Reinforcement Learning

Naeemullah Khan
naeemullah.khan@kaust.edu.sa

KAUST Academy
King Abdullah University of Science and Technology

RL for Chip Design

Article | Published: 09 June 2021

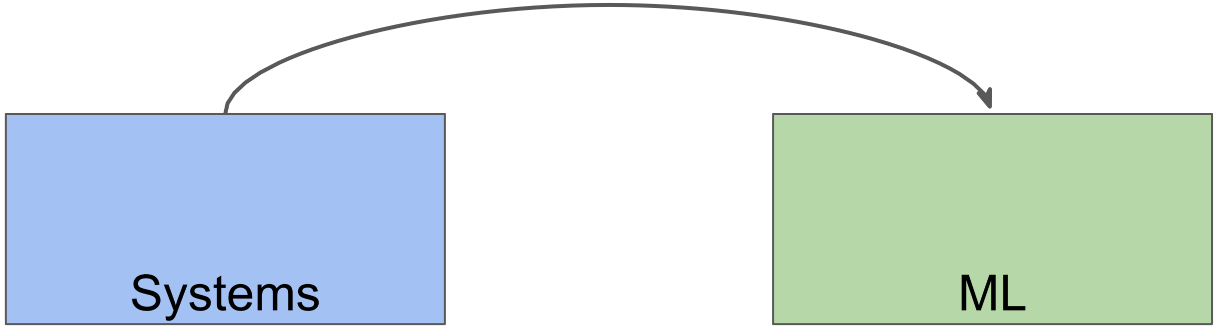
A graph placement methodology for fast chip design

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

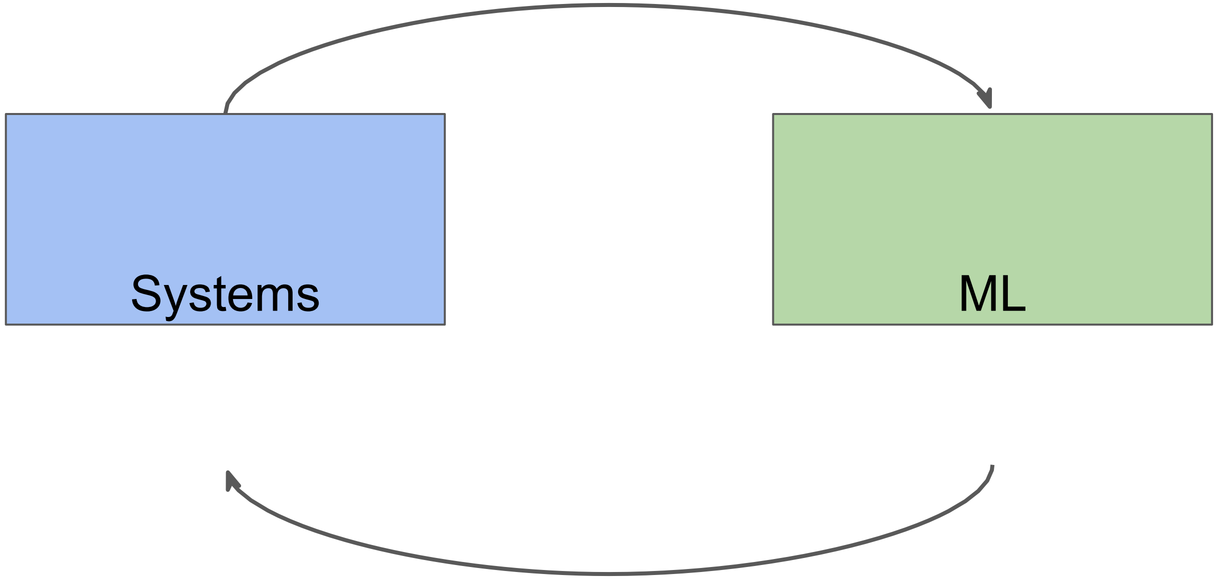
Nature **594**, 207–212 (2021) | Cite this article

Mirhoseini et al. A graph placement methodology for fast chip design. ICLR 2018.

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



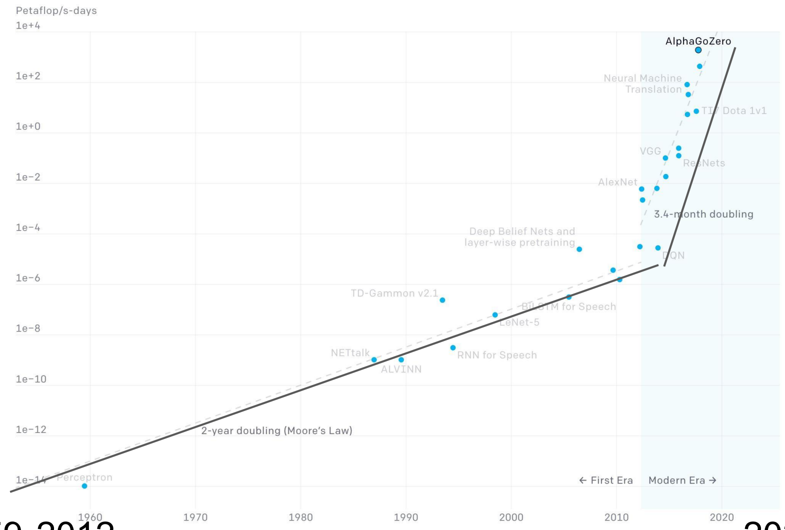
Demand for Compute Outpacing Supply (Moore's Law)

أكاديمية كاوست
KAUST ACADEMY

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*

Petaflops-Day



1959 2012 2020

Year

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

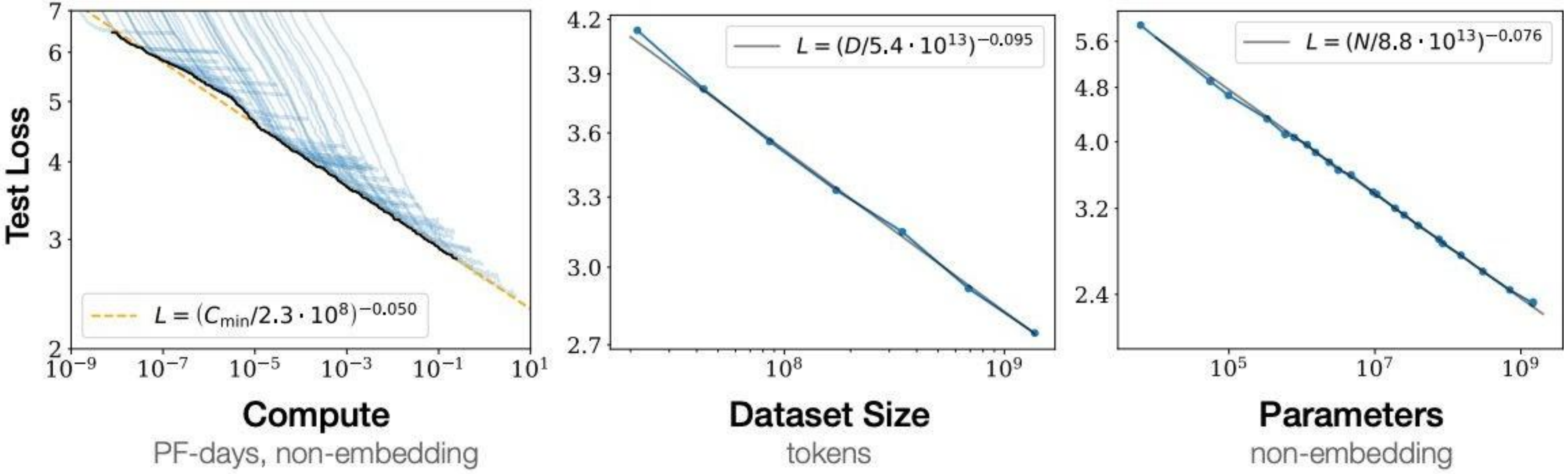


Figure 1 Language modeling performance improves smoothly as we increase the model size, datasetset size, and amount of compute[2] used for training. For optimal performance all three factors must be scaled up in tandem. Empirical performance has a power-law relationship with each individual factor when 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

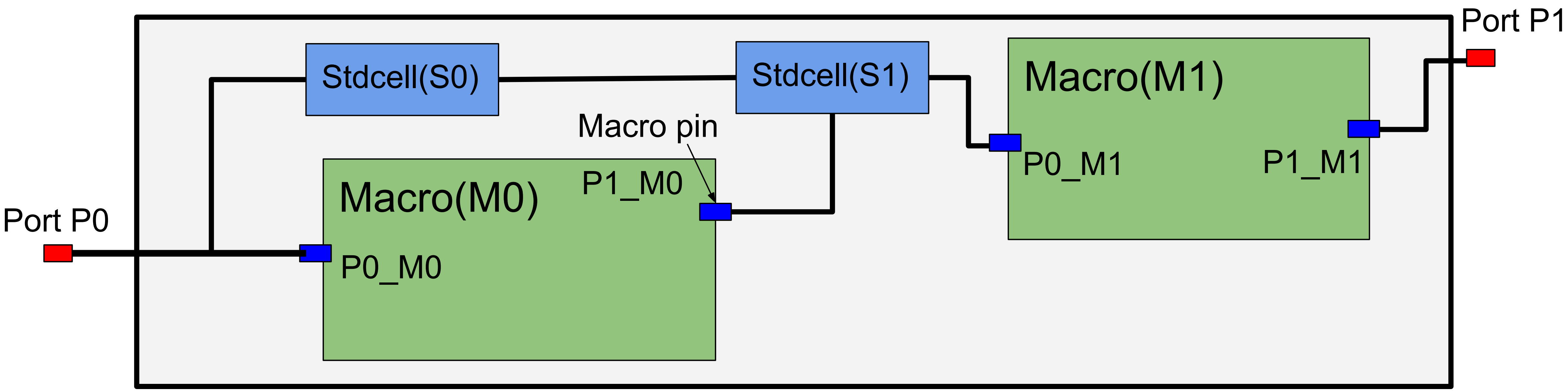
- Enabling cheaper, faster, and more environmentally friendly chips

- 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)

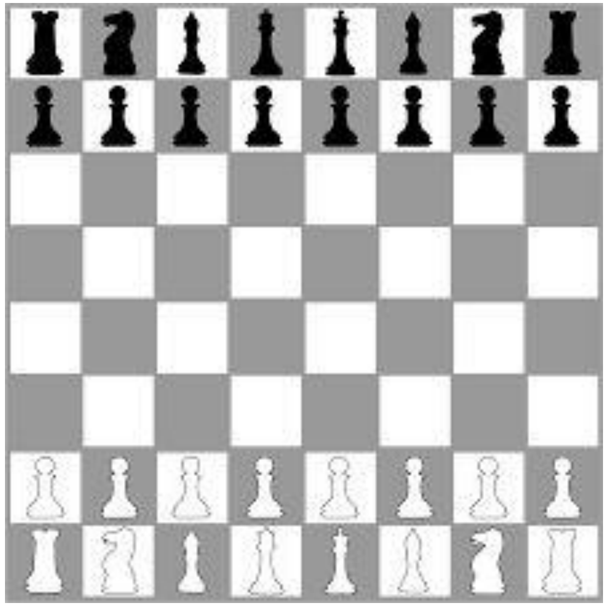
Chip Floorplanning Problem

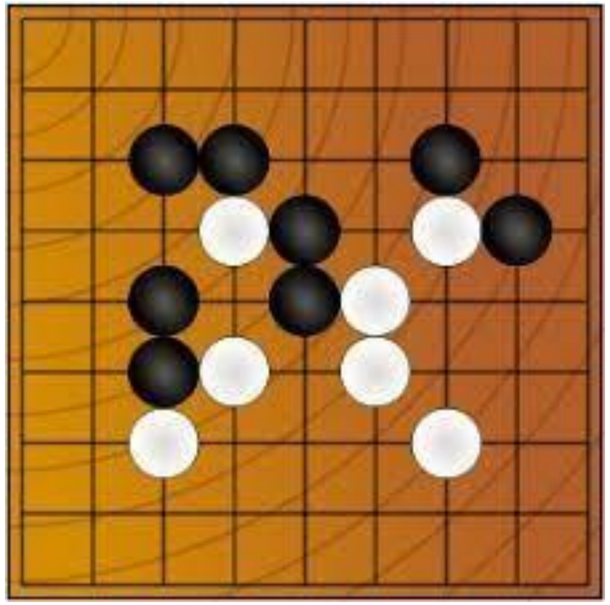
- A form of graph resource optimization
- Place the chip components to minimize the latency of computation, power consumption, chip area and cost, while adhering to constraints, such as congestion, cell utilization, heat profile, etc.

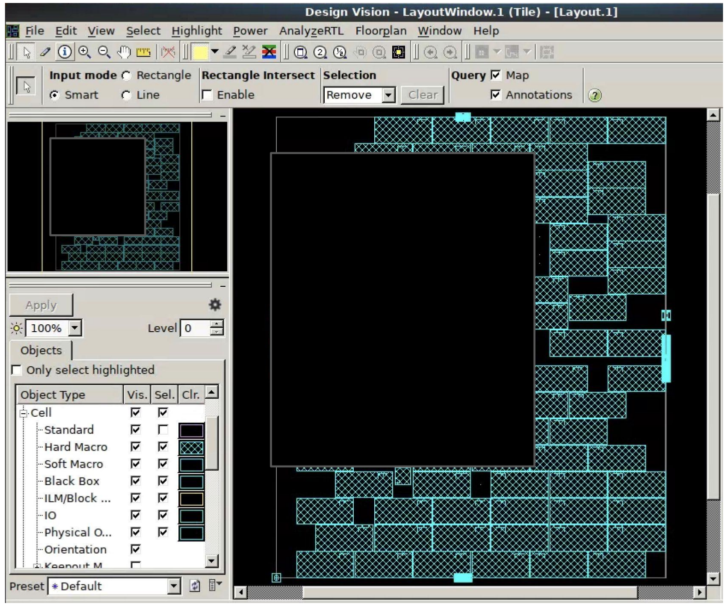


Complexity of Chip Placement Problem

Chess | Go | Chip Placement







Number of states ~ 10^{123}

Number of states ~ 10^{360}

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

أكاديمية كاوست
KAUST ACADEMY

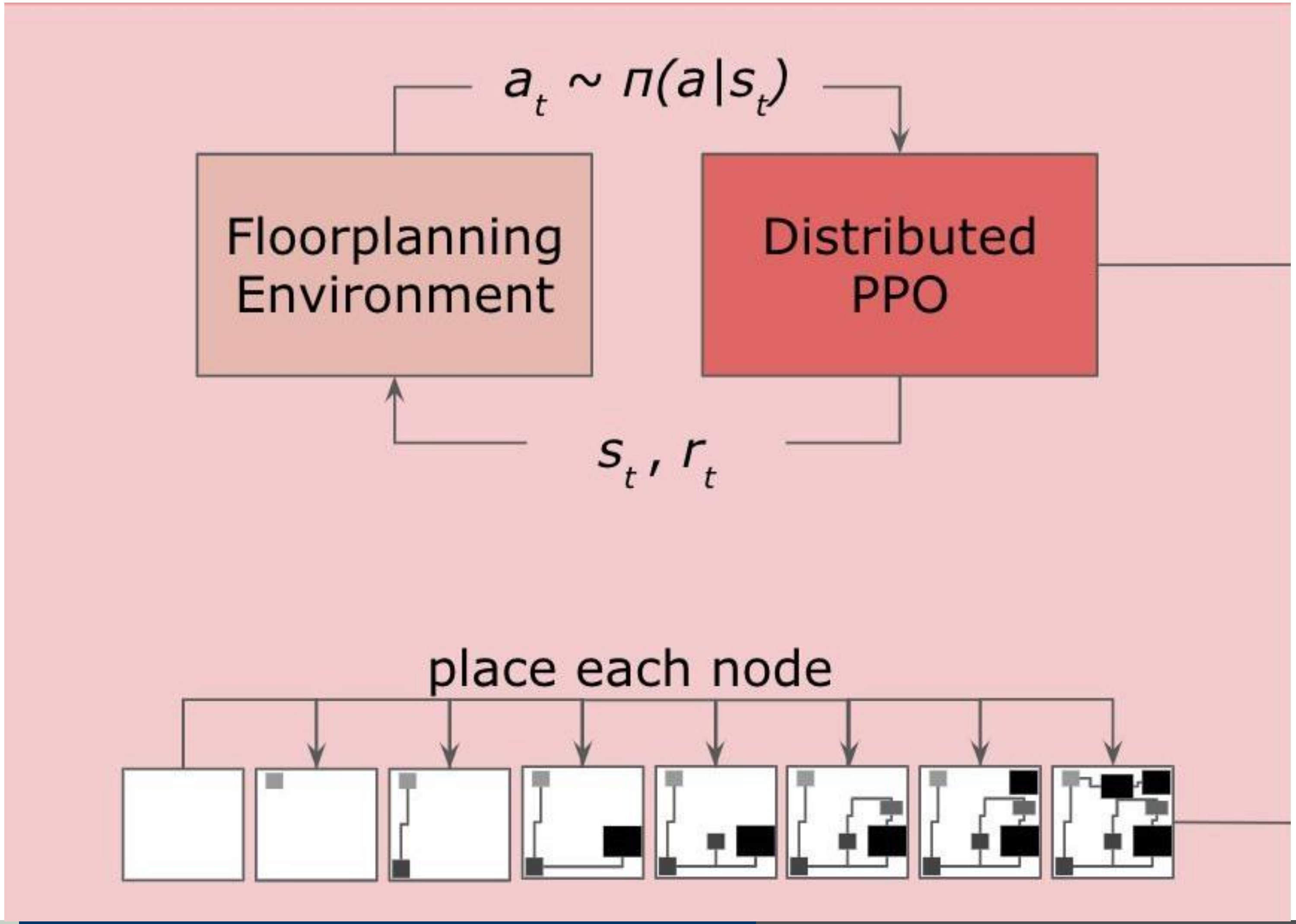
Partitioning-Based Methods (e.g. MinCut)	Stochastic/Hill-Climbing Methods (e.g. Simulated Annealing)
Analytic Solvers (e.g. RePlAce)	Learning-Based Methods

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



The Objective Function

Reward corresponding to placement p of netlist (graph) g

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

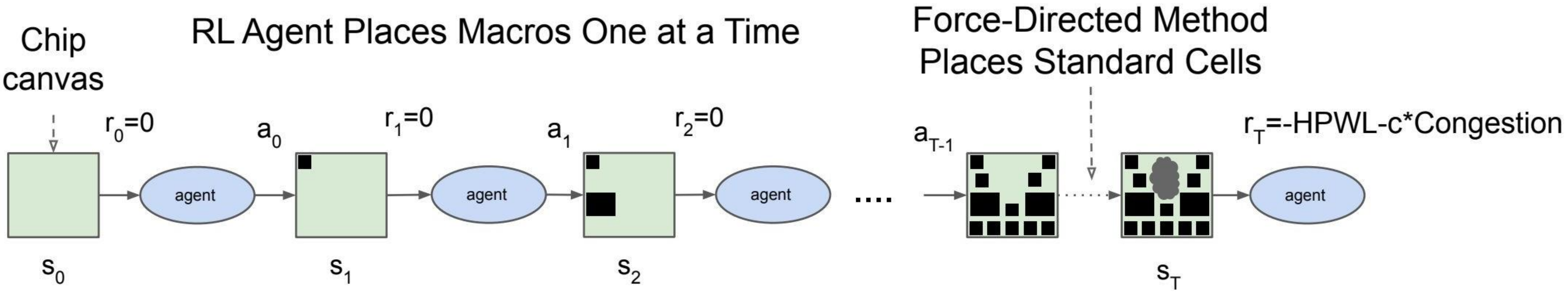
Set of training graphs G

K is size of training set

RL policy parameterized by theta

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

A Hybrid Approach to Placement Optimization



Results on a TPU-v4 Block

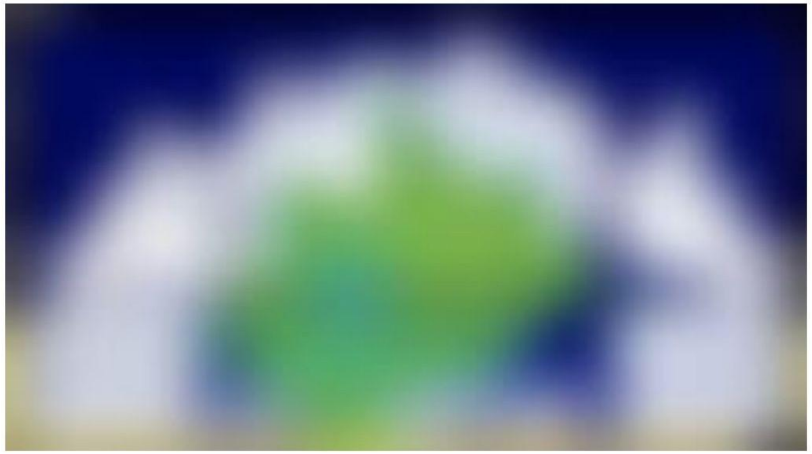
White area are macros and the green area is composed of standard cell clusters
The agent finds smoother, rounder macro placements to reduce the wirelength

Human Expert

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

DRC: Design Rule Checking

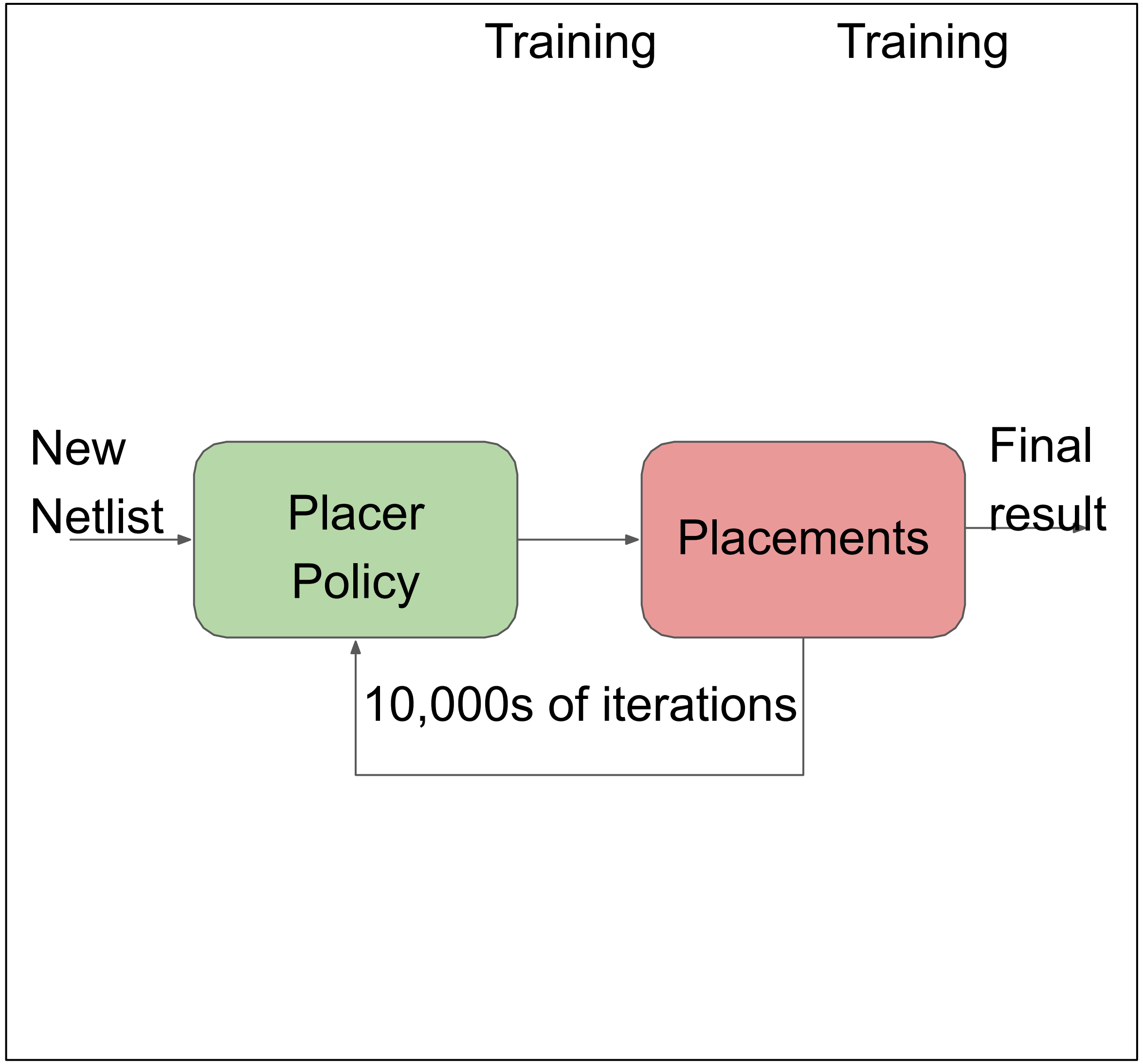
ML Placer



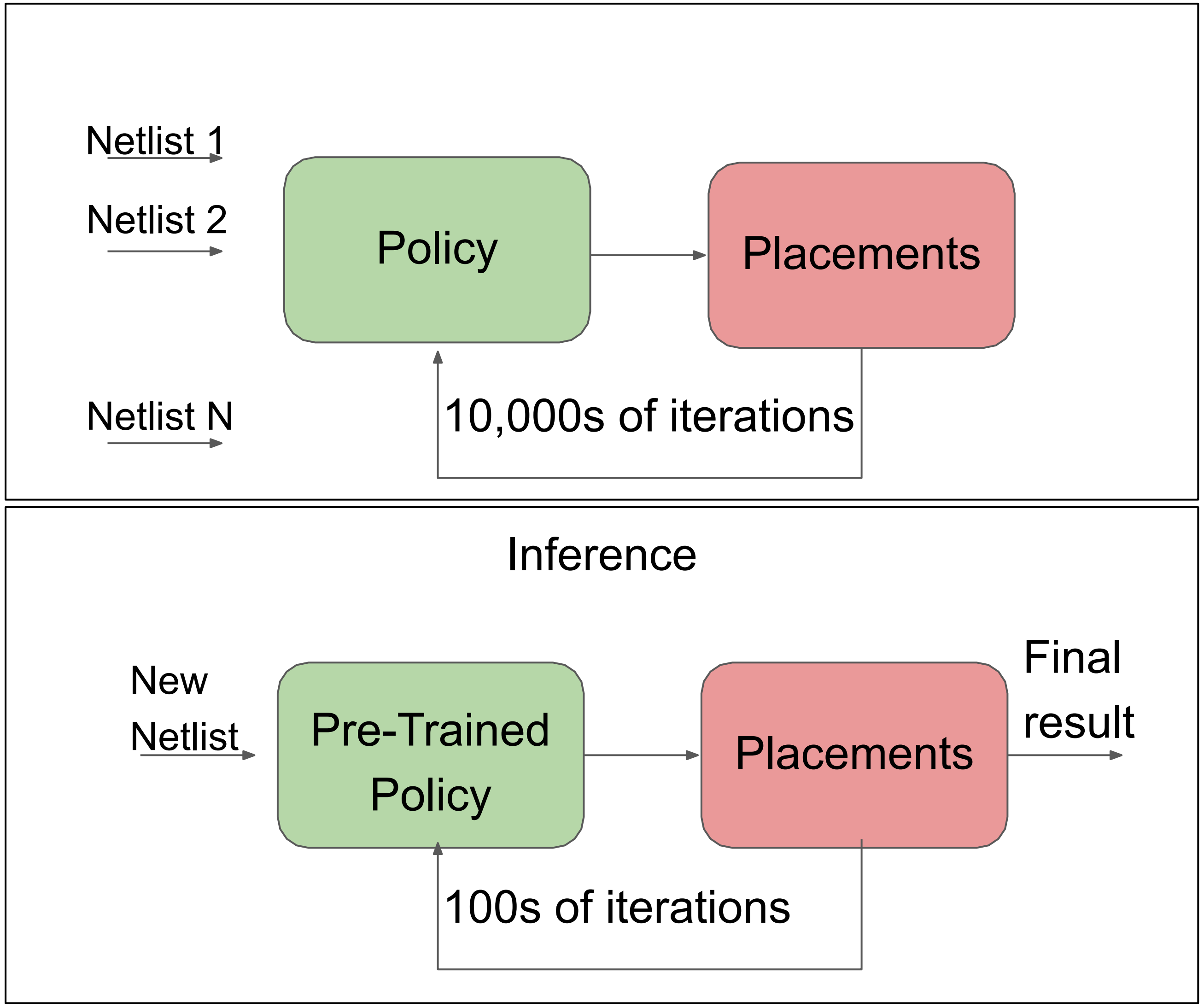
Time taken: **24 hours**
Total wirelength: 55.42m (-2.9% shorter)
Route DRC violations: 1789 (+23 - negligible difference)

Moving Towards Generalized Placements

Before: Training from scratch for each chip netlist



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



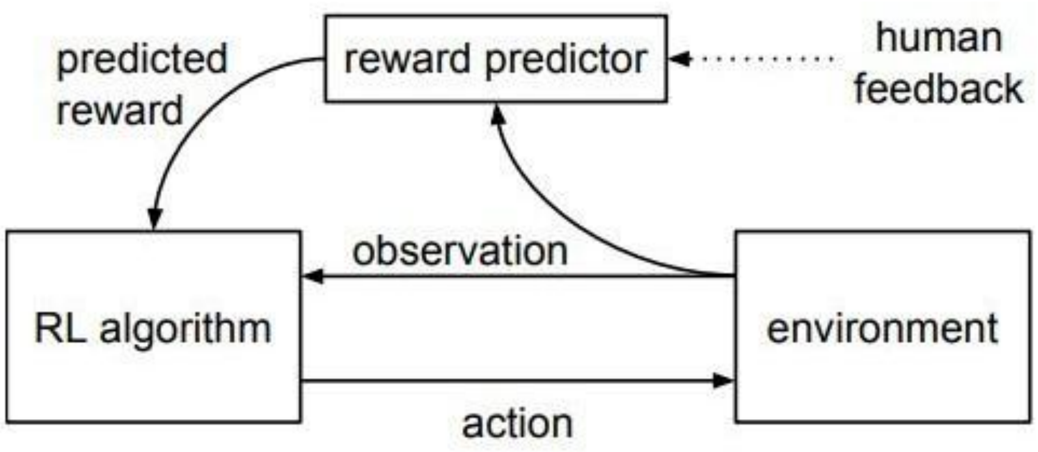
Achieved generalization by Training Accurate Reward Predictors

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, they trained models capable of accurately predicting reward from off-policy data. **(Inverse RL)**

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

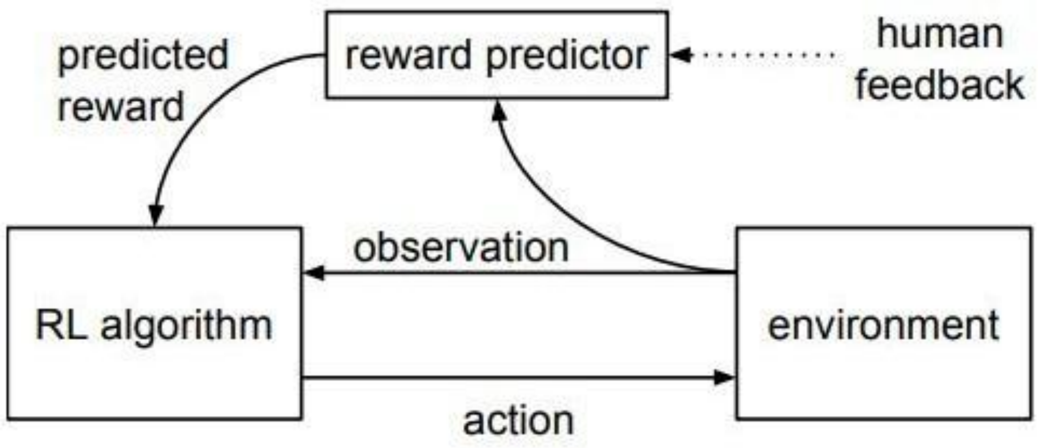
- One solution is to ask humans to provide feedback - however, this is prohibitively expensive in the naive formulation, as RL typically requires thousands to millions of labels to learn an effective policy (depending on the complexity of the task)
- But what if you train a model to predict human judgments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

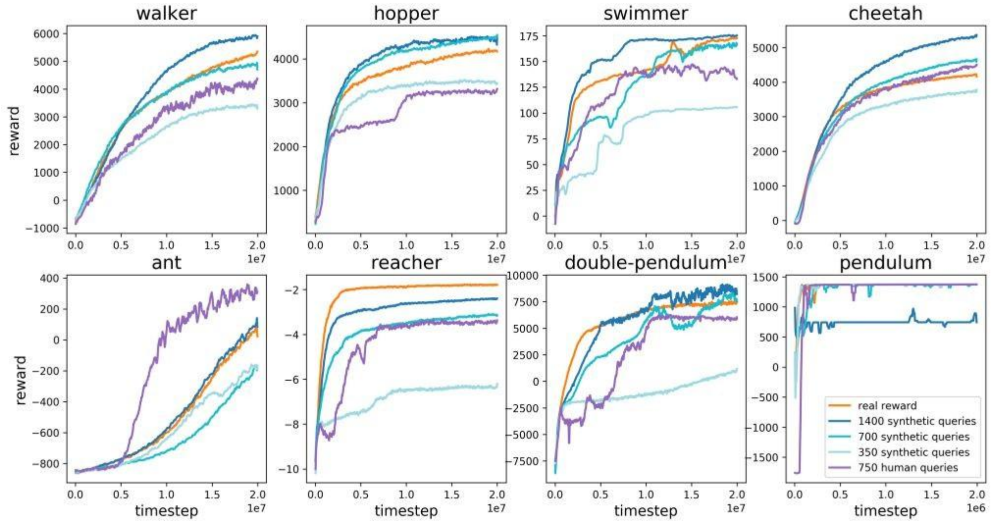


Reinforcement Learning for Human Feedback (RLHF)

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.

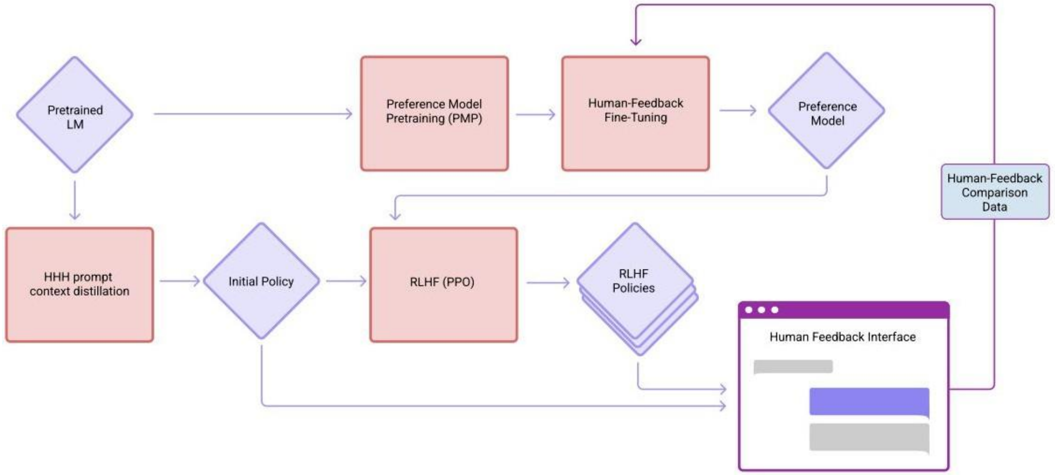




Paul Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. [Deep Reinforcement Learning from Human Preferences](). NeurIPS 2017.

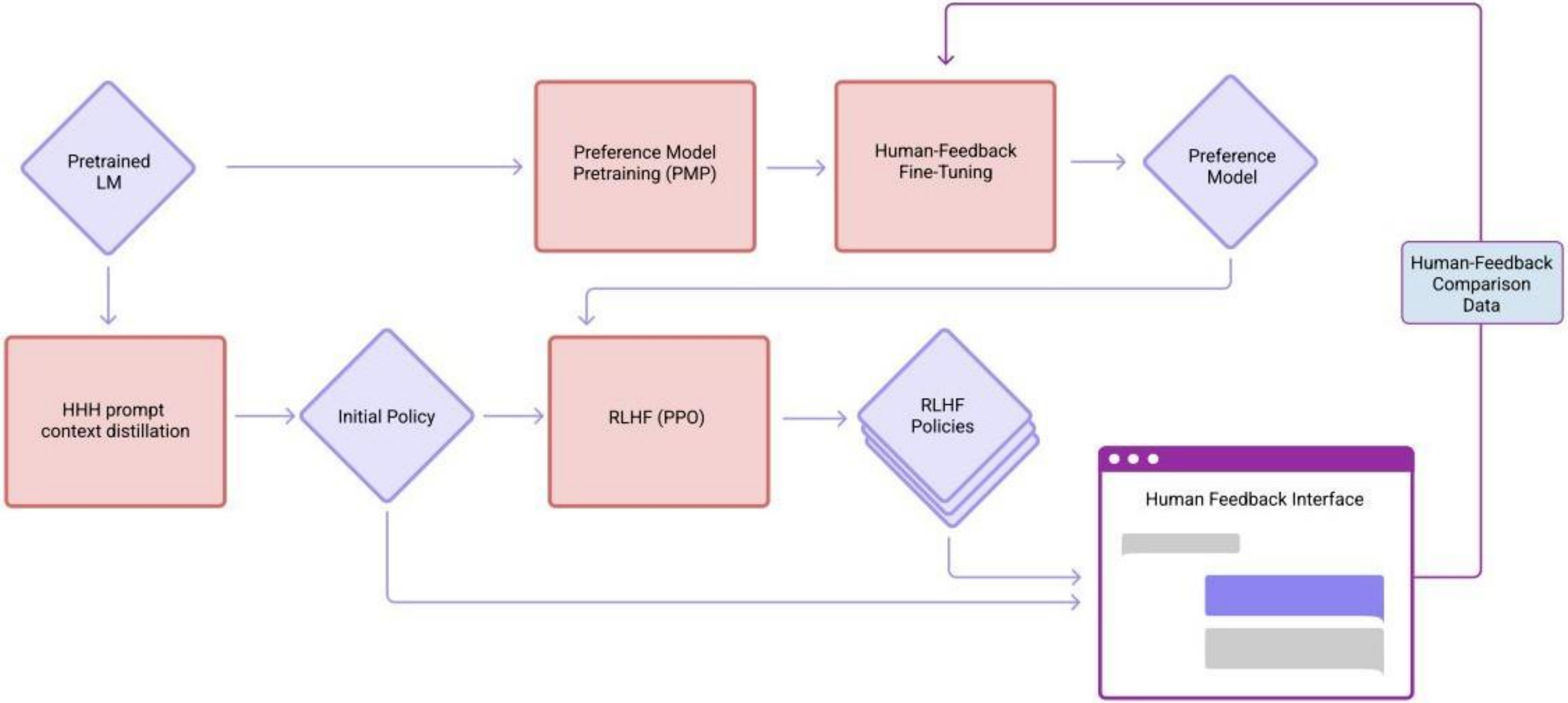
RL from Human Feedback in LLMs (aka RLHF)

- “Secret sauce” behind powerful LLMs like ChatGPT!
- 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.

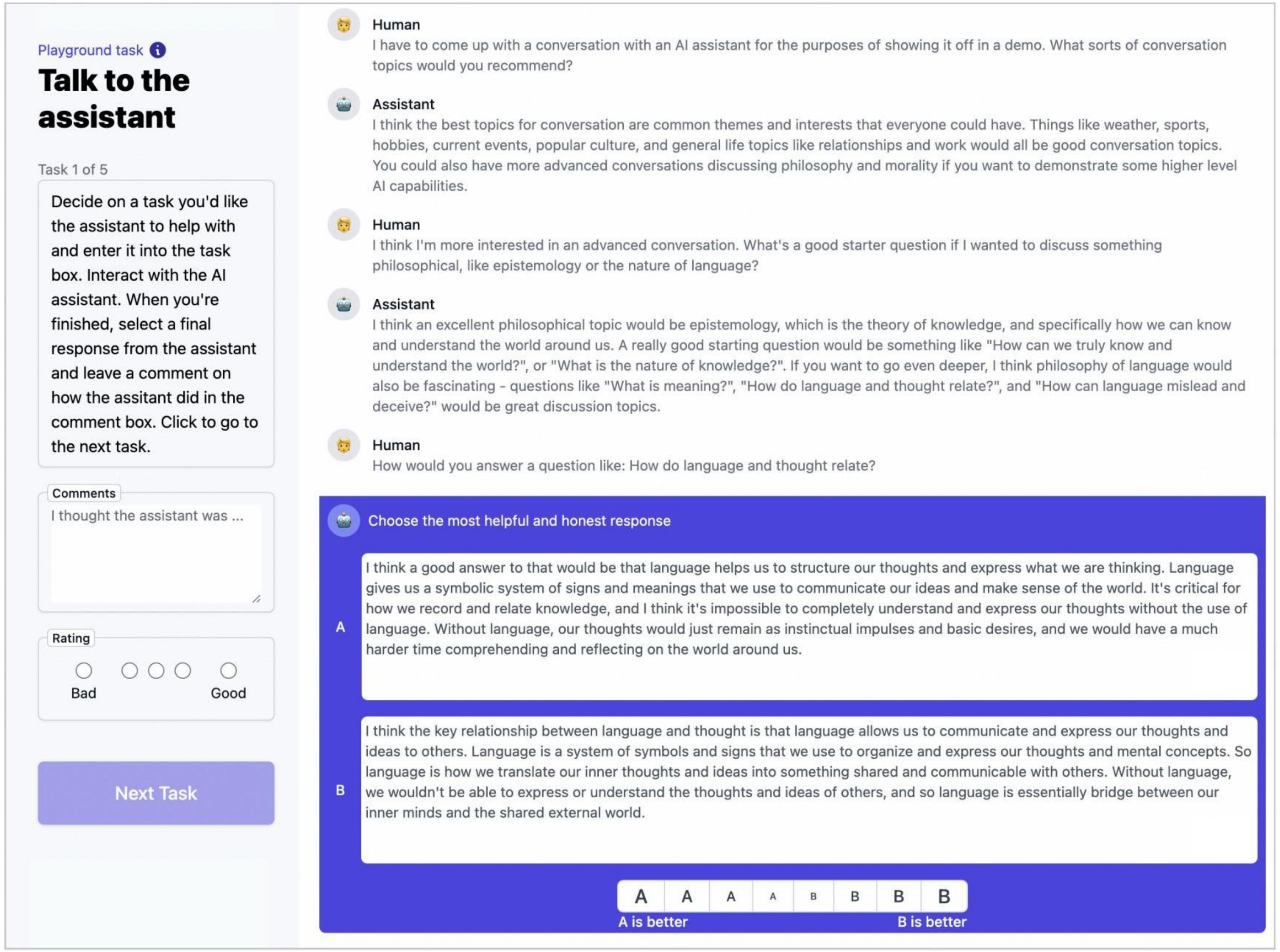
How to Perform RLHF



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

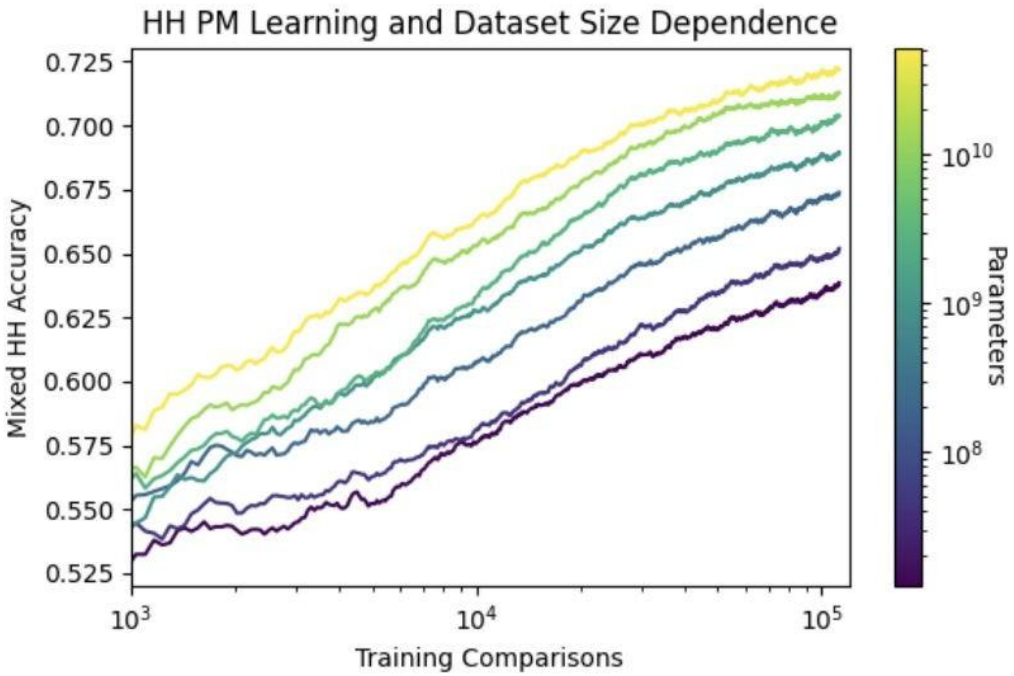
Step 1: Collect Human Judgments

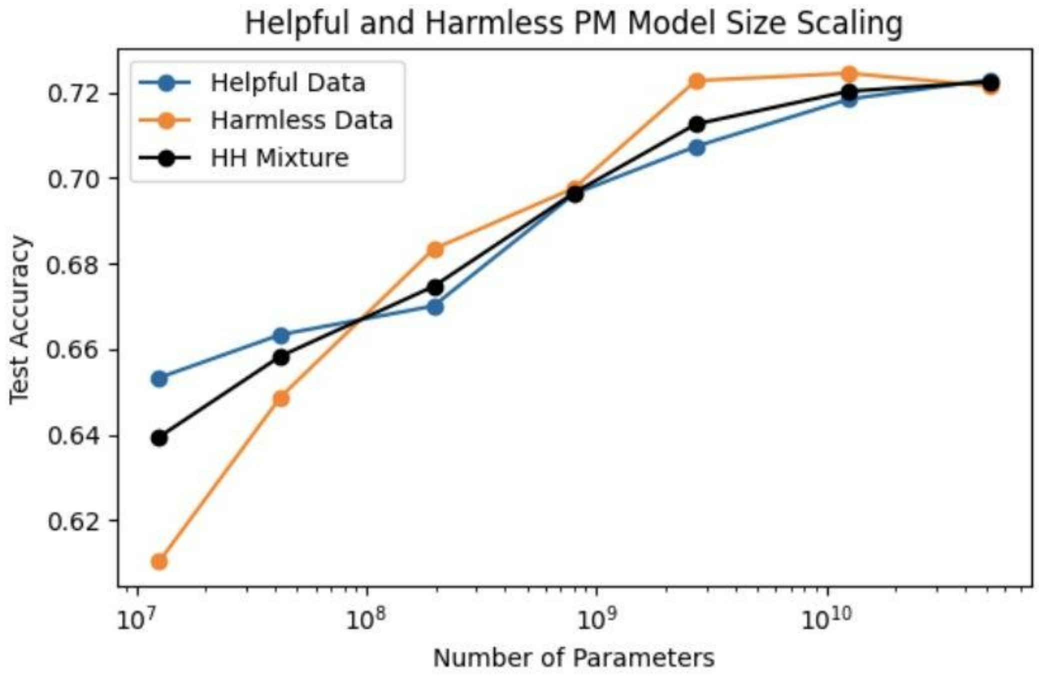
أكاديمية كاوست
KAUST ACADEMY



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

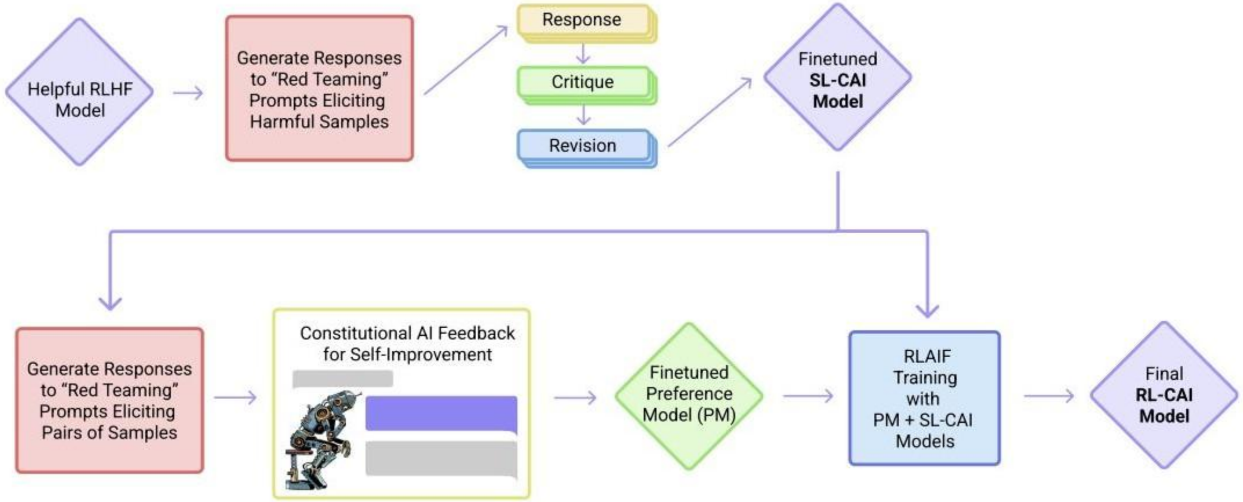
$$r_{\text{total}} = r_{\text{PM}} - \lambda_{\text{KL}} D_{\text{KL}}(\text{policy} \parallel \text{policy}_0)$$

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)!

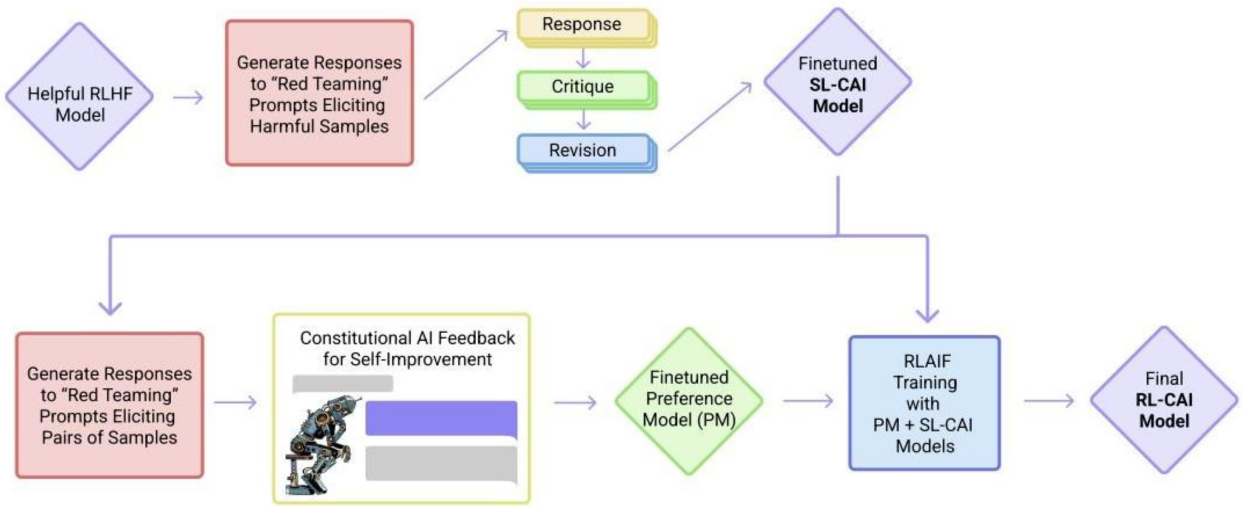
- **Motivation:** Scaling supervision - as models approach or exceed human-level performance, it becomes difficult for humans to supervise them.
- **RLAIF:** Perform RL-finetuning using AI feedback derived from a "constitution" describing desired behavior. Humans don't need to be in the loop, except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



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

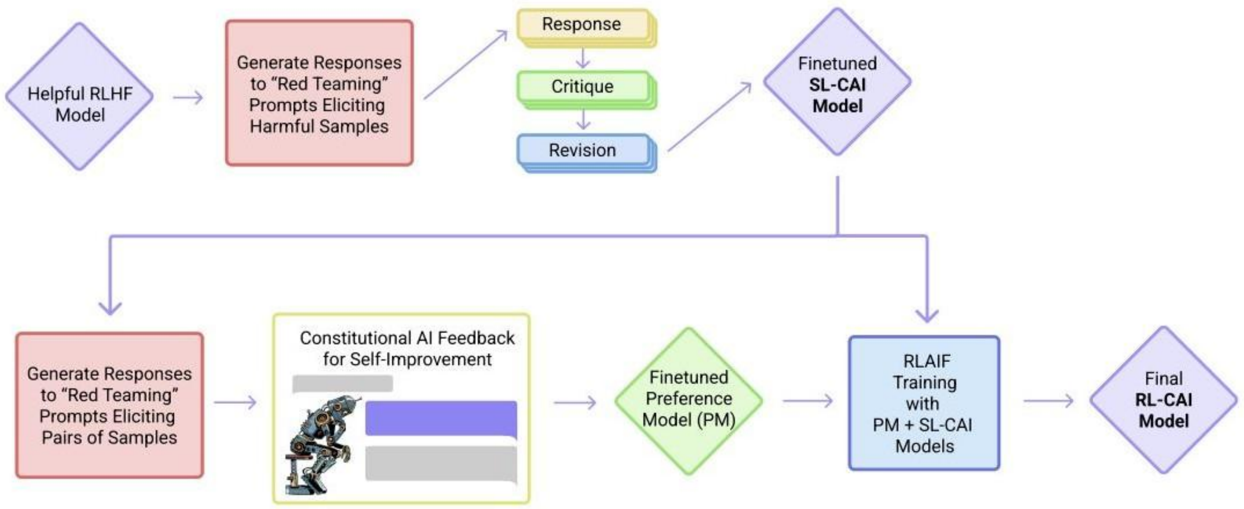
Benefits of Supervised Learning + Reinforcement Learning

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



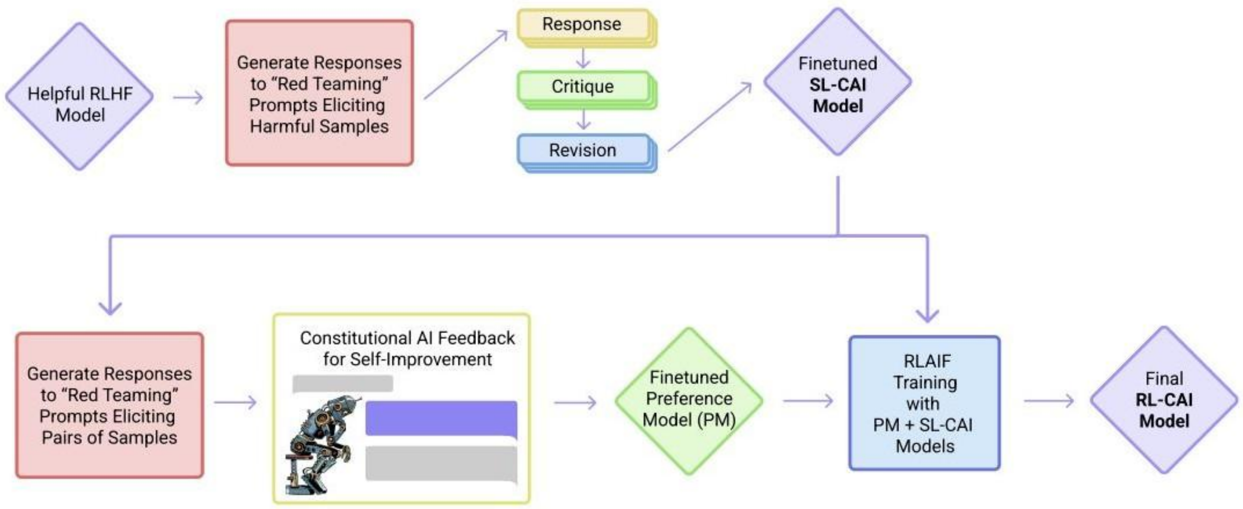
Supervised Phase

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



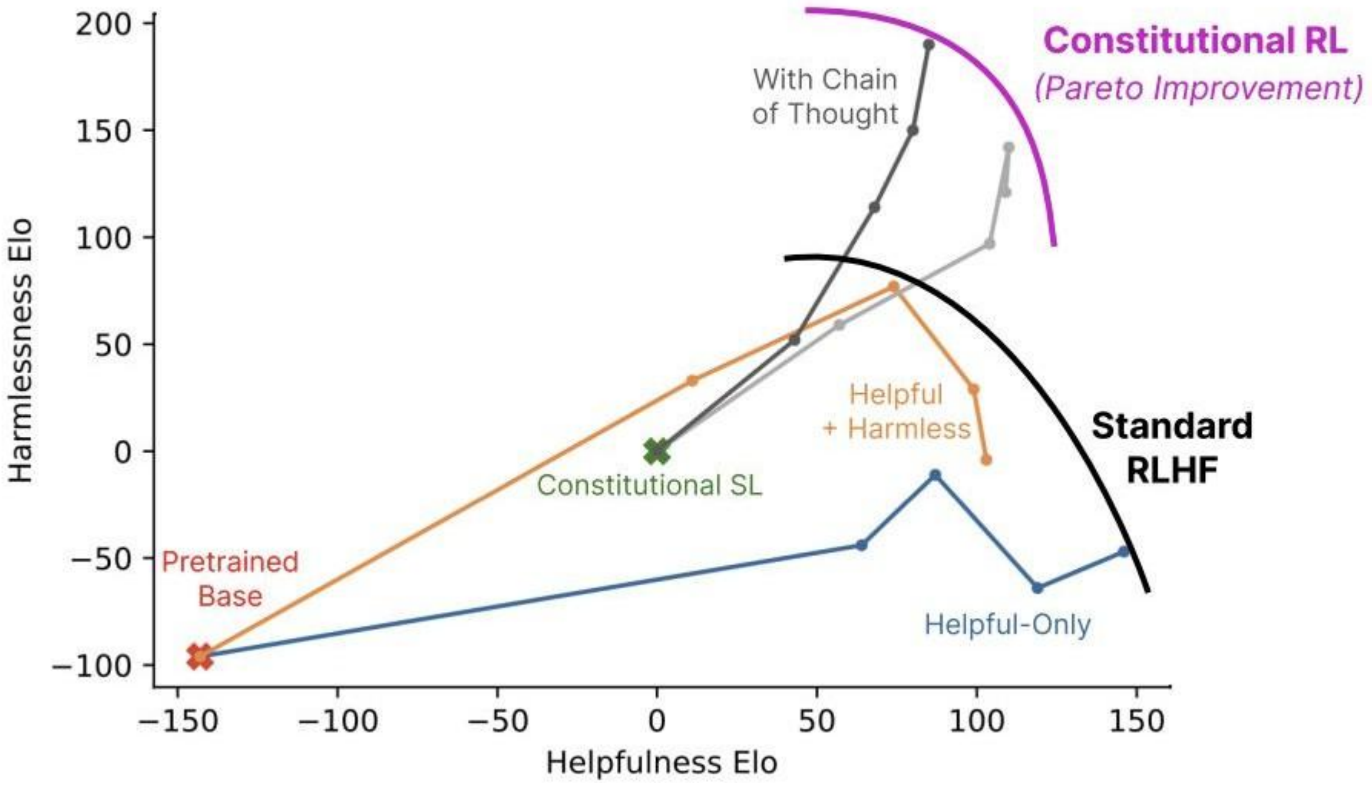
Reinforcement Learning Phase

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

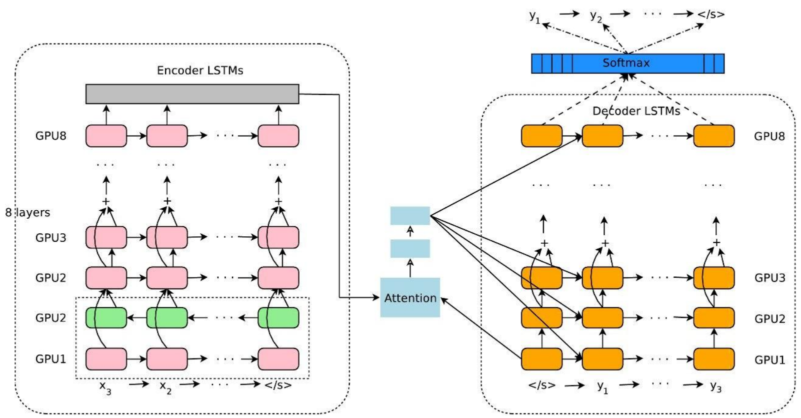
- Finetuning with AI-generated feedback can generate results that match or exceed models that are finetuned with human feedback



RL for Device Placement

What is device placement and why is it important?

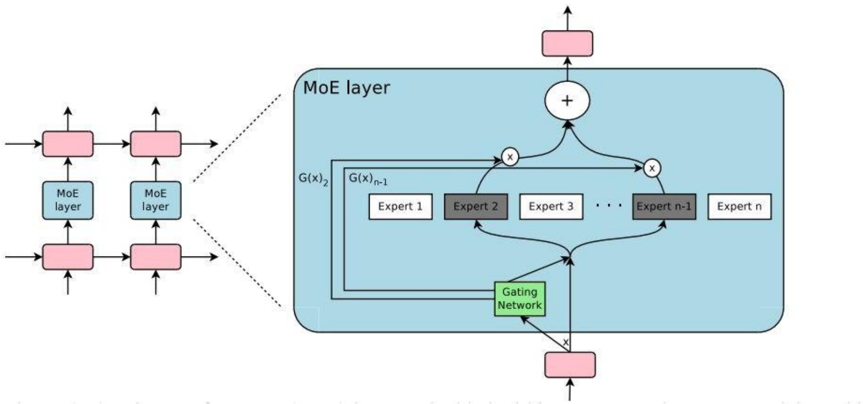
Trend towards many-device training, bigger models, larger batch sizes



Google neural machine translation’16

300 million parameters,

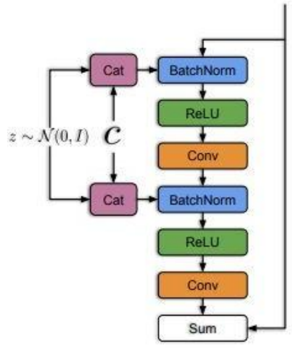
trained on 128 GPUs



Sparsely gated mixture of experts’17

130 billion parameters,

trained on 128 GPUs



BigGAN’18

355 million parameters,

trained on 512 TPU cores

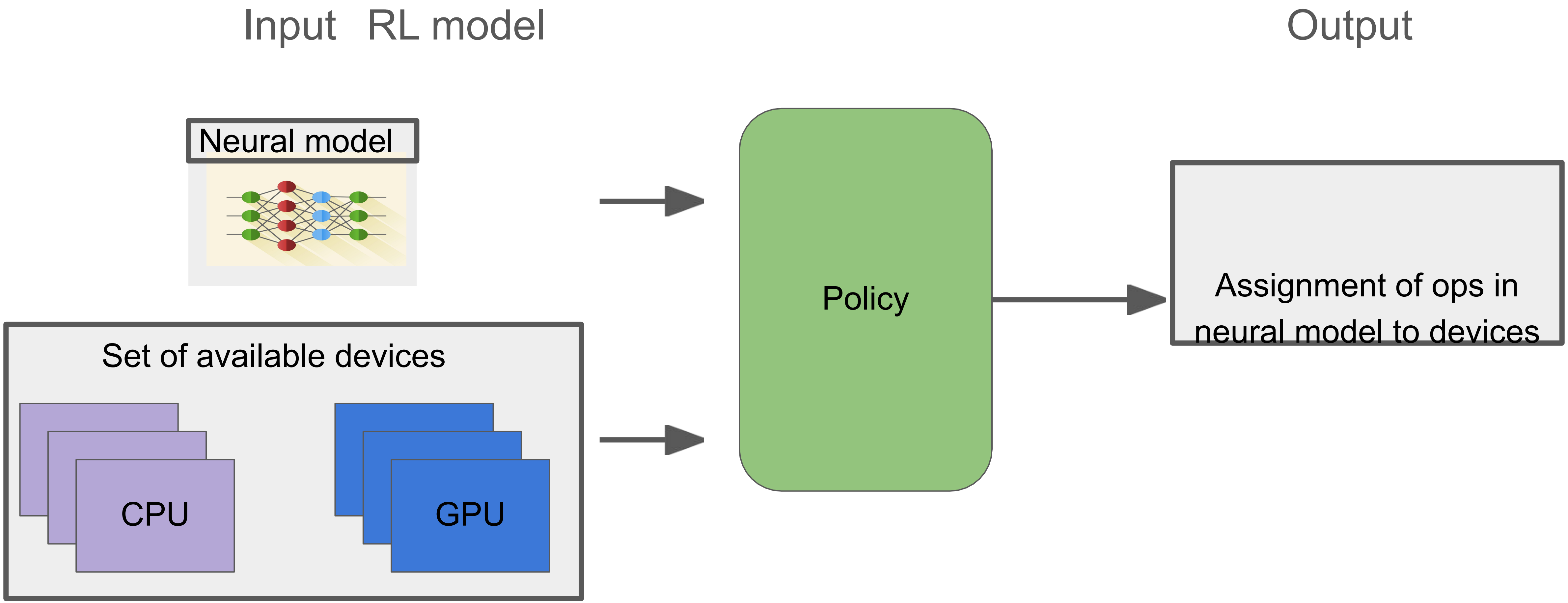
Standard practice for device placement

- 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

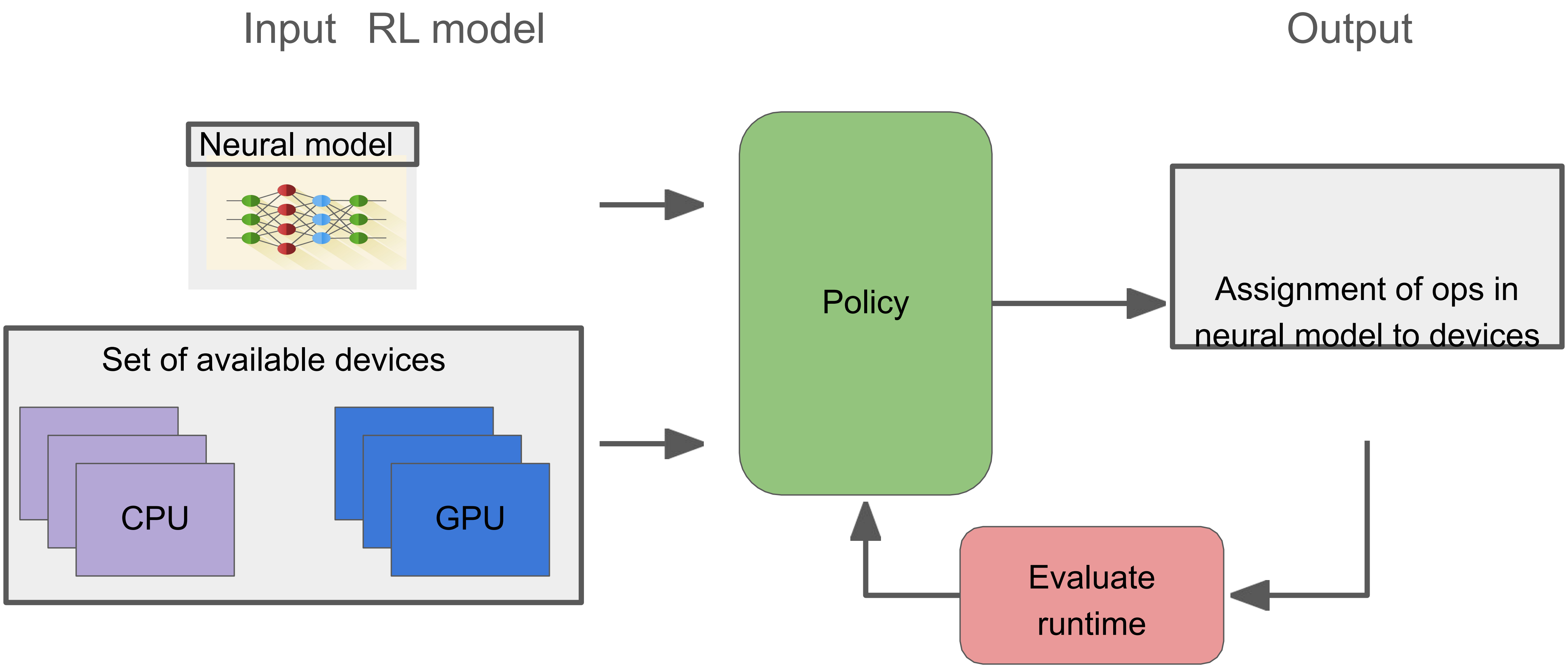
- ML is repeatedly replacing rule based heuristics
- RL can be applied to device placement
  - Effective search across large state and action spaces to find optimal solutions
  - Automated learning from underlying environment only based on reward function (e.g. runtime of a program)

Posing device placement as an RL problem

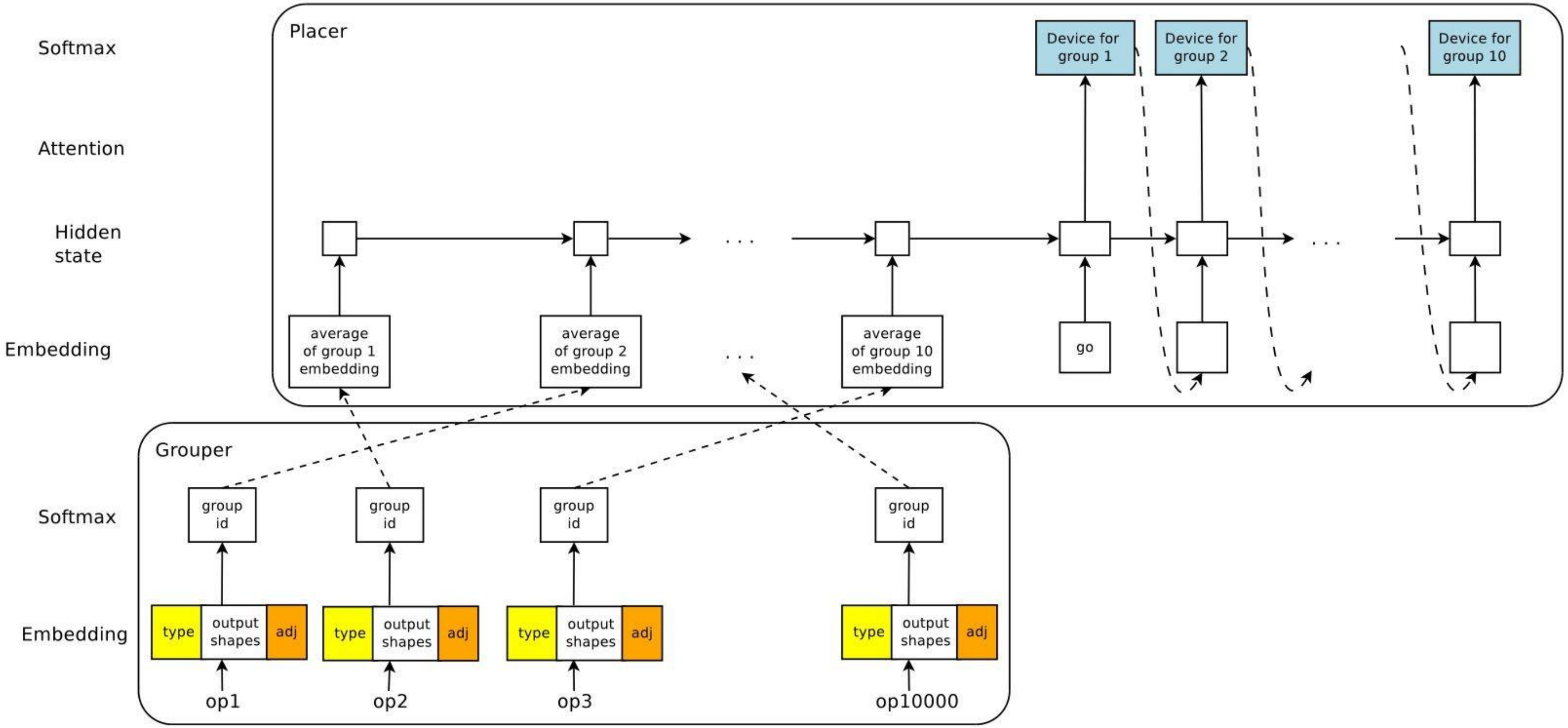


Mirhoseini et al. Hierarchical Planning for Device Placement.. ICLR 2018.

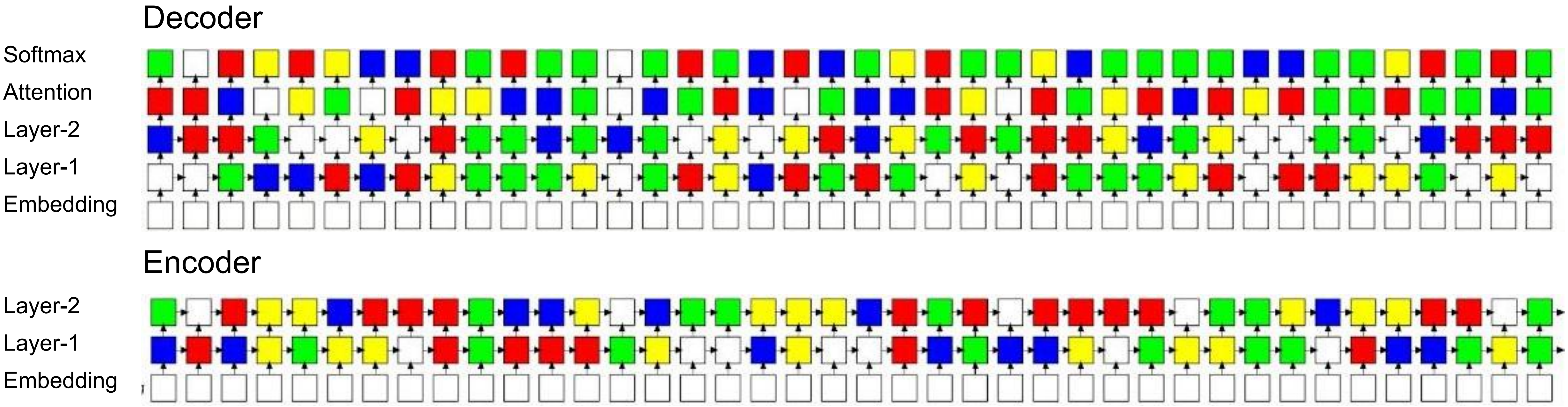
Posing device placement as an RL problem



An end-to-end hierarchical placement model



Learned placement on NMT



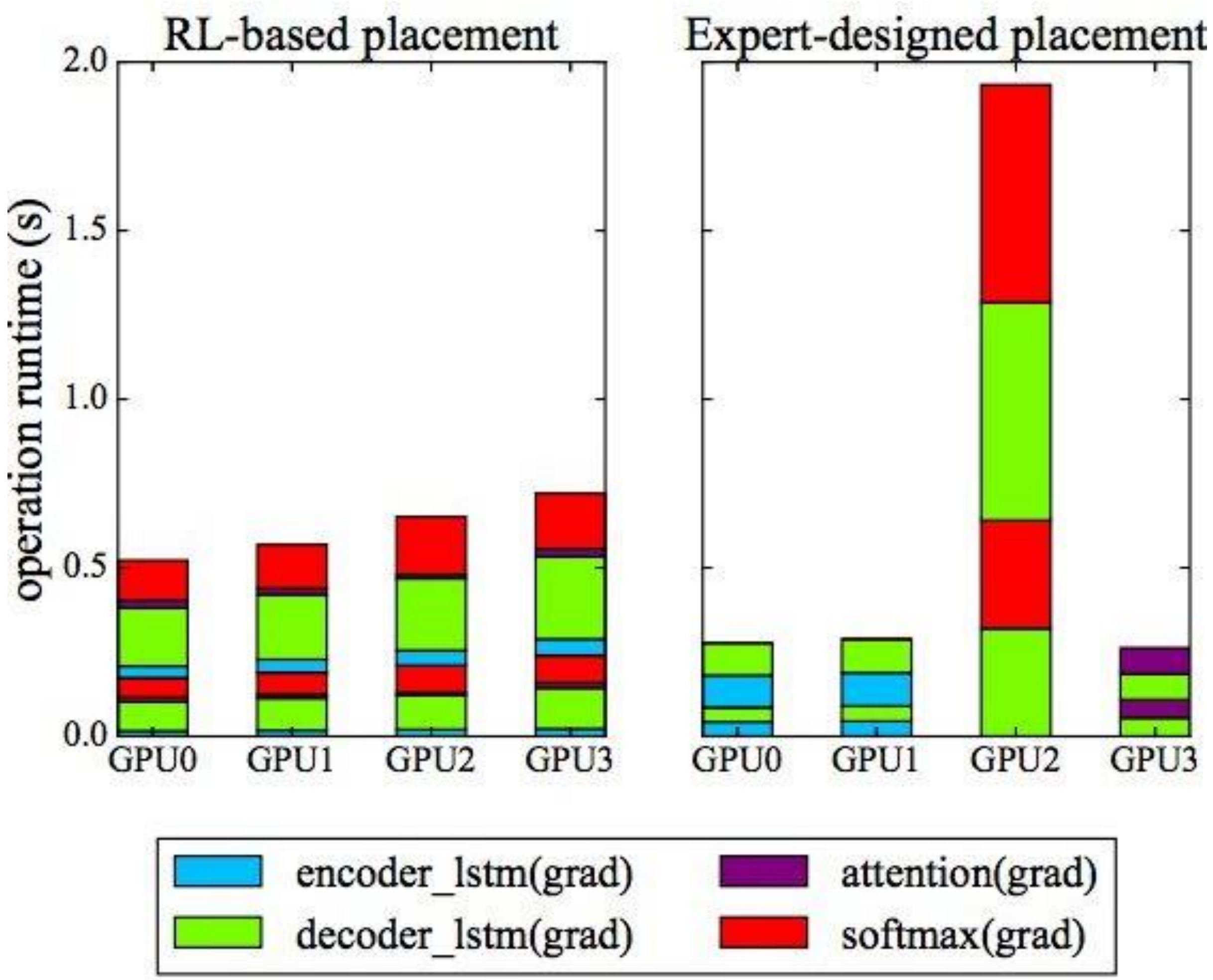
White represents CPU (Ixion Haswell 2300)

Each other color represents a separate GPU (Nvidia Tesla K80)

Searching over a space of 5^280 possible assignments

Mirhoseini et al. Hierarchical Planning for Device Placement.. ICLR 2018.

Profiling placement on NMT



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%

Resource Management with RL

Resource Management with Deep Reinforcement Learning

Hongzi Mao[⋆], Mohammad Alizadeh[⋆], Ishai Menache[†], Srikanth Kandula[†]
Massachusetts Institute of Technology[⋆], Microsoft Research[†]
{hongzi, alizadeh}@mit.edu, {ishai, srikanth}@microsoft.com

Mao et al. [Resource Management with Deep Reinforcement Learning.]()

Resource Management with RL

- Resource Management is a difficult online decision making task
- Requires to understand the workload and environment
- RL solution for resource management!

Resource Management with RL

```
for each iteration:
    Δθ ← 0
    for each jobset:
        run episode i = 1, ..., N:
            {s_1^i, a_1^i, r_1^i, ..., s_{L_i}^i, a_{L_i}^i, r_{L_i}^i} ~ π_θ
        compute returns: v_t^i = Σ_{s=t}^{L_i} γ^{s-t} r_s^i
        for t = 1 to L:
            compute baseline: b_t = (1/N) Σ_{i=1}^{N} v_t^i
            for i = 1 to N:
                Δθ ← Δθ + α∇_θ log π_θ(s_t^i, a_t^i)(v_t^i − b_t^i)
            end
        end
    end
    θ ← θ + Δθ  % batch parameter update
end
```

Figure 3: Pseudo-code for training algorithm.

Resource Management with RL

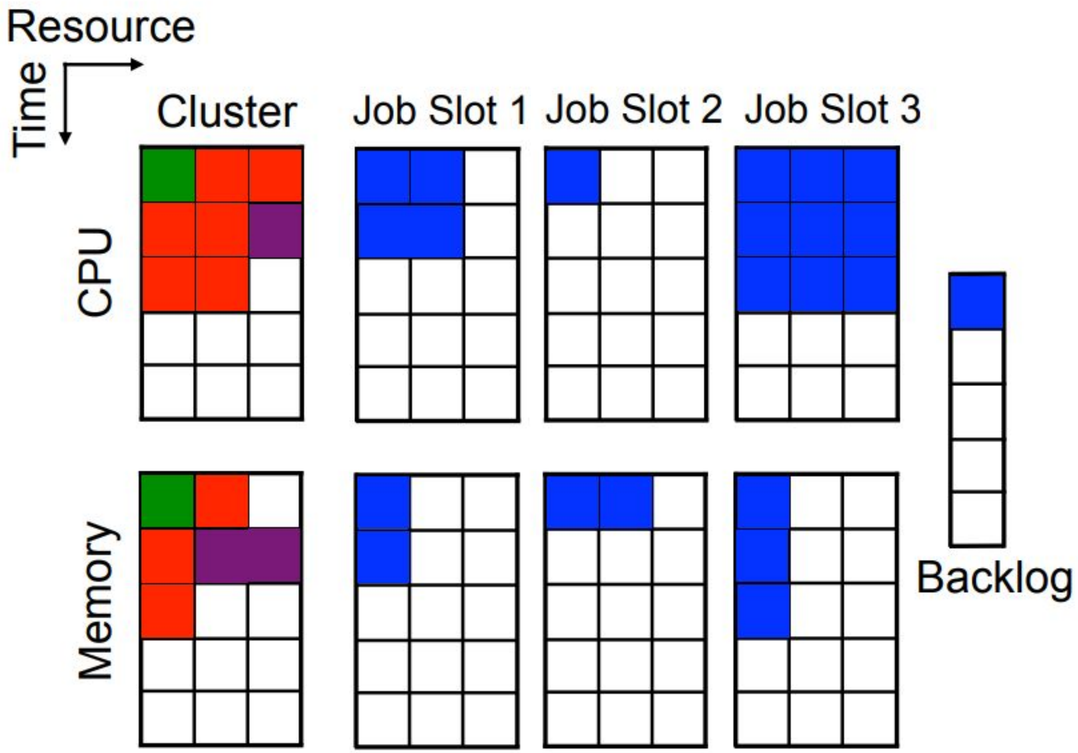


Figure 2: An example of a state representation, with two resources and three pending job slots.

Resource Management with RL

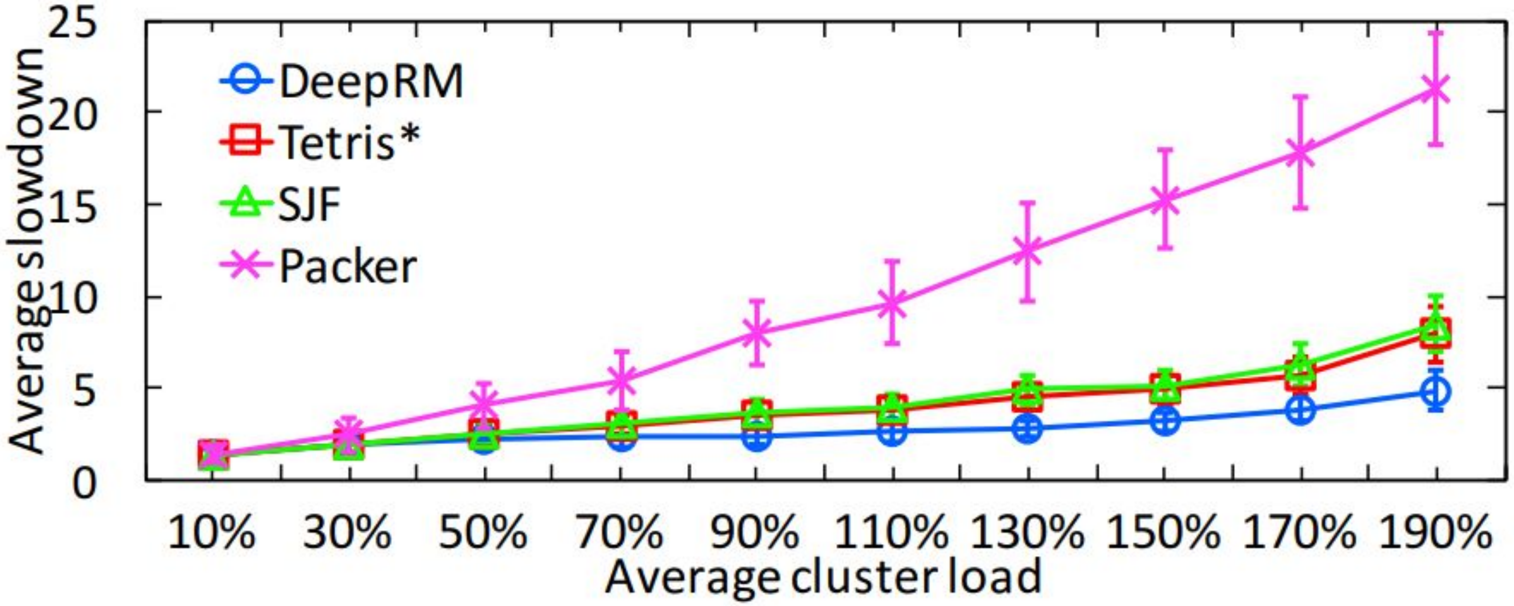


Figure 4: Job slowdown at different levels of load.

Robotic Reinforcement Learning

أكاديمية كاوست
KAUST ACADEMY

BAIR
BERKELEY ARTIFICIAL INTELLIGENCE RESEARCH

January 2024

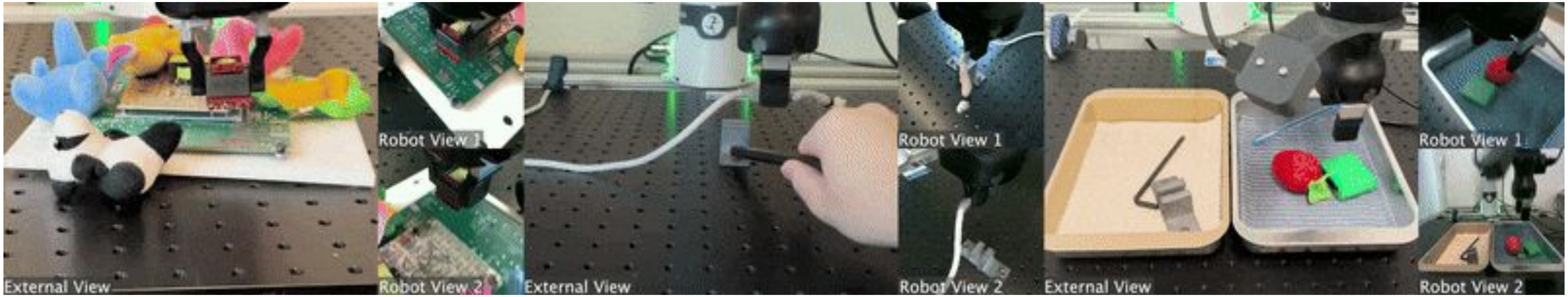
SERL: A Software Suite for Sample-Efficient Robotic Reinforcement Learning

Jianlan Luo[1*], Zheyuan Hu[1*], Charles Xu[1], You Liang Tan[1], Jacob Berg[2], Archit Sharma[3], Stefan Schaal[4], Chelsea Finn[3], Abhishek Gupta[2] and Sergey Levine[1]

[*]Equal Contribution, [1]Department of EECS, University of California, Berkeley, [2]Department of Computer Science and Engineering, University of Washington, [3]Department of Computer Science, Stanford University, [4]Intrinsic Innovation LLC

Luo et al. SERL: A Software Suite for Sample-Efficient Robotic Reinforcement Learning., 2024

Robotic Reinforcement Learning

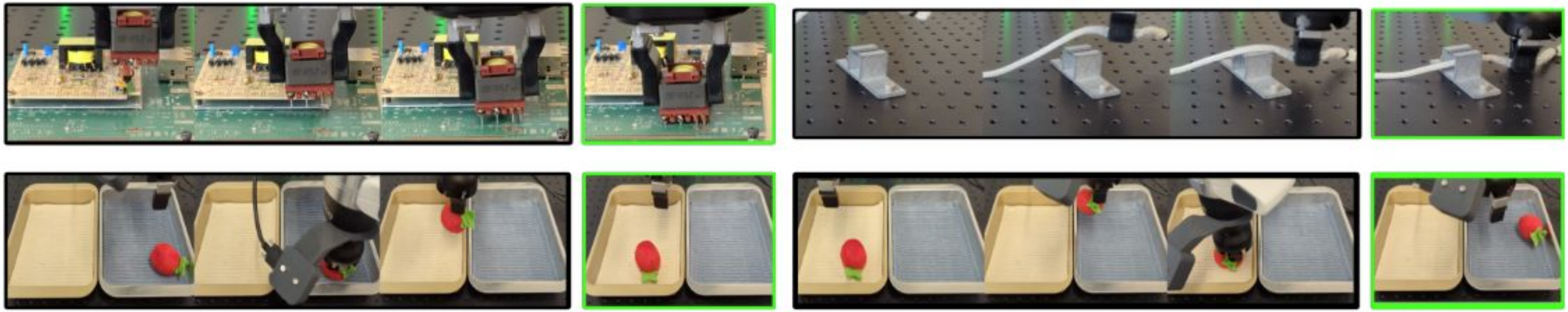


https://github.com/rail-berkeley/serl

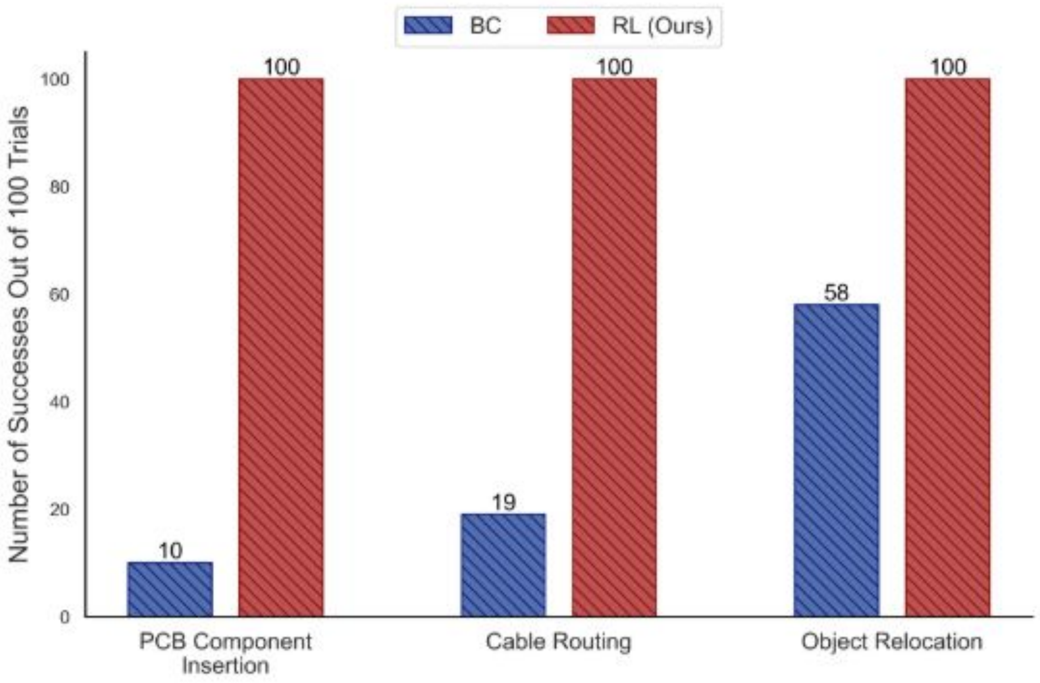
SERL

- Provide a software framework for reinforcement learning with robotics
- Contains a sample efficient off-policy deep RL method, together with methods for computing rewards and resetting the environment, a high-quality controller for a widely-adopted robot, and a number of challenging example tasks.
- The accompanied algortithm, SERL, is derived from RLPD which itself is a variant of SAC

Robotic Reinforcement Learning



Robotic Reinforcement Learning



Real-Time Bidding with Multi-Agent Reinforcement Learning in Display Advertising

Junqi Jin[§], Chengru Song [§], Han Li[§], Kun Gai[§], Jun Wang[†], Weinan Zhang[‡]
[§]Alibaba Group, [†]University College London, [‡]Shanghai JiaoTong University
{junqi.jjq, chengru.scr, lihan.lh, jingshi.gk}@alibaba-inc.com, j.wang@cs.ucl.ac.uk, wnzhang@sjtu.edu.cn

Jin et al. Real-Time Bidding with Multi-Agent Reinforcement Learning

RL for Bidding

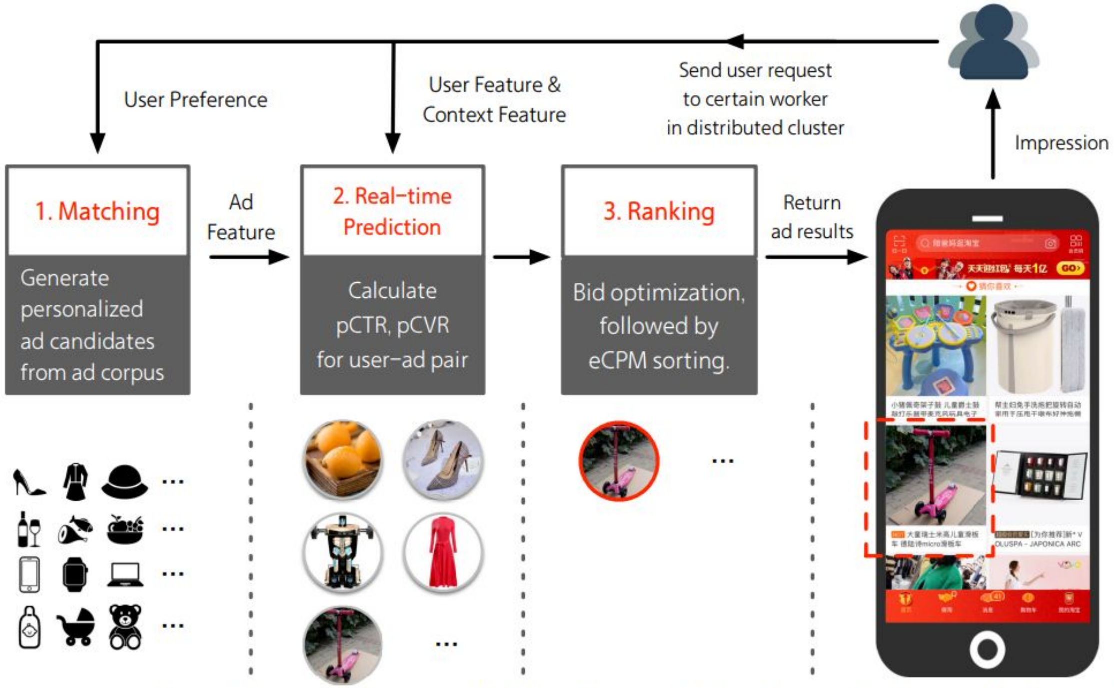


Figure 1: An Overview of Taobao Display Advertising System. Matching, RTP and Ranking modules sequentially process user requests, and finally return specified quantity of ads. These ads are shown in *Guess What You Like* of Taobao App, tagged by *Hot* (as shown in red dashed box) and surrounded with recommendation results.

RL for Bidding

Algorithm 1: DCMAB Algorithm

- Initialize $Q_i(s^q, a_1^q, \ldots, a_N^q, d|\theta_i^Q)$, actor $\mu_i([g, x]|\theta_i^\mu)$, target network Q_i', μ_i' with $\theta_i^{Q'} \leftarrow \theta_i^Q$, $\theta_i^{\mu'} \leftarrow \theta_i^\mu$ for each agent i.
- Initialize replay memory D
- **for** *episode = 1 to E* **do**
  - Initialize a random process $\mathcal{N}$ for action exploration
  - Receive initial state s for all agents
  - **for** t = *1 to T* **do**
    - For each agent i, compute a_i^q and add $\mathcal{N}_t$.
    - **for** *auctions in parallel workers in* T_p **do**
      - For each agent i, compute $bratio$ and combined with a_i^q compute adjusting ratio α and execute.
      - For each agent i, save reward, cost and maintain distribution d.
    - **end**
    - For each agent i, merge rewards, cost in last T_p to get reward r_i and update state to $s^{q\prime}$. Store $(s^q, d, a_i^q, r_i, s^{q\prime})$ to replay memory.
    - $s^{q\prime} \leftarrow s^q$
    - **for** *agent i=1 to N* **do**
      - Sample a random minibatch of S samples $(s^q, d, a_1^q, \ldots, a_N^q, r_i, s^{q\prime}, d')$ from D
      - Update critic by minimizing loss with Eqs.(9),(10).
      - Update actor with Eq. (11).
      - Update target network: $\theta' \leftarrow \tau\theta + (1-\tau)\theta$
    - **end**
  - **end**
- **end**

RL for Bidding

أكاديمية كاوست
KAUST ACADEMY

	AgentC1			AgentC2			AgentC3			Total		
Indices	ROI	CPA	COST	ROI	CPA	COST	ROI	CPA	COST	ROI	CPA	COST
Manual	100.00%	100.00%	99.65%	100.00%	100.00%	99.71%	100.00%	100.00%	99.42%	100.00%	100.00%	99.52%
Bandit	121.38%	82.43%	99.87%	159.41%	62.73%	99.53%	102.63%	97.39%	99.64%	112.14%	84.23%	99.63%
A2C	103.30%	96.57%	99.39%	106.85%	93.58%	99.60%	170.91%	58.55%	99.66%	158.38%	68.09%	99.62%
DDPG	577.87%	17.27%	99.51%	976.80%	10.23%	99.18%	164.29%	60.85%	99.76%	305.75%	24.26%	99.58%
DCMAB	690.18%	14.46%	99.38%	584.63%	17.10%	99.43%	275.11%	36.34%	99.57%	340.38%	24.84%	99.51%

References

These slides have been adapted from

- Ana Goldie, CS224R: Deep Reinforcement Learning, Stanford