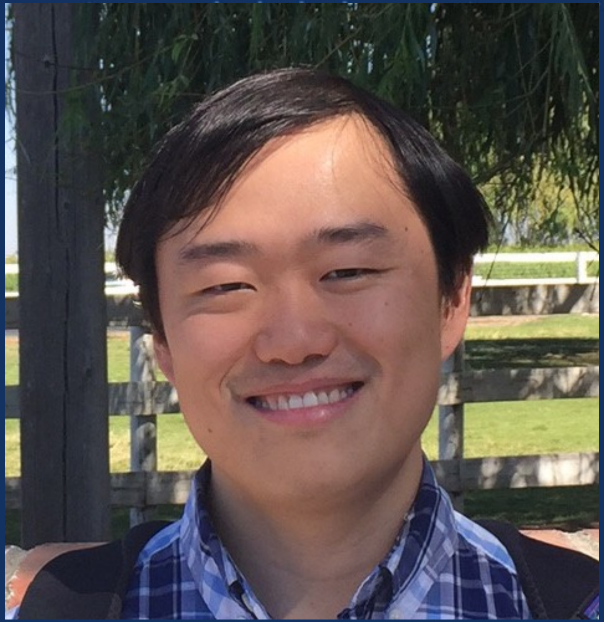
Going beyond games: towards decision making in the real-world

Yuandong Tian

Research Scientist and Senior Manager

Meta AI (FAIR)



facebook Artificial Intelligence

Reinforcement Learning


Go


Chess


Shogi


Poker


DoTA 2


StarCraft II

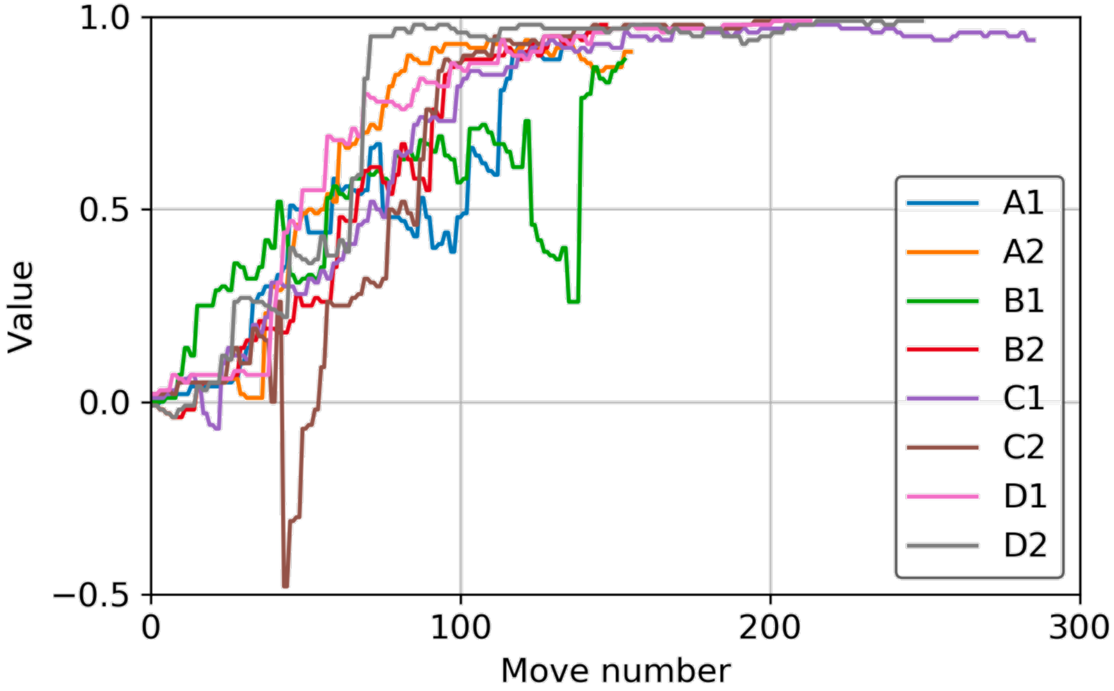
Big Success in Games

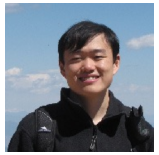
ELF OpenGo

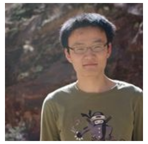
Vs top professional players

Name (rank)	ELO (world rank)	Result
Kim Ji-seok	3590 (#3)	5-0
Shin Jin-seo	3570 (#5)	5-0
Park Yeonghun	3481 (#23)	5-0
Choi Cheolhan	3466 (#30)	5-0

Single GPU, 80k rollouts, 50 seconds
Offer unlimited thinking time for the players








[ELF OpenGo: An Analysis and Open Reimplementation of AlphaZero, Y. Tian et al, ICML 2019]

What’s Beyond Games?

Chip Design (Google)

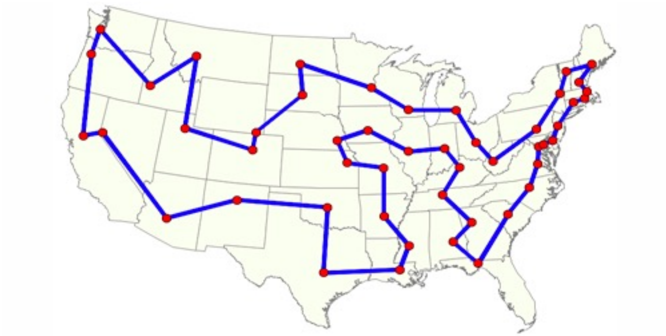
Several weeks with **human experts** in the loop

→

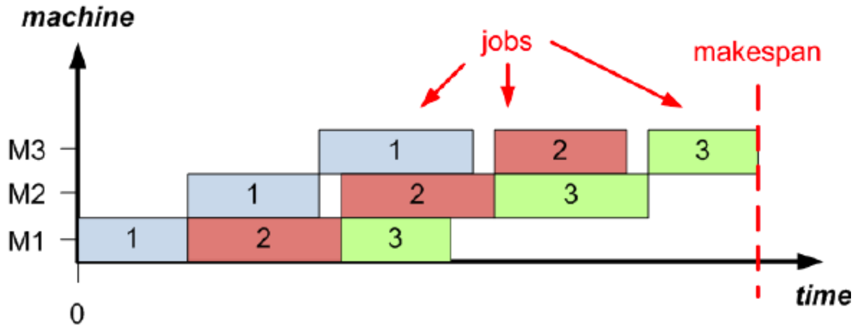
Fully automatic design in 6 hours

[A. Mirhoseini, A. Goldie, *A graph placement methodology for fast chip design,* Nature'21]

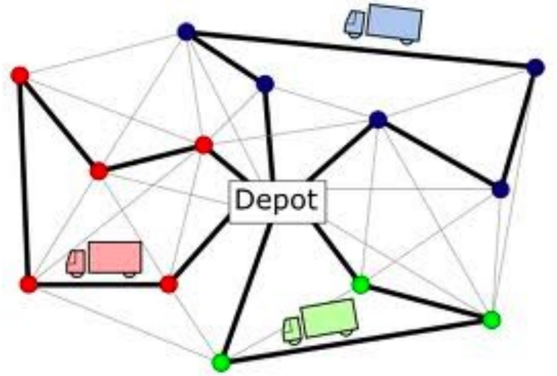
Optimization Problems



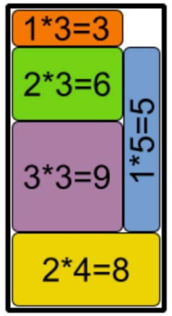
Travel Salesman Problem



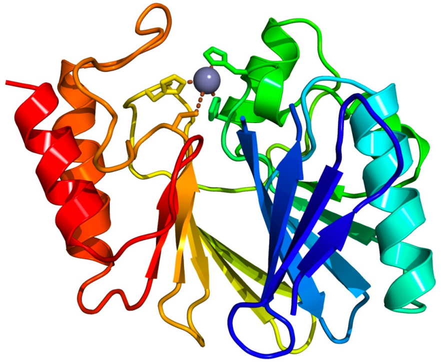
Job Scheduling



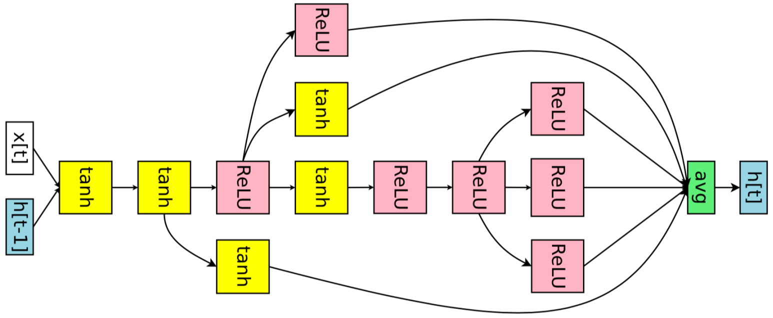
Vehicle Routing



Bin Packing



Protein Folding



Model-Search

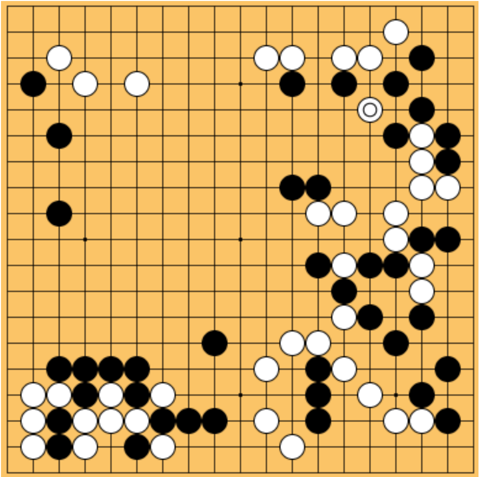
$$x^* = \arg\max_{x \in \Omega} f(x)$$

Wait…What?

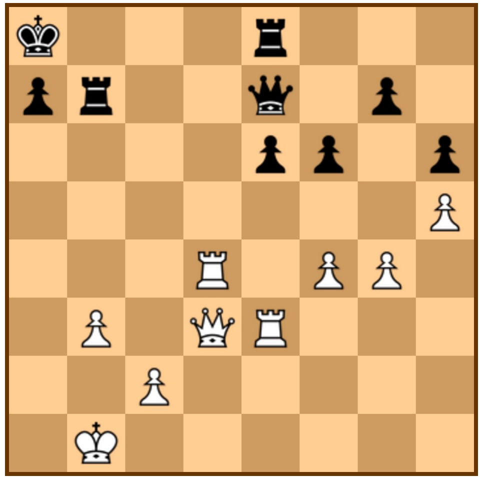
- Many problems are NP-hard problems.
  - No good algorithm unless $P = NP$
- These guarantees are worst-case ones.
  - To prove a lower-bound, construct an adversarial example to fail the algorithm
- For specific distribution, there might be better heuristics.
  - Human heuristics are good but may not be suitable for everything

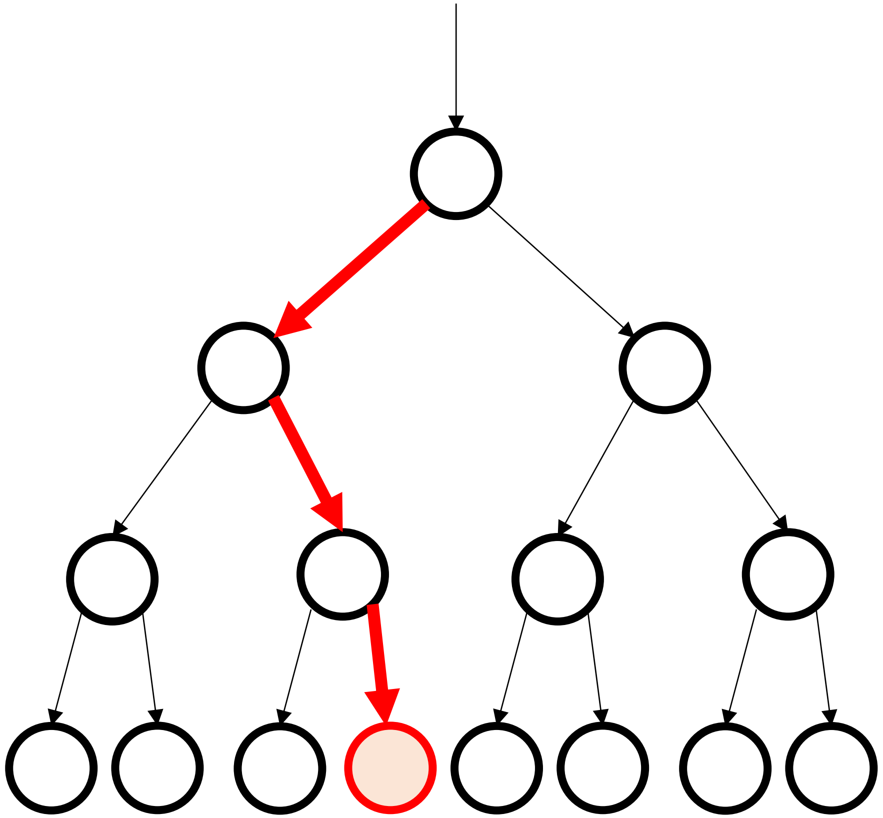
Efficient Search for Games

Go



Chess

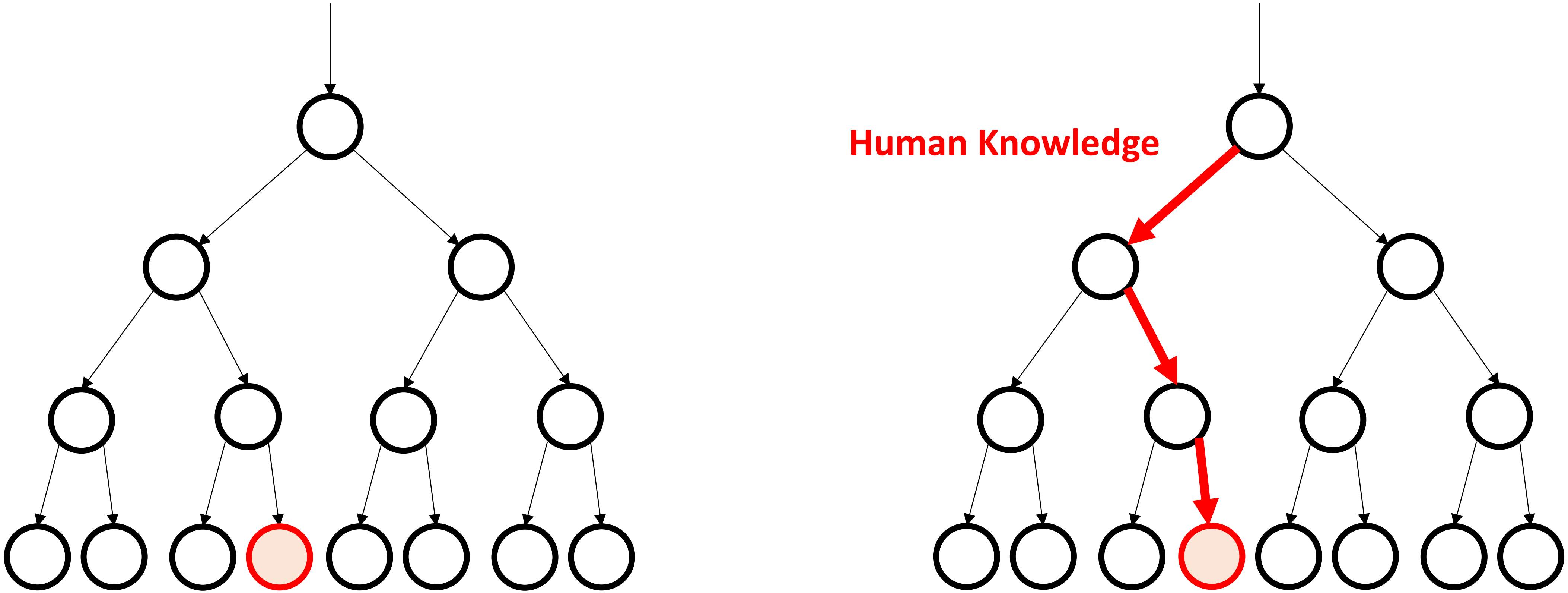




~~**Human Knowledge**~~

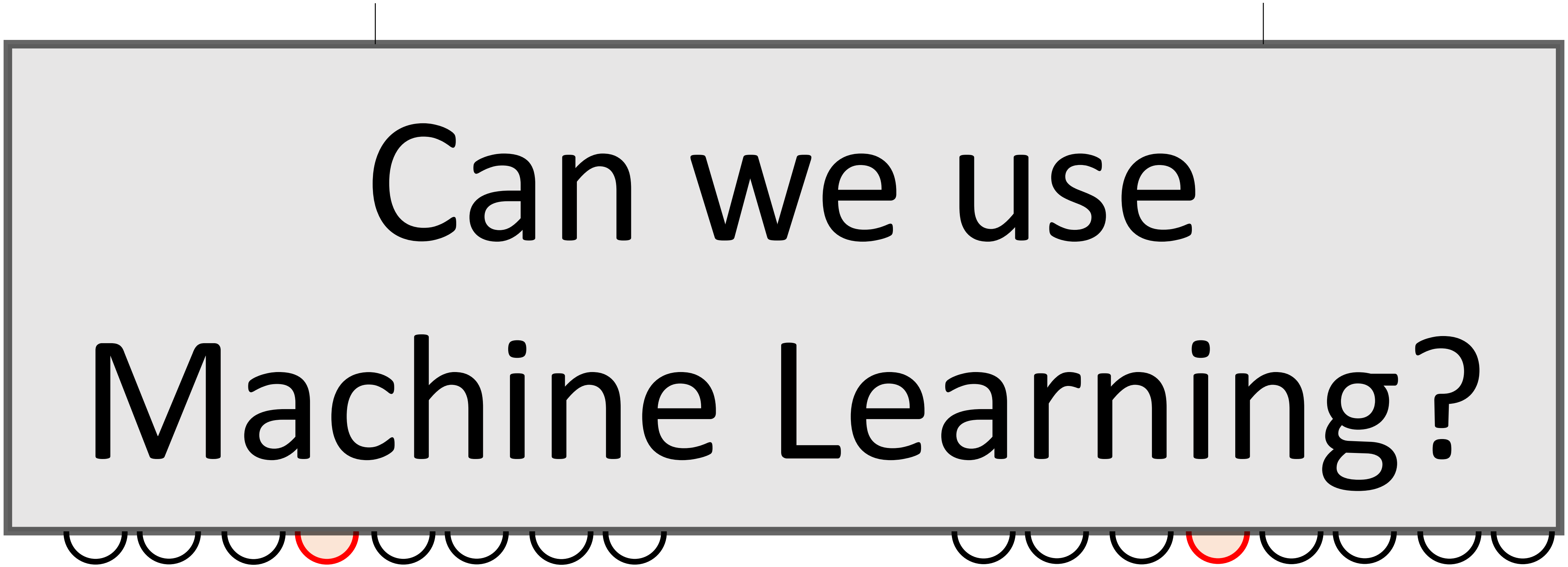
Machine learned models

Efficient Search for Optimization



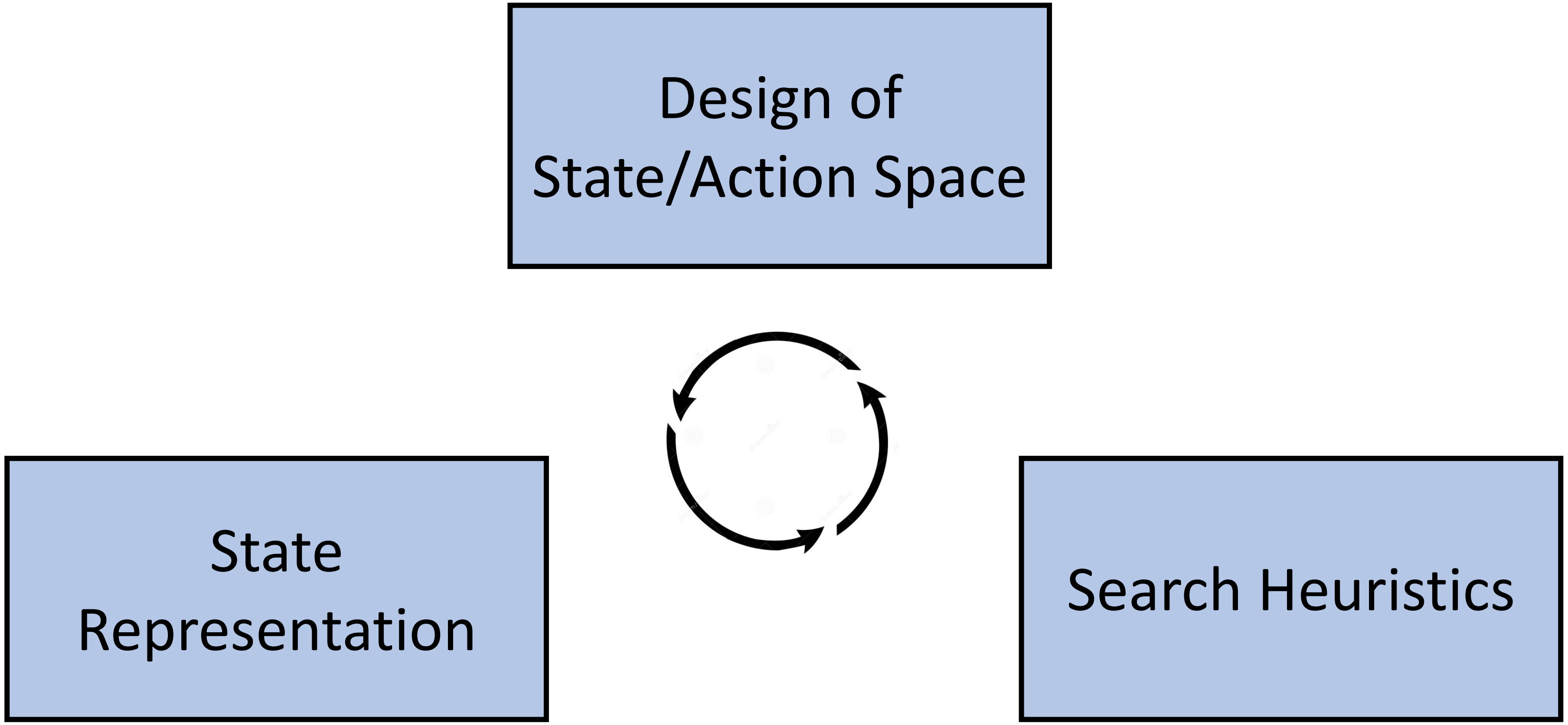
Exhaustive search to get **a good solution**

More Efficient Search for Optimization



Exhaustive search to get **a good solution**

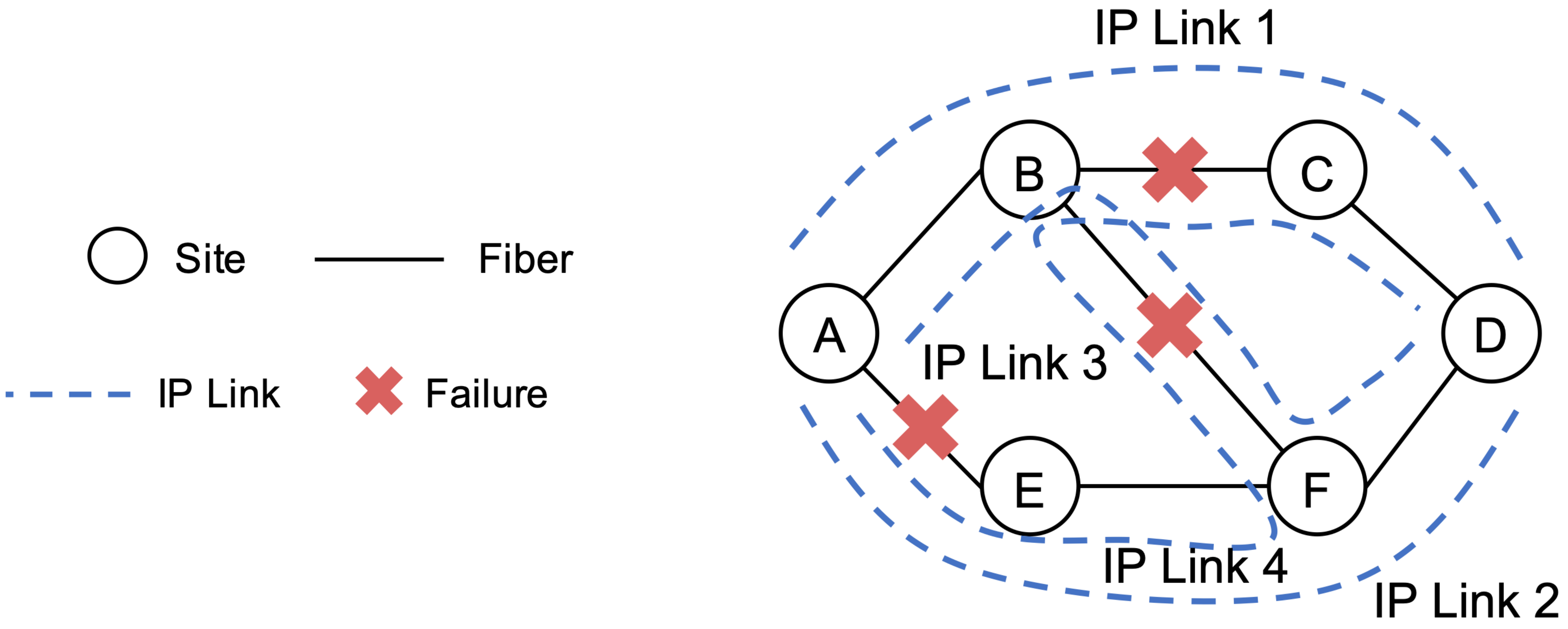
Components of Search



Part I: Learning Search Heuristics

NeuroPlan: Network planning problem

A->D: 100Gbps, under several single-fiber failures



[H. Zhu et al, **Network planning with deep reinforcement learning**, *ACM SIGCOMM'21]*

Existing approach

ILP problem

Both C_l and $Y(l, \omega, \lambda)$ are decision variables.

$$\min \sum_{l \in L} (C_l \times cost_{IP} + \sum_{f \in \Psi_l} cost_f) \quad (1)$$

Objective

$$\text{s.t.} \sum_{l:l_{src}=n} Y(l,\omega,\lambda) - \sum_{l:l_{dst}=n} Y(l,\omega,\lambda) = Traffic(\omega, n)$$

$$\forall \omega \in \Omega, \lambda \in \Lambda \quad (2)$$

Flow conservation constraints

$$C_l \geq \sum_{\omega} Y(l,\omega,\lambda), \forall \lambda \in \Lambda \quad (3)$$

Link capacity constraints

$$\sum_{l \in \Delta_f} C_l \times \phi_{lf} \leq S_f \quad (4)$$

Spectrum efficiency constraints

$$C_l \geq C_l^{min} \quad (5)$$

Existing topology constraints

Existing approach

ILP problem

$$\min \sum_{l \in L} (C_l \times cost_{IP} + \sum_{f \in \Psi_l} cost_f) \quad (1)$$

$$\text{s.t.} \sum_{l:l_{src}=n} Y(l,\omega,\lambda) - \sum_{l:l_{dst}=n} Y(l,\omega,\lambda) = Traffic(\omega,n)$$

$$\forall \omega \in \Omega, \lambda \in \Lambda \quad (2)$$

$$C_l \geq \sum_{\omega} Y(l,\omega,\lambda), \ \forall \lambda \in \Lambda \quad (3)$$

$$\sum_{l \in \Delta_f} C_l \times \phi_{lf} \leq S_f \quad (4)$$

$$C_l \geq C_l^{min} \quad (5)$$

ILP solvers (Gurobi / CPLEX / SCIP)

Scalability issues: takes days or weeks

Human-designed heuristics to trade optimality for tractability

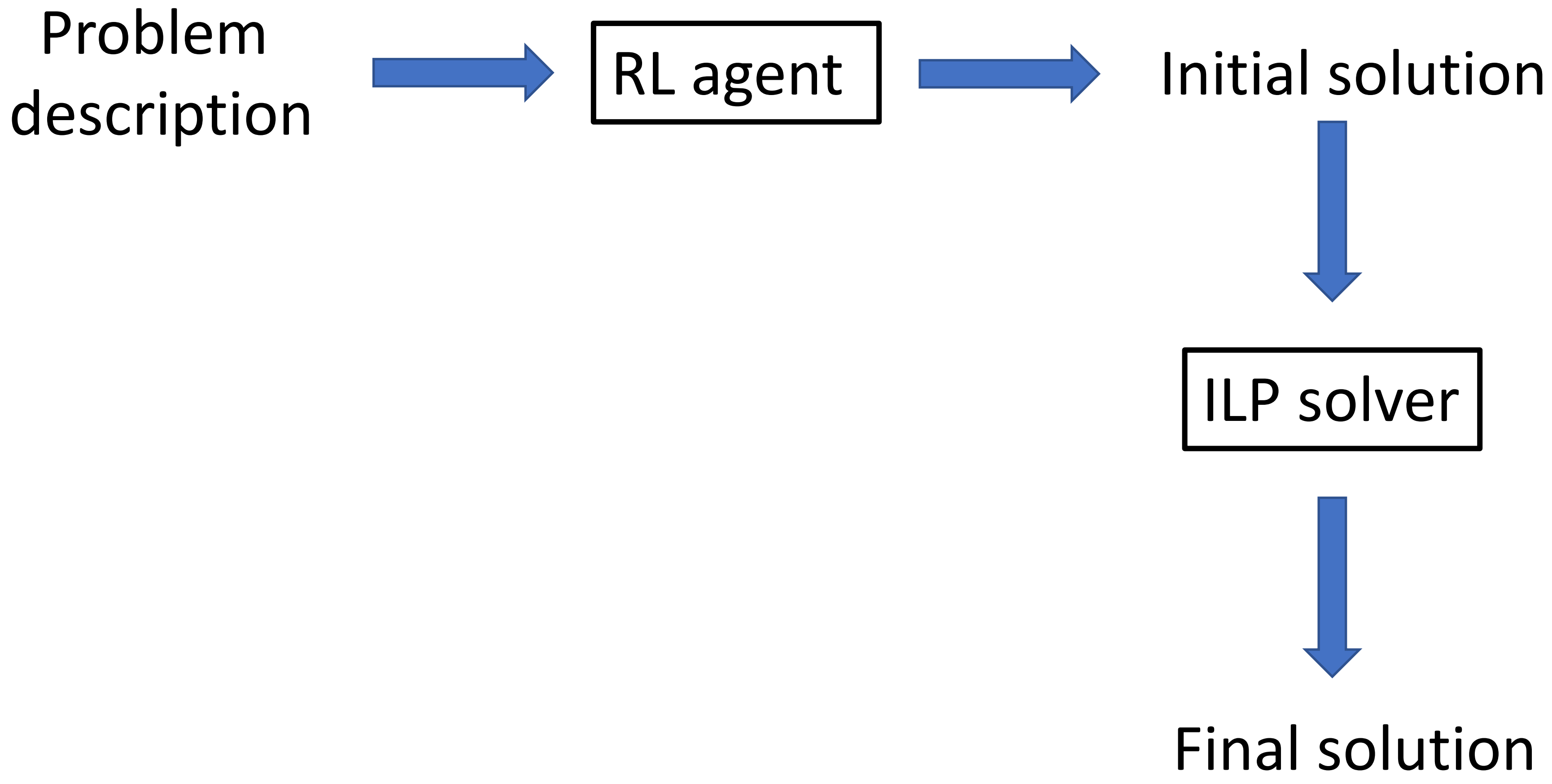
Topology decomposition

Topology transformation

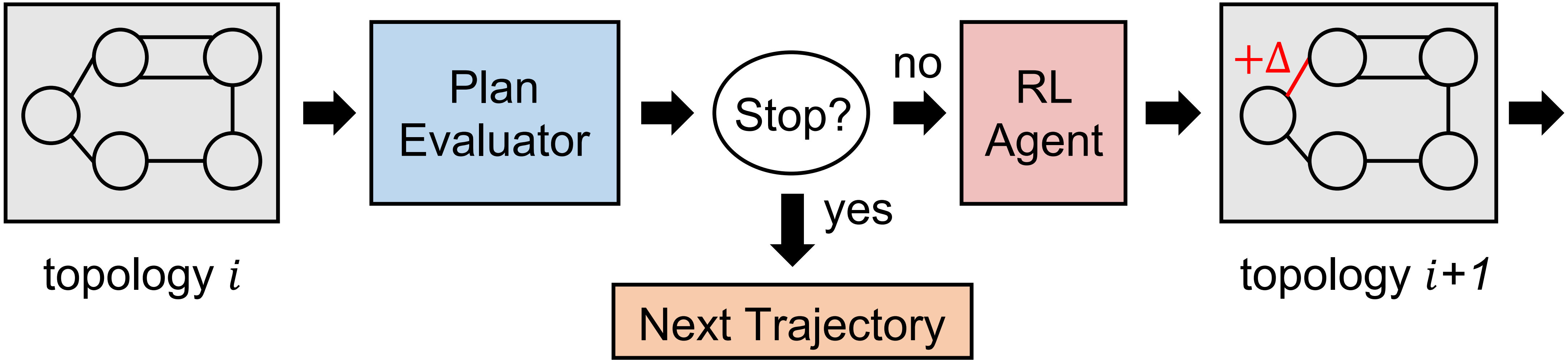
Failure selection

Problem: Hard to find feasible solutions!

Two-stage approach



Stage One



RL agent to learn topology (capacity variable C_l) that leads to feasible solutions

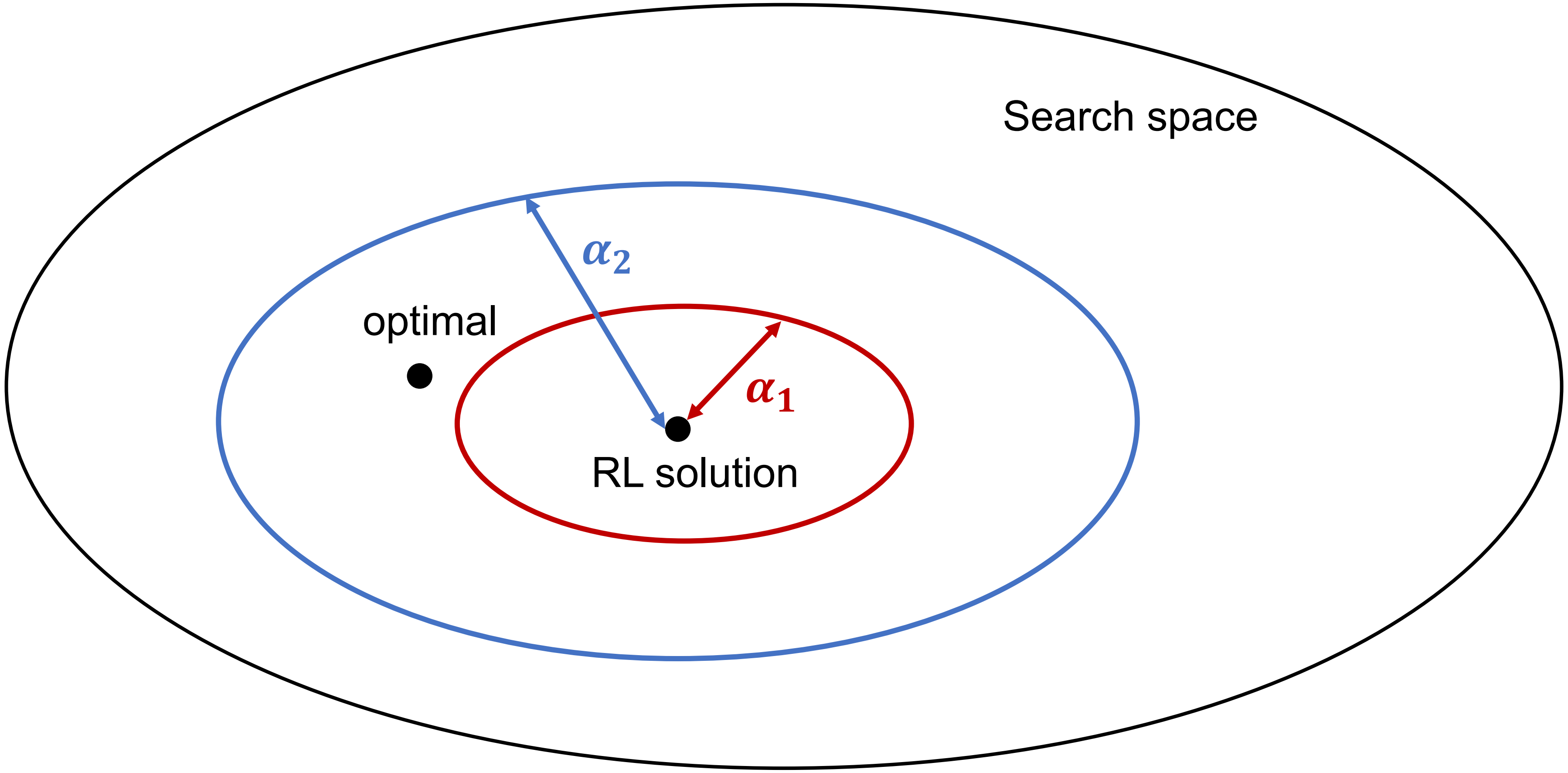
Plan evaluator to give rewards based on feasibility.

→ Checking feasibility is much faster than minimizing cost

→ Faster way to check: **source aggregation** (SA) and **stateful failure checking**

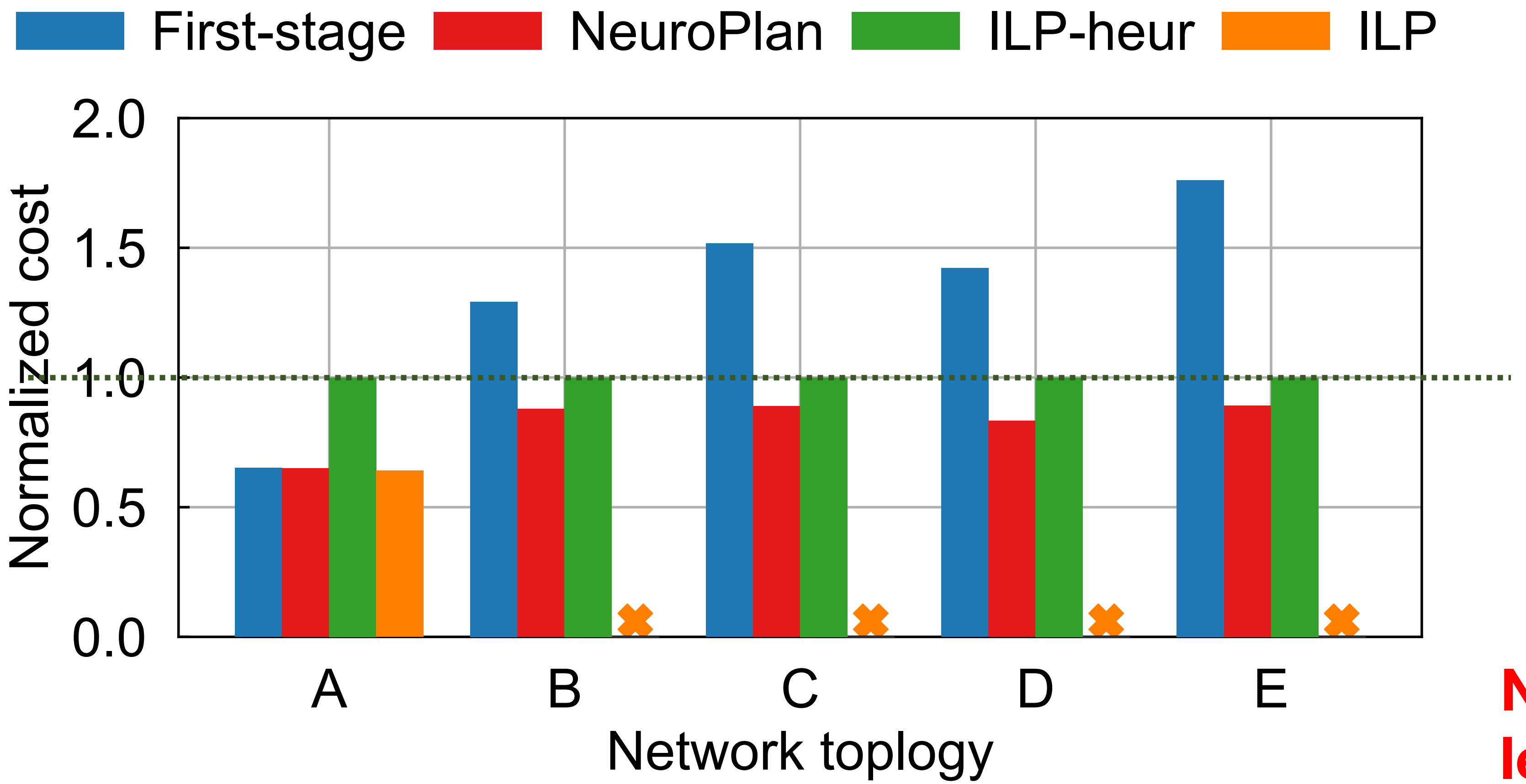
Stage Two

α: Relaxed Factors



Only search a local region of the RL solution to save ILP computation.

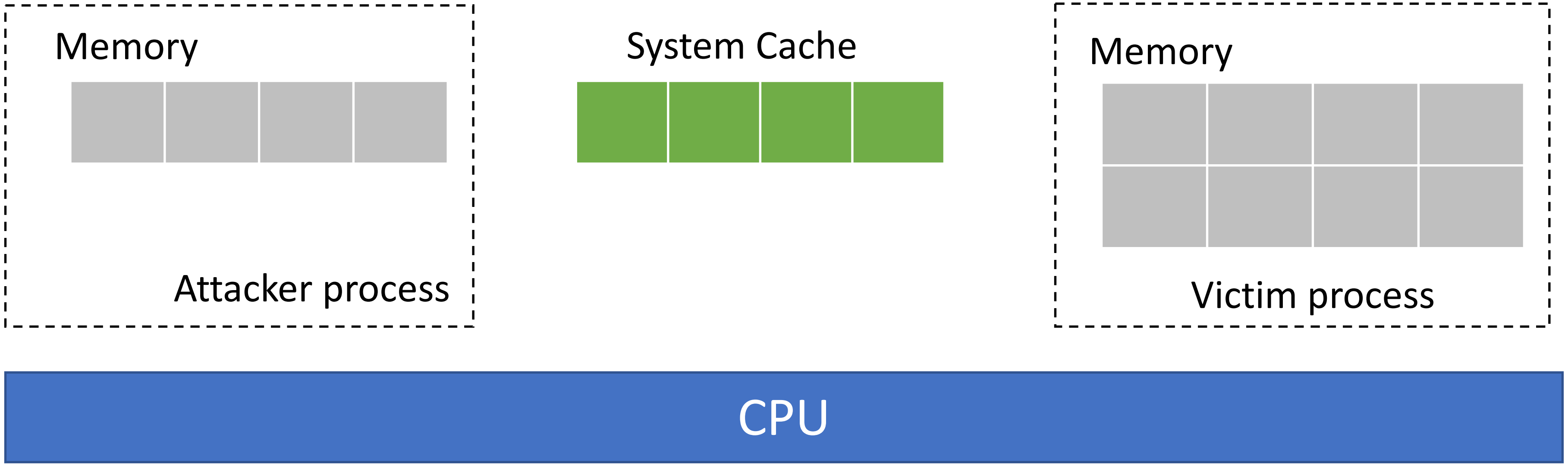
Solution quality versus Scalability



NeuroPlan automatically learns good heuristics.

https://github.com/netx-repo/neuroplan

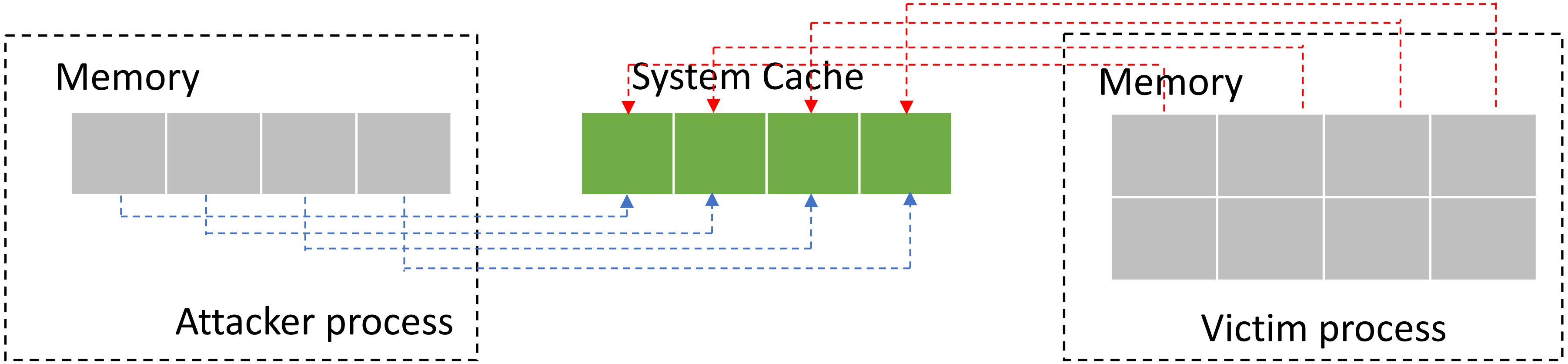
AutoCAT: Cache Side-Channel Attack



1. For different processes, their memory spaces are separate.
2. Different processes share the **same** system cache → **Security issues**

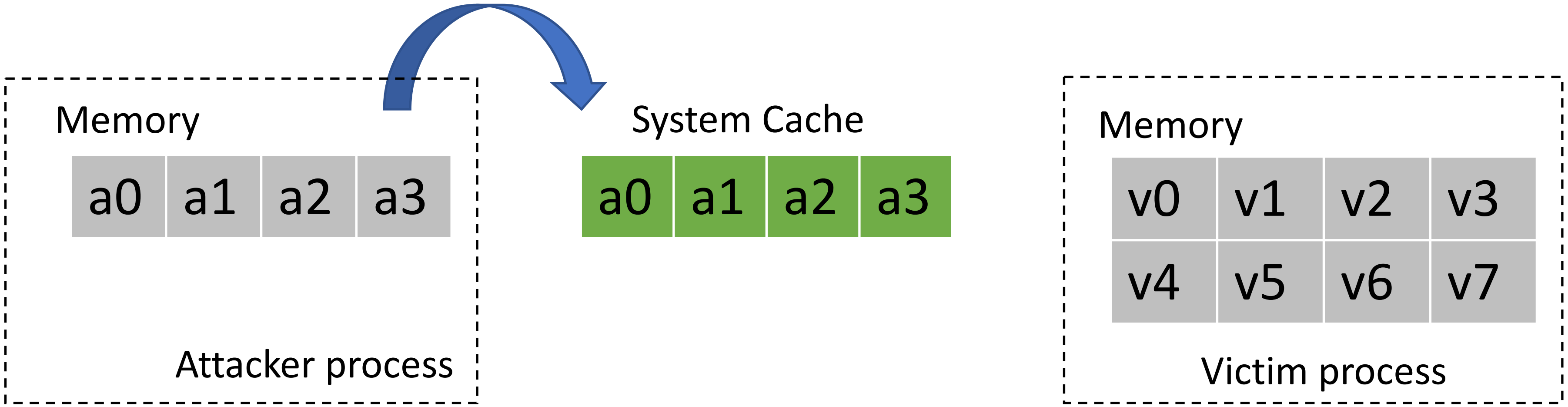
[M. Luo, W. Xiong*, et al, **AutoCAT:** Reinforcement Learning for Automated Exploration of Cache Timing-Channel Attacks]*

A Simple Example (Prime+Probe)



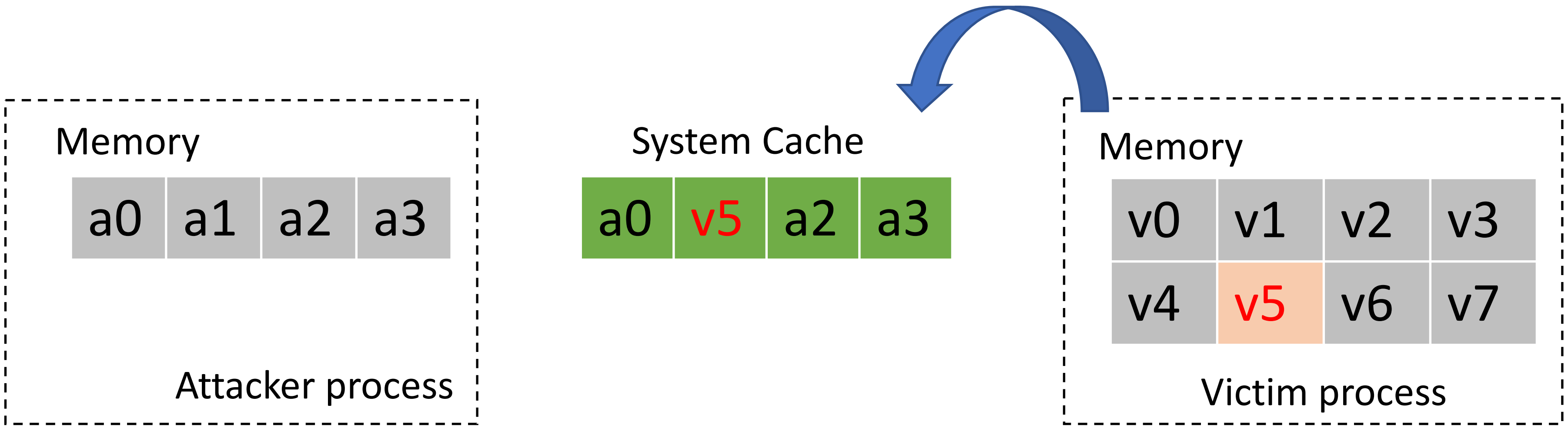
Direct-Mapped cache

A Simple Example (Prime+Probe)



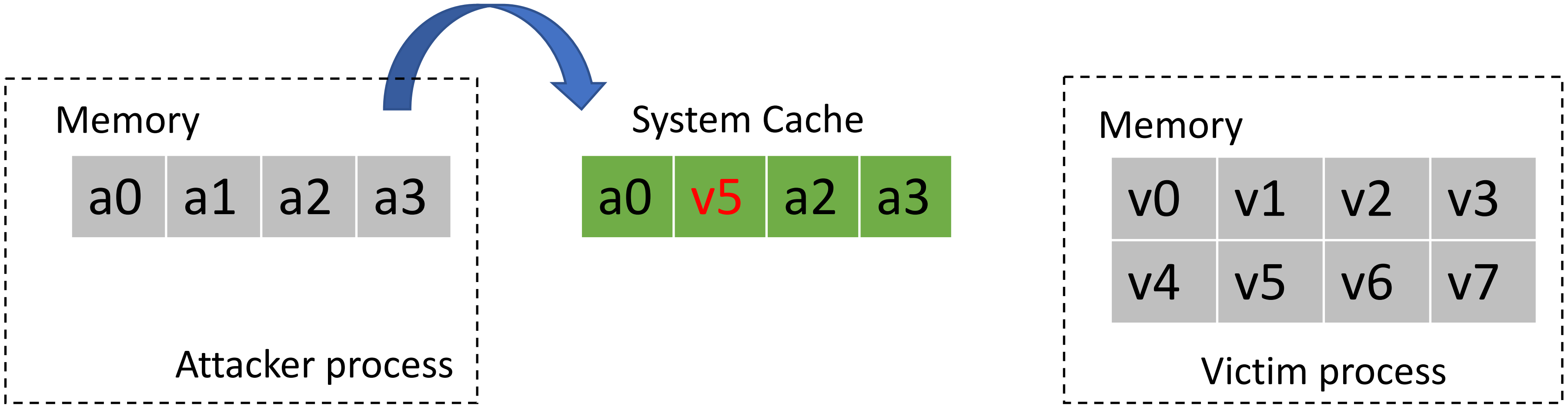
Attacker accesses a0..a3

A Simple Example (Prime+Probe)



Victim accesses *secret address* **v5**

A Simple Example (Prime+Probe)



Attacker accesses a0, cache hit, fast memory access

Attacker accesses a1, cache **miss**, **slow** memory access
→ Victim's **secret address** must be v1 or v5.

AutoCAT: RL to discover strategies automatically

- Discover novel form of side-channel attack is hard
  - Require expert knowledge with time commitment
  - New architecture / platforms emerge
  - Existing domain knowledge may lead to local optimal solutions, need "creativity"

Why not use Reinforcement Learning?

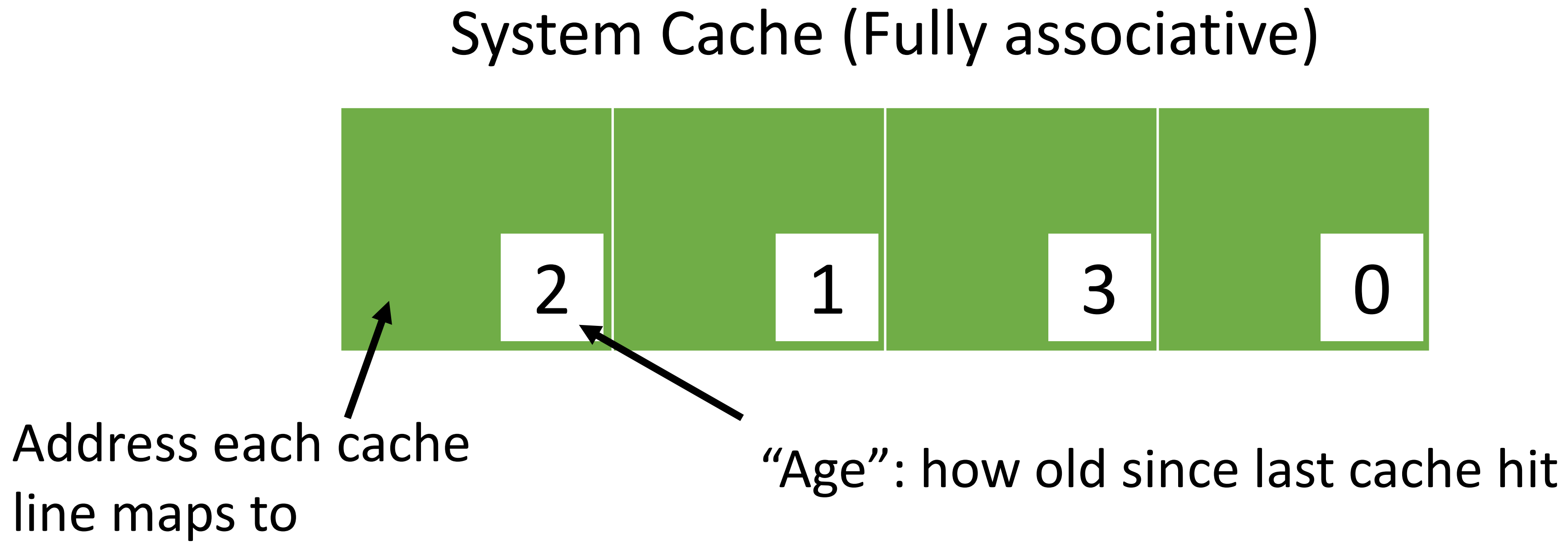
AutoCAT: RL to discover strategies automatically

No.	Cache config.		Attacker& victim config.			Expected attacks	Example Attack found by AutoCAT	
	Type†	Ways used	Sets	Victim addr	Attack addr	Flush inst	Possible attacks‡	Attack sequence (p indicates prefetch)	Attack Category
1	DM	1	4	0-3	4-7	no	PP	$5 \to 4 \to 7 \to v \to 5 \to 7 \to 4 \to g$	PP
2	DM+PFnextline	1	4	0-3	4-7	no	PP	$6(p7) \to 4(p5) \to v \to 4(p5) \to 5(p6) \to g$	PP
3	DM	1	4	0-3	0-3	yes	FR	$\ldots \to f1 \to v \to 1 \to f0 \to v \to f2 \to v \to 2 \to f3 \to 0 \to g$	FR
4	DM	1	4	0-3	0-7	no	ER, PP	$\ldots \to 3 \to 7 \to 4 \to 6 \to v \to 3 \to 0 \to 6 \to 4 \to g$	ER and PP
5	FA	4	1	0/E	4-7	no	PP, LRU	$4 \to 6 \to 7 \to v \to 5 \to 4 \to g$	LRU
6	FA	4	1	0/E	0-3	yes	FR, LRU	$0 \to 3 \to 1 \to 2 \to f0 \to 2 \to v \to 3 \to 0 \to g$	FR
7	FA	4	1	0/E	0-7	no	ER, PP, LRU	$v \to 4 \to 1 \to 6 \to 7 \to v \to 1 \to v \to 5 \to 6 \to g$	LRU
8	FA	4	1	0-3	0-3	yes	FR, LRU	$f3 \to f2 \to v \to 2 \to 3 \to f0 \to v \to 0 \to g$	FR
9	FA	4	1	0-3	0-7	yes	FR, LRU	$f0 \to f2 \to f1 \to v \to 2 \to 1 \to 0 \to g$	FR
10	DM	1	8	0-7	0-7	yes	FR	$f2 \to v \to 2 \to f4 \to f0 \to v \to 0 \to 4 \to f3 \to f7 \to v \to 3 \to v \to$ $7 \to f1 \to f6 \to v \to 6 \to 1 \to g$	FR
11	FA	8	1	0/E	0-7	yes	FR, LRU	$f0 \to v \to 0 \to g$	FR
12	FA	8	1	0/E	0-15	no	ER, PP, LRU	$7 \to 11 \to 10 \to 5 \to 4 \to 2 \to 3 \to 1 \to v \to 0 \to g$	ER
13	FA+PFnextline	8	1	0/E	0-15	no	ER, PP, LRU	$4\ (p5) \to 9\ (p10) \to 15\ (p16) \to 2\ (p3) \to v \to 0\ (p1) \to g$	ER
14	FA+PFstream	8	1	0/E	0-15	no	ER, PP, LRU	$15 \to 9 \to 8 \to 7(p6) \to 11 \to 6 \to 12 \to 14 \to v \to 0 \to g$	ER

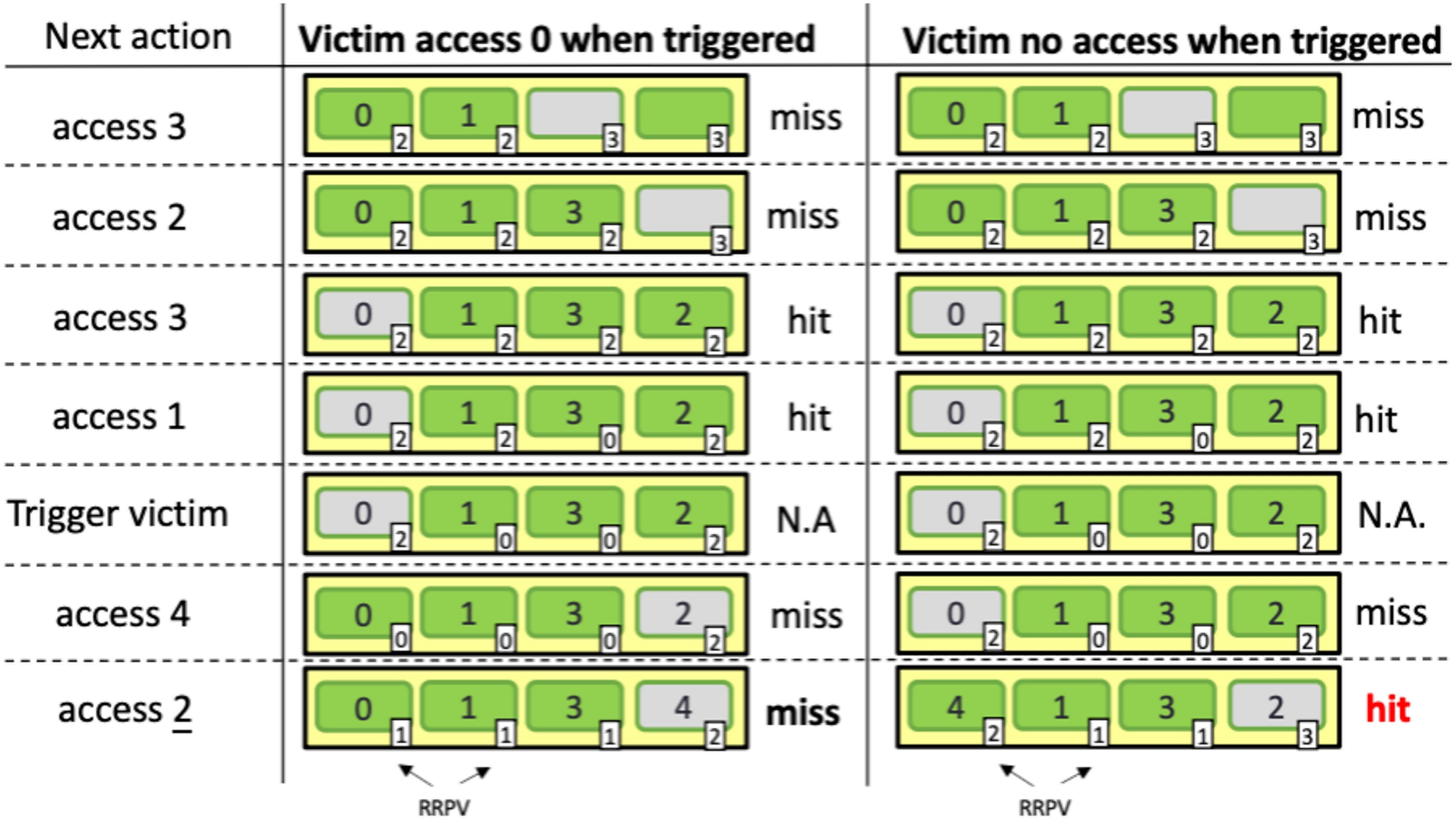
† FA: fully-associative, DM:direct-mapped, PFnextline: nextline prefetcher, PFstream: stream prefetcher. ‡ FR: flush+reload, ER: evict+reload, PP: prime+probe.

[M. Luo*, W. Xiong*, et al, **AutoCAT:** *Reinforcement Learning for Automated Exploration of Cache Timing-Channel Attacks*]
https://arxiv.org/abs/2208.08025

AutoCAT: Real hardware



AutoCAT: Real hardware



Victim access affects RRPV of address 0, affecting which line will be replaced when attacker accesses address 4

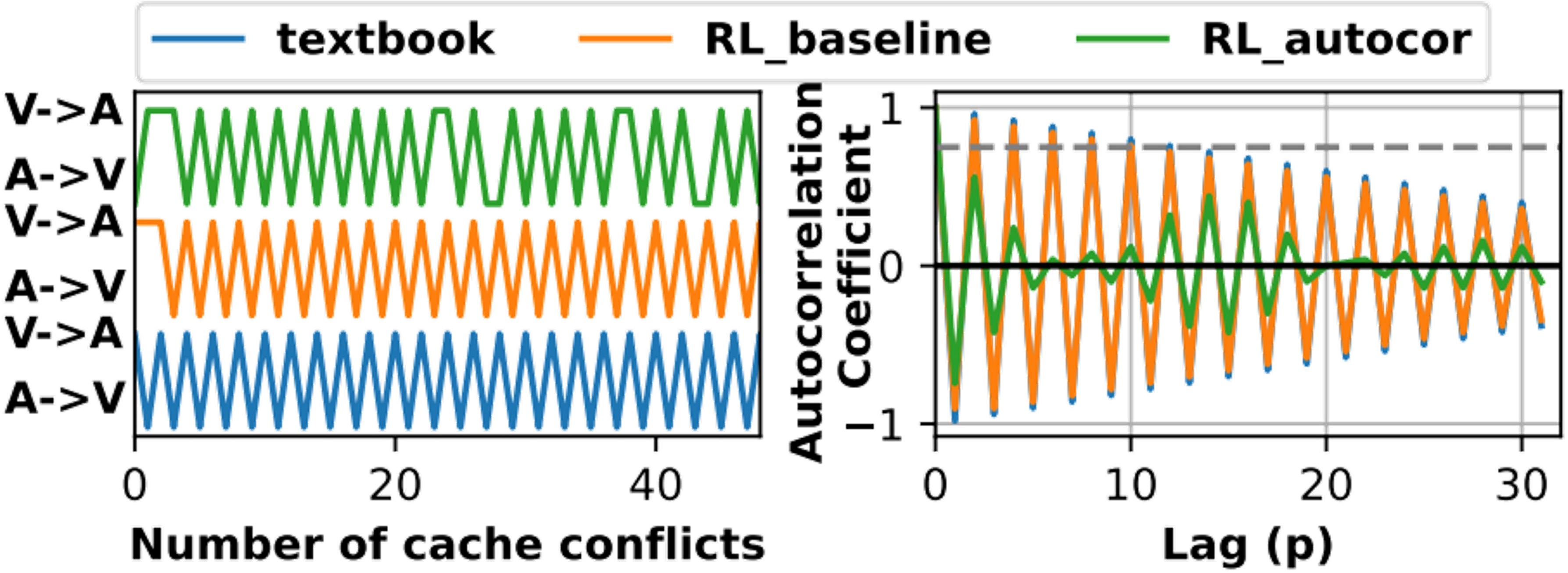
AutoCAT: Real hardware

CPU	Cache level	#Ways	Rep. Pol.	Victim addr.	Attacker addr.	Example attack sequence found by AutoCAT	Accuracy
Xeon E5-2660v3 (Haswell)	L1	8	PLRU	0/E	0-8	$2 \to 1 \to 5 \to 6 \to 4 \to 4 \to 7 \to 8 \to 4 \to 8 \to v \to 3 \to 4 \to v \to 0 \to g$	0.999
	L2	8	PLRU	0/E	0-8	$1 \to 8 \to 2 \to 3 \to 4 \to 7 \to 2 \to 5 \to 4 \to 2 \to 8 \to 6 \to v \to 6 \to 3 \to 6 \to 7 \to 1 \to g$	0.999
Core i7-6700 (SkyLake)	L1	8	PLRU	0/E	0-8	$1 \to v \to 4 \to v \to 5 \to v \to 5 \to 5 \to 3 \to 8 \to 4 \to v \to 0 \to 2 \to 0 \to 1 \to v \to 8 \to 4 \to v \to g$	0.996
	L2	4	N.O.D.‡	0/E	0-8	$0 \to 1 \to 7 \to 3 \to 6 \to 6 \to 6 \to 6 \to v \to 5 \to 0 \to 4 \to 1 \to 7 \to 5 \to g$	0.997
	L3	4†	N.O.D.	0/E	0-8	$v \to v \to 4 \to 0 \to 5 \to 1 \to 1 \to 4 \to 2 \to 7 \to 3 \to 3 \to v \to v \to 3 \to 0 \to g$	1.0
	L3	8†	N.O.D.	0/E	0-8	$\ldots \to 3 \to v \to 3 \to v \to 6 \to 7 \to 3 \to 3 \to 5 \to 1 \to 5 \to 1 \to 6 \to g$	0.966
Core i7-7700K (KabyLake)	L3	4†	N.O.D.	0/E	0-8	$1 \to 2 \to 6 \to 6 \to 8 \to 8 \to 8 \to v \to 0 \to g$	1.0
	L3	8†	N.O.D.	0/E	0-8	$7 \to 7 \to 3 \to 4 \to 6 \to 0 \to 2 \to 1 \to 6 \to 5 \to 3 \to 2 \to v \to 5 \to 4 \to 1 \to 2 \to 8 \to v \to 8 \to 7 \to 6 \to 6 \to 3 \to 3 \to 4 \to g$	0.991

RL finds long sequence of memory access patterns to setup the **cache state** properly, before trigger to victim

AutoCAT: RL to learn attacker to bypass defenders

Against Rule-based defender (autocorrelation)



(a) Event train

(b) Autocorrelograms

AutoCAT: RL to learn attacker to bypass defenders

Against ML-based defender

Attackers	Bit rate (guess/step)	Attack accuracy	SVM detection rate
Textbook	0.1625	1	1
RL_baseline	0.228	0.990	0.907
RL_SVM	0.150	0.964	0.021

AutoCAT: RL method and backbone

- Method to use
  - Proximal Policy Optimization (PPO)
  - Takes several hours to find good policies.
- Backbone
  - Transformers work much better than MLP, which is surprising.
  - Transformer may pick up the promising memory access sequences efficiently, given initial random explorations.

AutoCAT: Future works

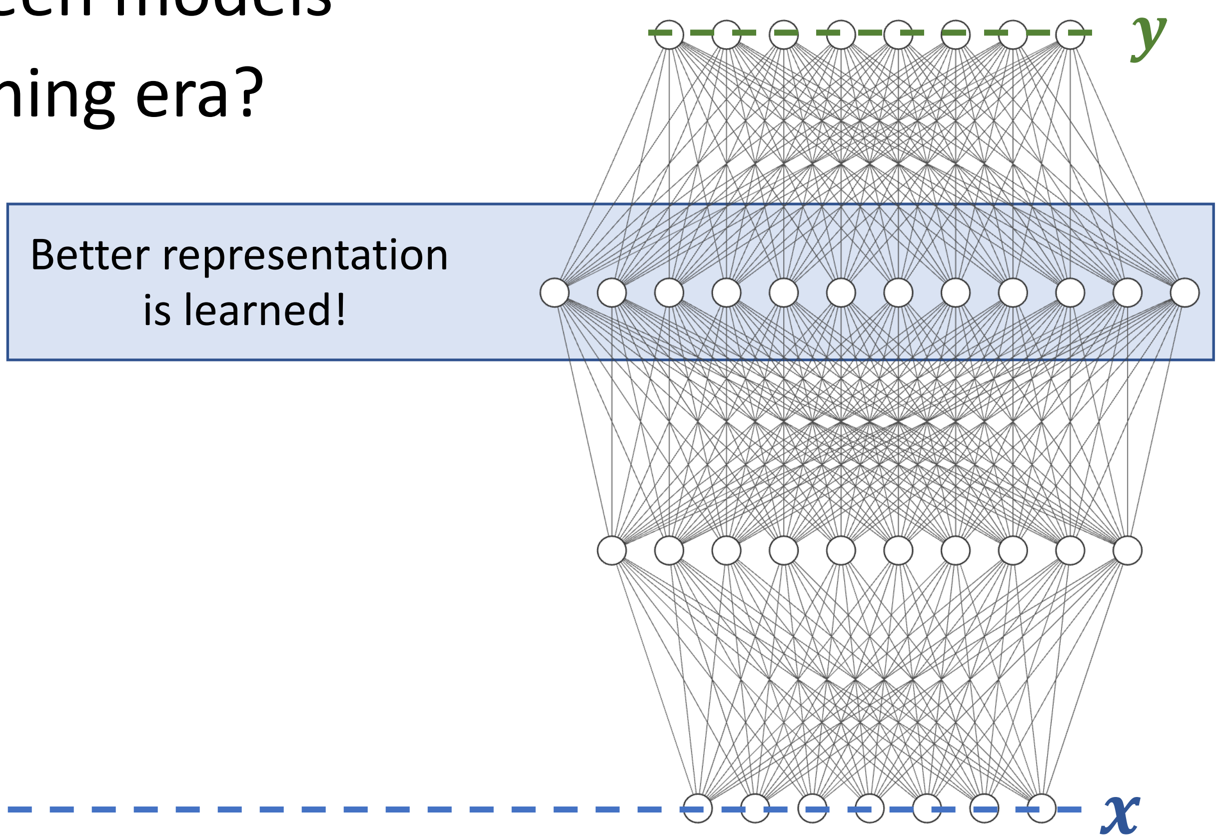
- Generalizable policies across different system settings
  - Memory-Cache mapping (e.g., Direct Mapping, Set-Associative)
  - Cache perfetch/replacement strategies
  - Context-switch among processes
  - The presence of normal programs / defenders

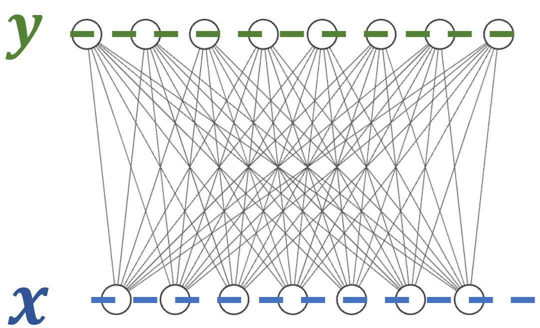
- Game settings
  - Partially observable game between attackers and defenders.
  - Learn better defender as well.

Part II: Learning Representation of State Space

Representation Learning

What's the difference between models
before/after deep learning era?





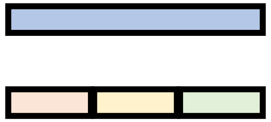
Linear Regression

Deep Models

Representation Learning in RL

Vectorized State

Action



Game Images

What the current AI sees

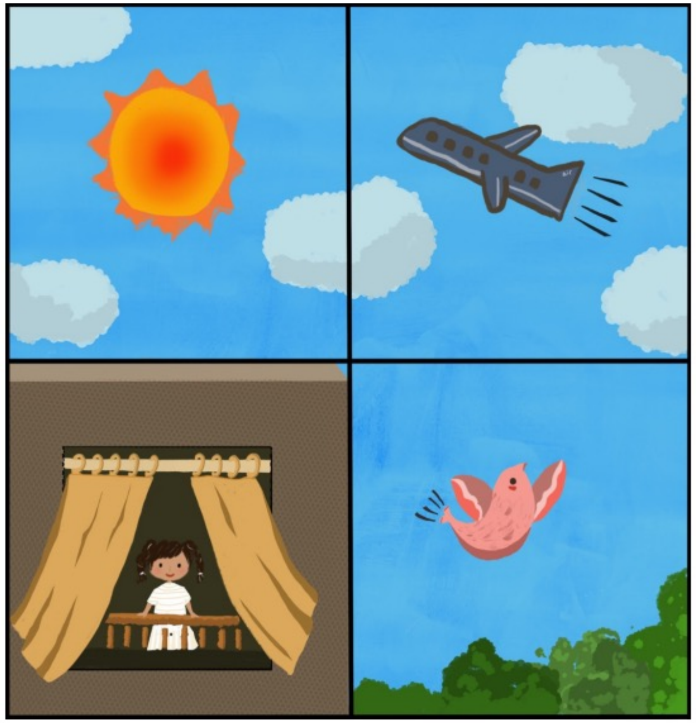
What human players really see

Denoised MDP

Reward-relevant Reward-irrelevant

Uncontrollable

Controllable

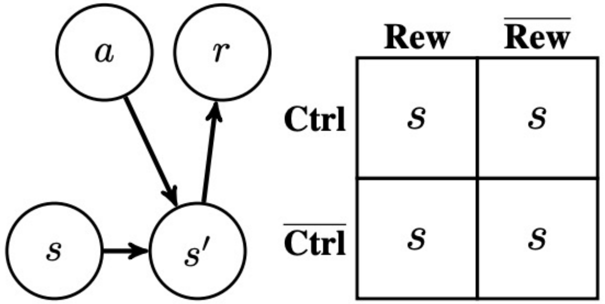


Denoise MDP

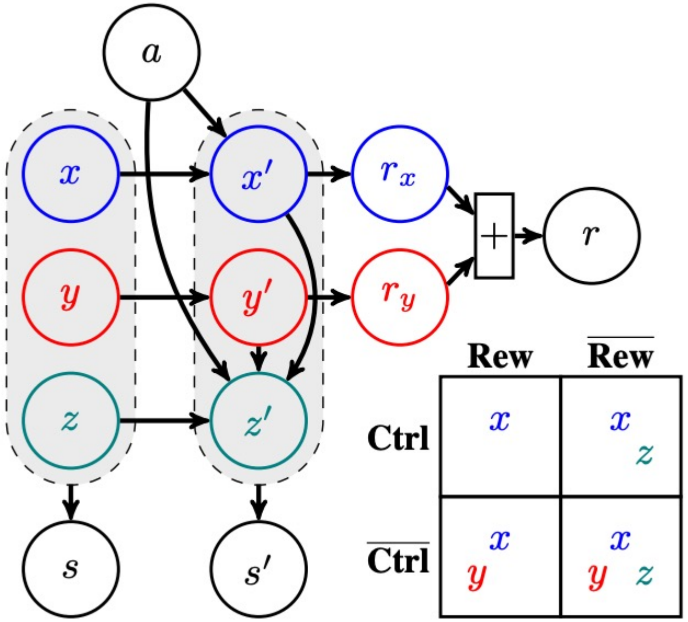
GOAL: Letting in as much sunlight as possible

[*T. Wang, et al,* **Denoised MDPs:** Learning World Models Better Than The World Itself, ICML 2022]

Denoised MDP



(a) Transition without useful structure. s may contain any type of information.



(c) Transition that factorizes out uncontrollable y and reward-irrelevant z.

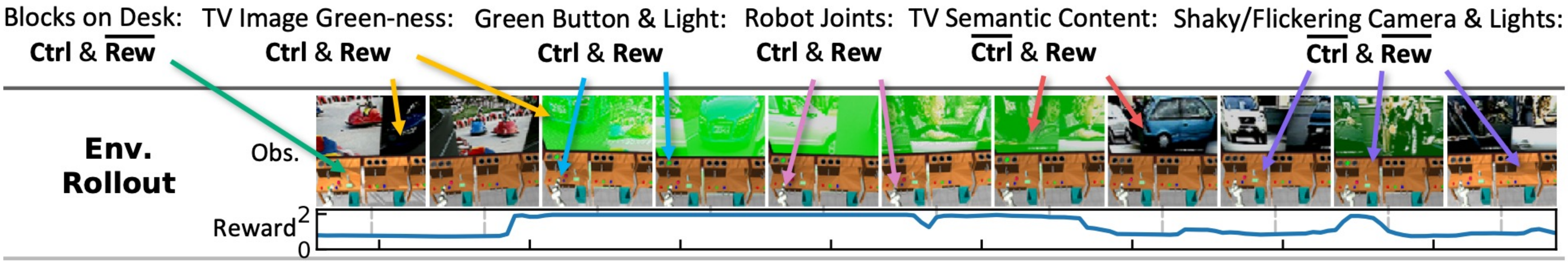
Controllability: Uncontrollable factor dynamics are independent with other factors and actions (and only optionally additively affects rewards so the the set of optimal policies stay the same).

Reward Relevance: Reward-irrelevant factor does not affect any other factor or reward.
Equivalently, if MDP transition can be represented as <u>below right</u>, latent y is **uncontrollable**, latent z is **reward-irrelevant**.

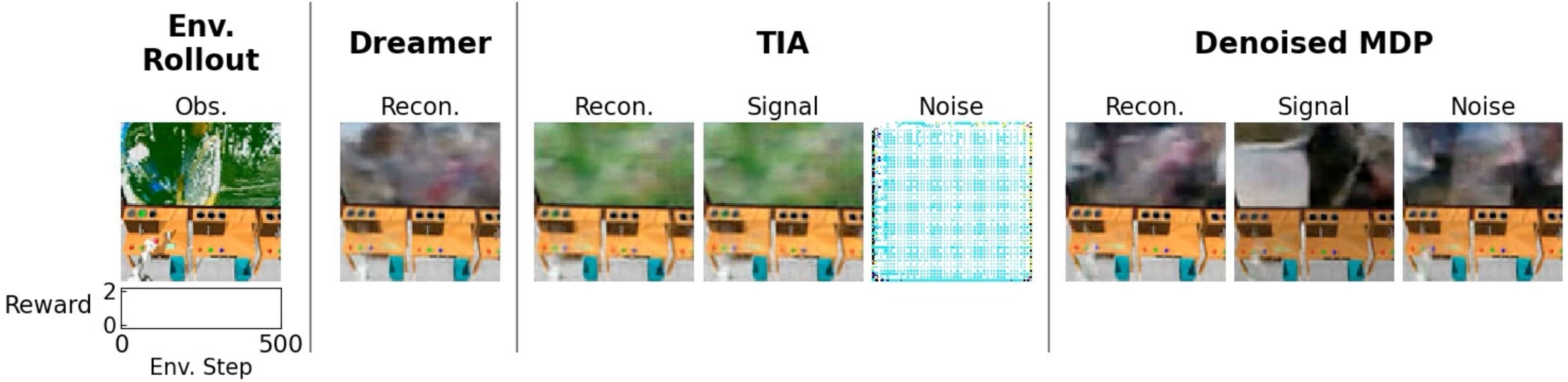
Objective: Minimizing reconstruction error **with the designed state/action/reward structure**

Experiments on (extended) RoboDesk

Original RoboDesk Env



Experiments on (extended) RoboDesk

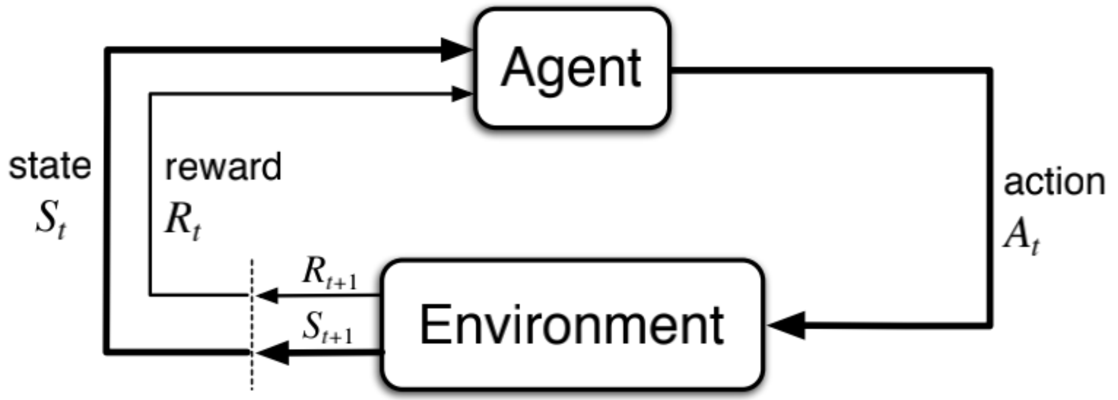


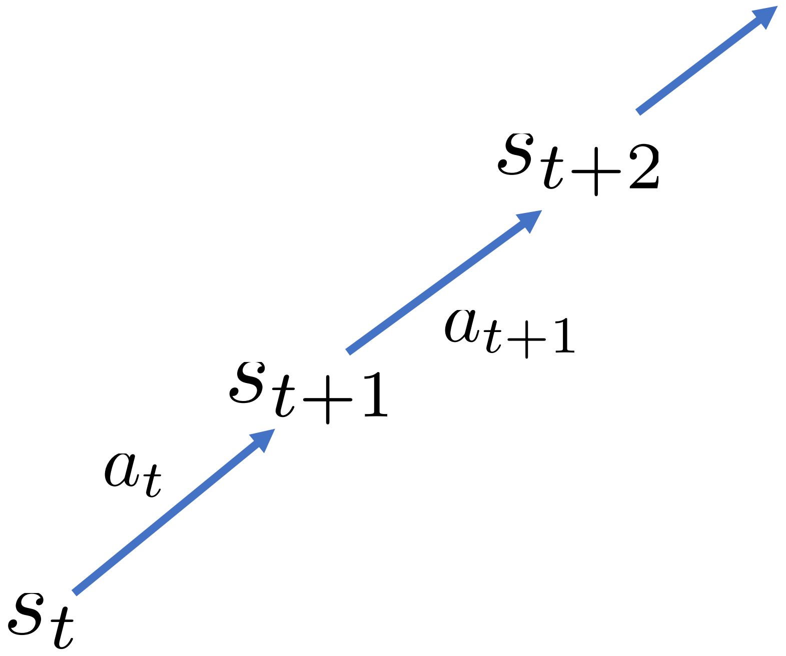
DMC policy optimization with learned representation

	Model-based			Model-free						
	Policy Learning: Backprop via Dynamics			**Policy Learning:** SAC (Latent-Space)			DBC	PI-SAC (No Aug.)	CURL (Use Aug.)	State-Space SAC (Upper Bound)
	Denoised MDP	TIA	Dreamer	Denoised MDP	TIA	Dreamer				
Noiseless	801.4 ± 96.6	769.7 ± 97.1	**848.6 ± 137.1**	**587.1 ± 98.7**	480.2 ± 125.5	575.4 ± 146.2	297.4 ± 72.5	246.4 ± 56.6	417.3 ± 183.2	910.3 ± 28.2
Video Background	**597.7 ± 117.8**	407.1 ± 225.4	227.8 ± 102.7	309.8 ± 153.0	**318.1 ± 123.7**	188.7 ± 78.2	188.0 ± 67.4	131.7 ± 20.1	478.0 ± 113.5	910.3 ± 28.2
Video Background + Noisy Sensor	**563.1 ± 143.0**	261.2 ± 200.4	212.4 ± 89.7	**288.2 ± 123.4**	197.3 ± 124.2	218.2 ± 58.1	79.9 ± 36.0	152.5 ± 12.6	354.3 ± 119.9	919.8 ± 100.7
Video Background + Camera Jittering	**254.1 ± 114.2**	151.7 ± 160.5	98.6 ± 27.7	**186.8 ± 47.7**	126.5 ± 125.6	105.2 ± 33.8	68.0 ± 38.4	91.6 ± 7.6	**390.4 ± 64.9**	910.3 ± 28.2

Project page: https://ssnl.github.io/denoised_mdp/

Representation Learning in RL





Is the temporal nature a **blessing** or a **curse**?

Planning In a Trajectory Latent Space

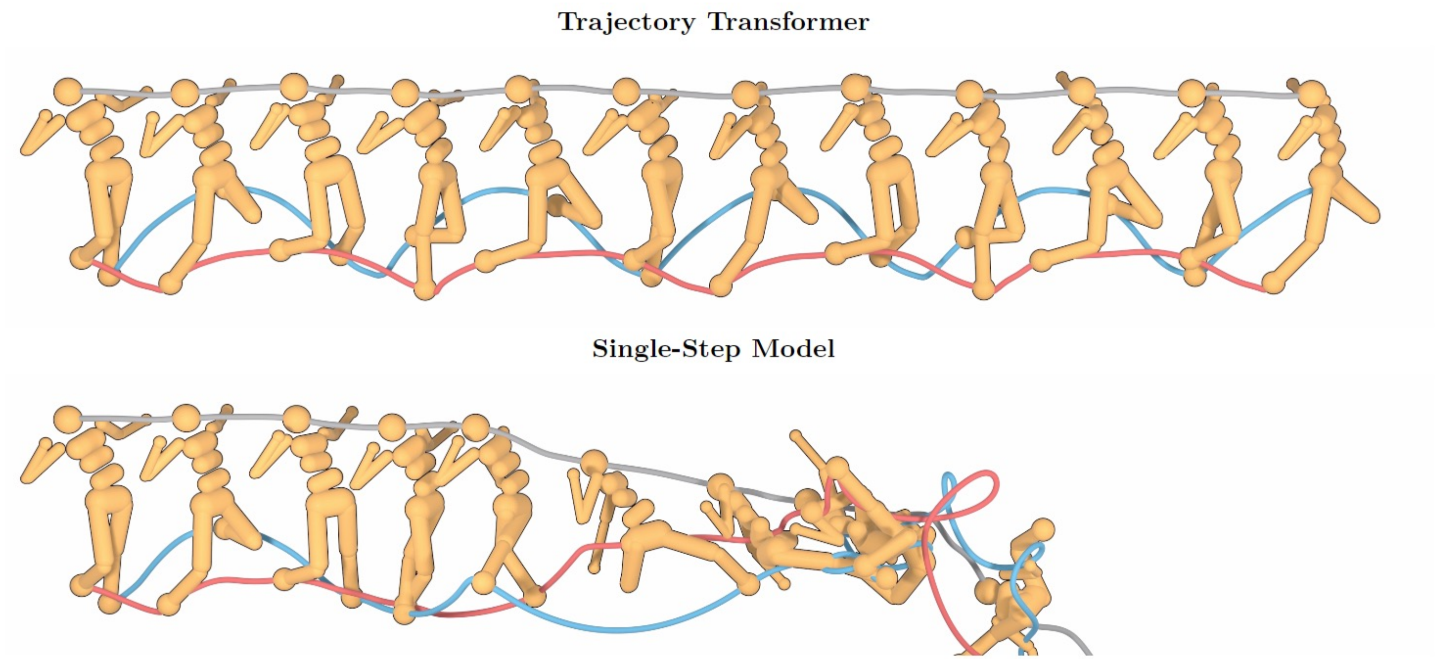


Figure from [M. Janner et al, Trajectory Transformer, NeurIPS'21]
https://trajectory-transformer.github.io/

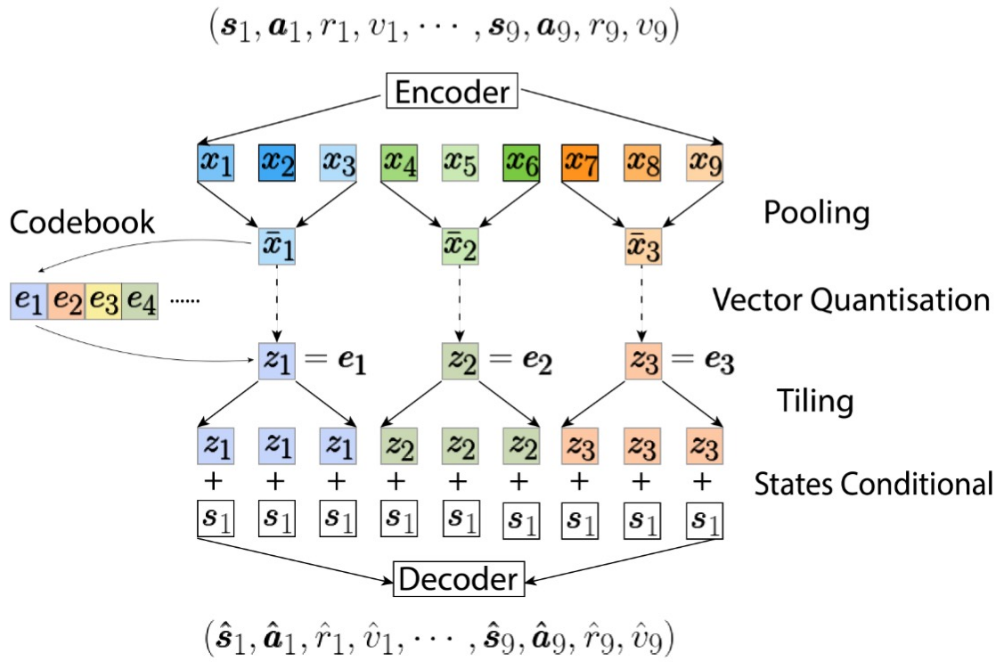
Transformer as sequential modeling for Reinforcement Learning

☺ Accurate long-term prediction
☹ Quadratic time complexity

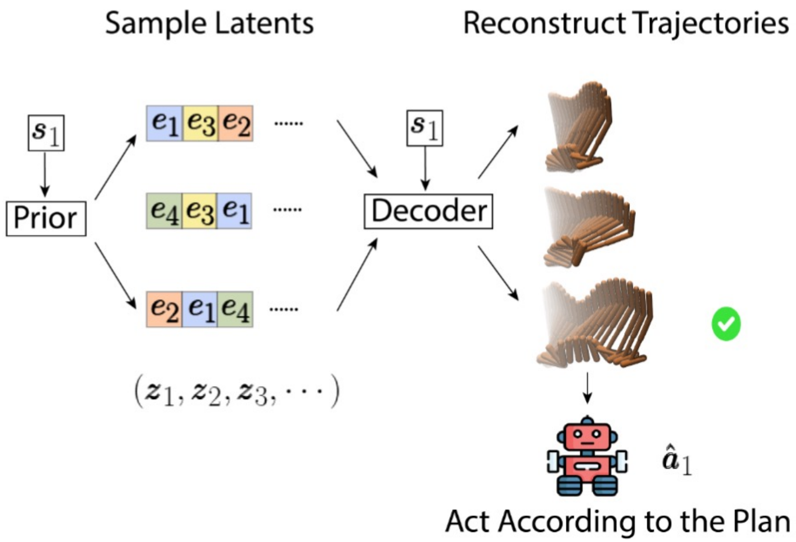
[Z. Jiang, et al, Offline Reinforcement Learning with a Scalable Trajectory Generative Model]

Trajectory Autoencoding Planner (TAP)

Training



Planning / Search

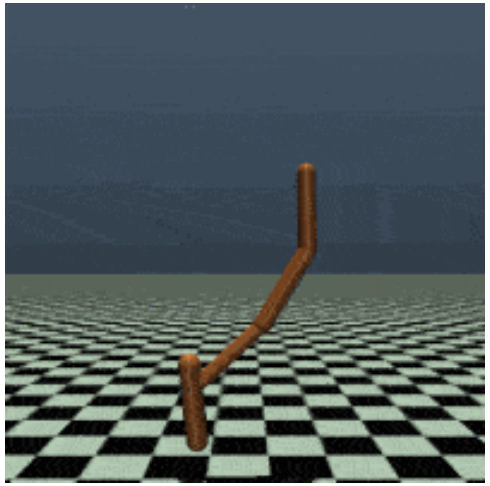


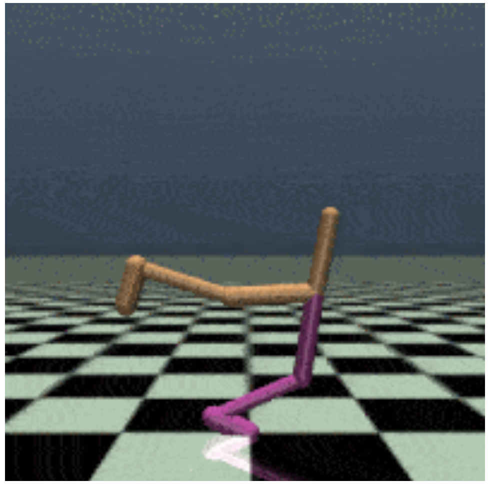
Planning Criterion = $\overbrace{\sum_i \gamma^i r_i + \gamma^T v_T}^{\text{return}} + \alpha \overbrace{\ln\left(\mathrm{clip}(p(z_1, z_2, ..., z_{T/L}|s), 0, \beta)\right)}^{\text{likelihood}}$

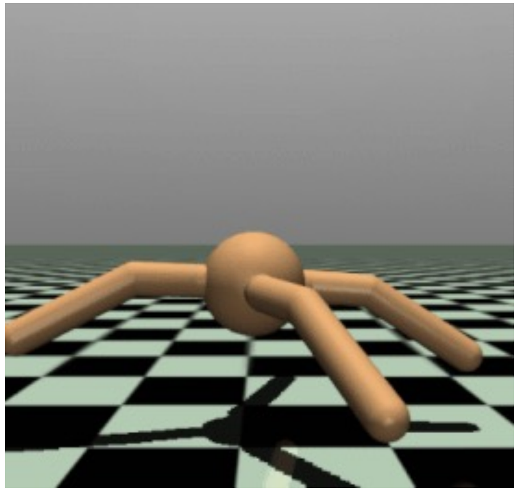
1. Use VQ-VAE latent code to encode temporal segments
2. Train transformer on top of latent code

1. Sample latent codes from prior model
2. Do beam search for the planning

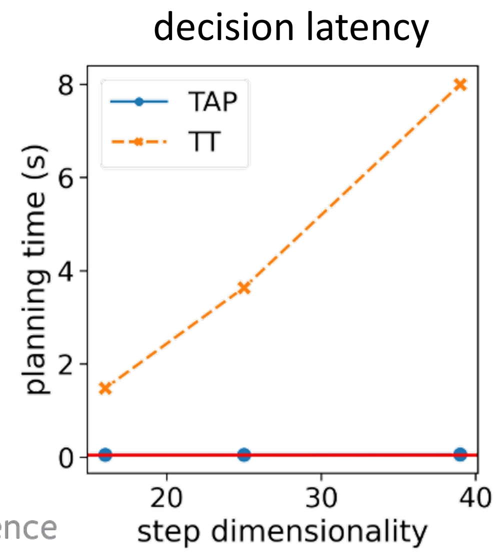
Scale to Higher Dimensionality

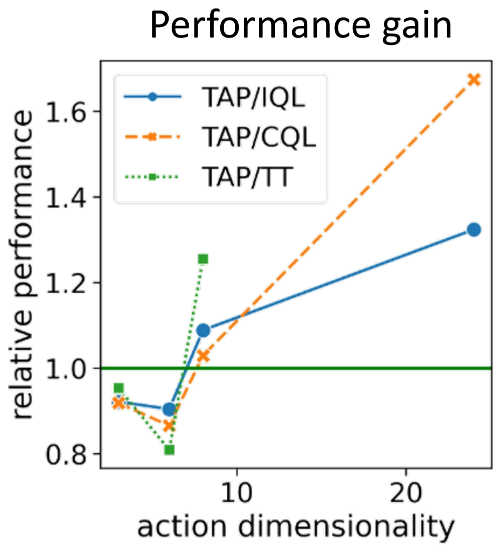






Increase Dimensionality



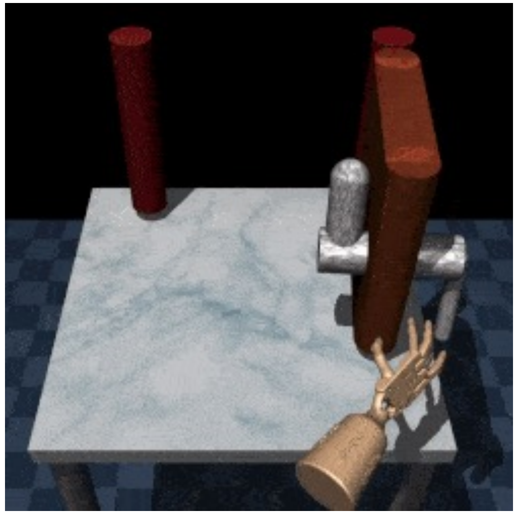


TAP scales better both in terms of decision latency and the performance in D4RL

Strong Performance in Adroit D4RL (robotic hand control)

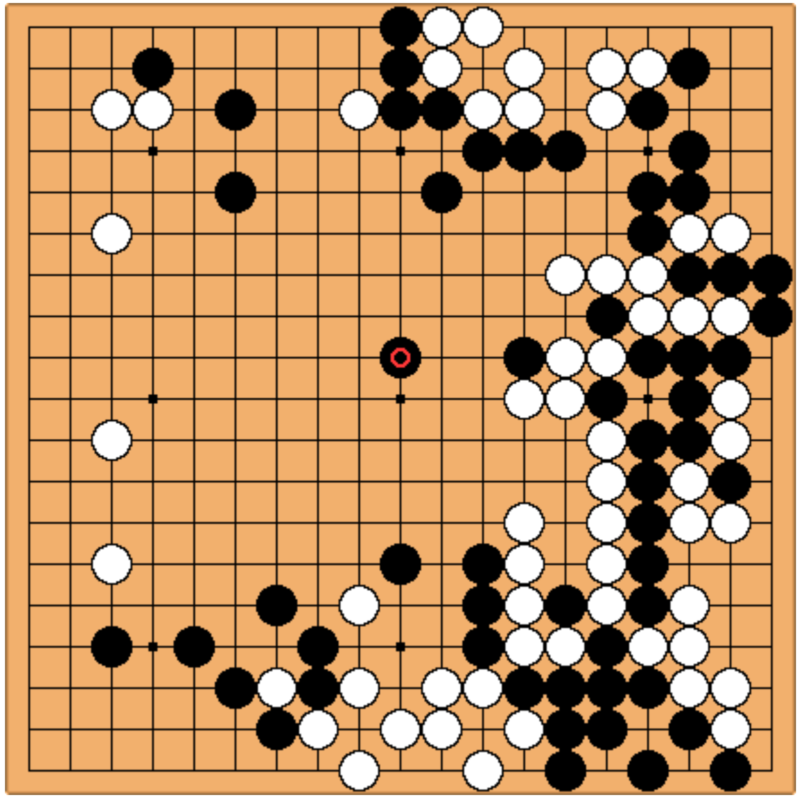
high state and action dimensionality

Dataset	Environment	BC	CQL	IQL	MOPO	Opt-MOPO	TT	TAP (Ours)
Human	Pen	34.4	37.5	71.5	6.2	19.0	36.4	**76.5** ±8.5
Human	Hammer	1.5	**4.4**	1.4	0.2	0.5	0.8	1.4 ±0.1
Human	Door	0.5	**9.9**	4.3	–	–	0.1	8.8 ±1.1
Human	Relocate	0.0	0.2	0.1	–	–	0.0	0.2 ±0.1
Cloned	Pen	56.9	39.2	37.3	6.2	23.0	11.4	**57.4** ±8.7
Cloned	Hammer	0.8	2.1	2.1	0.2	**5.2**	0.5	1.2 ±0.1
Cloned	Door	−0.1	0.4	1.6	–	–	−0.1	**11.7** ±1.5
Cloned	Relocate	−0.1	−0.1	−0.2	–	–	−0.1	−0.2 ±0.0
Expert	Pen	85.1	107.0	–	15.1	50.6	72.0	**127.4** ±7.7
Expert	Hammer	125.6	86.7	–	6.2	23.3	15.5	**127.6** ±1.7
Expert	Door	34.9	101.5	–	–	–	94.1	**104.8** ±0.8
Expert	Relocate	101.3	95.0	–	–	–	10.3	**105.8** ±2.7
Mean (without Expert)		11.7	11.7	14.8	–	–	6.1	**19.6**
Mean (all settings)		36.7	40.3	–	–	–	20.1	**51.9**

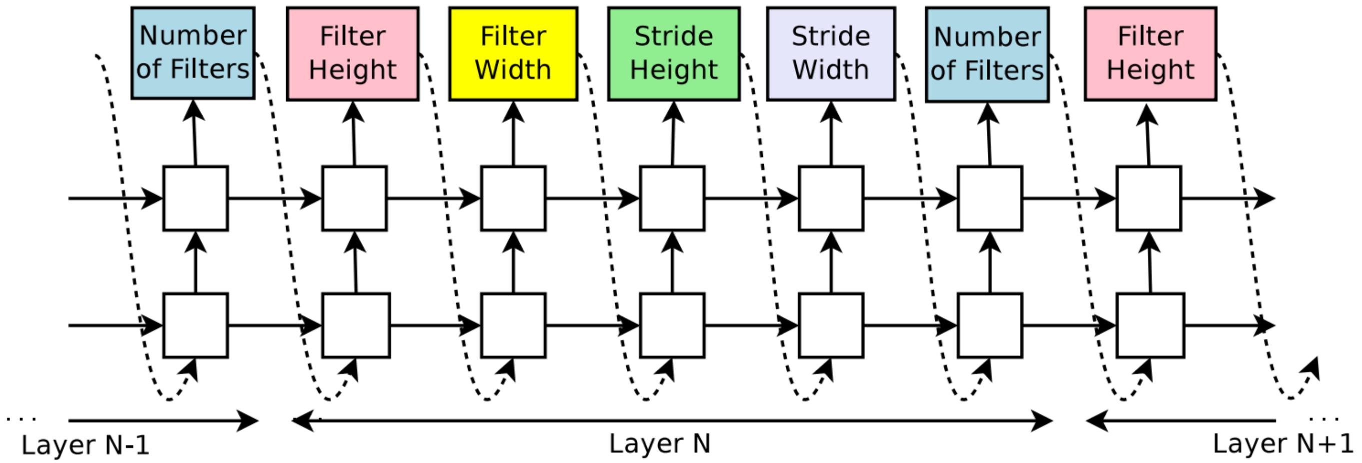


Part III: Learning Design of State/Action Space

Predefined Action Space



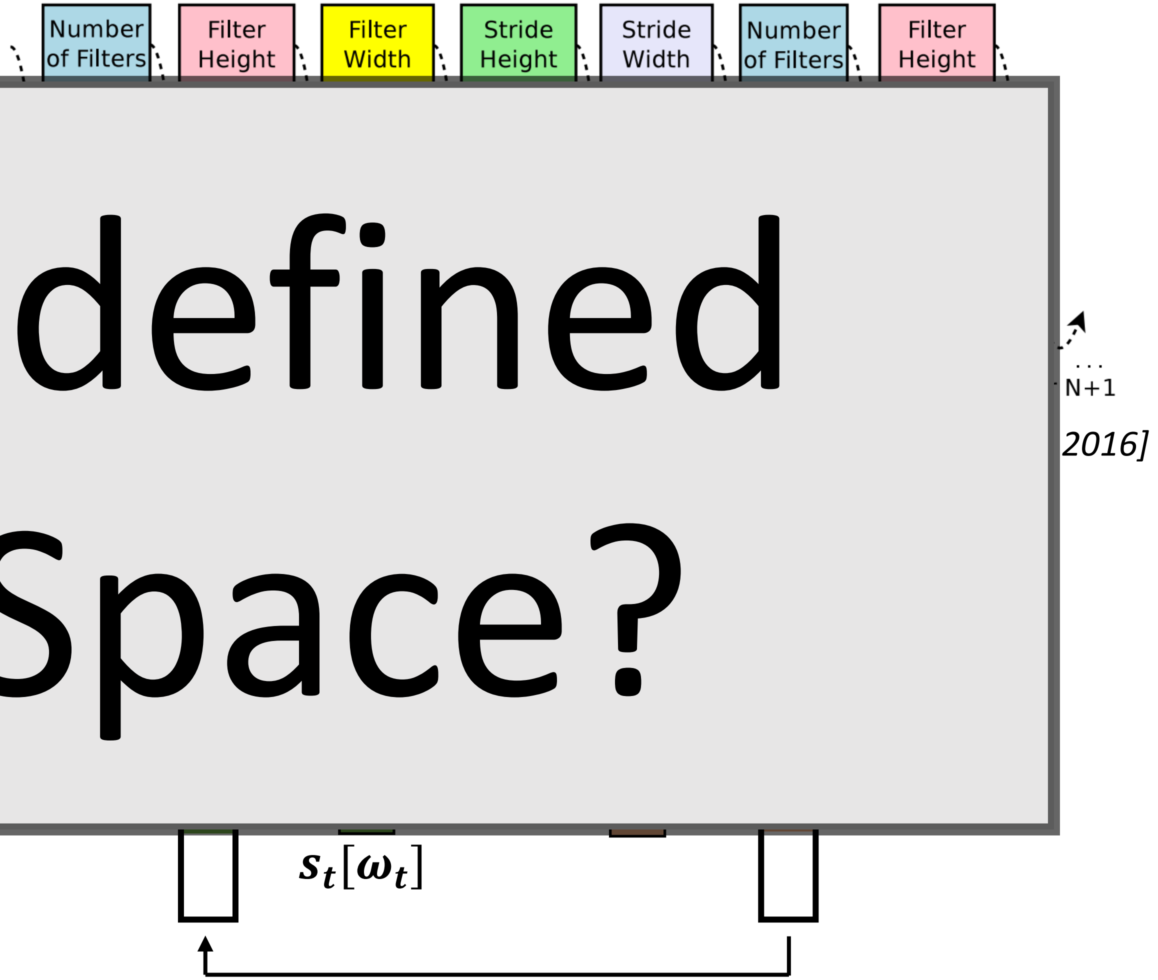
Fixed action space = R^{361}



[B. Zoph and Q. Le, Neural Architecture Search with Reinforcement Learning, 2016]

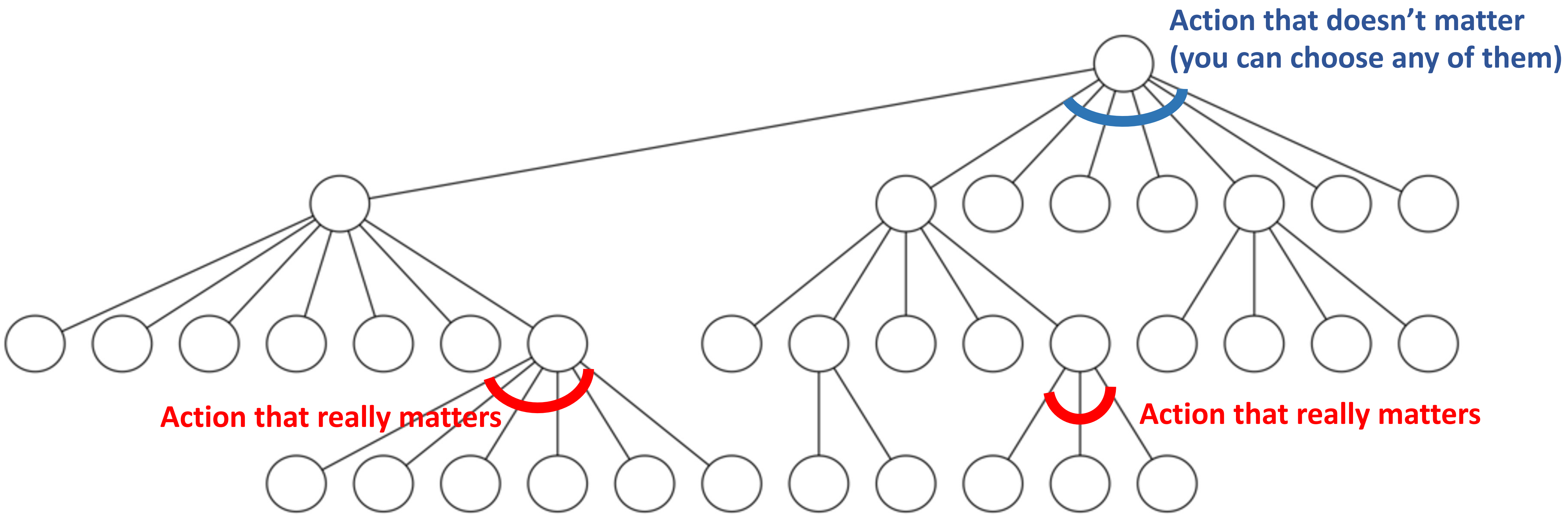
Predefined Action Space

Fixed action space = R^{361}



Why Predefined Action Space?

What is a Good Representation for MDP itself?

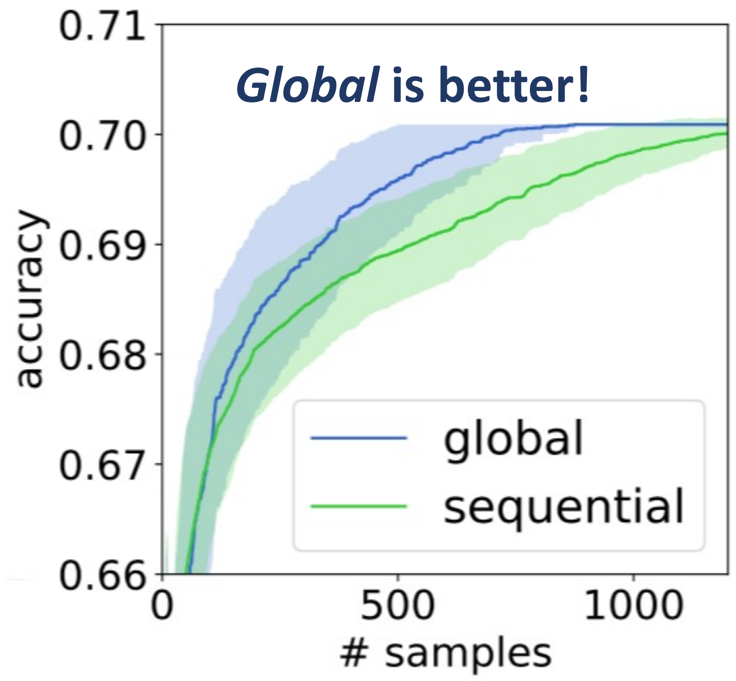


If useful actions only happen after **50** binary moves, then we will waste our efforts in this $\mathbf{2^{50}}$ possibilities.

Different Representation matters

Depth = {1, 2, 3, 4, 5}
Channels = {32, 64}
KernelSize = {3x3, 5x5}

1364 networks.

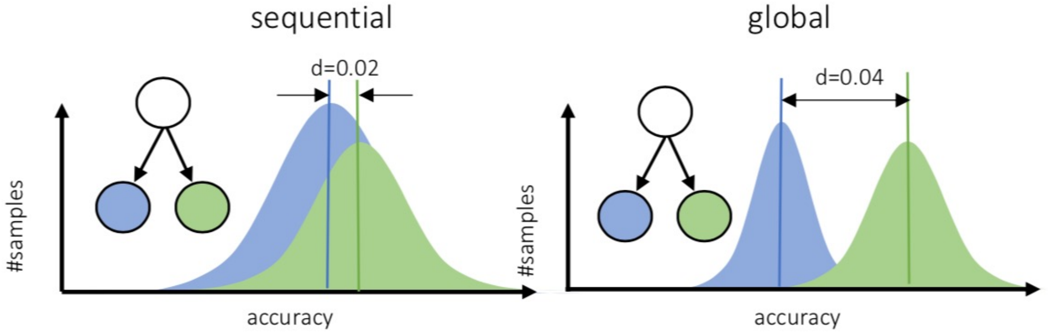


Goal: Find the network with the best accuracy using fewest trials.

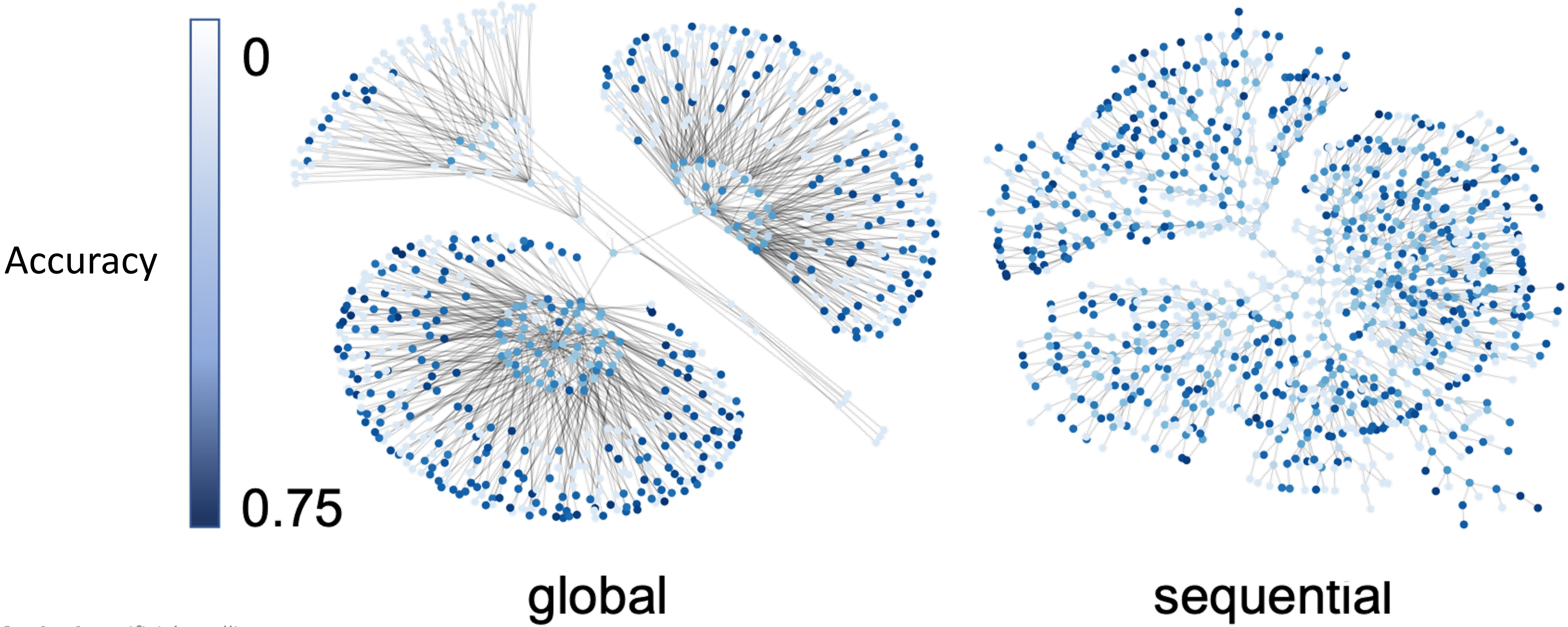
Representation of action space

Sequential = { add a layer, set K, set C }

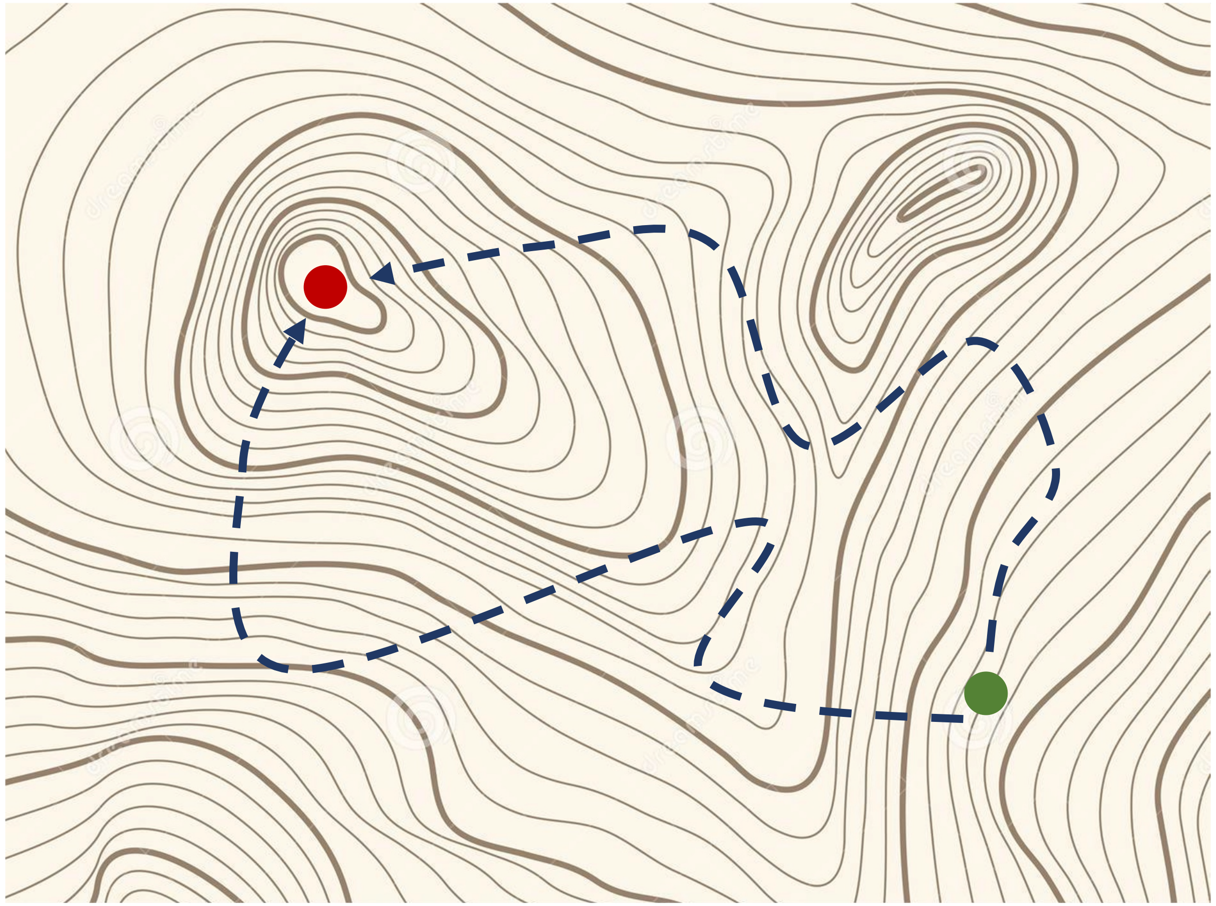
Global = { Set depth, set all K, set all C }



Different Partition → Different Value Distribution



Why Predefined Action Space?



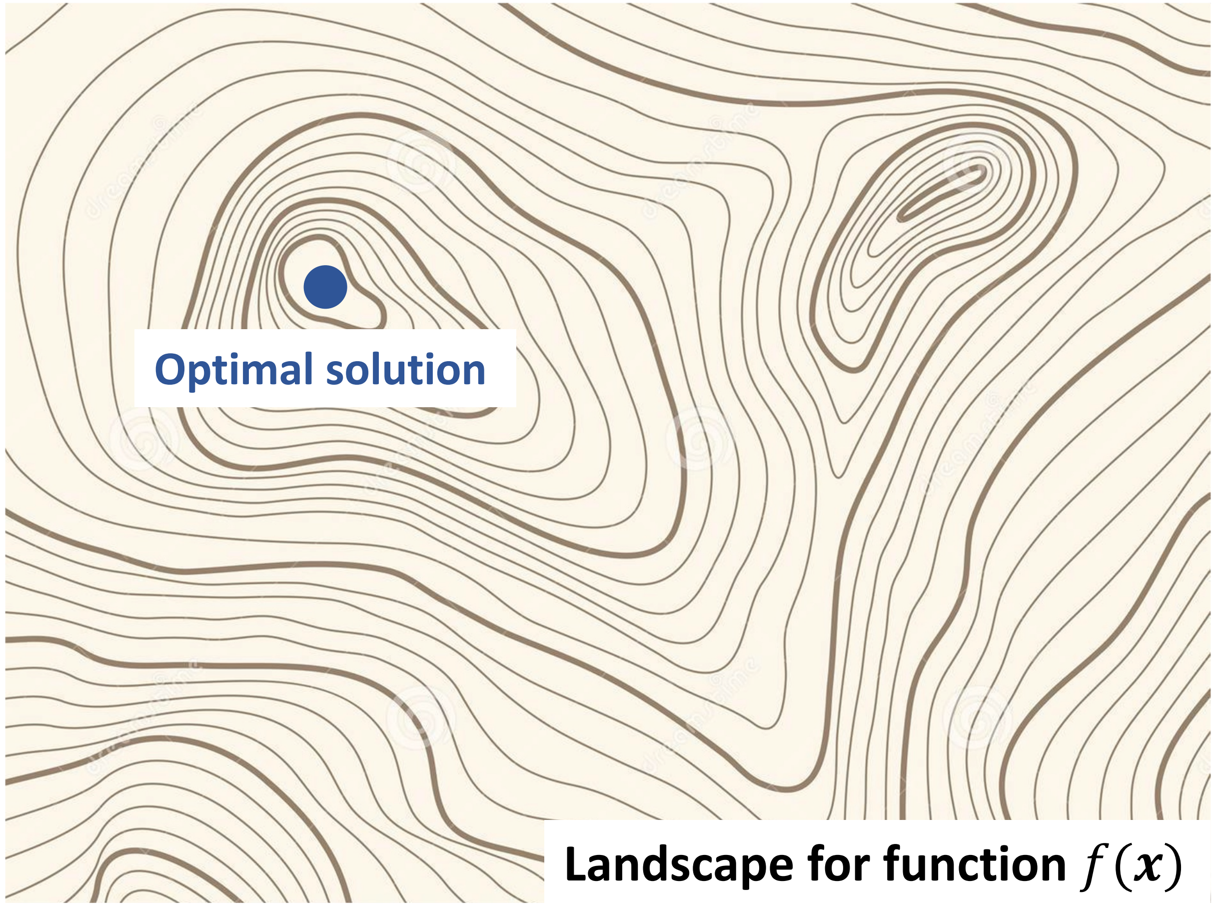
<u>Optimization problems</u>

$$\min_{\boldsymbol{x}} f(\boldsymbol{x})$$

We only care the final solution

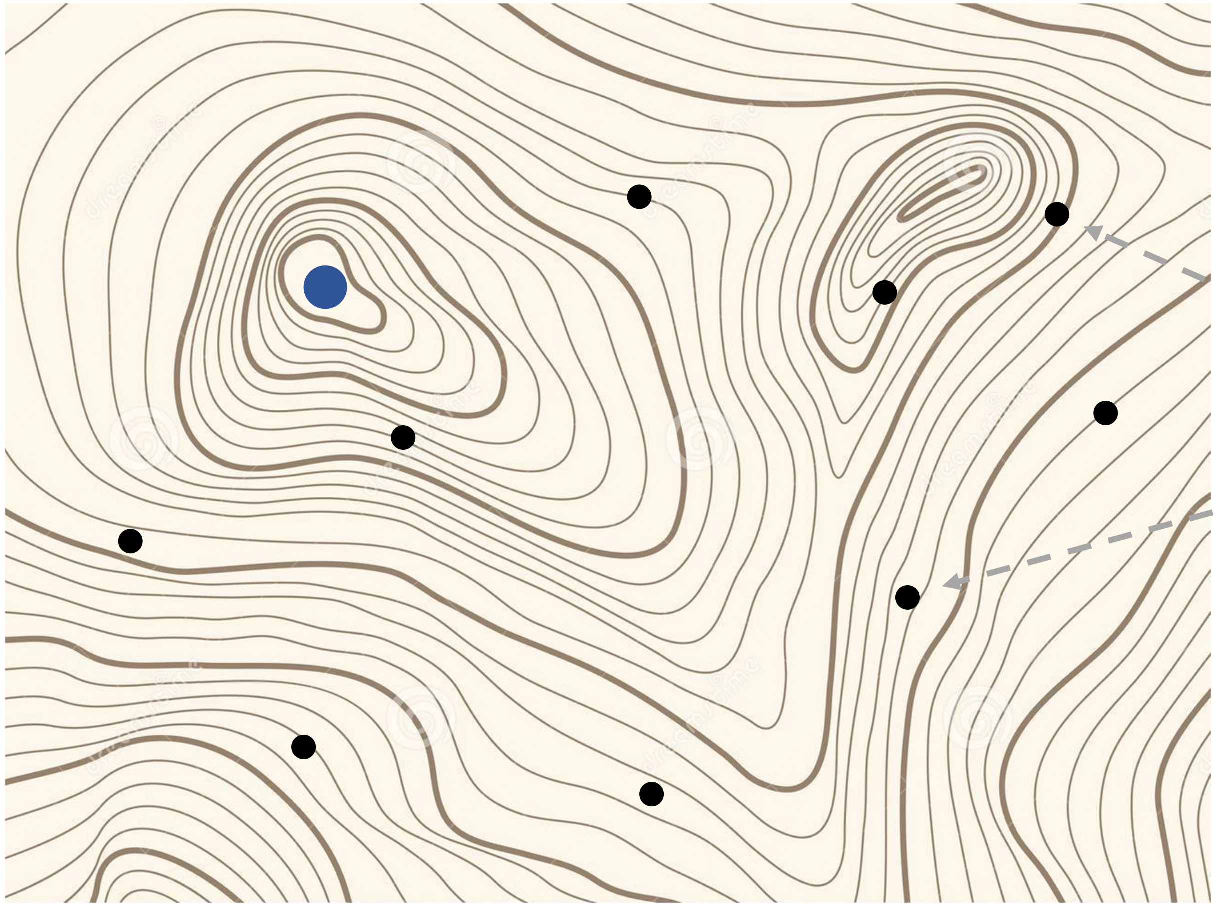
We don't care how we get it.

Learning to Partition --- How it works?



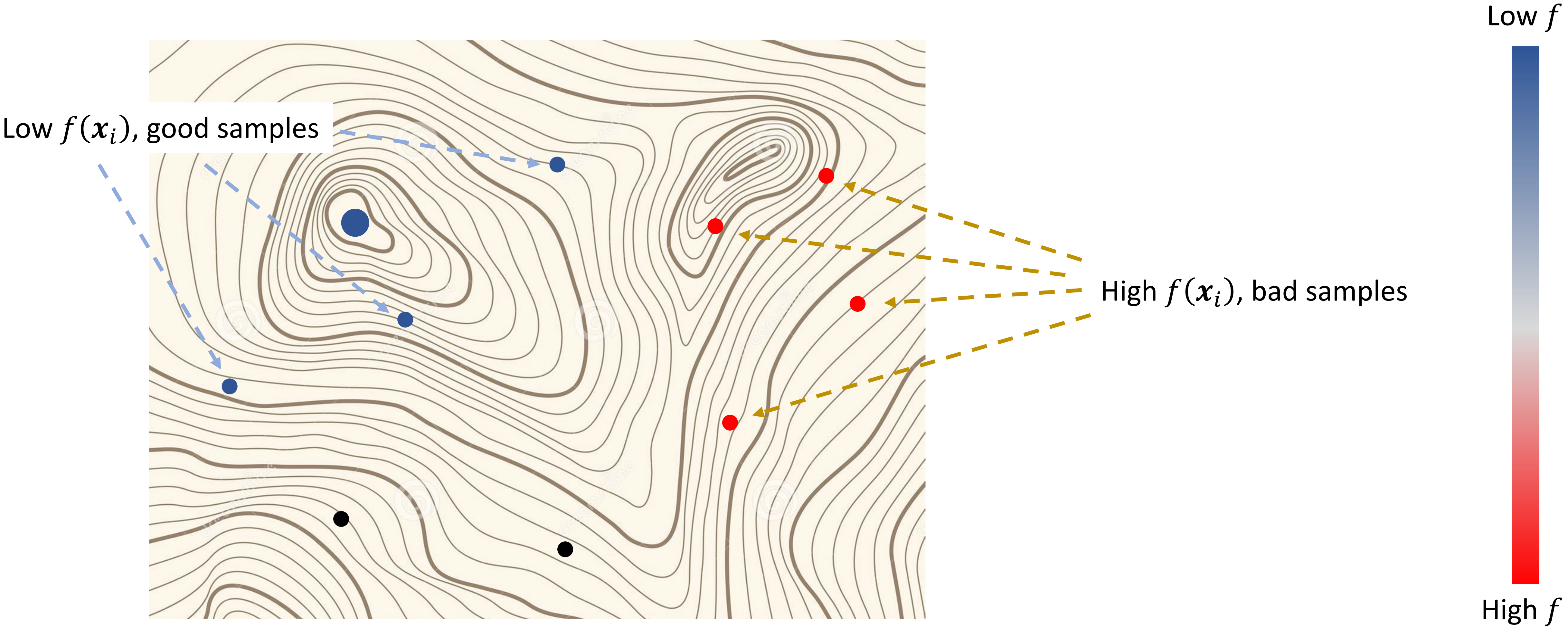
$$\min_{\boldsymbol{x}} f(\boldsymbol{x})$$

Learning to Partition

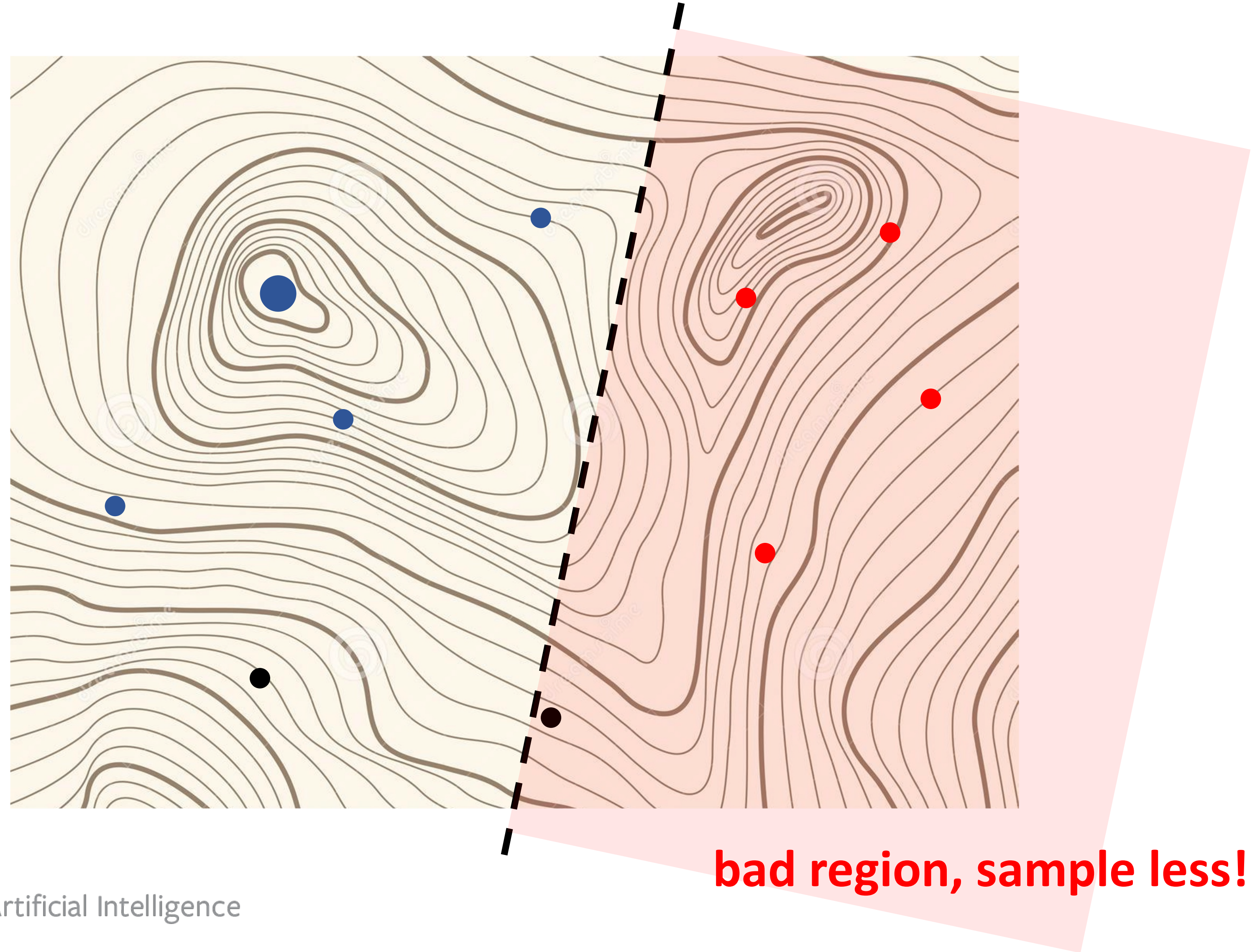


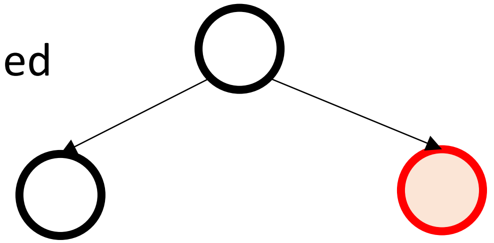
Sample of the function $f(\boldsymbol{x}_i)$

Learning to Partition

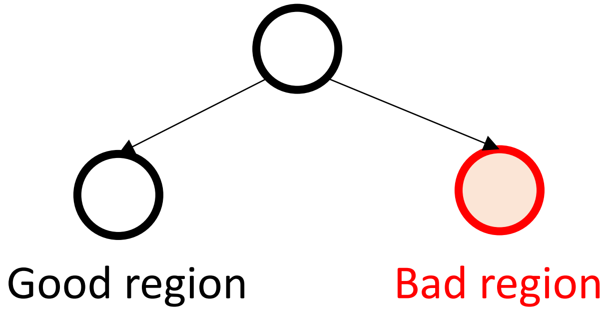


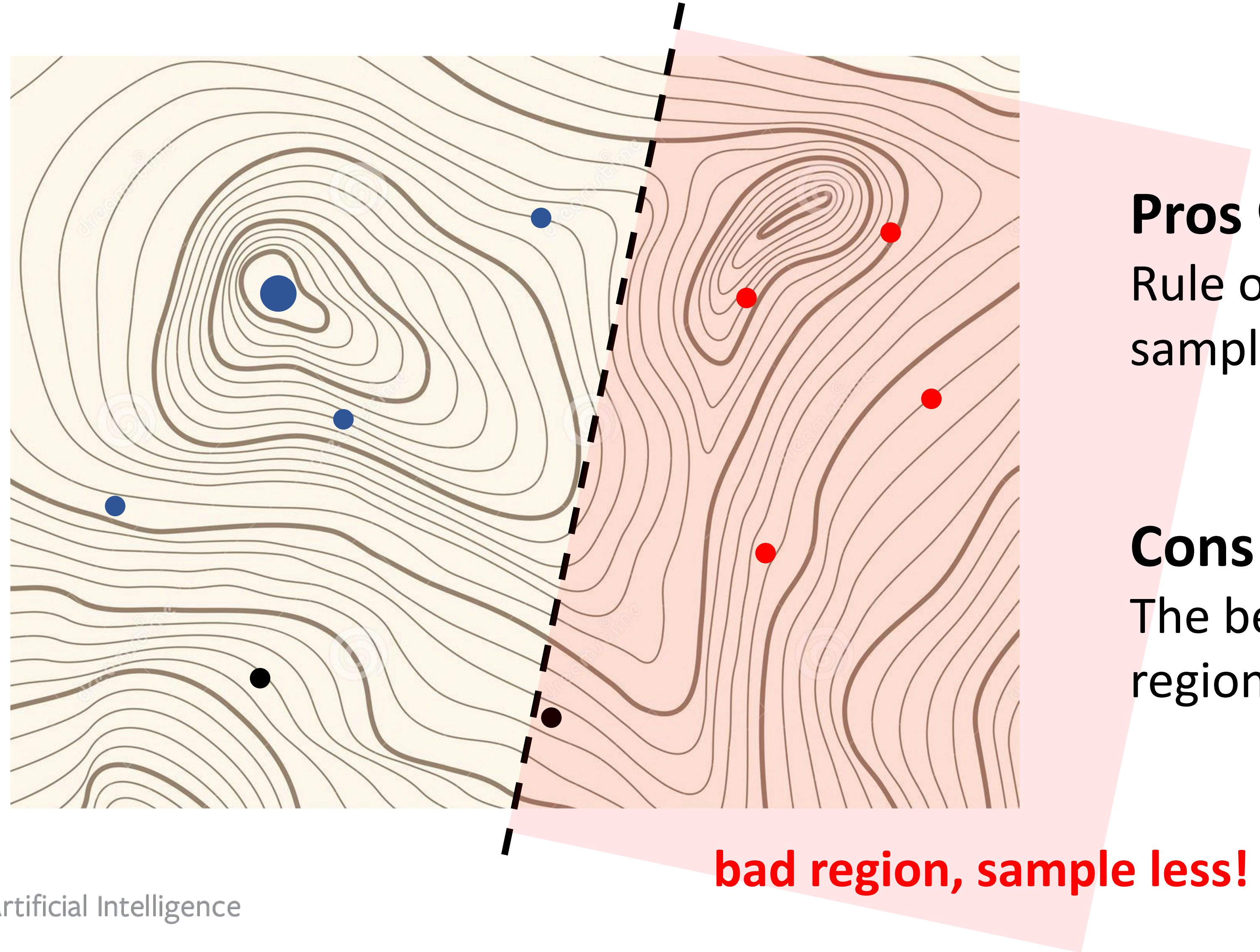
Learning to Partition





Learning to Partition





Pros ☺:

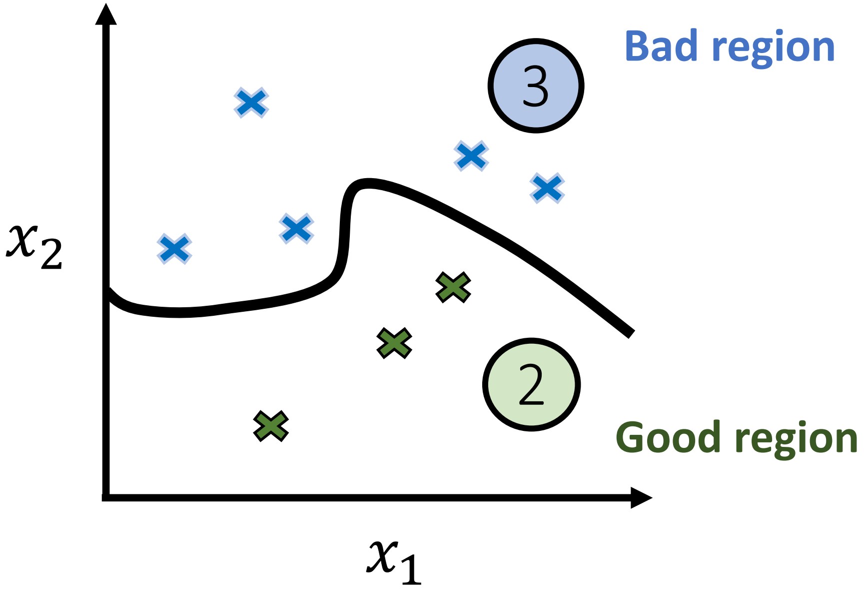
Rule out a lot of regions so that the sampling can be more efficient.

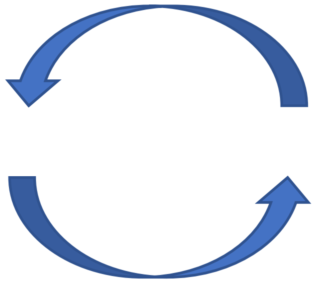
Cons ☹:

The best solution can be in “bad” regions.

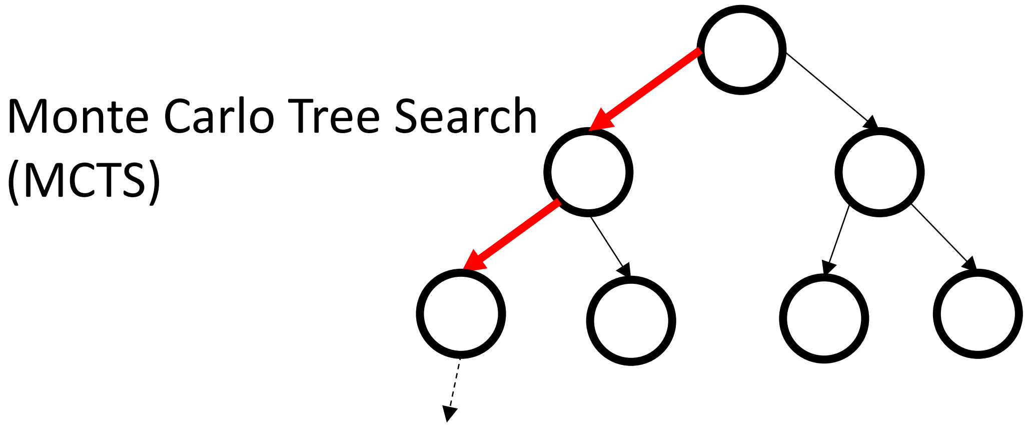
Latent Space Monte Carlo Tree Search (LaMCTS)

(a) Learn the action space.





(b) Search using learned action space until a fixed #rollouts are used.



Getting the true quality $f(x)$ for the solution x

[L. Wang, R. Fonseca, Y. Tian, **Learning Search Space Partition for Black-box Optimization using Monte Carlo Tree Search,** *NeurIPS 2020]*
[L. Wang, S. Xie, T. Li, R. Fonseca, Y. Tian, **Sample-Efficient Neural Architecture Search by Learning Action Space,** *TPAMI 2021]*

Code is public now!

LA-MCTS

https://github.com/facebookresearch/LaMCTS

Both 3rd and 8th teams in NeurIPS 2020 Black-box optimization competition use our method!

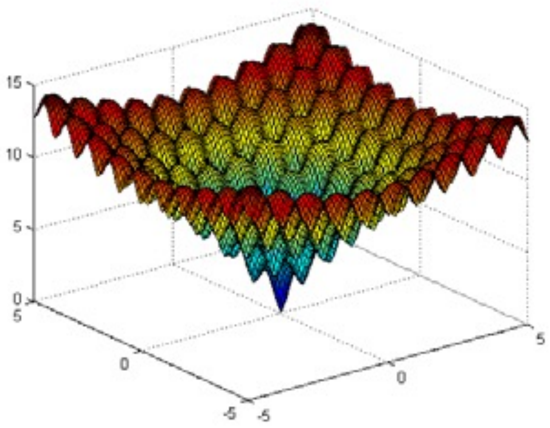
Open Domain

ImageNet
(mobile setting
Flop < 600M)

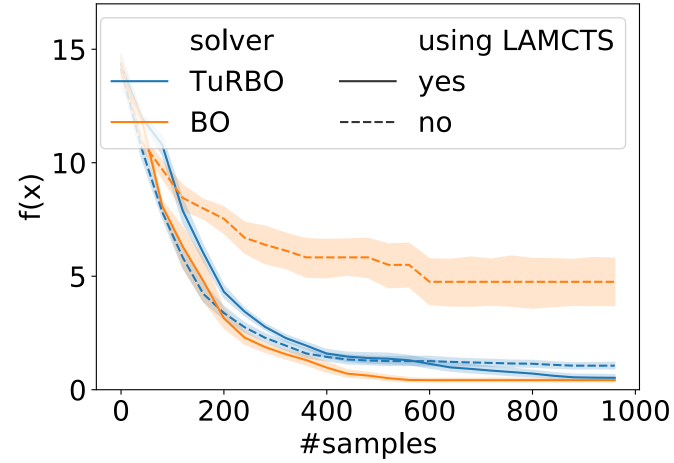
Model	FLOPs	Params	top1 / top5 err
NASNet-A (Zoph et al. (2018))	564M	5.3 M	26.0 / 8.4
NASNet-B (Zoph et al. (2018))	488M	5.3 M	27.2 / 8.7
NASNet-C (Zoph et al. (2018))	558M	4.9 M	27.5 / 9.0
AmoebaNet-A (Real et al. (2018))	555M	5.1 M	25.5 / 8.0
AmoebaNet-B (Real et al. (2018))	555M	5.3 M	26.0 / 8.5
AmoebaNet-C (Real et al. (2018))	570M	6.4 M	**24.3 / 7.6**
PNASNet-5 (Liu et al. (2018a))	588M	5.1 M	25.8 / 8.1
DARTS (Liu et al. (2018b))	574M	4.7 M	26.7 / 8.7
FBNet-C (Wu et al. (2018))	375M	5.5 M	25.1 / -
RandWire-WS (Xie et al. (2019))	583M	5.6 M	25.3 / 7.8
BayesNAS (Zhou et al. (2019))	-	3.9 M	26.5 / 8.9
LaNet	570M	5.1 M	**25.0 / 7.7**

La-MCTS as a meta method

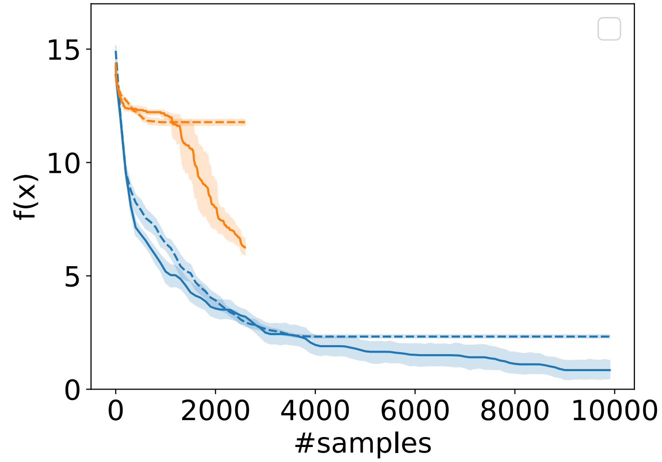
$$x^* = \arg\min_{x \in \Omega} f(x)$$



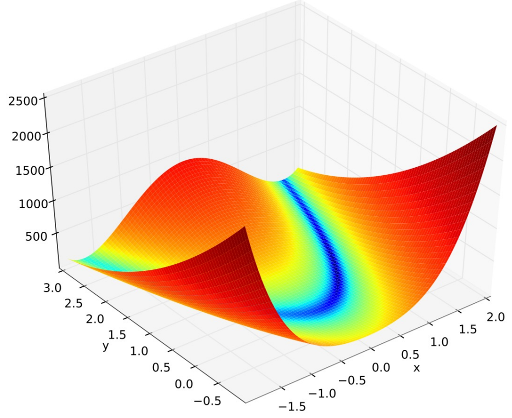
Ackley-2d



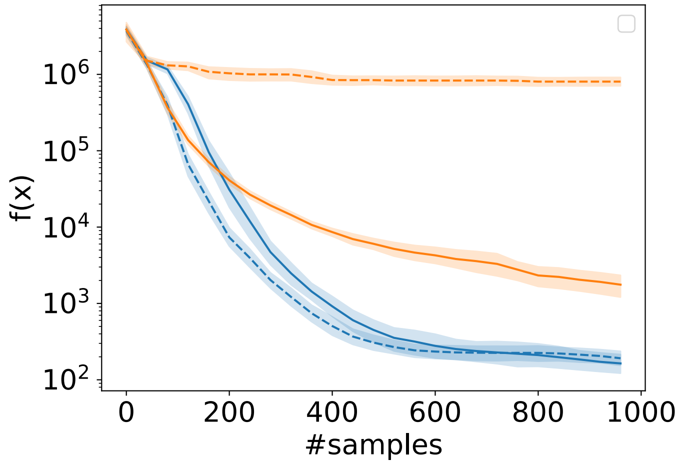
Ackley-20d



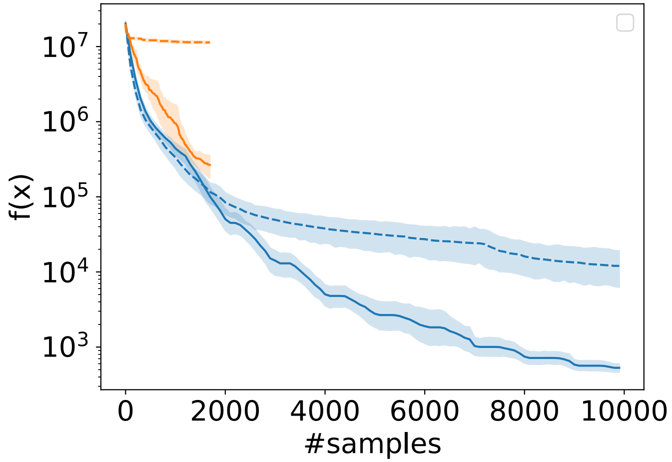
Ackley-100d



Rosenbrock-2d



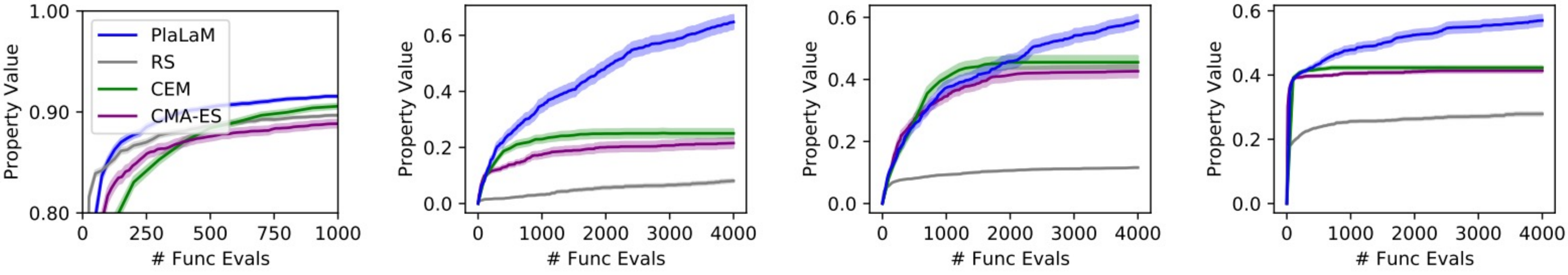
Rosenbrock-20d



Rosenbrock-100d

Molecule Design

Latent representation learned from unlabeled molecule dataset (1.8M molecules)



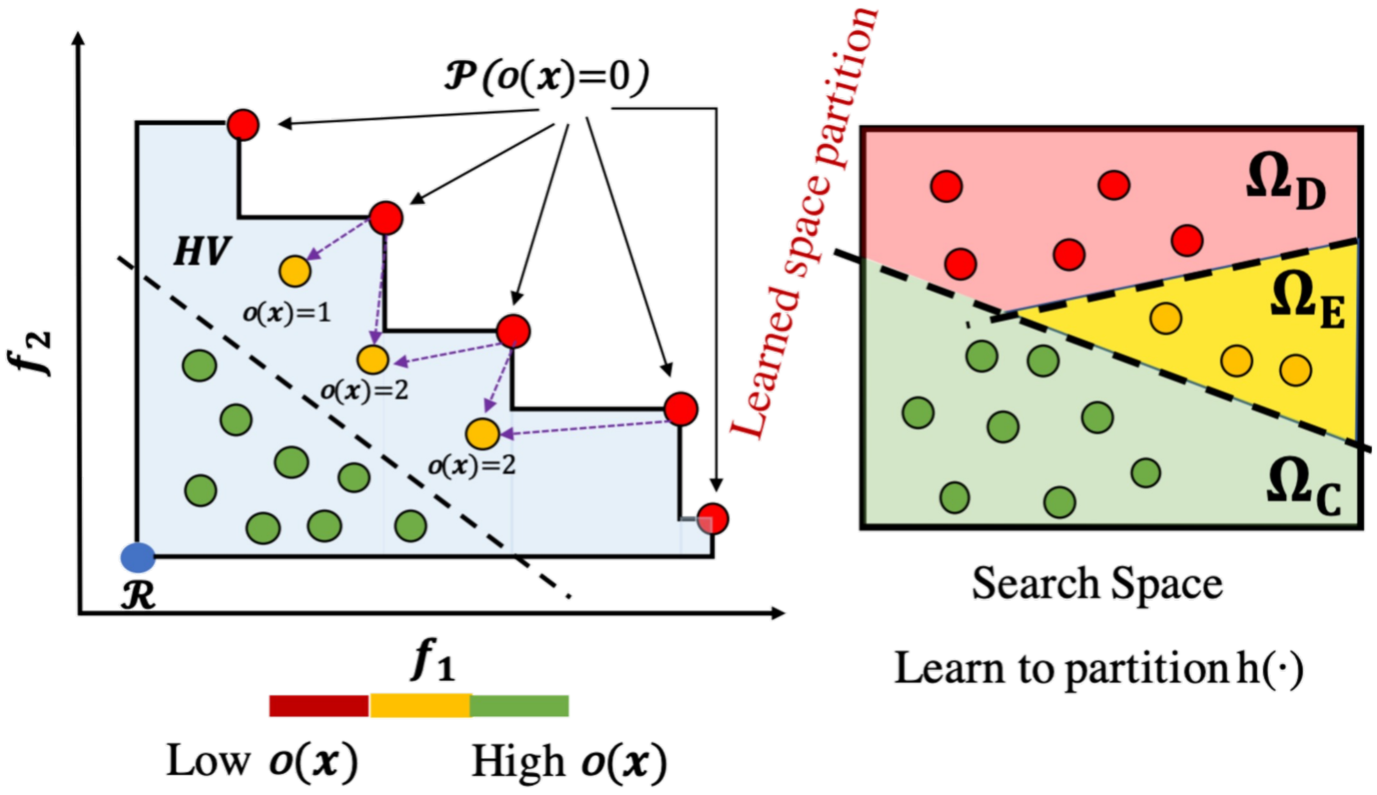
(a) QED (b) DRD2 (c) HIV (d) SARS

QED: a synthetic measure of drug-likeness (easy property)
DRD2: a measure of binding affinity to a human dopamine receptor
HIV: the potential inhibition probability for HIV
SARS: the potential inhibition probability for COVID-19

Code is available

*[K. Yang, T. Zhang, … **Y. Tian**,* **Learning Space Partitions for Path Planning**, *NeurIPS 2021]*

Multi-Objective Optimization (LaMOO)

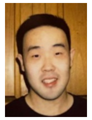


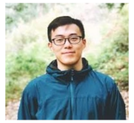
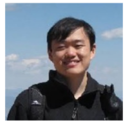
Compute Dominant Number $o(\boldsymbol{x})$

$$o_{t,j}(\mathbf{x}) := \sum_{\mathbf{x}_i \in D_{t,j}} \mathbb{I}[\mathbf{x} \prec_{\mathbf{f}} \mathbf{x}_i,\ \mathbf{x} \neq \mathbf{x}_i]$$

$o(\boldsymbol{x})$ can be computed in $O(n \log n)$

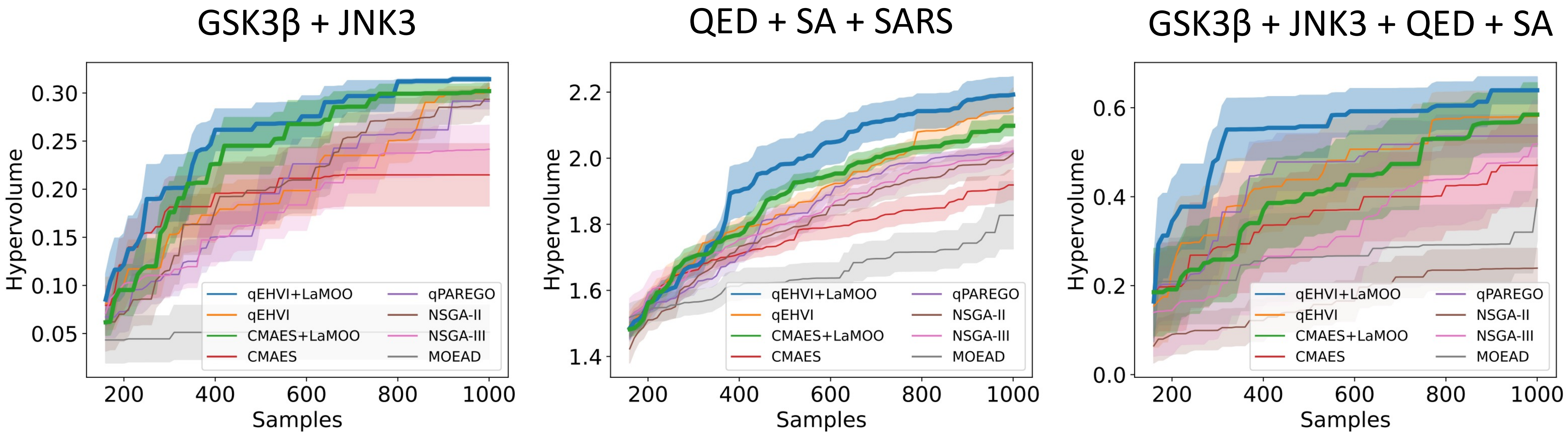
Learn a space partition to separate good / bad regions



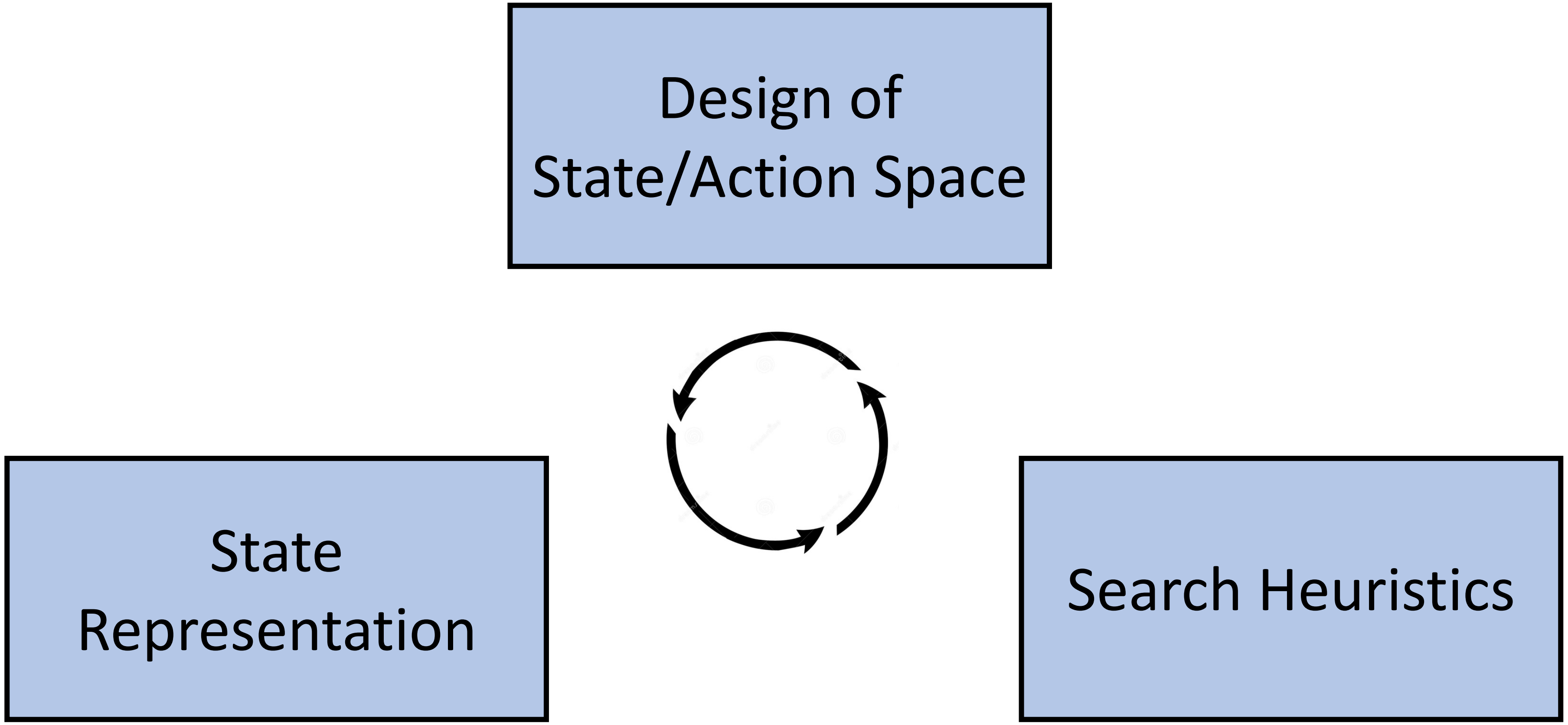
[Y. Zhao, …, Y. Tian, **Multi-objective Optimization by Learning Space Partitions**, ICLR'22]

Molecule Design (32 dimensional input)



GSK3β, JNK3: biological targets
SA: a standard measure of synthetic accessibility
QED: a synthetic measure of drug-likeness
SARS: the potential inhibition probability for COVID-19

Future Work


Design of State/Action Space
State Representation
Search Heuristics

Thanks!