Introduction to Applied Reinforcement Learning

Shah Rukh Qasim

Department of Mathematical Modeling and Machine Learning
&&
Physik Institut
University of Zurich

shahrukh.qasim@physik.uzh.ch

25.11.2025

Today's Agenda

- We will skip the tutorial today
    - Although I will give a link to an example towards the end to play
    - Interest of time: there is a lot to learn to get started with RL
- Reinforcement Learning (RL) is hard and there is a lot to learn
    - We will cover the core concepts
- The order of the contents of this lecture is non traditional in how RL is taught: Fundamentals are discussed later
    - Expectations is that it will make it easier to understand quickly
- References in Sutton and Barto 2nd Edition
    - Freely available at: http://incompleteideas.net/book/the-book-2nd.html

https://tinyurl.com/AppliedRLNotebook

https://colab.research.google.com/drive/1suKXuHlf6M0N2yh1K_OAq-qJwpMe4ytM?usp=sharing

Why Reinforcement Learning?

- Playing chess
  - Input: state of the board
  - Output: the next move
  - Supervised learning:
    - Collect a whole bunch of samples with the next best move and then train the NN on it
  - Reinforcement learning:
    - Let the agent make decisions
    - … to maximize the reward function
    - It’s the job of the agent to collect data and take the best decision at every point

Reinforcement Learning

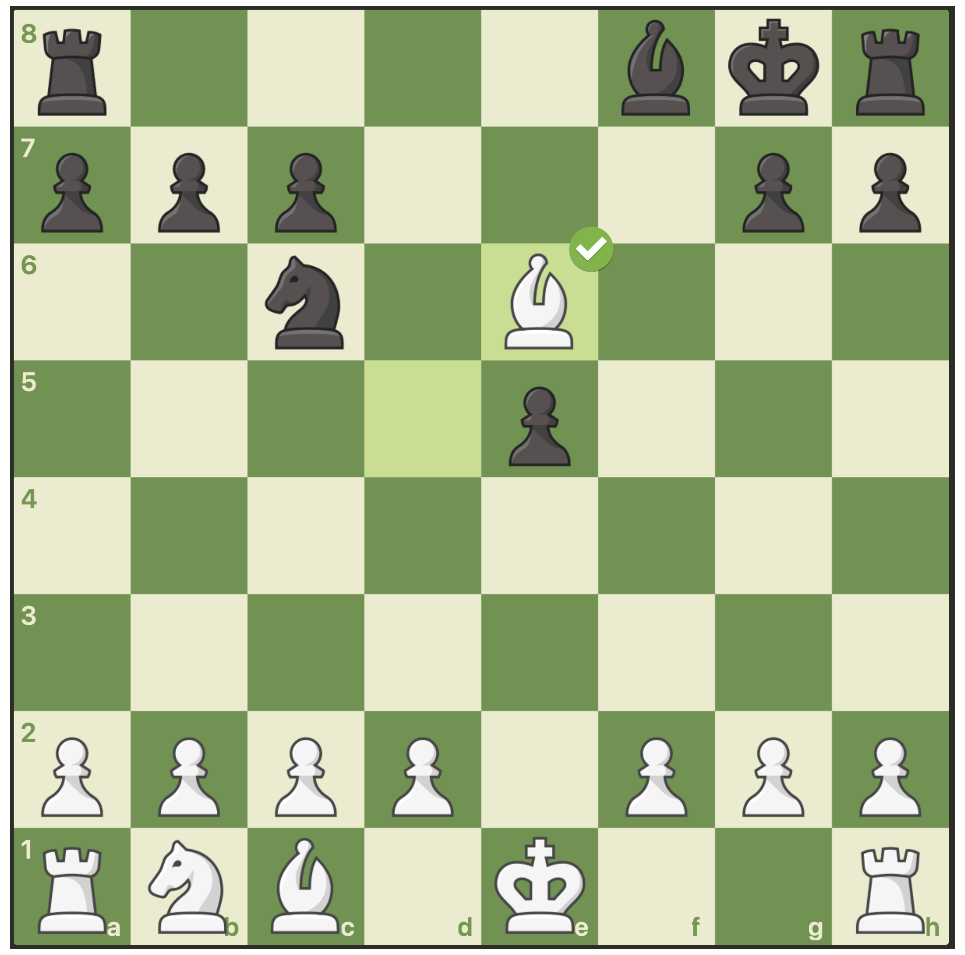
$r_{t+1} = 0$ $r_{t+2} = r_T = 1$

s_t

$\xrightarrow{a_t}$

s_{t+1}

$\xrightarrow{a_{t+1}}$



$s_{t+2} = s_T$

- RL agent gets a state s_t
- Takes an action a_t
- Gets a reward r_{t+1}
- The state gets updated s_{t+1}

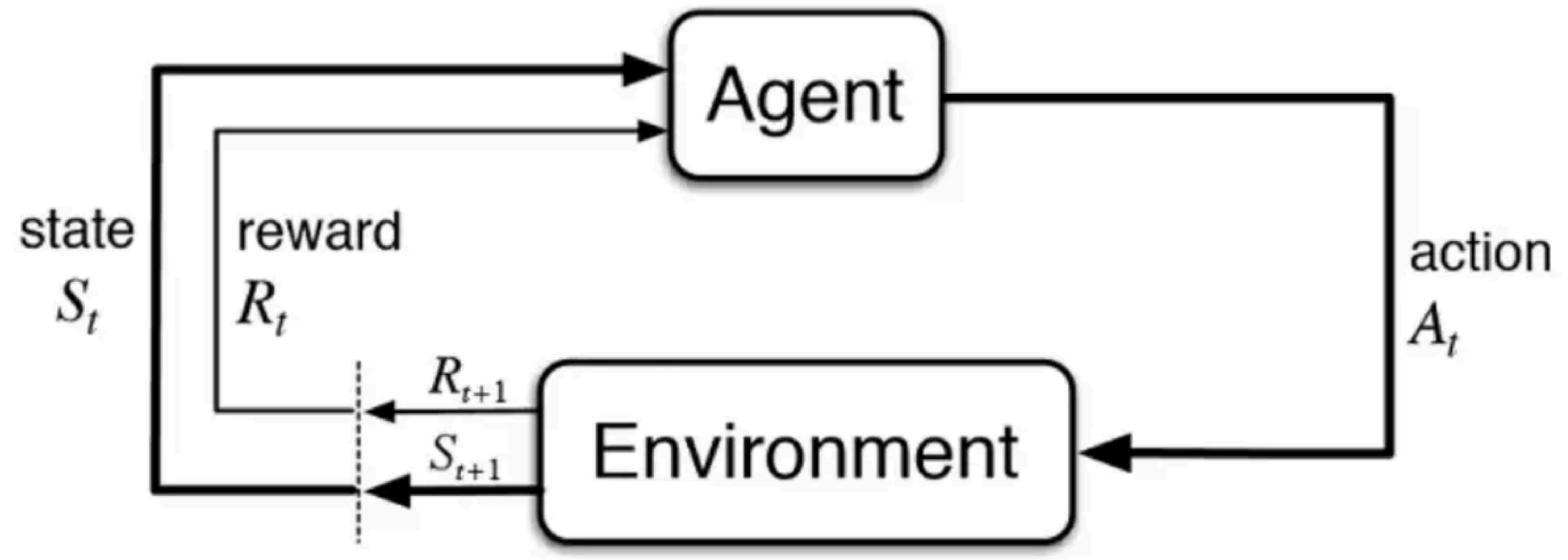
Remember: At every time step, an agent makes a decision — ONLY based on the current state!

If the history is important, append it to the state!

Reinforcement Learning

- Capital letters represent random variables: not exact, we blur the boundaries
- An episode is then:
  - $(s_0, a_0, r_1, s_1, a_1, r_2, s_2, \ldots, a_{T-1}, r_T, s_T)$
- Maximize:
  - $G_t = R_{t+1} + R_{t+2} + \cdots + R_T$
    - (incomplete)
  - Cumulative reward
  - Also called return
  - G_0 is episode return

Sutton, R. S. and Barto, A. G. Introduction to Reinforcement Learning



Agent's Goal: Maximize Return G_t at every step, not R_{t+1}

Discounted Returns

- We need to add a discount factor
  - $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T$
  - $\gamma \leq 1$, immediate rewards matter more
    - You can invest money now.
    - There is inflation.
    - There is uncertainty about the future.

- Or infinitely long episodes:
  - $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$

Policy

- What is a policy?
  - A rule or strategy that tells an agent how to act in each situation.
    - State: A customer asks to return an item (with or without receipt)
    - Policy: "If the customer has a receipt, accept the return; otherwise, decline”
- In RL:
  - A policy can be **deterministic**
    - $a_t = \pi(s_t)$
    - $\pi : \mathcal{S} \to \mathcal{A}$
  - A policy can be **stochastic**
    - $\pi(a \mid s) = \Pr(A_t = a \mid S_t = s)$
    - $\pi : \mathcal{S} \times \mathcal{A} \to [0,1]$

Why stochastic?

- A deterministic policy can cause a vacuum robot to loop forever by repeating the same action in the same state, while a stochastic policy can break the loop.
- When several actions are equally good, a stochastic policy represents this better than forcing one arbitrary choice.
- Stochasticity enables learning (discussed later)

Policy Gradient: REINFORCE

- The NN is
  - π_θ: It products output logits — same as a normal classification task
  - It can also be a continuous output
- Sample full episodes
- Compute returns
- Change policies to make actions more likely which led to higher returns
  - Proof: 13.3 in Sutton and Barto
- Wait till the end of episode: **Monte Carlo** learning

Algorithm 1 REINFORCE (Monte Carlo Policy Gradient) with Sampled Trajectories

Require: learning rate α, discount factor γ, initial parameters θ

```
for episodes n = 1, 2, ... do
    Generate a trajectory (s_0, a_0, r_1, s_1, ..., s_T) using policy π_θ(a_t | s_t):
    for t = 0 to T − 1 do
        Sample action a_t ∼ π_θ(· | s_t)
        Environment produces reward r_{t+1} and next state s_{t+1}
    end for
    Compute returns G_t for t = T − 1, ..., 0:
    for t = T − 1 down to 0 do
        G_t ← r_{t+1} + γG_{t+1}                    (with G_T ← 0)
    end for
    Compute policy gradient estimate:
        g ← Σ_{t=0}^{T−1} G_t ∇_θ log π_θ(a_t | s_t)
    Update parameters:
        θ ← θ + αg
end for
```

Compute policy gradient estimate:

$$g \leftarrow \sum_{t=0}^{T-1} G_t \nabla_\theta \log \pi_\theta(a_t \mid s_t)$$

Update parameters:

$$\theta \leftarrow \theta + \alpha g$$

Value function

- An agent can predict how promising each situation (state) is in the long run.
  - Done through state-value function:
    - $V^\pi(s) = \mathbb{E}_\pi \left[G_t \mid S_t = s \right]$
      - The state-value function tells us how good it is to be in a given state when the agent follows a particular policy π
      - It is the expected return starting from state s and acting according to π thereafter
  - Or Q function
    - $Q^\pi(s, a) = \mathbb{E}_\pi \left[G_t \;\middle|\; S_t = s, A_t = a \right]$

REINFORCE with baseline

- Why does it help?
  - Without a baseline: grading students on raw scores
  - Baseline: grading relative to class average
- Reduces the variance
- This is an **Actor–Critic** algorithm
  - the policy network acts by selecting actions, while the value network critiques those actions by estimating their expected returns and providing the advantage signal for learning
- Unfortunately, REINFORCE is **not** the foundation of modern RL algorithms

Algorithm 2 REINFORCE with Learned Baseline (Monte Carlo Actor–Critic)

Require: learning rate α for policy, learning rate β for value network, discount factor γ

Require: initial policy parameters θ, initial value network parameters ϕ

for episodes $n = 1, 2, \dots$ **do**
  Generate a trajectory $(s_0, a_0, r_1, s_1, \dots, s_T)$ using the policy network $\pi_\theta(a_t \mid s_t)$ *(and record baseline estimates* $V_\phi(s_t)$ *along the way)*:
  for $t = 0$ to $T - 1$ **do**
    Sample action $a_t \sim \pi_\theta(\cdot \mid s_t)$
    Evaluate baseline $V_\phi(s_t)$
    Environment produces reward r_{t+1} and next state s_{t+1}
  end for
  Compute returns G_t for $t = T-1, \dots, 0$:
  for $t = T - 1$ down to 0 **do**
    $G_t \leftarrow r_{t+1} + \gamma G_{t+1}$ (with $G_T \leftarrow 0$)
  end for
  Compute advantages:

$$A_t \leftarrow G_t - V_\phi(s_t)$$

  Update the value network:

$$\phi \leftarrow \phi - \beta \sum_{t=0}^{T-1} \nabla_\phi \left(G_t - V_\phi(s_t)\right)^2$$

  Update the policy network:

$$\theta \leftarrow \theta + \alpha \sum_{t=0}^{T-1} A_t \nabla_\theta \log \pi_\theta(a_t \mid s_t)$$

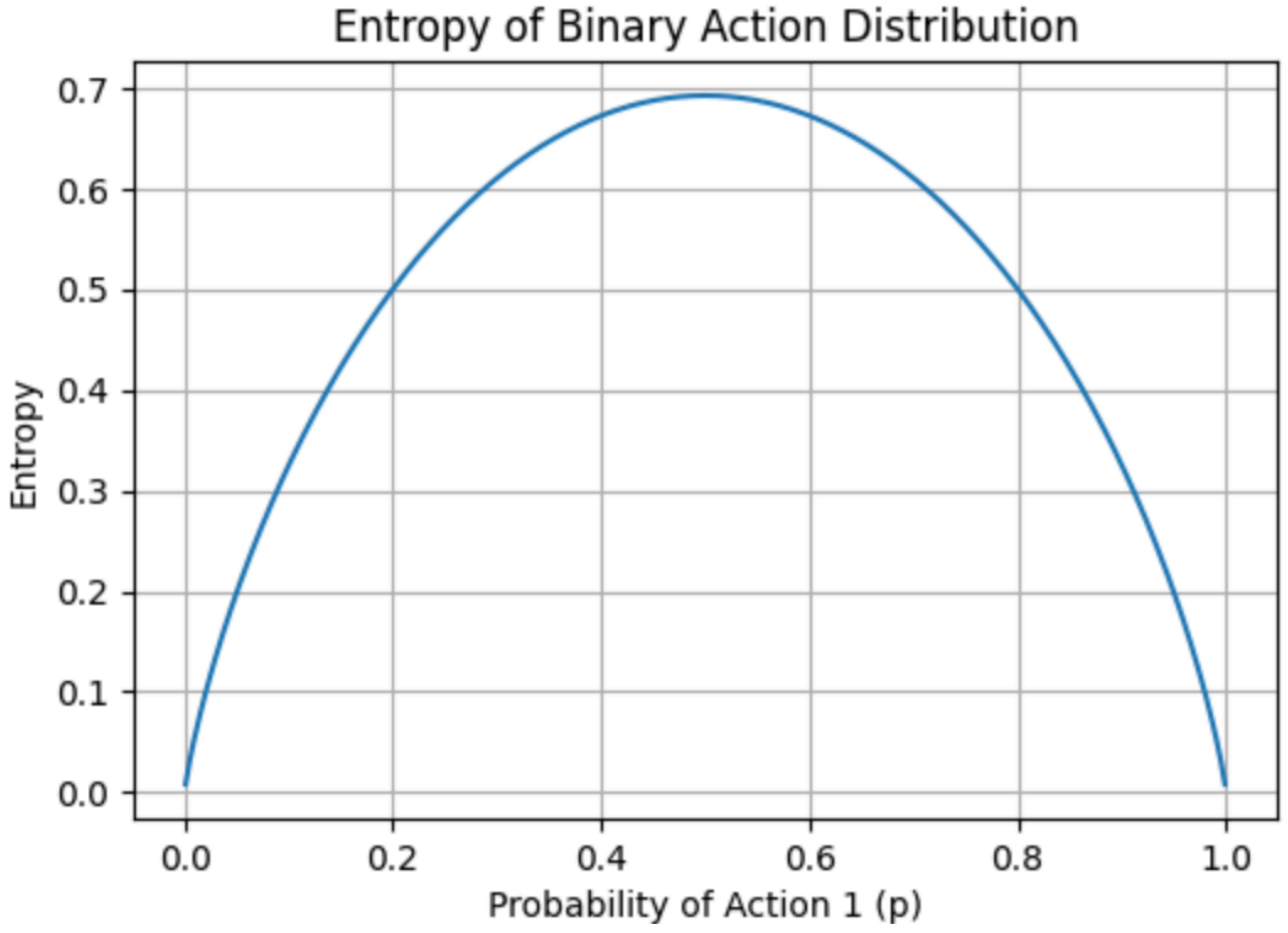
end for

Value based control

- Control = Simply Learning (a policy or otherwise)
- Instead of policy gradient [$a_t \sim \pi_\theta(\cdot \mid s_t)$]
  - We can also do value based learning with the Q function. Here we only have $Q(s, a)$ defined as a NN and action selection has to be added externally
    - $a_t = \begin{cases} \text{a random action from } \mathscr{A}(s_t), & \text{with probability } \varepsilon, \\ \arg\max_a Q(s_t, a), & \text{with probability } 1 - \varepsilon\,. \end{cases}$
    - $Q^\pi(s, a) = \mathbb{E}_\pi\left[G_t \,\middle|\, S_t = s, A_t = a\right]$
- Exploration vs exploitation
  - Why we don't have ϵ greedy in policy gradient?
  - We can add entropy:

$$H(\pi(\cdot \mid s)) = -\sum_a \pi(a \mid s) \log \pi(a \mid s)$$

Example if you have only two actions



Temporal Difference Learning

- Foundation of RL: Temporal Difference Learning
- Let's first explain with a real world example:
    - What happens if I move the queen to the red star?
        - Do I need to wait for the game to finish?
    - What happens if I take a wrong turn and end up on the opposite end of the highway?
        - Do I need to wait for an accident to happen?
- Also called **bootstrapping**
- Why is this better?

SARSA

- See how we are learning at every step instead of waiting for the end?
  - That's temporal difference learning
- Reading: TD-λ:
  - Instead of doing one-step-back update, we go back further with λ decay

Algorithm 5 SARSA with Neural Network Function Approximation (Explicit Loss Form)

Require: step size α, discount factor γ, exploration parameter ε
Require: neural network $Q_\theta(s, a)$ with parameters θ

for episodes $n = 1, 2, \dots$ **do**
  Initialize starting state s_0
  Choose initial action a_0 using an ε-greedy policy from $Q_\theta(s_0, \cdot)$
  for $t = 0, 1, 2, \dots$ until s_t is terminal **do**
    Take action a_t, observe reward r_{t+1} and next state s_{t+1}
    if s_{t+1} is terminal **then**
      Set TD target:
$$\hat{q}_t = r_{t+1}$$
    else
      Choose next action a_{t+1} using an ε-greedy policy from $Q_\theta(s_{t+1}, \cdot)$
      Set TD target:
$$\hat{q}_t = r_{t+1} + \gamma Q_\theta(s_{t+1}, a_{t+1})$$
    end if
    Compute current prediction:
$$q_t = Q_\theta(s_t, a_t)$$
    Define instantaneous loss:
$$\mathcal{L}_t = \frac{1}{2}\left(\hat{q}_t - q_t\right)^2$$
    Update network parameters by (stochastic) gradient descent:
$$\theta \leftarrow \theta - \alpha \, \nabla_\theta \mathcal{L}_t$$
    Set $s_t \leftarrow s_{t+1}$; $a_t \leftarrow a_{t+1}$ (if non-terminal)
  end for
end for

Markov Decision Processes

- Generally RL courses and books start with MDP formulation of the RL problem
- Remember how the agent's decisions depend only on the current state (e.g. $a_t \sim \pi_\theta(\cdot \mid s_t)$)?
  - That assumption — the current state contains all the relevant information — is called the Markov property
- When this holds, we say the environment is Markov, and that allows us to model it as a Markov Decision Process (MDP)
  - The RL problem is defined as MDP problem: $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$
    - $\mathcal{S}$ is the set of all possible states.
    - $\mathcal{A}$ is the set of all the actions available to the agent
    - $P(s' \mid s, a)$ **is transition probability (how the environment changes).**
    - $R(s, a)$ is the immediate reward
    - γ is the discount factor

Early RL: Tabular data

- In this lecture we directly went to the use of NNs for modeling the policy functions, value functions etc.
  - In Sutton and Barto, you will see a use of tabular objects:
    - Works if the spaces are limited
  - State-value tables (V-tables)
  - Action-value tables (Q-tables)
  - Transition matrix representing transition probabilities
- Useful for very very simple problems (like tic-tac-toe) and for understanding algorithms but modern RL usage mostly requires NNs

Reading only: Dynamic Programming and Bellman Equations

- Transition dynamics are perfectly known as a MDP
  - The algorithms to solve for the optimal policies = Dynamic Programming
- Read about the bellman equations and proofs of how value iteration converges to optimal policies in Sutton and Barto (Chapter 4 with connections to Chapter 3)

- $V^{\pi}(s) = \sum_{a} \pi(a|s) \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V^{\pi}(s')\big]$
- $Q^{\pi}(s,a) = \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma \sum_{a'} \pi(a'|s') Q^{\pi}(s', a')\right]$

Off-policy RL

- Both SARSA and REINFORCE that we saw are on-policy
  - We are working with one episode at a time:
    - Can also work with batches for stability similar to supervised RL
- We can also keep the past experience in a buffer
  - And train the NNs again: **experience replay**
  - However, need to be careful that now you are optimizing a policy using data generated by other: off-policy reinforcement learning (RL)
- Learning online:
  - Challenging: NNs forget
    - We want them to remember but also not always

Planning in RL

- We also learn the state-transition functions
    - Think about playing chess again
    - Or any form of imagination:
        - If I do this, this will happen
    - After doing this, you can update your value function for example: this is called planning
    - You have a model of the real world: **model-based RL**
- Using tabular algorithm for explanation only

- What else can you do?
- Generative Models
- Used in dreamer architectures
    - https://arxiv.org/pdf/2301.04104

Algorithm 6 Dyna-Q

Require: step size α, discount factor γ, exploration parameter ε
Require: number of planning steps n_{plan}

Initialize action-value estimates $Q(s,a)$ arbitrarily
Initialize learned model $\hat{P}(s' \mid s,a)$ and $\hat{R}(s,a)$ (e.g., empty tables)
for episodes $k = 1, 2, \ldots$ **do**
  Initialize starting state s
  for $t = 0, 1, 2, \ldots$ until s is terminal **do**
    Choose action a using ε-greedy policy derived from $Q(s, \cdot)$
    Take action a, observe reward r and next state s'
    (1) Direct RL Update

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

    (2) Update the Model
    Store transition and reward in the model:

$$\hat{P}(s' \mid s,a) \leftarrow 1, \qquad \hat{R}(s,a) \leftarrow r$$

    (3) Planning Updates
    for $i = 1, \ldots, n_{\text{plan}}$ **do**
      Randomly sample a previously observed state–action pair $(\tilde{s}, \tilde{a})$
      Query the learned model to obtain:

$$\tilde{r} = \hat{R}(\tilde{s}, \tilde{a}), \qquad \tilde{s}' \sim \hat{P}(\cdot \mid \tilde{s}, \tilde{a})$$

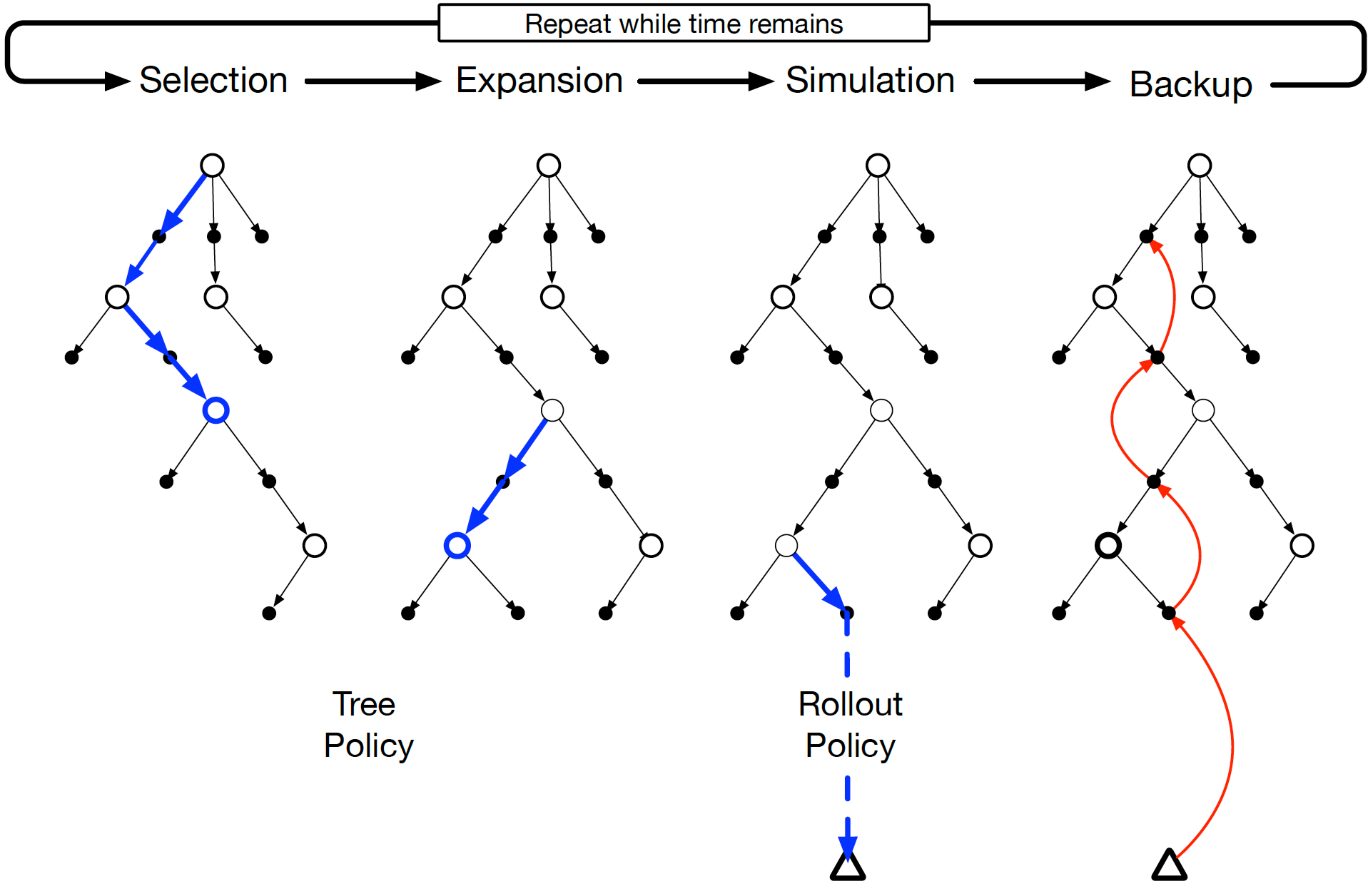
      Perform a simulated Q-learning update:

$$Q(\tilde{s},\tilde{a}) \leftarrow Q(\tilde{s},\tilde{a}) + \alpha \left[\tilde{r} + \gamma \max_{a'} Q(\tilde{s}',a') - Q(\tilde{s},\tilde{a}) \right]$$

    end for
    Set $s \leftarrow s'$
  end for
end for

Planning: Monte Carlo Tree Search

- Start with the current state, which becomes the root node of the search tree.
- Selection: From the root, follow a tree policy (e.g., UCB/PUCT) down the tree until you reach a leaf.
- Expansion: If that leaf has any unexplored actions, add one or more new child nodes.
- Simulation: From the leaf (or new child), run a rollout to the end of the episode using a simple policy.
- Backup: Take the return from the rollout and propagate it back up the tree, updating value estimates on all edges used during selection.
- Action choice: After many iterations, pick the action from the root with the highest visit count—this is treated as the best move.



Sutton and Barto

AlphaGo's Victory (2016)

- First AI to defeat a world champion Lee Sedol in Go on March 15, 2016, a game long considered intractable for classical search.
- Foundation: Monte Carlo Tree Search (MCTS) but with reinforcement learning, and deep neural networks
- Learned from both expert human games and self-play reinforcement learning.
- Used policy networks to propose promising moves and a value network to estimate win probability.

More successes from MCTS

- AlphaGo Zero (2017) learned only from self-play
- AlphaZero (2017): generalization of AlphaGo Zero to other games
- AlphaFold (2020-2021):
  - Applied deep learning to protein folding, a grand challenge in biology
  - Achieved near-experimental accuracy on protein structure prediction
  - Considered one of the biggest scientific breakthroughs of the decade
- MuZero (2019):
  - General model-based RL algorithm that learns the rules of the environment itself
  - Doesn’t need to know game dynamics beforehand

Using RL for your problem

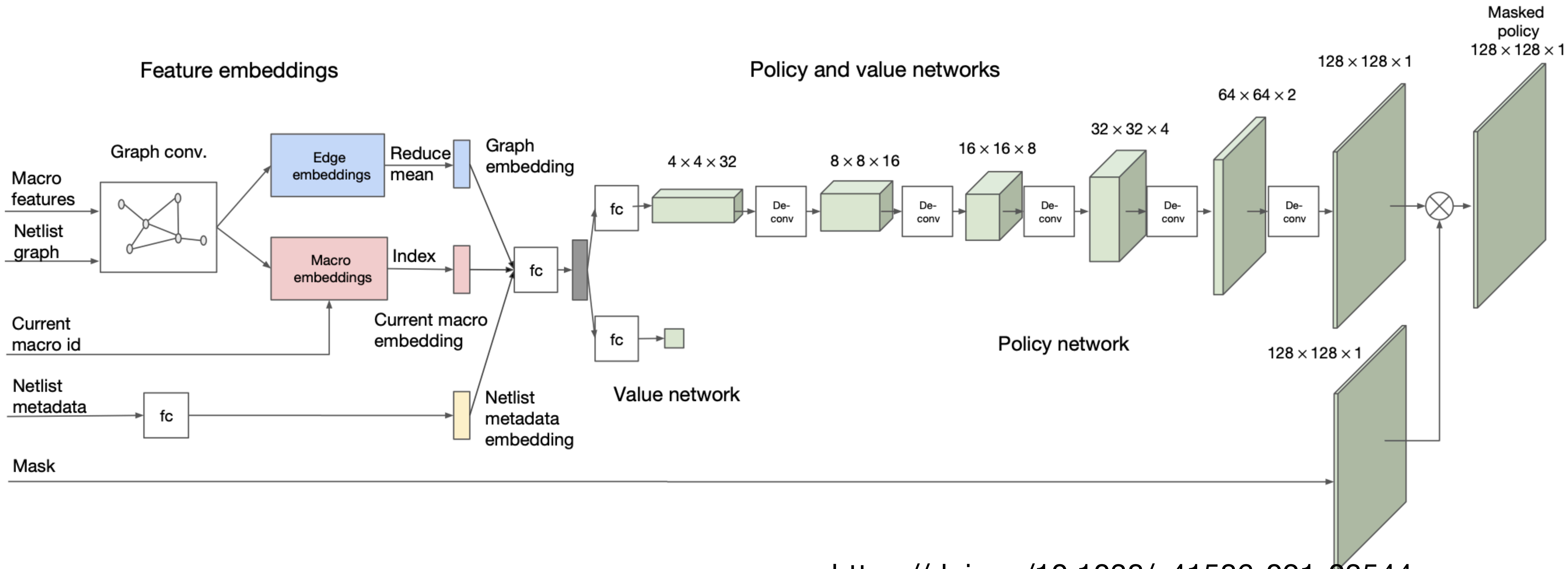
- Unfortunately, a lot of fine tuning needs to be done. Most algorithms are somewhat unstable
  - A lot of "tricks" (such as gradient clipping) need to be applied to make training works
    - Which is why it is hard to write your own algorithms from scratch: The algorithms we looked at in this lecture are for understanding mostly
  - Play with the parameters to find what works
- A RL library or your own algorithm
  - RLLib: https://docs.ray.io/en/latest/rllib/index.html
  - Stable baselines 3: https://stable-baselines3.readthedocs.io/en/master/
  - I always recommend Proximal Policy Optimization (PPO) as the first method to try depending on the problem
- Using a library, we only need two elements. What are they?
  - Defining a value network
  - Defining a policy network

```
.training(
    train_batch_size=512,
    minibatch_size=128,
    num_epochs=4,
    lr=3e-4,
    gamma=0.995,
    lambda_=0.95,
    use_gae=True,
    grad_clip=1.0,
    entropy_coeff=0.005,
    clip_param=0.8,
    vf_loss_coeff=0.1,
    vf_clip_param=1000.0,
)
```

Training configuration for PPO in RLLib

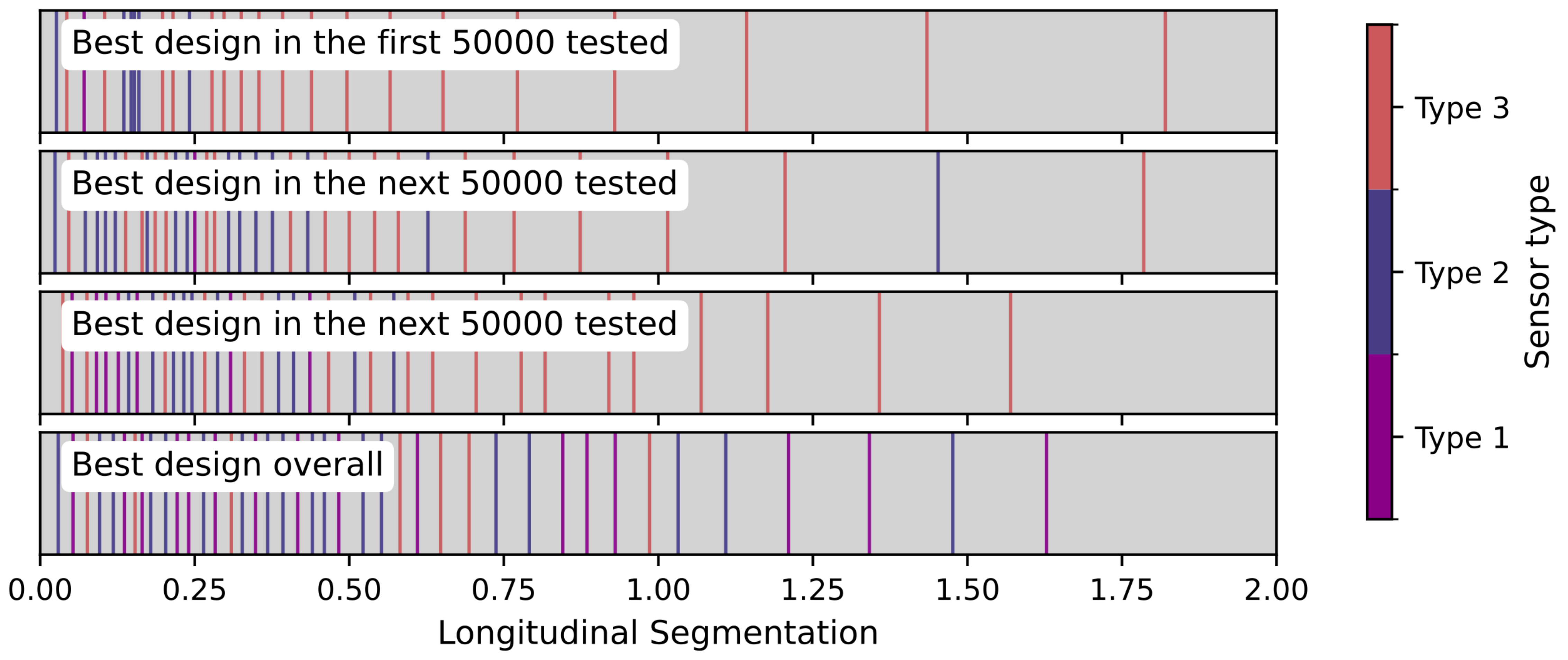
RL for Chip Placement

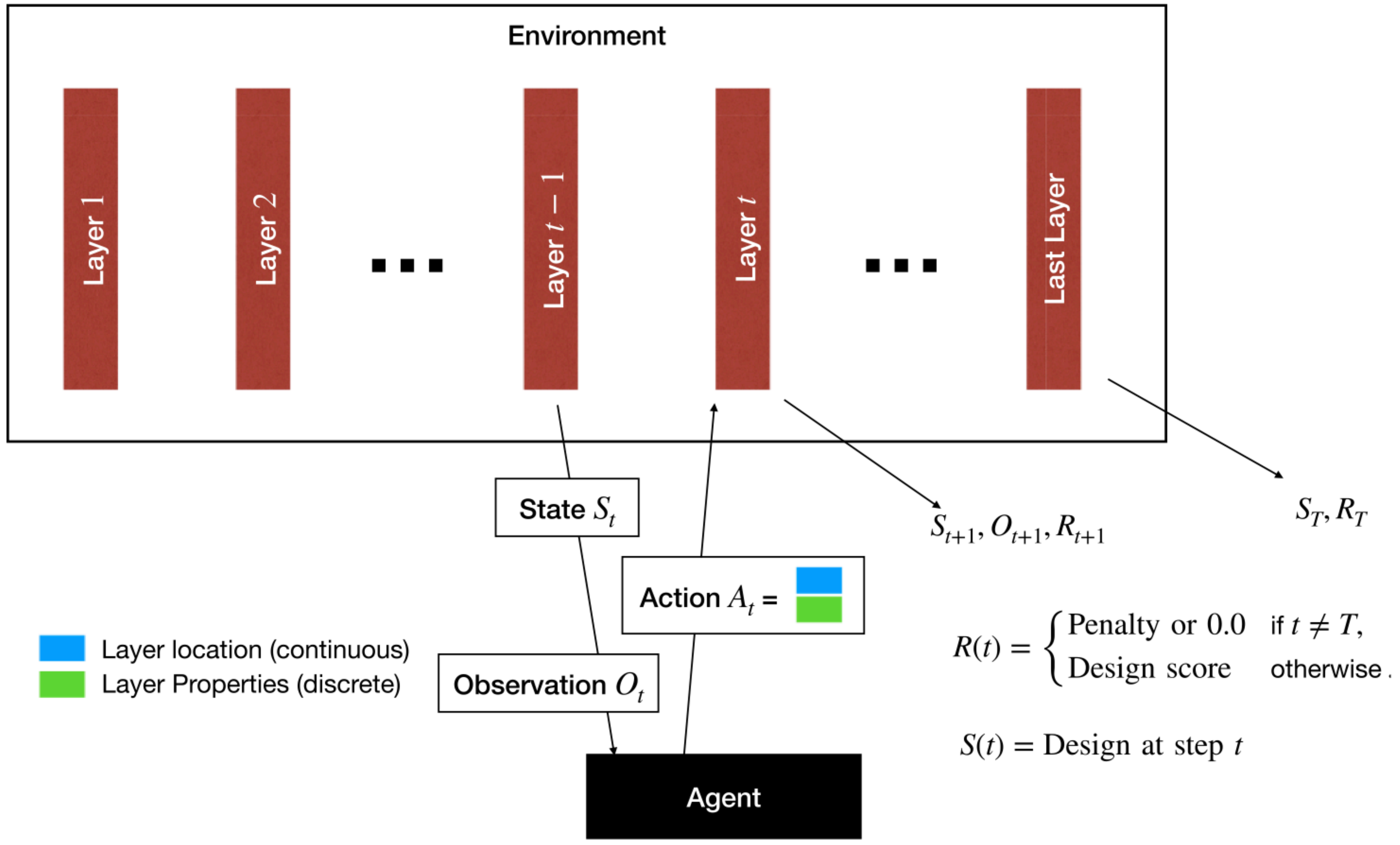
- Using PPO
- GNNs as value and policy network
- Achieves competitive or superior PPA (power, performance, area) compared to expert-designed or algorithmic methods, while reducing design iteration time.

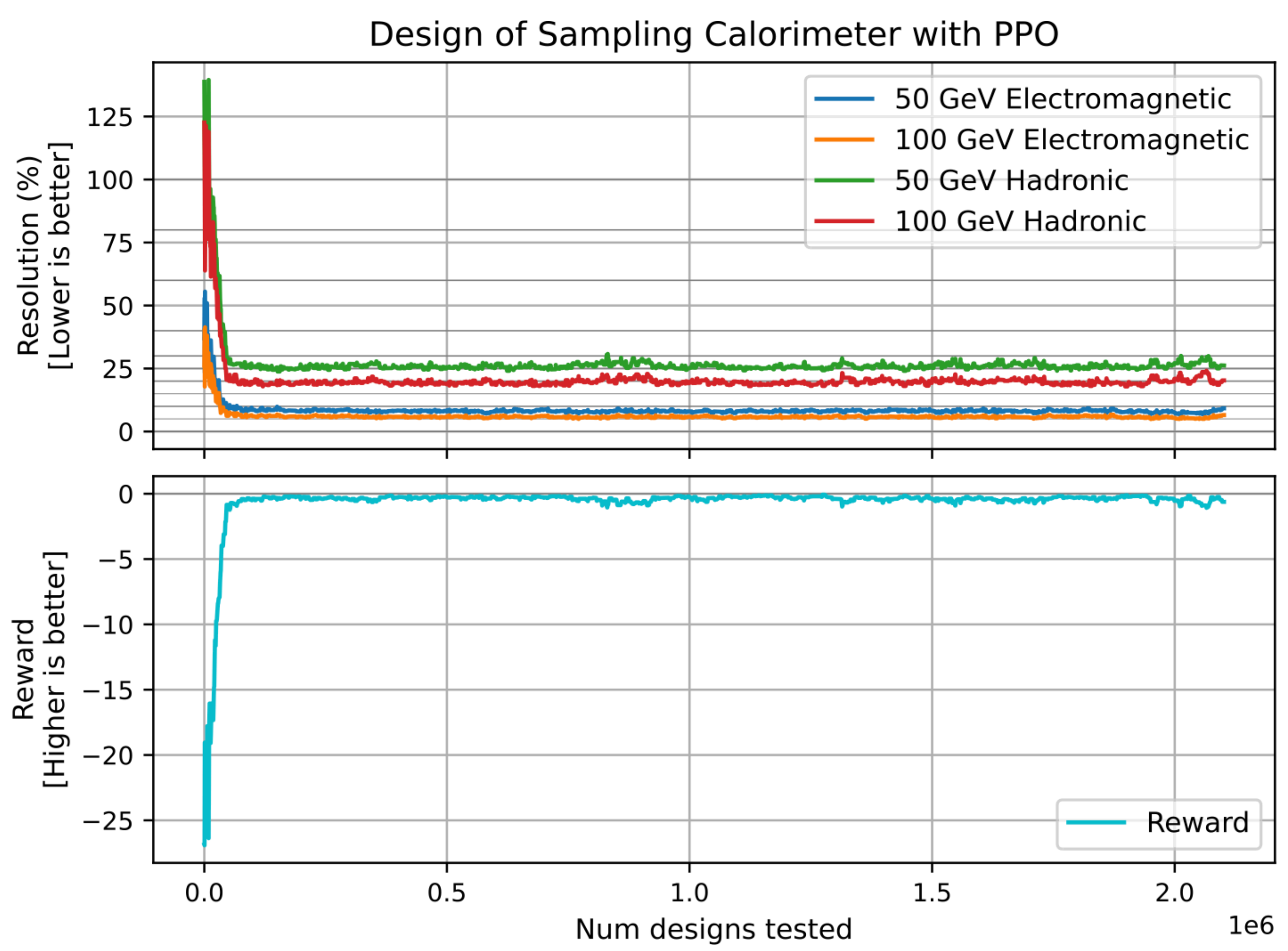


https://doi.org/10.1038/s41586-021-03544-w

RL for Instrument Design







- https://doi.org/10.1088/2632-2153/adf7ff
- Our group works a lot on using AI for design

Conclusion

- Reinforcement Learning is interesting
  - If you are interested in general theory of learning, look into RL not LLMs
- Already applied to a lot of problems (chess, design, optimization of many systems)
  - And we are expecting the usage to only increase in the future

- Thank you!