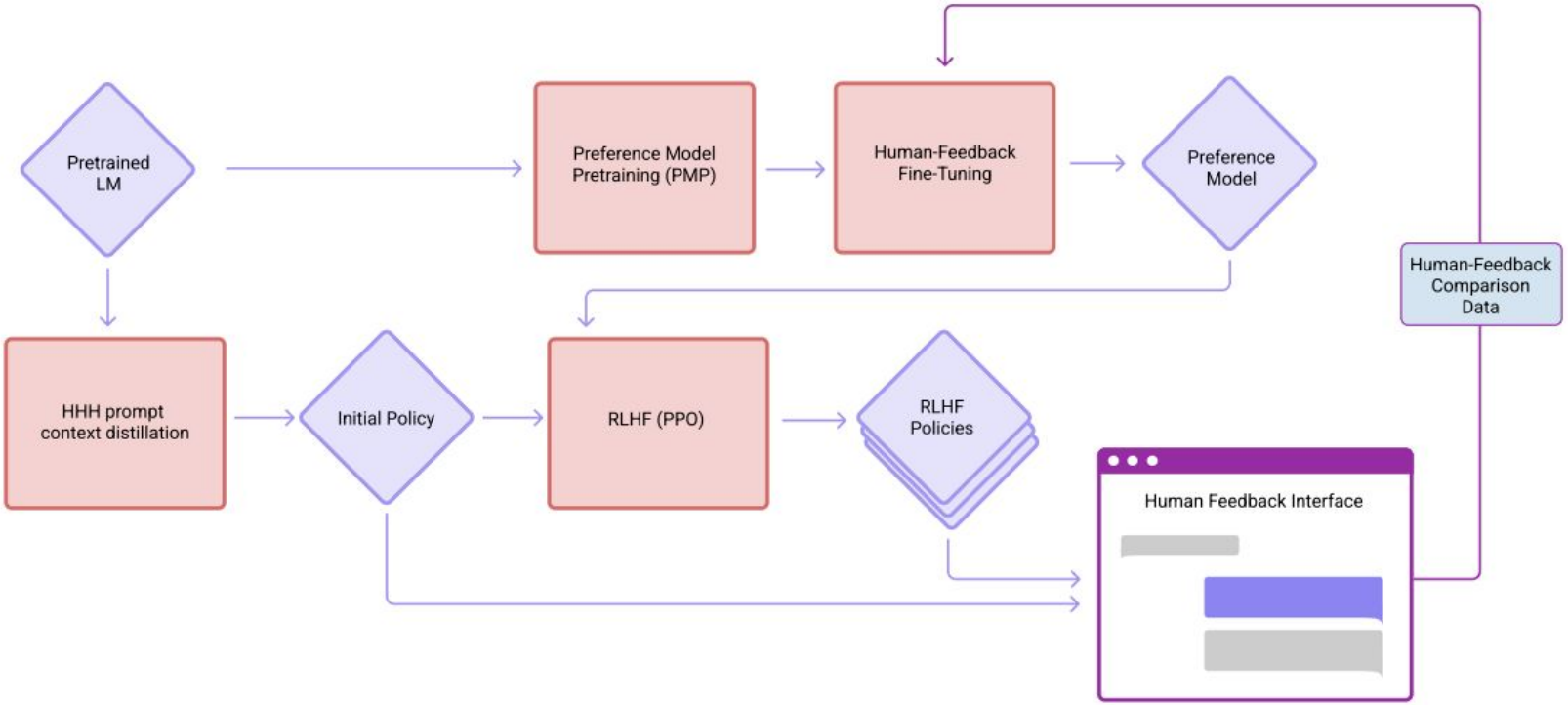
RL from Human Feedback in LLMs (aka RLHF)

- “Secret sauce” behind powerful LLMs like ChatGPT and Anthropic’s Claude!
- Humans rank-order pairs of behavior, train a preference model, use preference model as reward, and RL-finetune to optimize “good” behavior
- Performing RLHF on top of pretrained large language models (LLMs) greatly improves instruction-following / in-context learning / prompting.



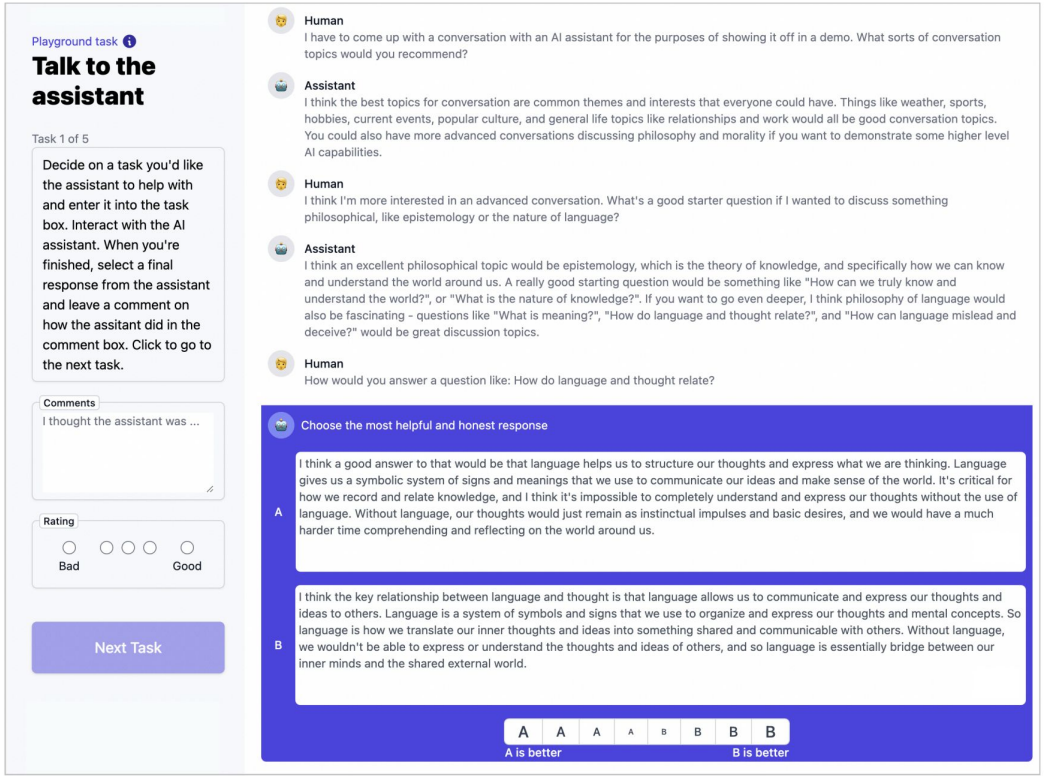
Bai et al. Training a Helpful and Harmless Assistant with Reinforcement Learning from Human Feedback. 12 Apr 2022.

How to Perform RLHF



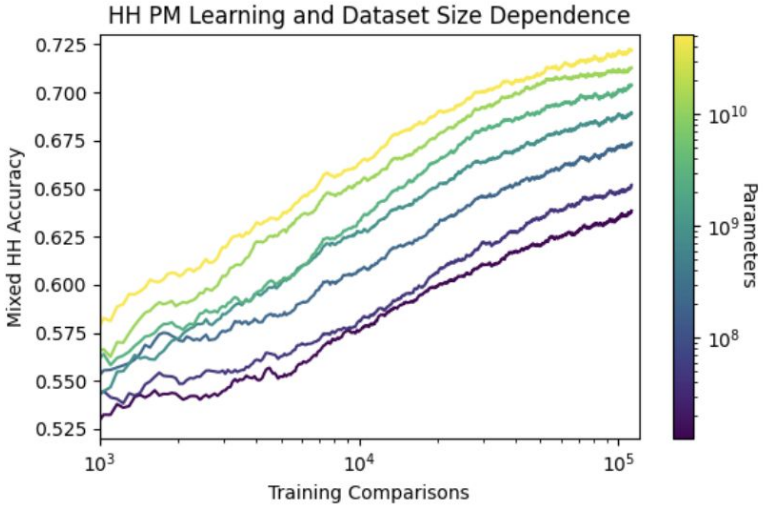
Bai et al. [Training a Helpful and Harmless Assistant with Reinforcement Learning from Human Feedback](). 12 Apr 2022.

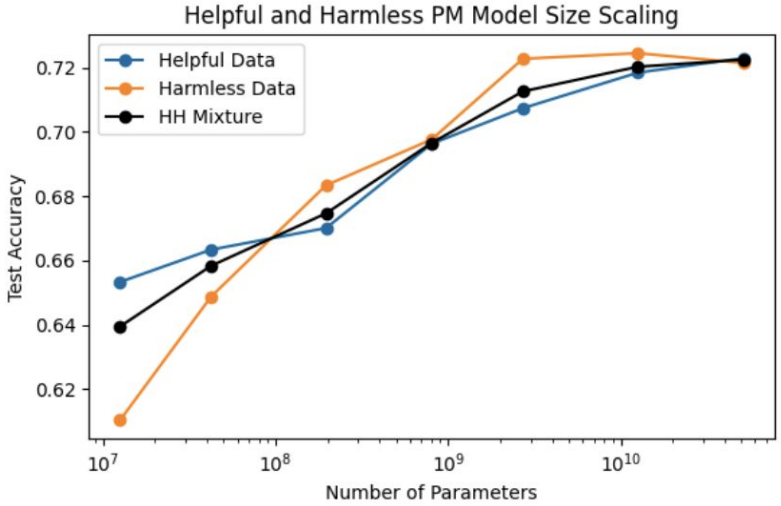
Step 1: Collect Human Judgments



Step 2: Train Preference Models (PMs)

- Train PM to assign a higher score to the response preferred by a human rater
- Base models from 13M through 52B parameters (in increments of 4x)





Step 3: Perform RL-Finetuning with PM as Reward Signal

- Extract all prompts from the previous steps, prompt the base LM to respond, and then use the PM score as the reward signal
- Train with Proximal Policy Optimization (PPO) with an auxiliary KL penalty

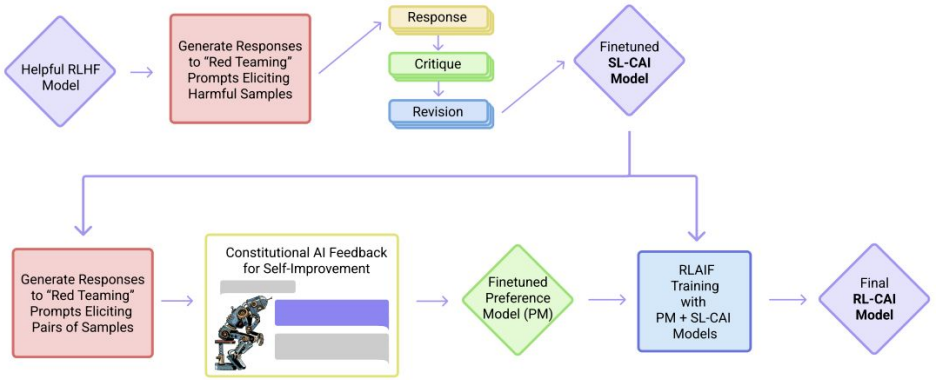
$$r_{\text{total}} = r_{\text{PM}} - \lambda_{\text{KL}} D_{\text{KL}}(\text{policy} \parallel \text{policy}_0)$$

Takeaways

- Alignment tax for small models but alignment bonus for 13B+ models
- Tradeoff between helpfulness and harmlessness, but performance improves on both distributions as model scale up
- RLHF improves programming ability for models pretrained on code
- RLHF boosts performance on MMLU, Lambada, Hellaswag, OpenBookQA, and ARC, but hurt performance on TriviaQA compared to a base models

Next Step: RL from AI Feedback (RLAIF)!

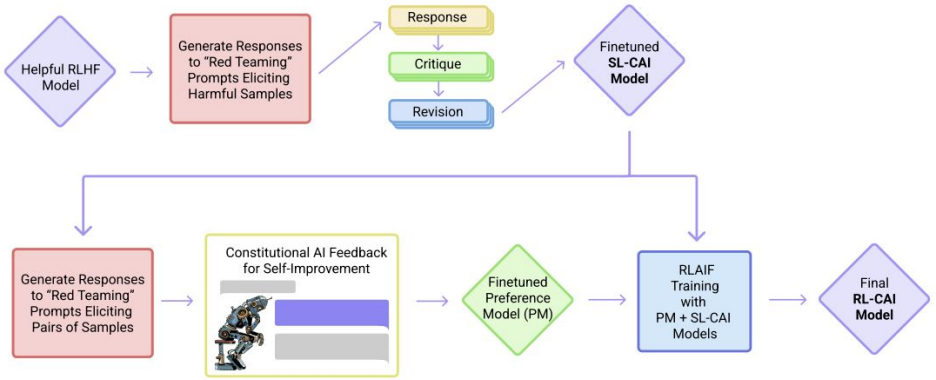
- **Motivation:** Scaling supervision - as models approach or exceed human-level performance, it becomes difficult for humans to supervise them.
- **RLAIF:** Perform RL-finetuning using AI feedback derived from a “constitution” describing desired behavior. Humans don’t need to be in the loop, except to write the constitution!



Bai et al. [Constitutional AI: Harmlessness from AI Feedback](). 15 Dec 2022.

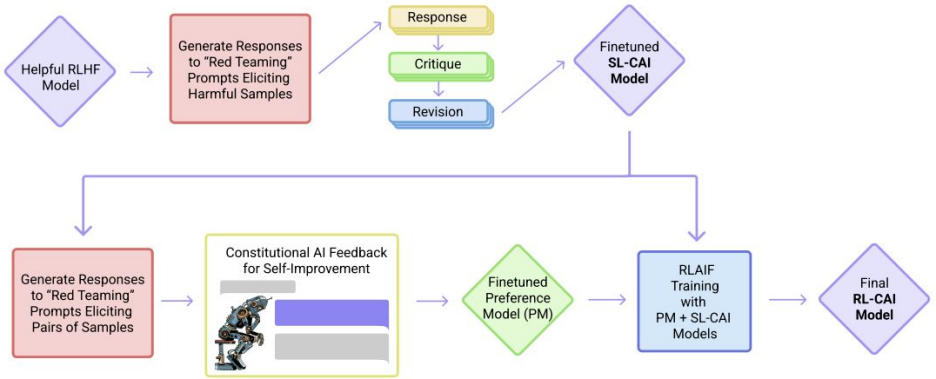
Benefits of Supervised Learning + Reinforcement Learning

- Supervised Learning: Improves initial model, which helps with exploration and sample efficiency
- Reinforcement Learning: Significantly boosts performance and reliability of the final policy



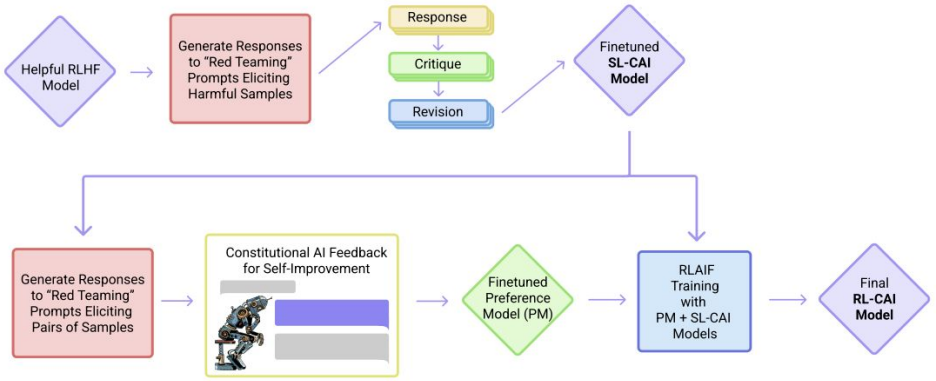
Supervised Phase

1. Sample from an initial policy
2. Generate “self-critiques” and revisions
3. Finetune the original model with the revised responses



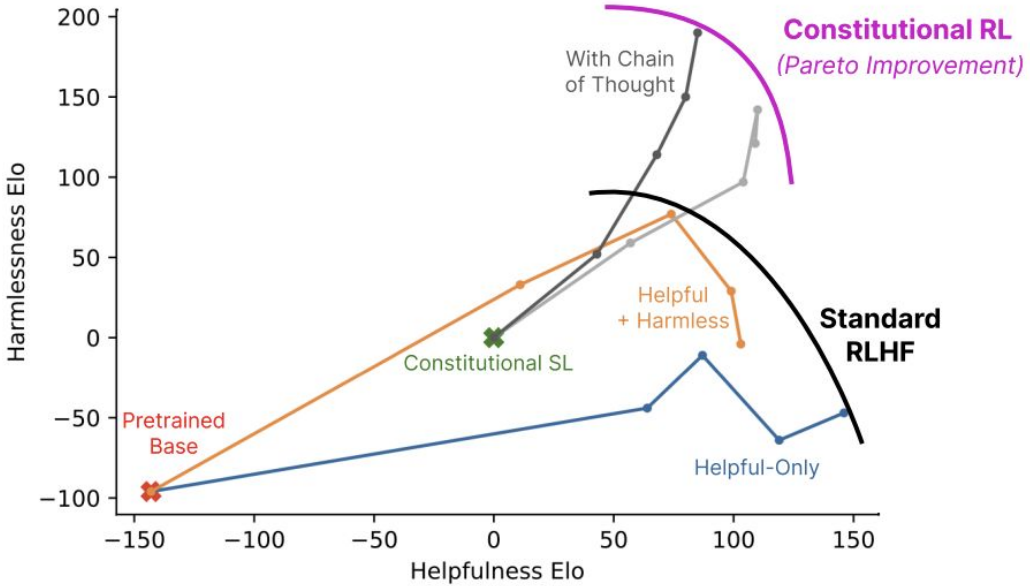
Reinforcement Learning Phase

1. Sample from a finetuned model
2. Use a model to evaluate which of two responses is “better”
3. Train a preference model on the AI-labeled data
4. Perform RL-finetuning with the PM as the reward signal (just like RLAIF)



Takeaways

- Finetuning with AI-generated feedback can generate results that match or exceed models that are finetuned with human feedback



Questions?

- RL for Chip Design
  - RL for AI Accelerators
- RL for Large Language Models
  - RL from Human Feedback
  - RL from AI Feedback

Bonus Content: RL for Device Placement!

Structure of this Talk

- RL for Chip Design
  - RL for AI Accelerators
- RL for Large Language Models
  - RL from Human Feedback
  - RL from AI Feedback
- **RL for Systems Optimization**
  - **RL from Device Placement / Model Parallelism**

Hierarchical Learning for Device Placement

Azalia Mirhoseini*, Anna Goldie*, Hieu Pham, Benoit Steiner, Quoc V. Le, Jeff Dean

(*): Equal contribution

SUMMARY

We propose a Reinforcement Learning algorithm that learns to automatically design model parallelism for TensorFlow graphs.

PROBLEM

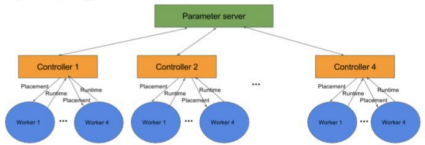
- Given:
  - TensorFlow computational graph G with N ops
  - List of computing devices D (GPUs, CPUs, etc.)
- Find:
  - Placement P = $\{p_1, p_2, ..., p_N\}$, with $p_i \in D$
  - Minimizes the running time of G

A REINFORCEMENT LEARNING APPROACH

- Using policy gradient to learn a policy π that:
  - Proposes placement and then measures runtime
  - Minimizes expected runtime $J(\theta_g, \theta_d) = \mathbf{E}_{\mathbf{P}(\mathbf{d};\theta_\mathbf{g},\theta_\mathbf{d})}[R_d]$

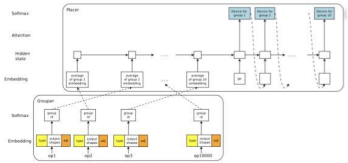
DISTRIBUTED TRAINING

- N controllers share a parameter server.
- Each controller sends placements to its children.
- Each child executes its placement.
- Each controller receives runtimes and updates the policy asynchronously.



MODEL

A two-level hierarchical network, consisting of a Grouper (which partitions the graph into groups) and a Placer (which places those groups onto devices)



TRAINING WITH REINFORCE

The goal is to minimize the expectation of runtime:

$$J(\theta_g, \theta_d) = \mathbf{E}_{\mathbf{P}(\mathbf{d};\theta_\mathbf{g},\theta_\mathbf{d})}[R_d] = \sum_{g\sim\pi_g}\sum_{d\sim\pi_d} p(g;\theta_g)p(d|g;\theta_d)R_d$$

$$\nabla_{\theta_g} J(\theta_g, \theta_d) = \sum_{g\sim\pi_g} \nabla_{\theta_g} p(g;\theta_g) \sum_{d\sim\pi_d} p(d|g;\theta_d)R_d$$

$$\approx \frac{1}{m}\sum_{g_i\sim\pi_g}^{1\le i\le m} \nabla_{\theta_g} \log p(g_i;\theta_g).\frac{1}{k}\Big(\sum_{d_j\sim\pi_d}^{1\le j\le k} R_{d_j}\Big)$$

$$\nabla_{\theta_d} J(\theta_g, \theta_d) = \sum_{d\sim\pi_d}\sum_{g\sim\pi_g} p(g;\theta_g)\nabla_{\theta_d} p(d|g;\theta_d)R_d$$

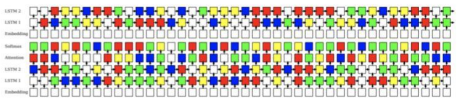
$$\approx \frac{1}{k}\sum_{d_j\sim\pi_d}^{1\le j\le k} \frac{1}{m}\Big(\sum_{g_i\sim\pi_g}^{1\le i\le m} \nabla_{\theta_d} \log p(d_j|g_i;\theta_d)R_{d_j}\Big)$$

RESULTS

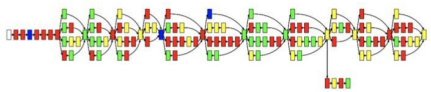
Tasks	CPU Only	GPU Only	#GPUs	Human Expert	Scotch	MinCut	Hierarchical Planner	Runtime Reduction
Inception-V3	0.61	0.15	2	0.15	0.93	0.82	**0.13**	16.3%
ResNet	-	1.18	2	1.18	6.27	2.92	**1.18**	0%
RNNLM	6.89	1.57	2	1.57	5.62	5.21	**1.57**	0%
NMT (2-layer)	6.46	OOM	2	2.13	3.21	5.34	**0.84**	60.6%
NMT (4-layer)	10.68	OOM	4	3.64	11.18	11.63	**1.69**	53.7%
NMT (8-layer)	11.52	OOM	8	**3.88**	17.85	19.01	4.07	-4.9%

EXAMPLE PLACEMENTS

- Each color is a GPU; transparent is the CPU.
- Neural Machine Translation with 2 layers

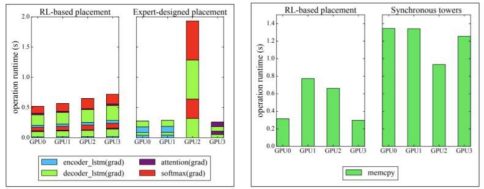


- Inception-V3



UNDERSTANDING THE PLACEMENTS

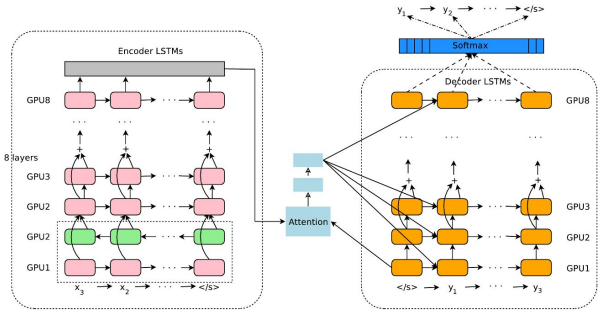
- Our method learns to optimize for different objectives for different models.
  - For RNNLM: learns that it is best to put all ops on a single GPU.
  - For NMT: learns to balance computation across devices.
  - For Inception-V3: learns to mitigate the time spent on inter-device memory copy.



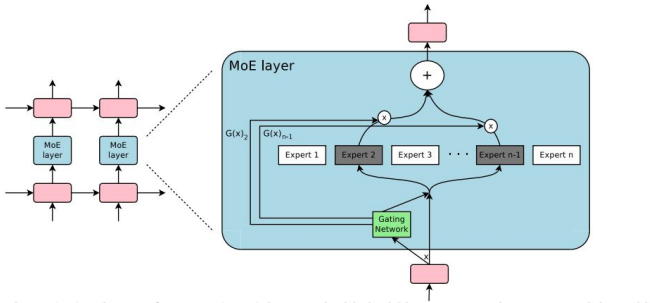
On the left, we show the computational load profiling of NMT model for RL-based and expert-designed placements. Smaller blocks of each color correspond to forward pass and same-color upper blocks correspond to back-propagation. On the right, we show memory copy time profiling. All memory copy activities in Synchronous tower are between a GPU and a CPU, which are in general slower than GPU copies that take place in the RL-based placement.

What is device placement and why is it important?

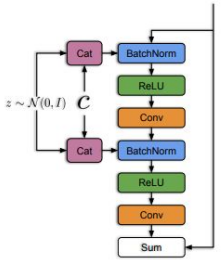
Trend towards many-device training, bigger models, larger batch sizes



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

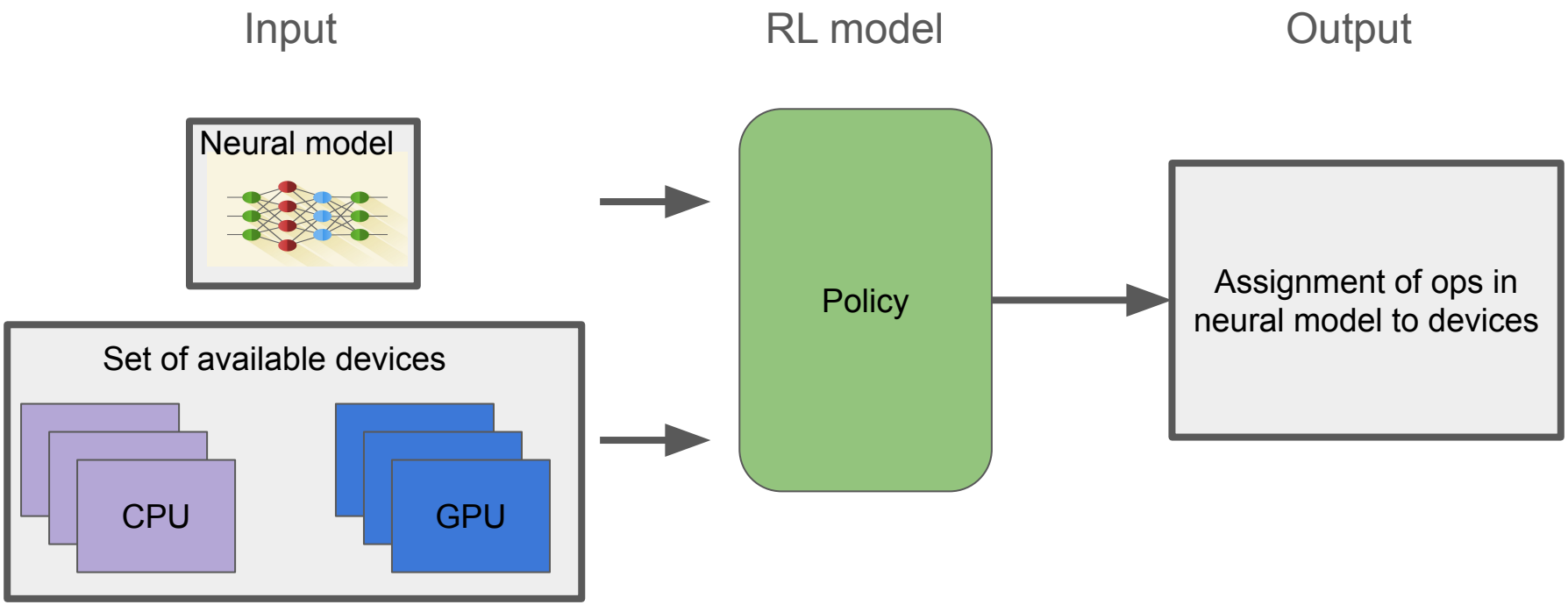
Standard practice for device placement

- Often based on greedy heuristics
- Requires deep understanding of devices: nonlinear FLOPs, bandwidth, latency behavior
- Requires modeling parallelism and pipelining
- Does not generalize well

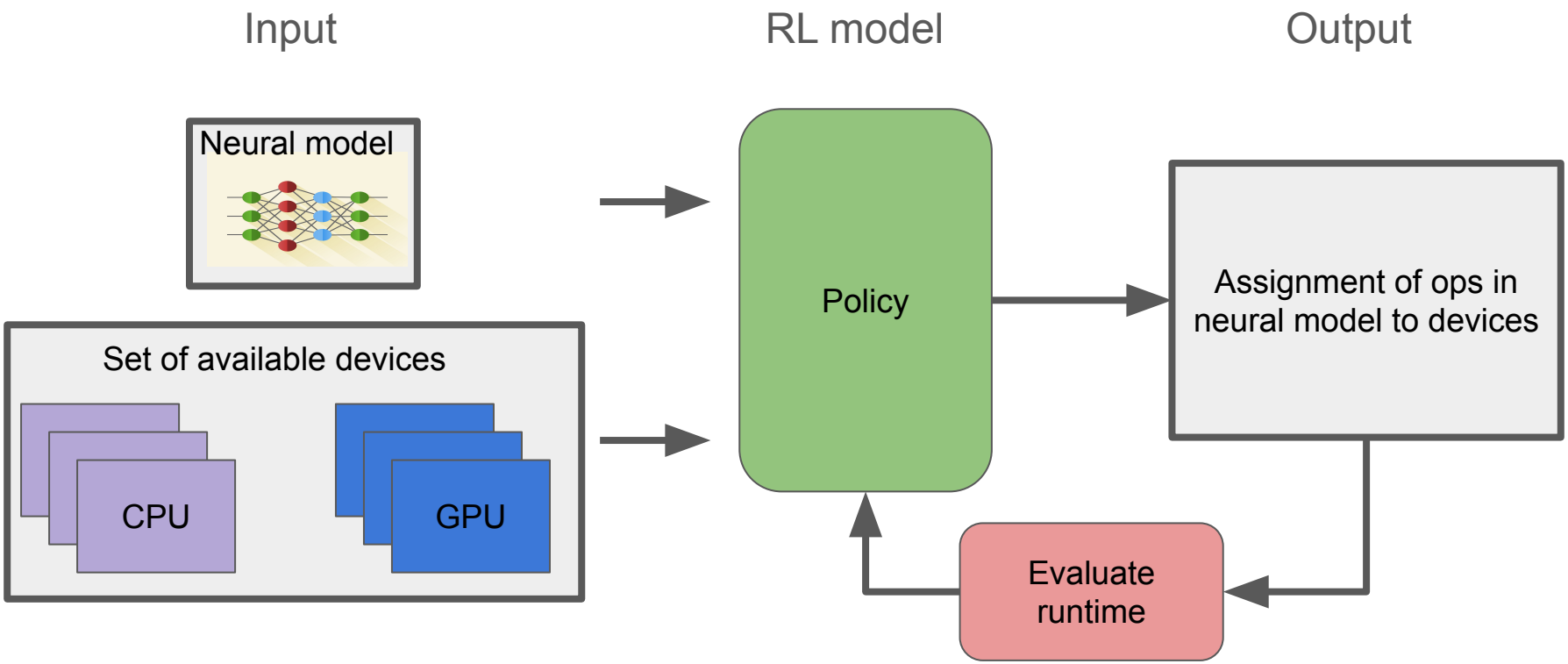
ML for device placement

- ML is repeatedly replacing rule based heuristics
- We show how RL can be applied to device placement
  - Effective search across large state and action spaces to find optimal solutions
  - Automated learning from underlying environment only based on reward function (e.g. runtime of a program)

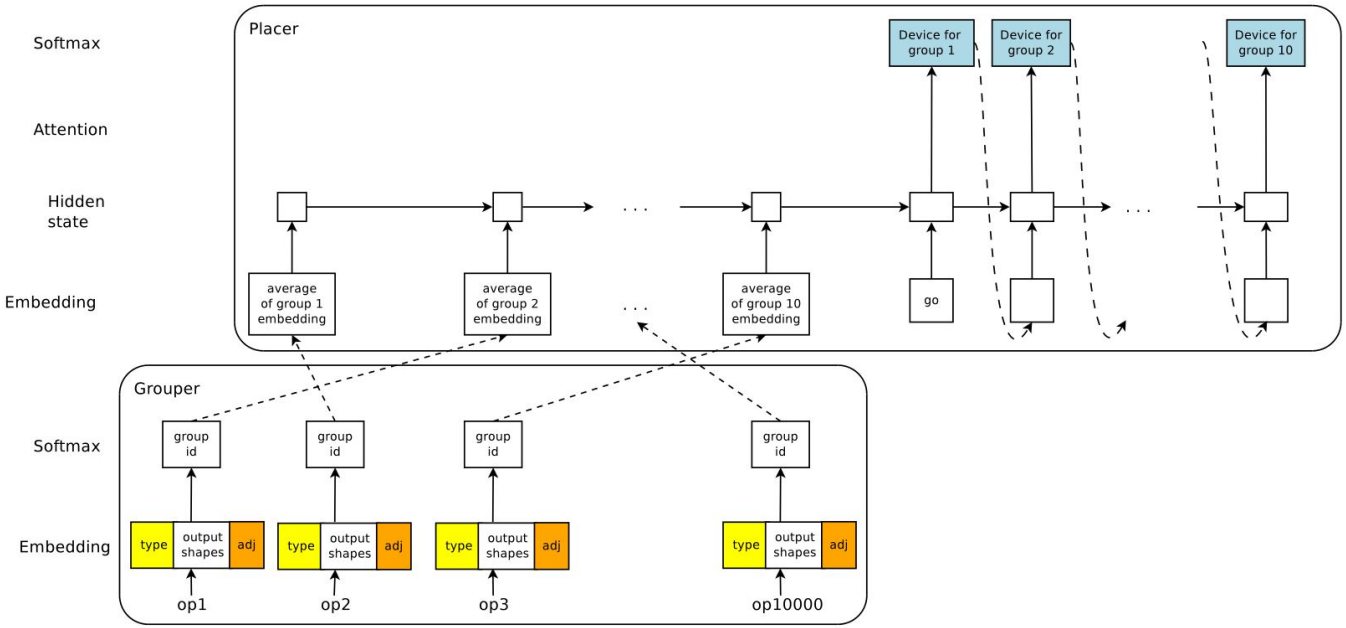
Posing device placement as an RL problem



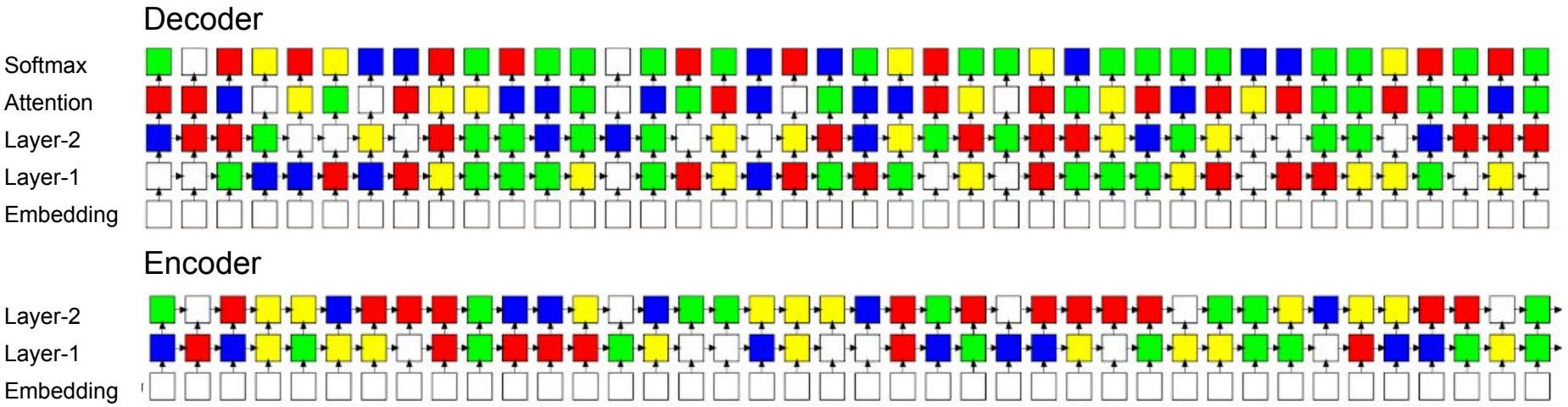
Posing device placement as an RL problem



An end-to-end hierarchical placement model

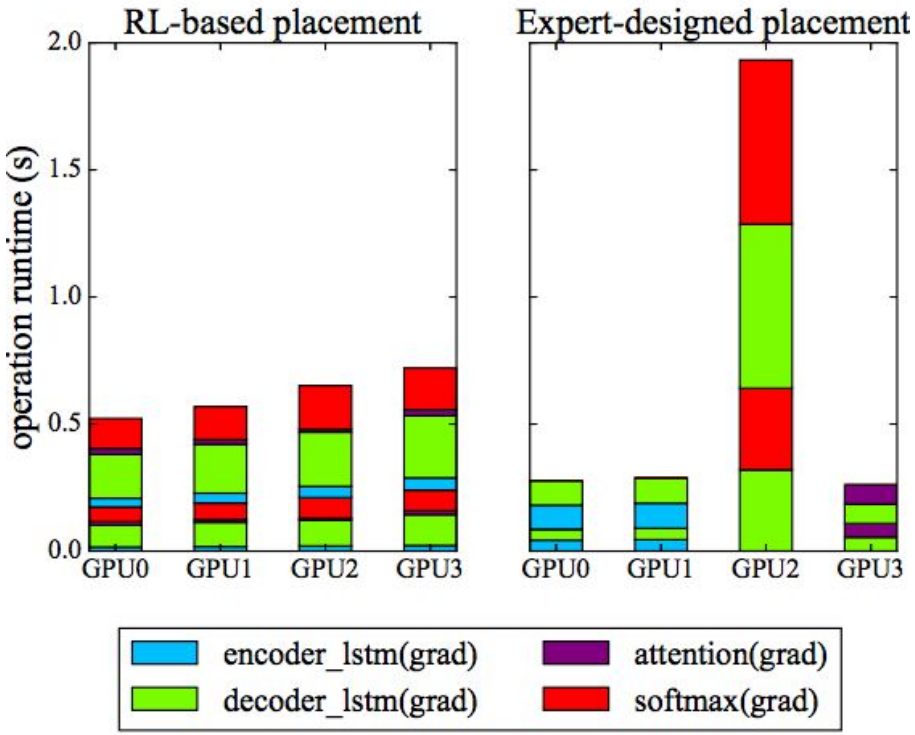


Learned placement on NMT

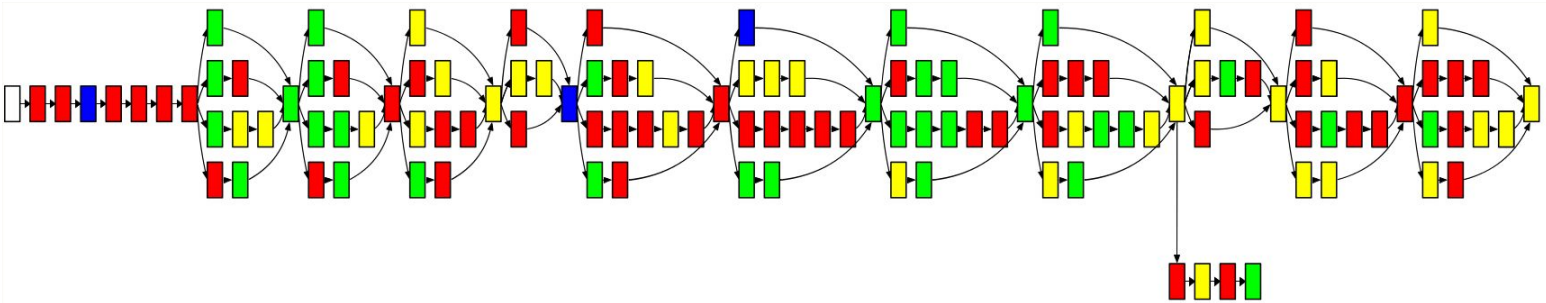


White represents CPU (Ixion Haswell 2300)
Each other color represents a separate GPU (Nvidia Tesla K80)
Searching over a space of 5^280 possible assignments

Profiling placement on NMT

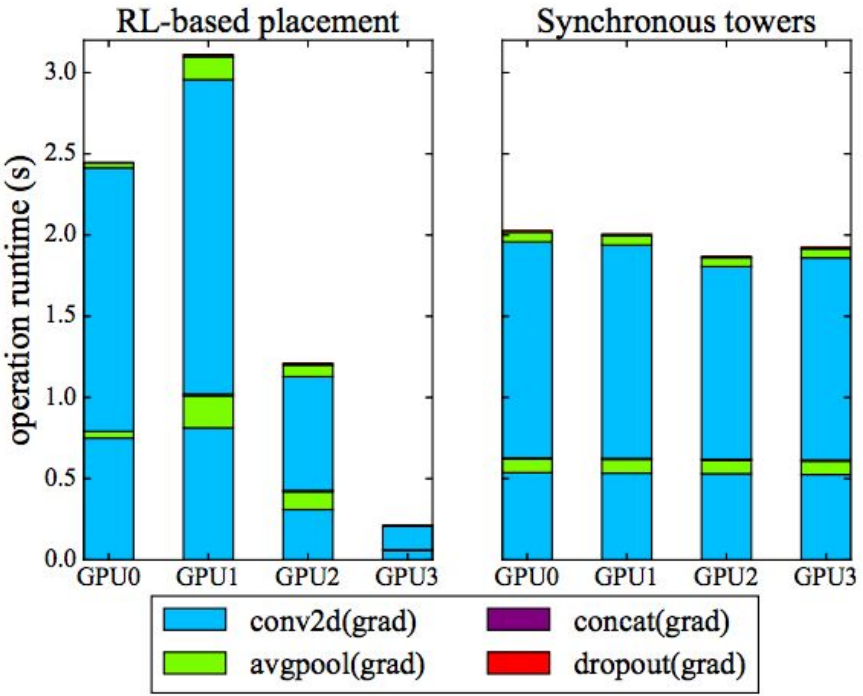


Learned placement on Inception-V3

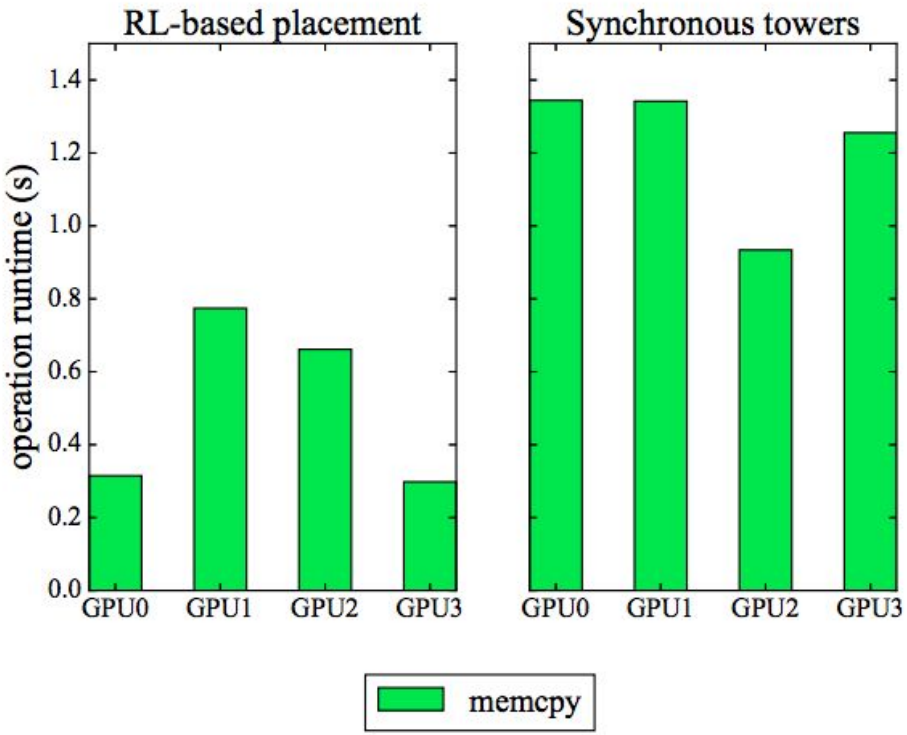


White represents CPU (Ixion Haswell 2300)
Each other color represents a separate GPU (Nvidia Tesla K80)
Searching over a space of 5^83 possible assignments

Profiling placement on Inception-V3



Profiling placement on Inception-V3



Results (runtime in seconds)

Tasks	CPU Only	GPU Only	#GPUs	Human Expert	Scotch	MinCut	Hierarchical Planner	Runtime Reduction
Inception-V3	0.61	0.15	2	0.15	0.93	0.82	**0.13**	16.3%
ResNet	-	1.18	2	1.18	6.27	2.92	**1.18**	0%
RNNLM	6.89	1.57	2	1.57	5.62	5.21	**1.57**	0%
NMT (2-layer)	6.46	OOM	2	2.13	3.21	5.34	**0.84**	60.6%
NMT (4-layer)	10.68	OOM	4	3.64	11.18	11.63	**1.69**	53.7%
NMT (8-layer)	11.52	OOM	8	**3.88**	17.85	19.01	4.07	-4.9%

Summary

- Deep RL for resource allocation optimization
- **Papers:**
  - ICLR’18: A Hierarchical Model for Device Placement,
    - Azalia Mirhoseini*, Anna Goldie*, Hieu Pham, Benoit Steiner, Quoc V. Le and Jeff Dean
  - ICML’17: 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
- **Open-source TensorFlow code:**

  https://github.com/tensorflow/tensorflow/blob/master/tensorflow/python/grappler

Comparing Models with Elo Scores

$$\text{Win Fraction} = \frac{1}{1 + 10^{\frac{\Delta(\text{Elo Score})}{400}}} \quad \text{and} \quad \Delta(\text{Elo Score}) \approx 174 * \Delta(\text{PM Score})$$