Reinforcement learning bootcamp 2025

Lecture 2
Advances and Challenges in Policy Gradient Reinforcement Learning: From Theoretical Foundations to Real-World Deployment

Simon Hirlaender

With many ideas from Emma Brunskills Lectures and some from Sergey Levine

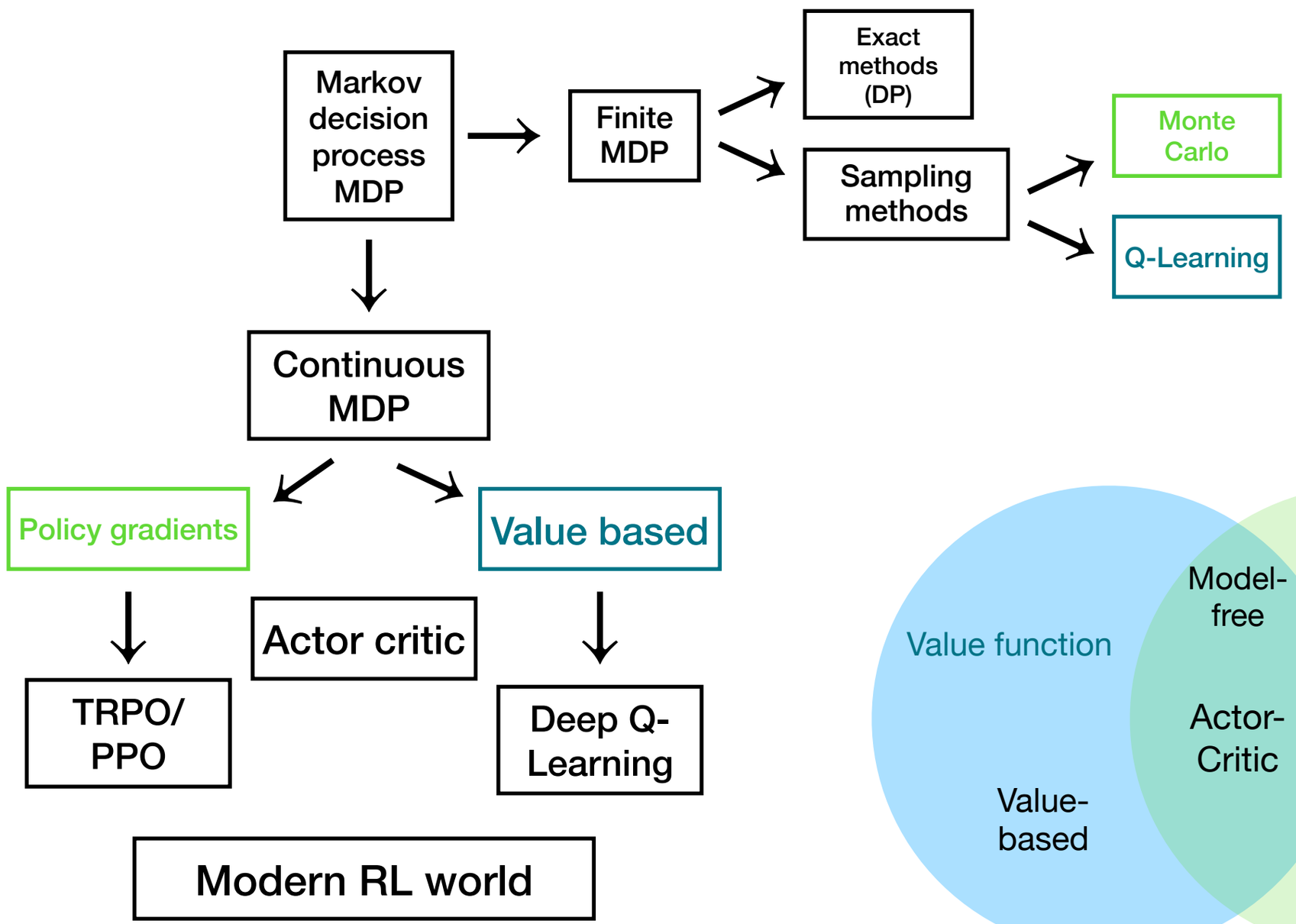
PARIS LODRON UNIVERSITÄT SALZBURG
Lab for Intelligent Data Analytics Salzburg

Outline

- Modern landscape characteristics and approaches
- PPO (modern policy gradients) in a nutshell
- RL and the real world
  - ➡ Capture **your** problem
  - ➡ **Inherent** RL problems
- Summary and takeaways

The modern deep RL world

Overview


Markov decision process MDP
Finite MDP
Exact methods (DP)
Sampling methods
Monte Carlo
Q-Learning
Continuous MDP
Policy gradients
Value based
Actor critic
TRPO/ PPO
Deep Q-Learning
Modern RL world

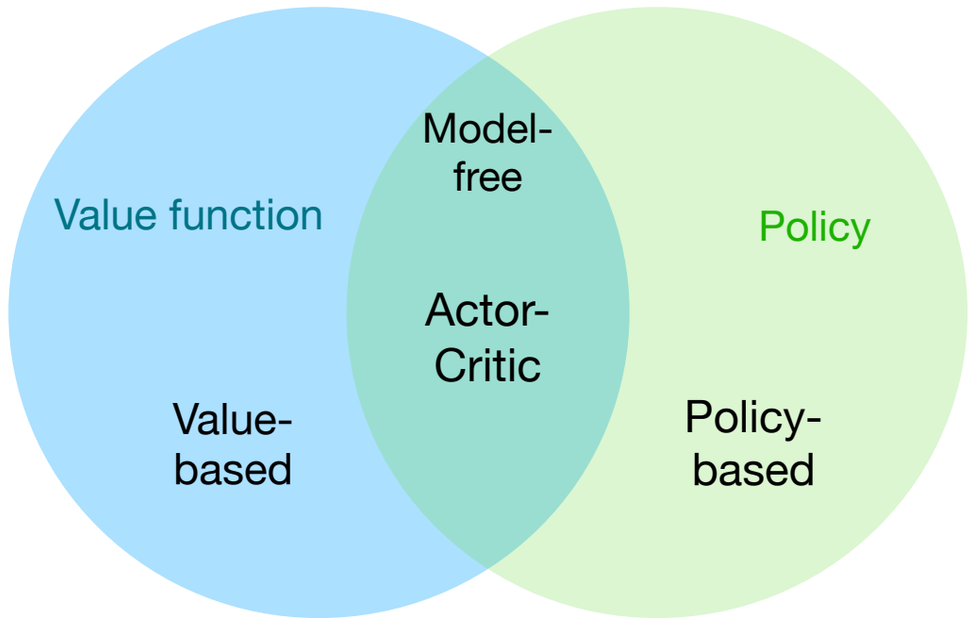
Value function
Model-free
Policy
Actor-Critic
Value-based
Policy-based

Characteristic inherited by all modern algorithms

- Return G_t ($G_t = \sum_k \gamma^k r_{t+k}$) is an unbiased estimator of $V^\pi(s_t)$
- TD[0]-Target: $[r_t + \gamma V^\pi(s_{t+1})]$ is a biased estimator of $V^\pi(s_t)$ (can be an n-step estimate

  $G_t^{(n)} = \sum_{k=0}^{n-1} \gamma^k r_{t+k} \; + \; \gamma^n V^\pi(s_{t+n}))$

  ➡ Usually much lower variance than single G_t
- Return G_t is function of multi-step sequence of random actions/states and rewards
  - TD[0] target has only one random action/state and reward
  - MC (Policy - based)
    - ➡ Unbiased
    - ➡ High variance
    - ➡ Consistent (converges to the true value) even with function approximation
  - TD (Value - based)
    - ➡ Biased
    - ➡ Low variance
    - ➡ TD[0] converges to the true value in the tabular case
    - ➡ TD[0] does not always converge with function approximation



Modern Deep RL world

Make more stable!

Value based relies on Bellman updates

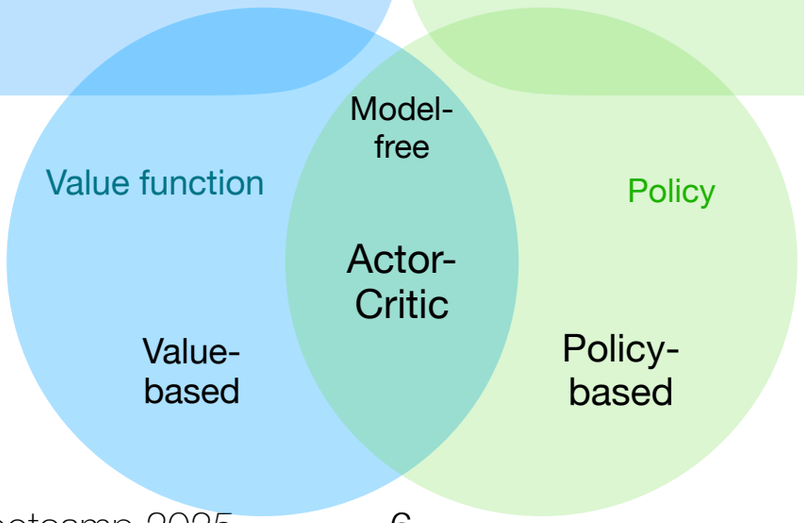
- Biased
- **Can be offline**
- **Brittle**
- No guarantees (deadly triad)
- Only Deterministic
- Sample efficient
- Examples: DDPG, TD3, SAC, DQN …

Actor/Critic

Reduce variance!

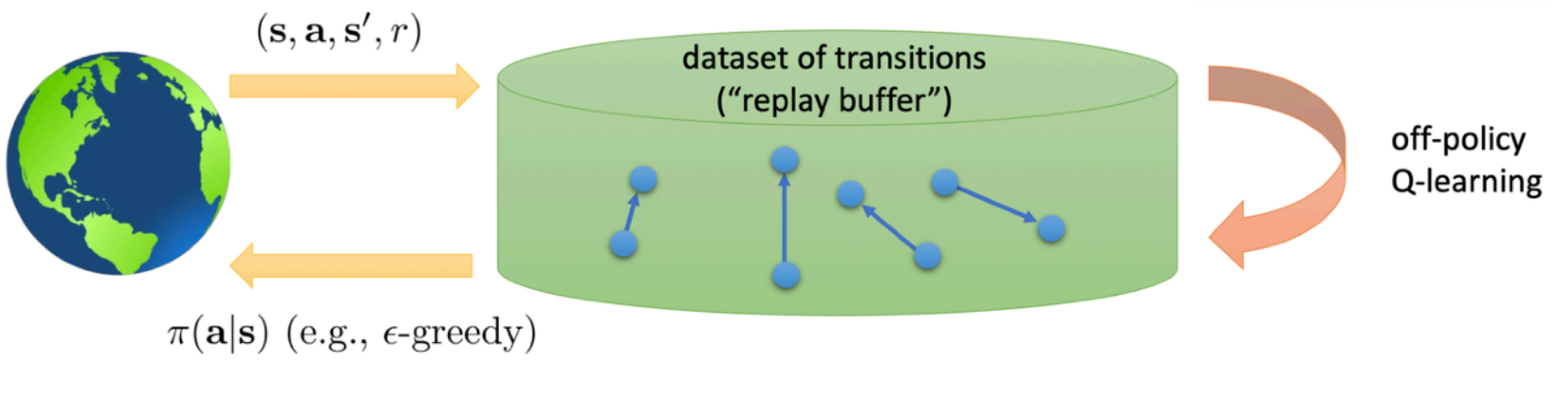
Policy based - stochastic optimisation

- High Variance
- Generally online
- **Stable**
- **Local performance guarantees**
- Allows for stochastic policies
- Low computational cost
- Examples: AC, TRPO, PPO…

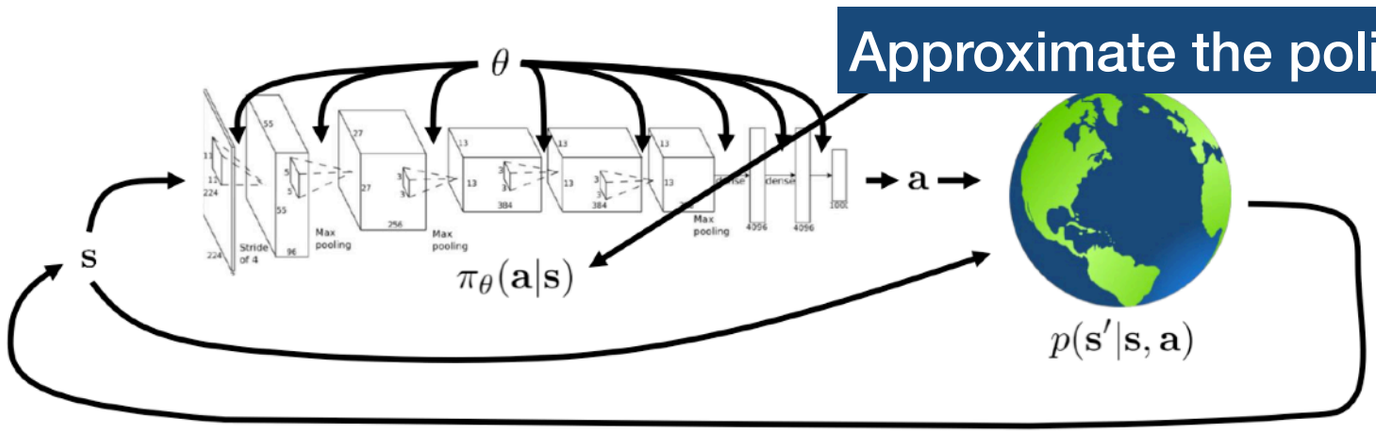


Difference between value based and policy based using function approximation

$$\pi(s \mid a) = \arg\max_{a'} \hat{Q}(s, a)$$

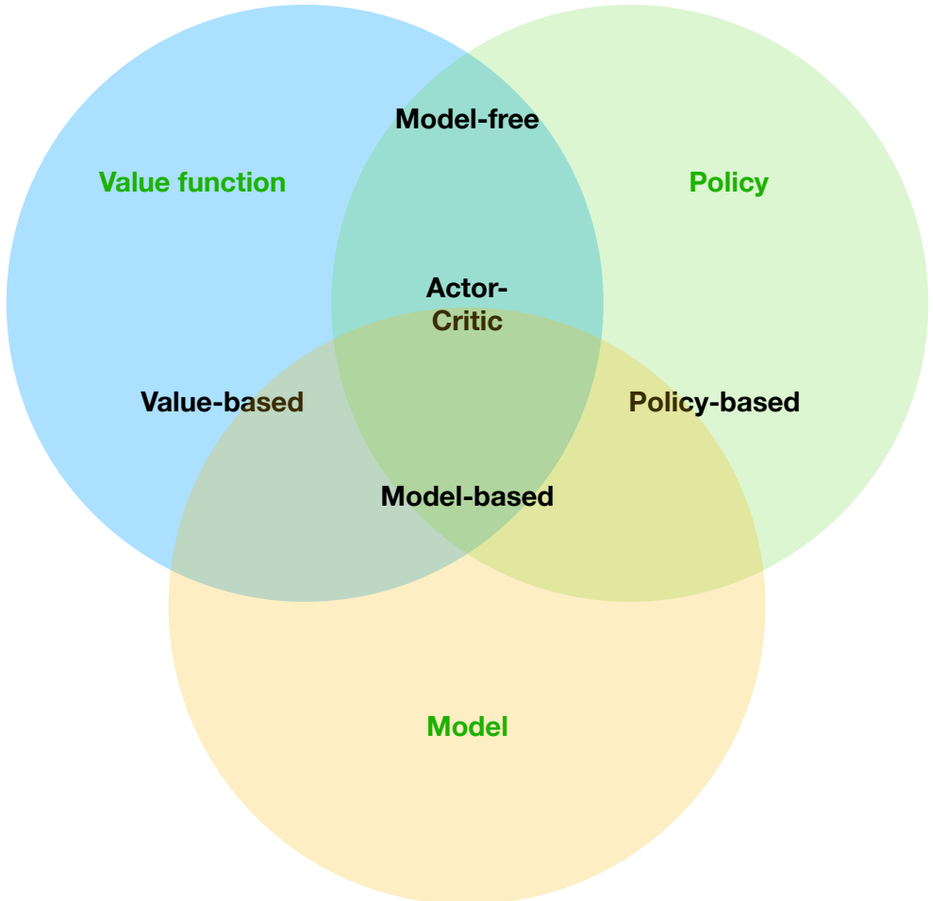


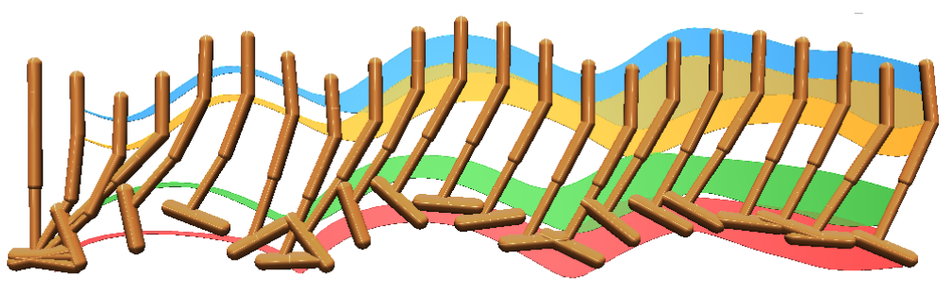
Approximate the policy directly



http://rail.eecs.berkeley.edu/deeprlcourse/

Some words about model-based





- Learn a model of the environment
  - ➡ Dynamics $p(s' \mid s, a)$, reward $r(s, a)$
  - ➡ Use the (probabilistic) model for planning
  - ➡ Simulate trajectories, lookahead, MPC, tree search
  - ➡ Algorithms: Dyna-Q, Monte Carlo Tree Search, Model Predictive Control.
- Sample efficiency
  - ➡ Fewer real interactions needed by reusing the model
- Bias–variance tradeoff
  - ➡ Imperfect models introduce **(model) bias**
  - ➡ But enable massive **variance reduction** via simulation
- Hybrid approaches
  - ➡ Combine with model-free (e.g. AlphaZero, Dreamer, MBPO)

PPO in a nutshell

Direct policy search

- **First: How to measure the quality of a policy π?**
- $$J(\pi) = \mathbb{E}_{\tau \sim p_\pi(\tau)} \left[\sum_t \gamma^t r(s_t, a_t) \right] = V^\pi(s_0)$$

  (episodic case for intuition)
- Now find the optimal policy $\pi^* = \arg\max_\pi J(\pi)$

- **This is an optimisation problem!**

Policy Optimisation Objective

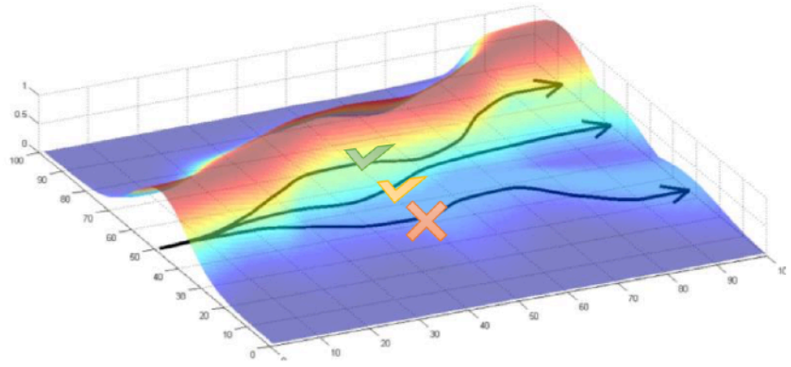
- **This is an optimisation problem!**
- Parametrise $\pi(a \mid s)$ by $\pi_\theta(a \mid s)$
- Goal: $\pi_\theta(a \mid s)$ with parameters θ be given, search θ
- Measure the quality of π_θ by the learning objective:

$$J(\pi_\theta) = \mathbb{E}_{\tau \sim p_{\pi_\theta}(\tau)} \left[\sum_t \gamma^t r(s_t, a_t) \right] = V^{\pi_\theta}$$

- The optimal policy $\pi^* = \arg\max_\pi J(\pi) \rightarrow \theta^* = \arg\max_\theta J(\pi_\theta))$
- **How to find the best parameters θ^*?**

Direct policy search/Policy gradients

- Objective if episodic: $J(\theta) = V^{\pi_\theta}(s_0) := V(\theta)$
  - ➡ Direct naïve approaches: stochastic search (random search, Nelder–Mead, Bayesian optimisation)
- Using more information - the gradient:
  - ➡ $V(\theta) = \sum_\tau P(\tau;\theta) R(\tau)$ (Trajectory probability; Trajectory reward)
  - ➡ $\nabla_\theta V(\theta) = \sum_\tau P(\tau;\theta) R(\tau) \nabla_\theta \log P(\tau;\theta) = \mathbb{E}[R(\tau) \nabla_\theta \log P(\tau;\theta)]$ (Log likelihood trick; Stochastic gradient)
  - ➡ Found by sampling of $A_t \sim p(\cdot \mid \tau_t;\theta)$



Cool but…

Trajectory probability

$$\nabla_\theta V(\theta) = \mathbb{E}[R(\tau)\nabla_\theta \log P(\tau;\theta)]$$

Trajectory reward

- Unbiased but very, very noisy
- Can be made practical by:
  - Temporal structure - causality - REINFORCE
  - Baseline - and near optimal choice is $V(s)$ (actor-critic)

General: Policy Gradient Theorem

- $\mathbf{g} = \nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_t \gamma^t \nabla_\theta \log \pi_\theta(a_t \mid s_t) A^{\pi_\theta}(s_t, a_t) \right]$

- Where:
  - $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$ (Advantage = how much better an action is than average at state s)
  - $Q^\pi(s,a) = \mathbb{E}[\sum_t \gamma^t r_t \mid s_0 = s, a_0 = a, \pi]$
  - $V^\pi(s) = \mathbb{E}[\sum_t \gamma^t r_t \mid s_0 = s, \pi]$

Still open challenges of PG

1. Sample efficiency is poor
2. Distance in parameter space ≠ distance in policy space!
   - What is policy space? For tabular case, set of matrices

$$\Pi = \left\{ \pi : \pi \in \mathbb{R}^{|S|\times|A|}, \sum_a \pi_{sa} = 1, \pi_{sa} \geq 0 \right\}$$

   - PG take steps in parameter space → step-size is hard to get right as a result

Cool but…

Trajectory probability

$$\nabla_\theta V(\theta) = \mathbb{E}[R(\tau)\nabla_\theta \log P(\tau;\theta)]$$

Trajectory reward

- Unbiased but very, very noisy
- Can be made practical by:
  - Temporal structure - causality - REINFORCE
  - Baseline - and near optimal choice is $V(s)$ (actor-critic)
  - **Other tricks:**
    - Reward & advantage normalisation
    - **n-step estimate/GAE baseline…**
    - **Trust-region / clipping (TRPO, PPO)**

Why Policy Gradients waste data

- Sample efficiency for vanilla policy gradient (as seen so far) methods is poor
- Discard each batch of data immediately after just one gradient step
  - ➡ Why? PG is an on-policy expectation.
- Two main approaches to obtaining an unbiased estimate of the policy gradient
  - ➡ Collect sample trajectories from policy, then form sample estimate. (More stable)
  - ➡ Use trajectories from other policies (Less stable)
- Opportunity: use old data to take multiple gradient steps before using the resulting new policy to gather more data
- Challenge: even if this is possible to use old data to estimate multiple gradients, how many steps should be taken?

Choosing a Step Size for Policy Gradients

- Policy gradient algorithms are stochastic gradient ascent: $\theta_{k+1} = \theta_k + \alpha_k \hat{\mathbf{g}}_k$ with step size $\Delta_k = \alpha_k \hat{\mathbf{g}}_k$.
- **If the step is too large, performance collapse**

Why are the step sizes so enormously important in policy gradients?

- Step-sizes are important in any problem in which the optima of a function is to be found
- In supervised learning: step too far → in the next update this will be corrected

- In RL:
  - ➡ Step too far→ bad policy
  - ➡ Next Batches: collected under bad policy
  - ➡ Policy determines the data! Controlling exploration/exploitation trade-off by certain policy parameters and the stochasticity of the policy
  - ➡ Possibly no recovery from bad selection→**Collapse of Performance**

Choosing a Step Size for Policy Gradients

- Policy gradient algorithms are stochastic gradient ascent:$\theta_{k+1} = \theta_k + \alpha_k \hat{\mathbf{g}}_k$ with step size $\Delta_k = \alpha_k \hat{\mathbf{g}}_k$.
- If the step is too large, performance collapse
- If the step is too small, progress is unacceptably slow
- “Right” step size changes based on θ
- Automatic learning rate adjustment like advantage normalisation, or Adam-style optimisers, can help. But does this solve the problem?

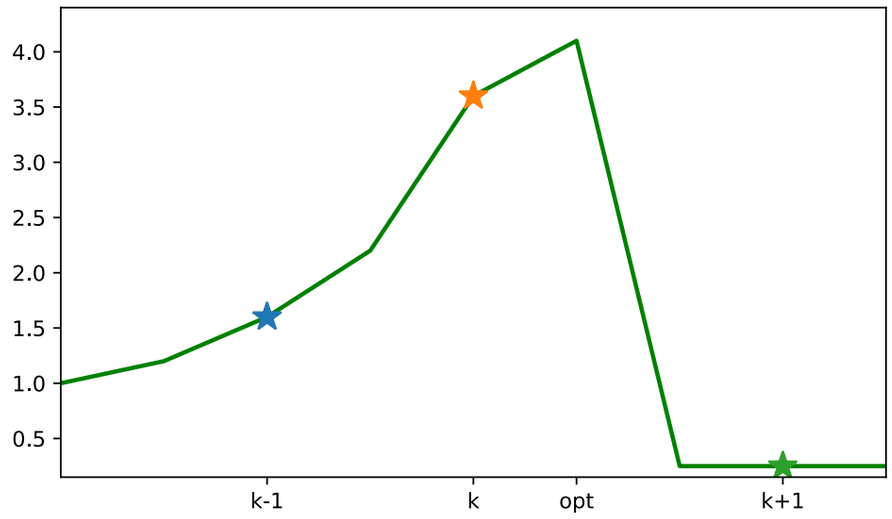


Figure: Policy parameters on x-axis and performance on y -axis. A bad step can lead to performance collapse, which may be hard to recover from.

Sensitivity of Policies to Parameterisation

- Consider a family of policies with parametrisation: $\pi_\theta(a) = \begin{cases} \sigma(\theta), & a = 1 \\ 1 - \sigma(\theta), & a = 2 \end{cases}$

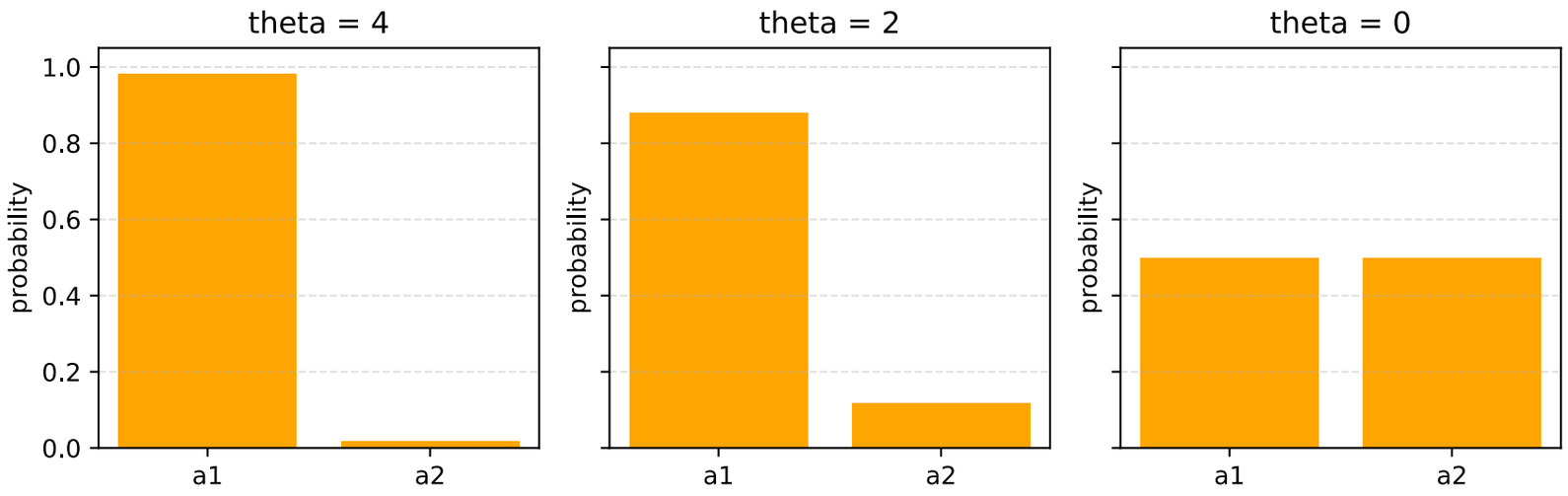


Figure: Small changes in the policy parameters can unexpectedly lead to big changes in the policy.

Big question: how do we come up with an update rule that doesn't ever change the policy more than we meant to?

What We Need from a Policy Update

- In a policy optimisation algorithm, we want an update step that
  - ➡ uses rollouts collected from the most recent policy as efficiently as possible,
  - ➡ and takes steps that respect distance in policy space as opposed to distance in parameter space.
- What can we do?

How much better/worse is new candidate π'?

- **Performance difference lemma**:

$J(\pi') - J(\pi) = \frac{1}{1-\gamma}\, \mathbb{E}_{s\sim d^{\pi'},\, a\sim\pi'}\left[A^{\pi}(s,a)\right]$, where

$d^{\pi'}(s) = (1-\gamma)\sum_t \gamma^t\, P(s_t = s \mid \pi')$ discounted state distribution of π'

- **Problem: we would need a lot of samples from the new policy π'**
- $J(\pi') \approx J(\pi) + \mathbb{E}_{s\sim d^{\pi},\, a\sim\pi}\left[\frac{\pi'(a\mid s)}{\pi(a\mid s)} A^{\pi}(s,a)\right]$ (importance weighting)
- $\mathcal{L}_{\pi}(\pi') := J(\pi) + \mathbb{E}_{s\sim d^{\pi},\, a\sim\pi}\left[\frac{\pi'(a\mid s)}{\pi(a\mid s)} A^{\pi}(s,a)\right]$
- **Nice but how good is this surrogate?**

Why is this useful?

- $J(\pi') - J(\pi) \approx \mathscr{L}_{\pi}(\pi')$
- $\mathscr{L}_{\pi}(\pi') = \frac{1}{1-\gamma}\,\mathbb{E}_{s\sim d^{\pi},\, a\sim\pi}\left[\frac{\pi'(a\,|\,s)}{\pi(a\,|\,s)}A^{\pi}(s,a)\right] = \mathbb{E}_{\tau\sim\pi}\left[\sum_t \gamma^t \frac{\pi'(a_t\,|\,s_t)}{\pi(a_t\,|\,s_t)}A^{\pi}(s_t,a_t)\right]$
- **This is something we can optimise using trajectories sampled from the old policy π!**
- Similar to using importance sampling ($w(\tau) = \prod_t \frac{\pi'(a_t\,|\,s_t)}{\pi(a_t\,|\,s_t)}$, but because weights only depend on current time step (and not preceding history), they don't vanish or explode.

Theorem: Relative policy performance bounds

- $$\left| J(\pi') - J(\pi) - \mathscr{L}_{\pi}(\pi') \right| \leq \quad C \cdot D_{\mathrm{KL}}^{\max}(\pi \,\|\, \pi'),$$
  - ➡ D_{KL} is the Kullback–Leibler divergence
  - ➡ $D_{\mathrm{KL}}^{\max}(\pi \,\|\, \pi') = \max_{s} D_{\mathrm{KL}}(\pi(\cdot \,|\, s) \,\|\, \pi'(\cdot \,|\, s))$
  - ➡ $C = \frac{4\gamma}{(1-\gamma)^2} \max_{s,a} |A^{\pi}(s,a)|$
- If policies are close in KL-divergence—the approximation is good!

Monotonic Policy Improvement — Key Idea

Theorem (Kakade, Schulman, Achiam)

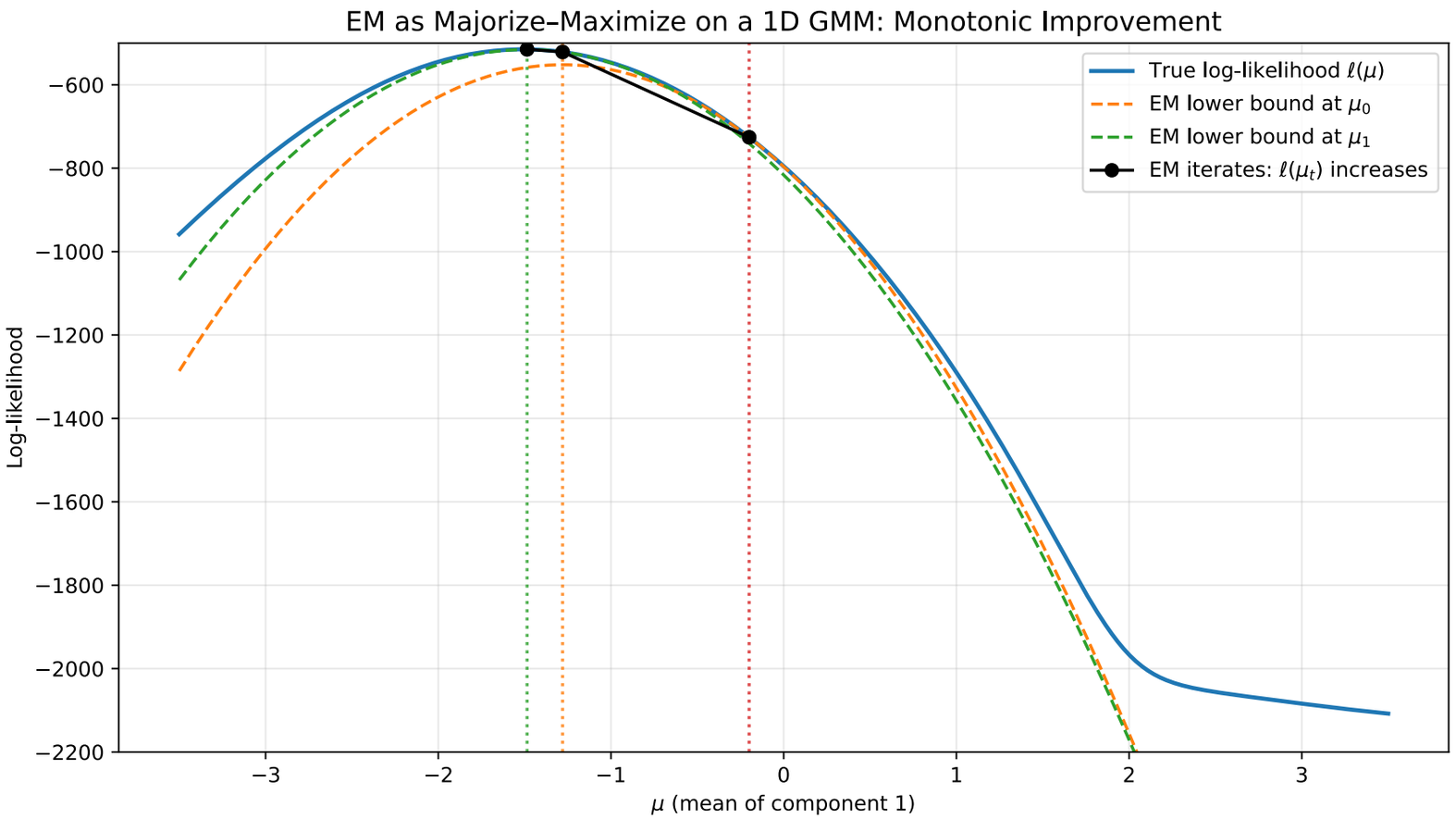
- If we maximise RHS over π', we guarantee $J(\pi') \geq J(\pi)$.
- $J(\pi') - J(\pi) \quad \geq \quad \mathscr{L}_{\pi}(\pi') \; - \; C\sqrt{\mathbb{E}_{s\sim d^{\pi}}[D_{KL}(\pi' \| \pi)[s]]}$.
- Proof sketch:
  - ➡ At $\pi' = \pi$, surrogate $\mathscr{L}_{\pi}(\pi) = 0$, $KL = 0$.
  - ➡ Therefore optimum ≥ 0 → improvement guaranteed.
  - ➡ Works for any policy class Π_{θ}.

KL Control for Stable Policy Improvement

- Limitations:
  - ➡ Constant C can be very large (esp. when $\gamma \approx 1$).
  - ➡ $\Rightarrow$ Updates from theory are too conservative.
- Practical Solutions:
  - ➡ TRPO: use explicit KL-constraint (“trust region”) → guaranteed monotonic improvement.
  - ➡ PPO: replace hard constraint with adaptive KL penalty or clipping.
- Takeaway:
  - ➡ Monotonic improvement proofs justify TRPO/PPO: optimising a surrogate with KL control yields guaranteed (or approximate) policy improvement.

PPO as Majorize–Maximize (like EM)

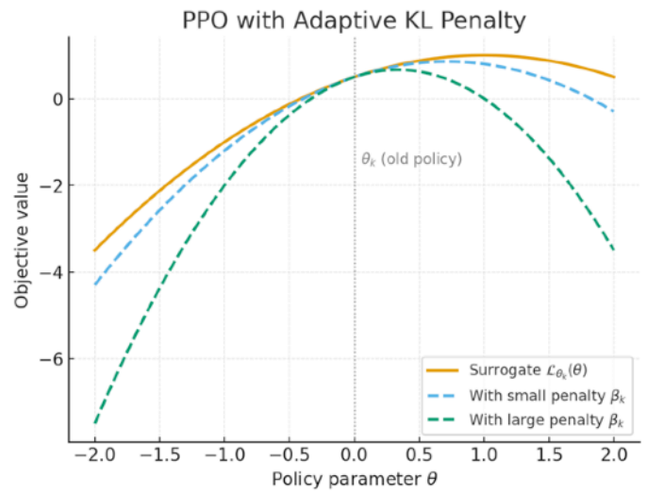
- EM:Each step maximizes a surrogate bound tight at the current iterate → monotonic likelihood improvement.
- TRPO (Trust Region Policy Optimization): Optimize a surrogate return bound subject to a KL constraint → guaranteed monotonic policy improvement (theory matches EM's guarantee).
- PPO:Each step maximizes a clipped surrogate bound around the current policy → approximate monotonic return improvement.



Proximal Policy Optimisation I

Proximal Policy Optimisation (PPO) is a family of methods that approximately penalise policies from changing too much between steps.

- Adaptive KL Penalty:
  - ➡ Policy update solves unconstrained optimisation problem:
    - $\theta_{k+1} = \arg\max_{\theta} \; \mathscr{L}_{\theta_k}(\theta) - \beta_k \, \overline{D}_{\mathrm{KL}}(\theta \,\|\, \theta_k)$
    - $\overline{D}_{\mathrm{KL}}(\theta \,\|\, \theta_k) = \mathbb{E}_{s \sim d^{\pi_k}} \left[D_{\mathrm{KL}}\Big(\pi_{\theta_k}(\cdot \mid s) \,\|\, \pi_{\theta}(\cdot \mid s) \Big) \right]$
  - ➡ Penalty coefficient β_k changes between iterations to approximately enforce KL-divergence constraint
  - ➡ β_k is the penalty coefficient



Additional ingredient: N-step estimator

- You can choose a middle ground between MC and TD estimators for the target function as a replacement for the true values function
  - $\hat{R}_t^{(1)} = r_t + \gamma V(s_{t+1})$
  - $\hat{R}_t^{(2)} = r_t + \gamma r_{t+2} + \gamma^2 V(s_{t+2})$
  - $\hat{R}_t^{(\infty)} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots$
- If you subtract the baseline from it, you get the advantage estimators:
  - $\hat{A}_t^{(1)} = r_t + \gamma V(s_{t+1}) - V(s_t)$
  - $\hat{A}_t^{(2)} = r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2}) - V(s_t)$
  - $\hat{A}_t^{(\infty)} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} \cdots - V(s_t)$
- $\hat{A}_t^a$ Has low variance but high bias. $\hat{A}_t^\infty$ Has high variance but small bias.
- An intermediate stage through e.g.: $k = 10 - 20$ is possible to achieve moderate bias/variance values.

Additional ingredient: GAE

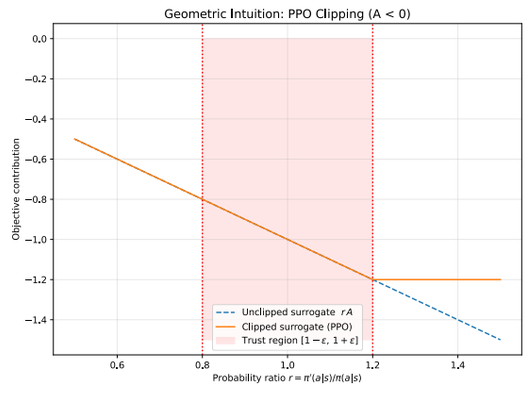
- How to update the advantage function?
  - ➡ Advantage Estimation
  - ➡ Advantage: $\hat{A}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s)$.
  - ➡ N-step returns trade off bias & variance.
- **Generalised Advantage Estimation (GAE)**
- Exponentially weighted average of k-step estimators:
- $$\hat{A}^{GAE}t = \sum_{l=0}^{\infty} (\gamma\lambda)^l \, \delta^V_{t+l}, \quad \delta^V_t = r_t + \gamma V(s_{t+1}) - V(s_t).$$
  - ➡ $\lambda \in [0,1]$ controls bias–variance tradeoff:
  - ➡ $\lambda = 0$: low variance, high bias (TD(0)-like).
  - ➡ $\lambda \approx 1$: low bias, higher variance (MC-like).
  - ➡ PPO uses truncated GAE for practical updates.

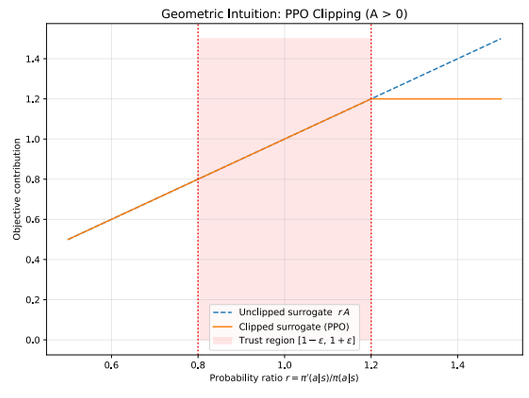
Code with Adaptive KL Penalty

- Input:
  - ➡ Initial policy parameters θ_0, Initial KL penalty β_0, Target KL-divergence δ
- Training Loop
- **For** k = 0, 1, 2, …:
  - ➡ Collect trajectories $\mathcal{D}_k$ under policy $\pi_k = \pi(\theta_k)$.
  - ➡ Estimate advantages $\hat{A}_t^{\pi_k}$ using any estimator.
  - ➡ Policy update:
    - $\theta_{k+1} = \arg\max_{\theta} \; \mathcal{L}_{\theta_k}(\theta) - \beta_k \; \bar{D}KL(\theta \,||\, \theta_k)$
    - Optimisation: Take K steps of mini-batch SGD (e.g., Adam).
  - ➡ Adaptive KL penalty update:
    - **If** $\bar{D}KL(\theta_{k+1} \,||\, \theta_k) \geq 1.5\delta$:
      - $\beta_{k+1} = 2\beta_k$
    - **Elif** $\bar{D}KL(\theta_{k+1} \,||\, \theta_k) \leq \delta/1.5$:
      - $\beta_{k+1} = \beta_k/2$
- Key Notes
  - ➡ Initial KL penalty not crucial — adapts quickly.
  - ➡ Some iterations may violate KL constraint, but most don’t.

Proximal Policy Optimisation II

- **Proximal Policy Optimisation (PPO) is a family of methods that approximately penalise policies from changing too much between steps.**
- Clipped Objective:
  - ➡ Define ratio: $r_t(\theta) = \dfrac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_k}(a_t \mid s_t)}$
  - ➡ Objective: $\mathscr{L}^{\text{CLIP}}\theta_k(\theta) = \mathbb{E}_{\tau\sim\pi_k}\left[\sum_{t=0}^{T} \min\left(r_t(\theta)\hat{A}_t^{\pi_k}, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t^{\pi_k}\right)\right]$
  - ➡ Clipping Hyperparameter: $\epsilon \approx 0.2$
  - ➡ Policy update:
  - ➡ $\theta_{k+1} = \arg\max_\theta \mathscr{L}^{\text{CLIP}}_{\theta_k}(\theta)$





- ➡ Summary
  - Adaptive KL Penalty: penalises deviations from KL target.
  - Clipped Objective: clips probability ratios for stable learning.

Code with clipped Objective

- Input: Initial policy parameters θ_0, Clipping threshold ϵ
- Training Loop
- For k = 0, 1, 2, …:
  - ➡ Collect trajectories $\mathcal{D}_k$ under policy $\pi_k = \pi(\theta_k)$.
  - ➡ Estimate advantages $\hat{A}_t^{\pi_k}$.
  - ➡ Policy update:
    - $\theta_{k+1} = \arg\max_\theta ; \mathcal{L}^{\text{CLIP}}_{\theta_k}(\theta)$
  - ➡ Optimization: Take K steps of minibatch SGD (e.g., Adam).
  - ➡ Clipped Objective
    - Define ratio:
    - $r_t(\theta) = \dfrac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_k}(a_t \mid s_t)}$
    - Objective:
    - $\mathcal{L}^{\text{CLIP}}\theta_k(\theta) = \mathbb{E}\tau \sim \pi_k \left[\sum_{t=0}^{T} \min \left(r_t(\theta)\hat{A}_t^{\pi_k}, , \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t^{\pi_k} \right) \right]$
  - ➡ Policy update uses clipped surrogate objective.
- Key Notes
  - ➡ Clipping prevents policy from making large updates that push it far from π_k.
  - ➡ Works as well as KL penalty version, but simpler to implement.

Key Properties of Modern Policy Gradient Methods

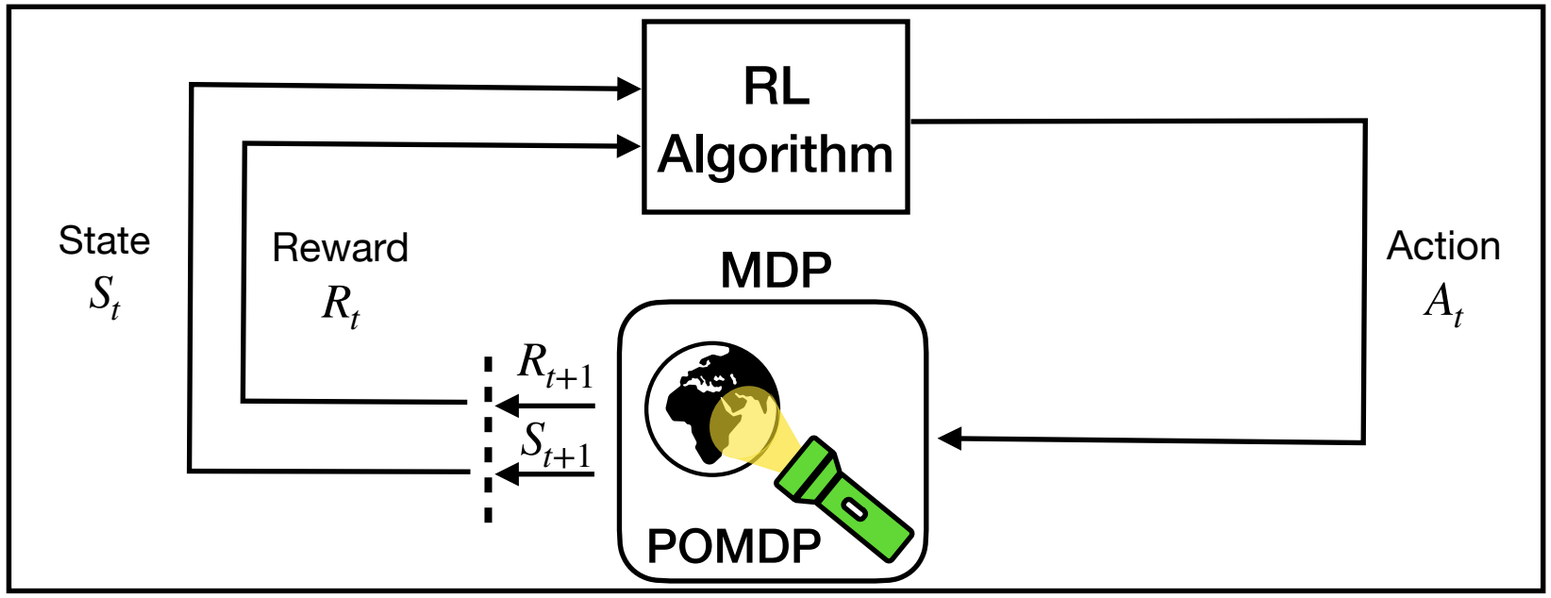
- Improved data efficiency: allows multiple gradient updates per batch before collecting new rollouts
- Stable learning: clipping or KL constraints prevent destructive updates and encourage monotonic improvement
- Conservative policy updates: a central principle in RL research since the early 2000s
- Convergence behaviour: guarantees convergence to local optima
- Practical impact: widely used in practice, simple to implement, and deployed in large-scale systems (e.g., ChatGPT fine-tuning)

PPO summary

- **Extremely popular and versatile**: many variants exist beyond this introduction.
- **Broad applicability**: works even when the **reward** function is **non-differentiable**.
- Commonly **combined with value methods**: especially in actor–critic frameworks.

RL and praxis

The entire problem



MDP Markov decision process
POMDP Partially observable Markov decision process

Challenges of RL (in the real world)

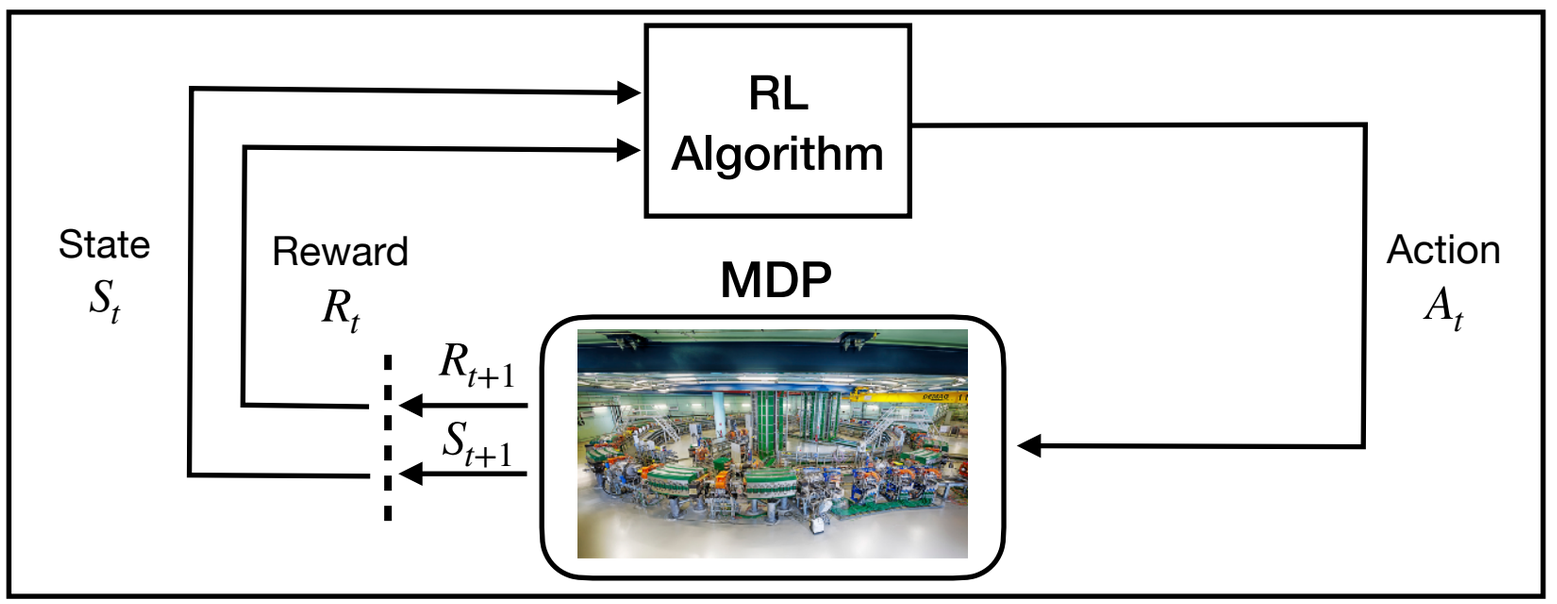
1. **Problem formulation** - capturing **your problem** in an MDP
   - ➡ State representation, Markov Property (e.g. non stationarity)
   - ➡ Reward engineering
   - ➡ ...
2. RL - core issues/RL research:
   - ➡ **Sample efficiency**
   - ➡ **Stability**
   - ➡ **Run time**
   - ➡ **Hyper-parameter tuning**
   - ➡ **Exploration**
   - ➡ **Safety**
   - ➡ **Robustness to Changes, Generalisation**
   - ➡ **...**

Formulating your MDP

The entire problem

Markov decision process - MDP

Problem



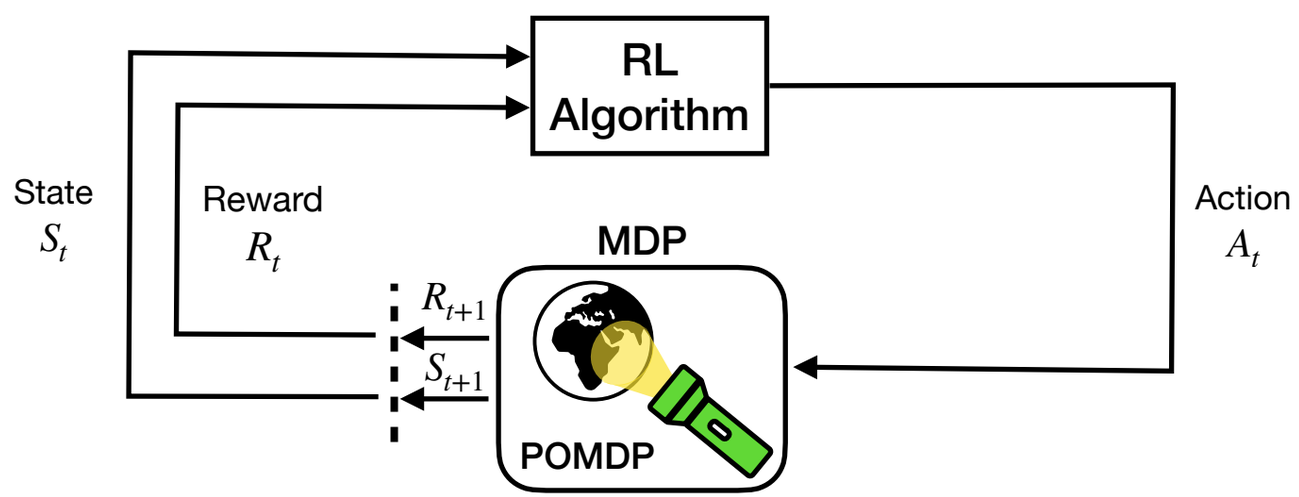
Challenges of RL (in the real world)

1. Problem formulation - capturing the problem in an MDP
   - ➡ State representation, Markov Property (e.g. non stationarity)
   - ➡ Reward engineering
   - ➡ ...

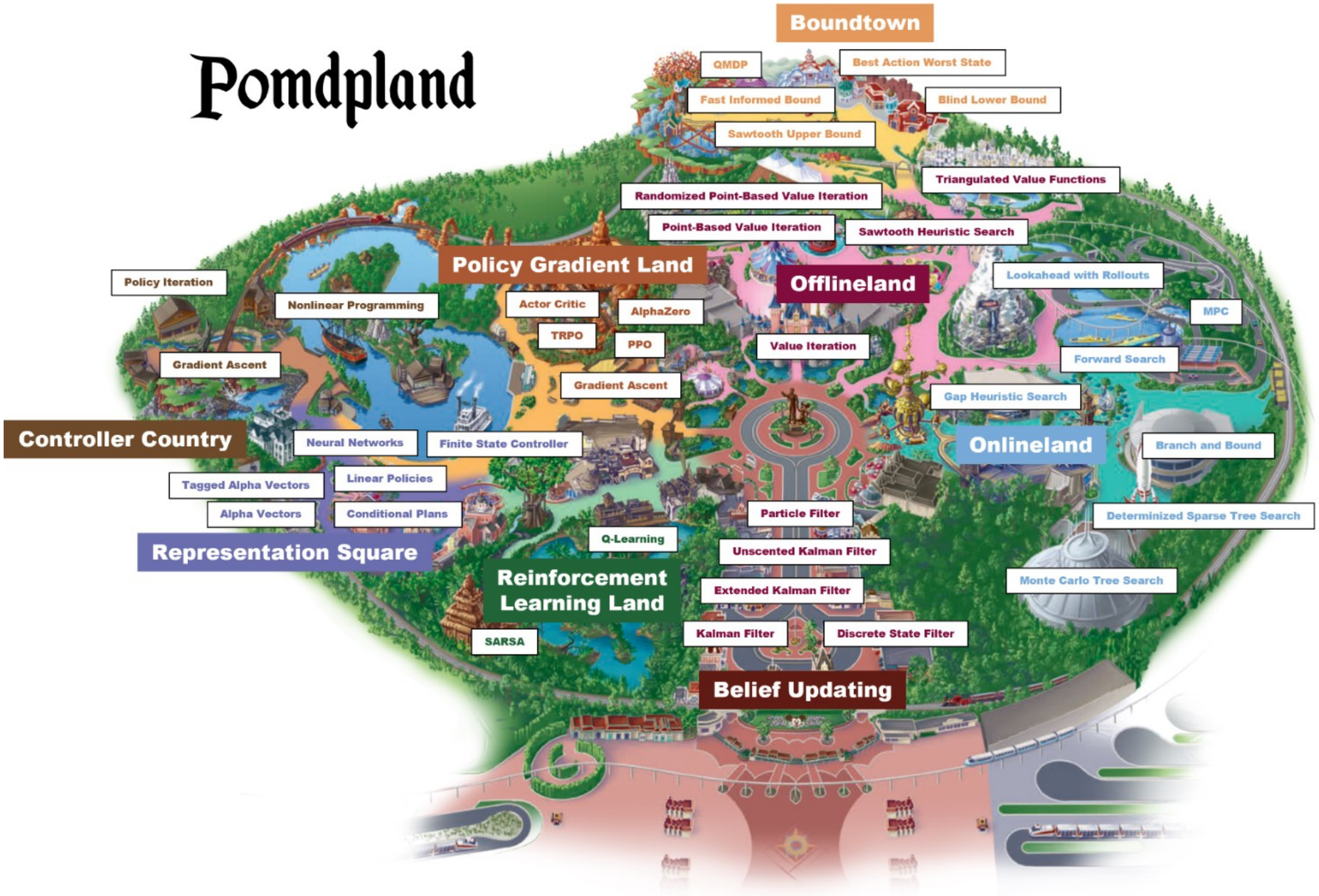
Markov Decision Process (MDP)

- Definition an MDP is a tuple $(\mathbb{S}, \mathbb{A}, P, R, \gamma, \rho)$:
  - $\mathbb{S}$ is a set of states ($s \in \mathbb{S}$)
  - $\mathbb{A}$ is a set of actions ($a \in \mathbb{A}$)
  - P is a dynamic model (transition kernel), defined for every action $P(s_{t+1} = s' \,|\, s_t = s, a_t = a)$ - in our case it is linear in state and action
  - R is a reward function $R(s_t = s, a_t = a) = \mathbb{E}[r_t \,|\, s_t = s, a_t = a]$
  - Discount factor $\gamma \in [0,1]$

- Definition partially observable MDP (POMDP) is a tuple $\mathcal{M} = (\mathbb{S}, \mathbb{A}, \mathbb{O}, P, R, E, \gamma, \rho)$:
  - $\mathbb{O}$ is the set of observations ($o \in \mathbb{O}$)
  - E is an emission function defining the distribution $E(o_t \,|\, s_t)$



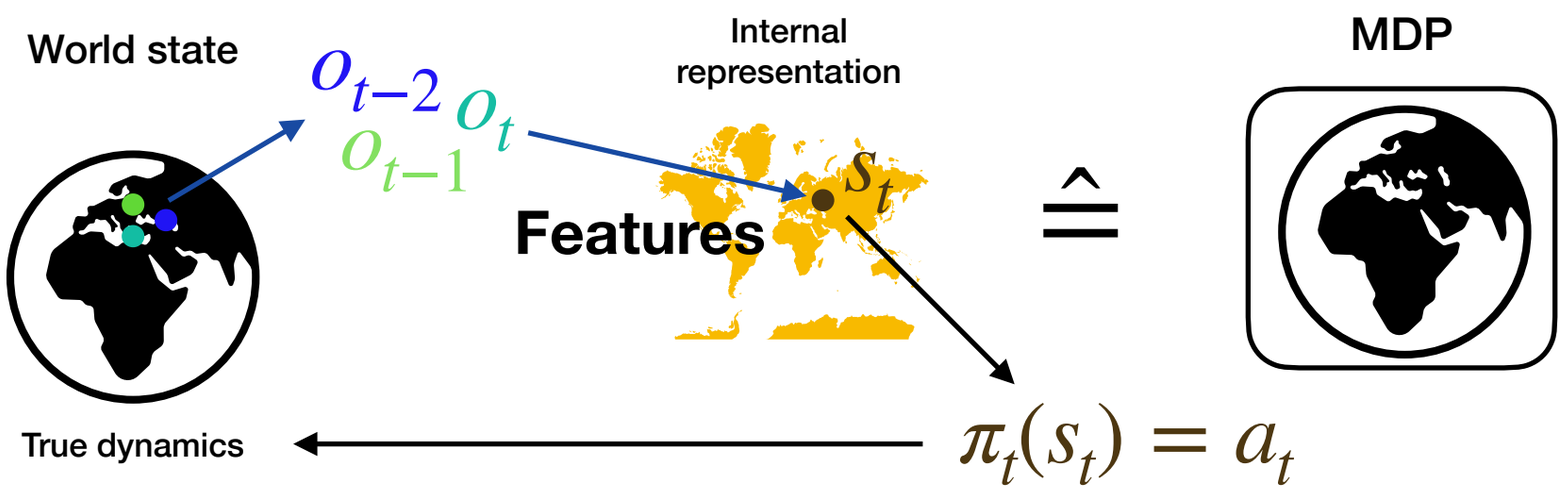
Wellcome to POMDPs



From Mykel Kochenderfer

Problem design - capture the right thing

- Solve an SDM problem: Information→Decision→Information→Decision→…
- Generally stochastic!
- Consequently we build a feedback system not planing too far in the future:
  - Define a **<u>state</u>** $s_t = h_t(o_t, a_{t-1}, o_{t-1}, a_{t-2}, o_{t-2} \dots)$, as a function holding **<u>sufficient statistics</u>** until time step t for a decision
  - Decision based on s_t via: $a_t = \pi_t(s_t)$ - the policy - optimise an expected aggregate of future rewards



- Rarely the observation o is the state s, the world state is, but often we assume it is certainty equivalence!
- POMDP ⇒ MDPs!

POMDPs and non stationarity

- To find a proper state we have to solve the additional prediction problem
  $s_t = h_t(o_t, a_{t-1}, o_{t-1}, a_{t-2}, o_{t-2} \dots)$
- In the non-stationary, finite horizon formulation the MDP has the form $(S, A, \{P\}_h, \{r\}_h, H, \rho_0) \Rightarrow$ Value-functions $Q_h(s, a)$ get time depended
  $\Rightarrow$ similar form of Bellman equations
- We can incorporate time into state e.g. $\tilde{s} = (s, h) \Rightarrow$ standard MDP
- Generally Bellman equation nice in discounted, stationary formulation $\Rightarrow$ this is what we usually see and most libraries build on this formulation

How bad is it?

- Linear POMDP: believe state - $O_t = h_t(S_t, A_t, W_t)$
  - ➡ **Static output feedback is NP hard** (linear in O_t and dynamics)
  - ➡ General **POMDPs** are **PSPACE hard**
- There are ways out - separation principle:
  - ➡ Filtering $\hat{s}_t = f(\{o_t\})$ - prediction problem
  - ➡ Action based on certainty equivalence
  - ➡ Optimal filtering - if dynamics are linear and noise is Gaussian - Kalman filtering - general belief propagation - LQG
  - ➡ Kalman filtered state - optimal in estimation and control
  - ➡ Estimate state with prediction $S_t = h(\tau_t)$, τ_t are time lags

Challenges of RL (in the real world)

2. RL - core issues:

- ➡ **Sample efficiency**
- ➡ **Stability**
- ➡ **Run time**
- ➡ **Hyper-parameter tuning**
- ➡ **Exploration**
- ➡ **Safety**
- ➡ **Robustness to Changes, Generalisation**
- ➡ **…**

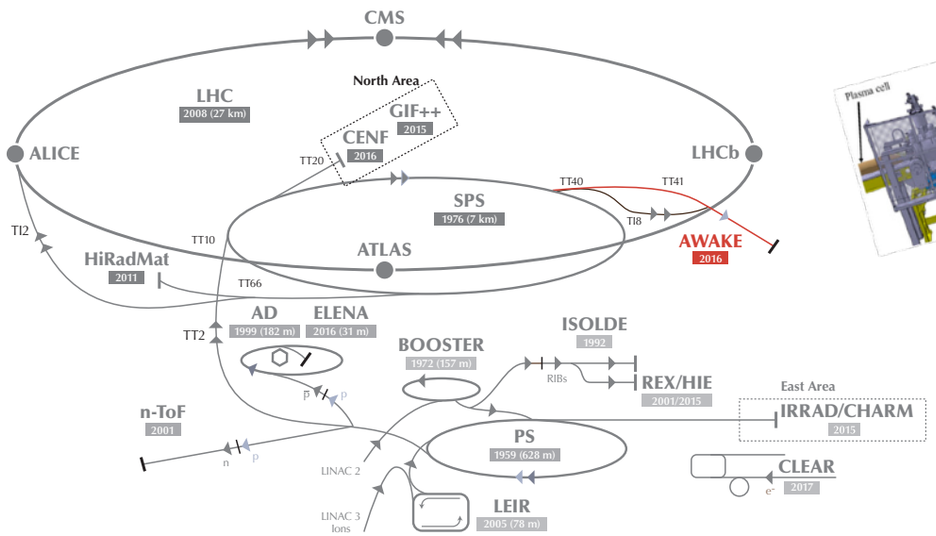
RL - core issues

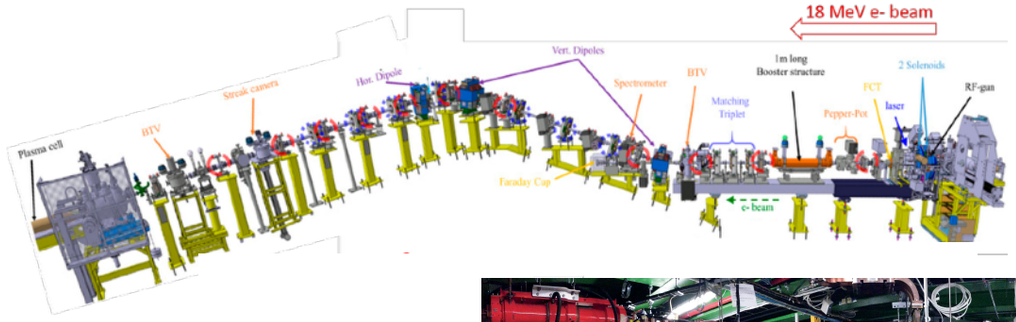
- Sample efficiency
- Stability
- Run time
- Hyper-parameter tuning
- Exploration
- Safety
- Robustness to Changes
- Generalisation
- …

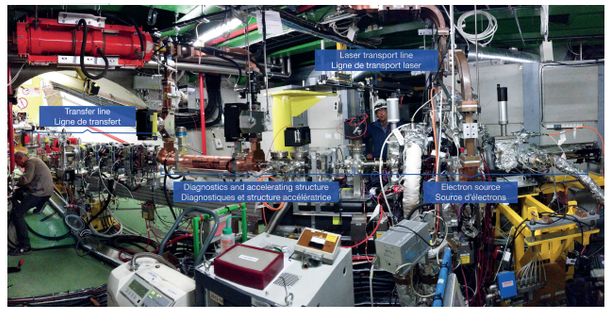
Possible solutions RL - core issues:

- Use a high fidelity model
  - ➡ Usually unavailable…
- Development of faster algorithms
  - ➡ SAC (very efficient maximum entropy algorithm)
  - ➡ TD3 (simple and efficient tricks accelerate DDPG-Style algorithms)
- Use data efficient method as MBRL
  - ➡ Hard to handle, low computational efficiency
- Reuse von prior knowledge to accelerate RL
  - ➡ Meta reinforcement learning

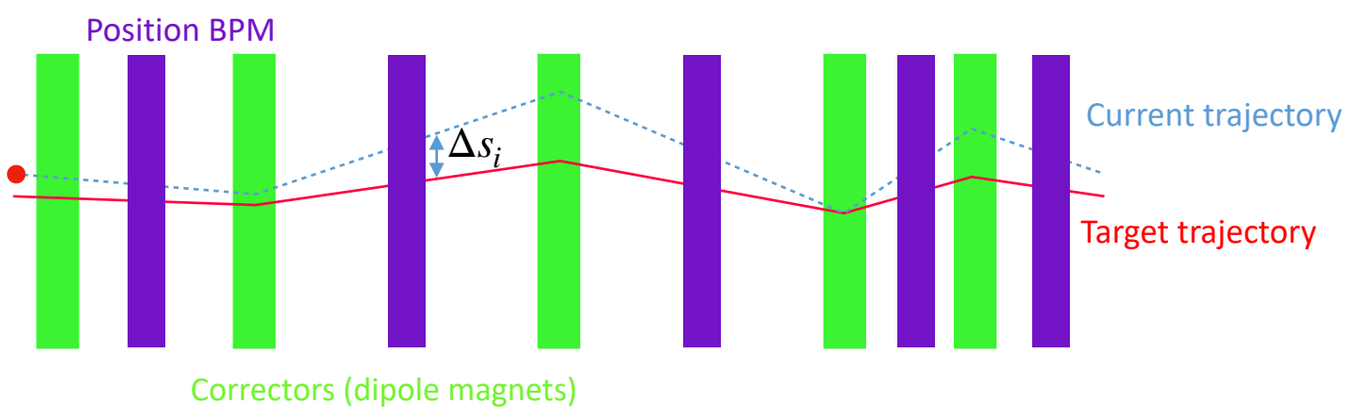
CERN AWAKE steering problem







- AWAKE electrons - start 5 MV (RF gun), accelerated to 18 MeV transported through beam line of 12 m to the AWAKE plasma cell.
- Vertical 1 m step and a 60° bend bring electron beam parallel SPS proton beam shortly plasma cell.
- The trajectory is controlled with 10 horizontal and 10 vertical steering dipoles according to the measurements of 10 beam position monitors (BPMs).



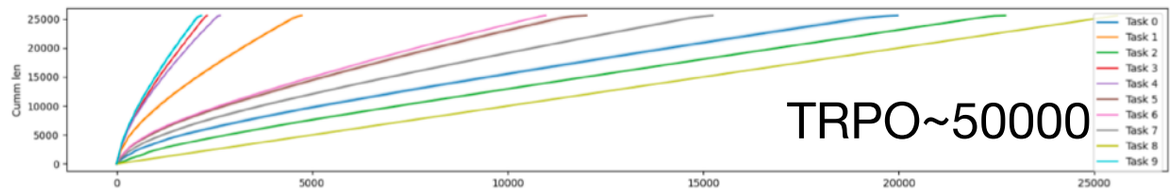
Sample efficiency

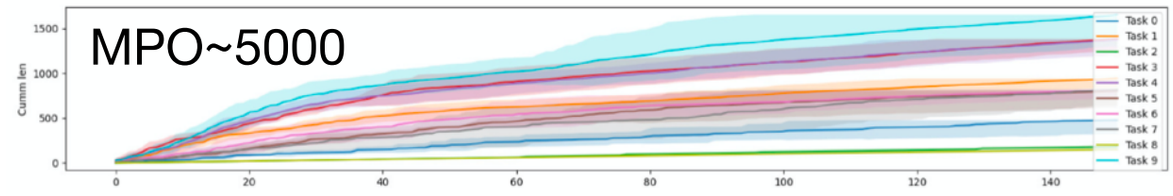
> 10e6 interactions

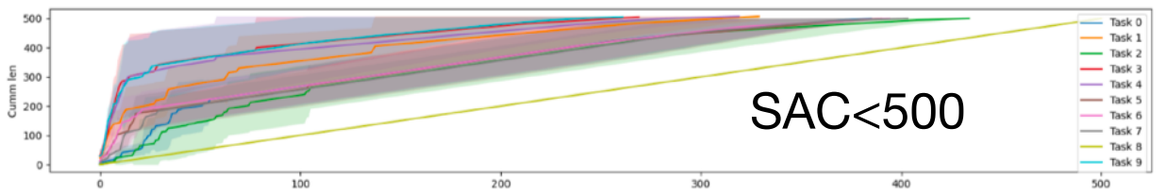
- Derivative free methods: (NES, CMA,..)
- 10 x Online methods (A3C)
- 10 x Batch policy-gradient methods (TRPO)
- 10 x Replay-Buffer + Value function estimation (Q-Learning, DDPG, TD3, NAF, SAC,…)
- 10 x Model-based RL methods (MPO, Guided Policy Search, Dyna)
- 10 x Model-based shallow methods (no NNs) Few shot GPs...

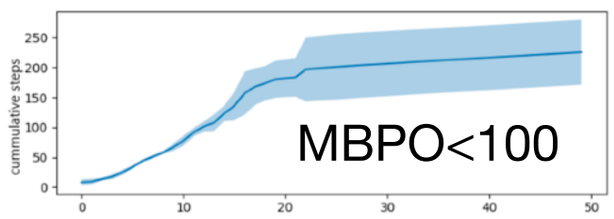
< 100 interactions

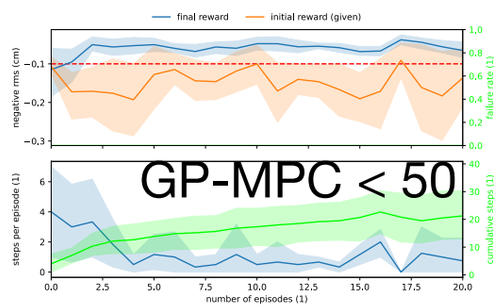
Colours are different tasks (optics)



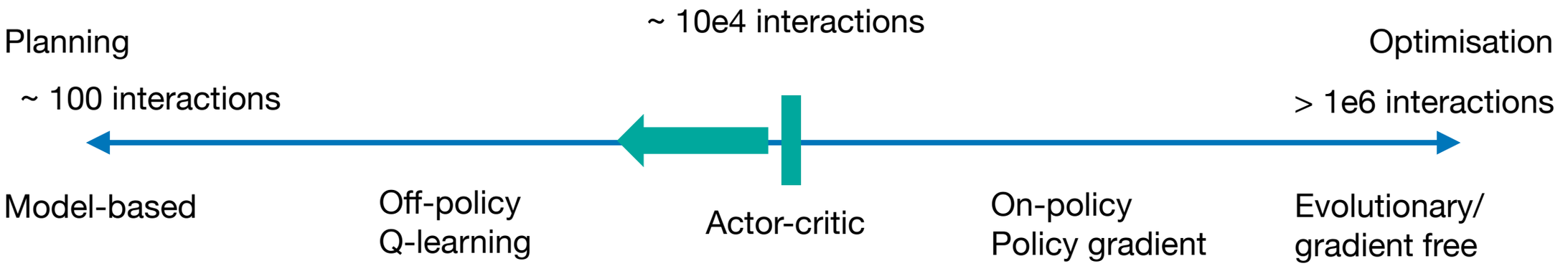






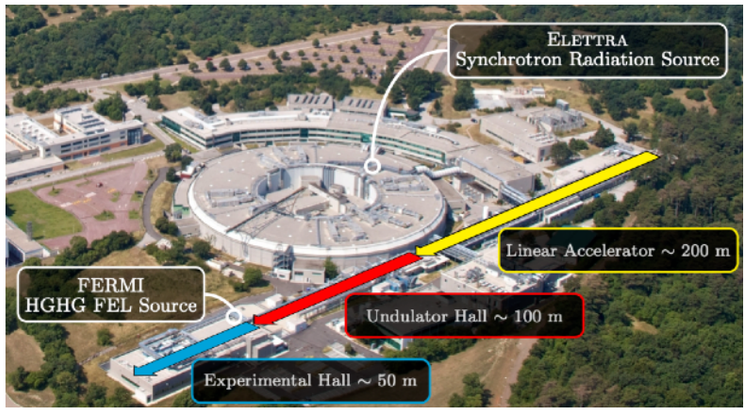


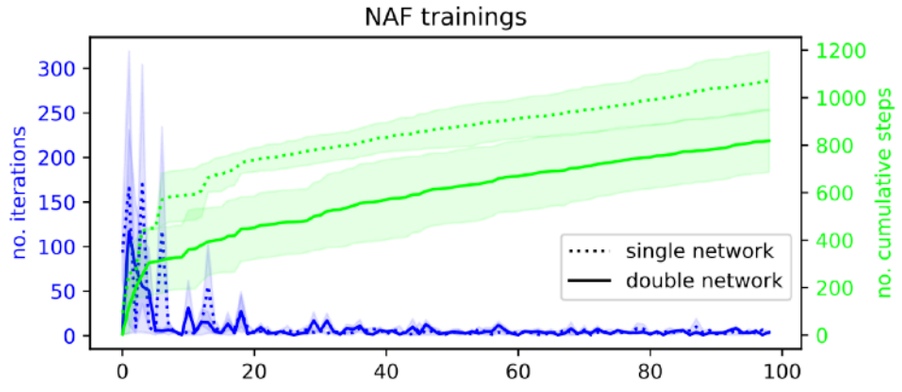
But sample efficiency is not all



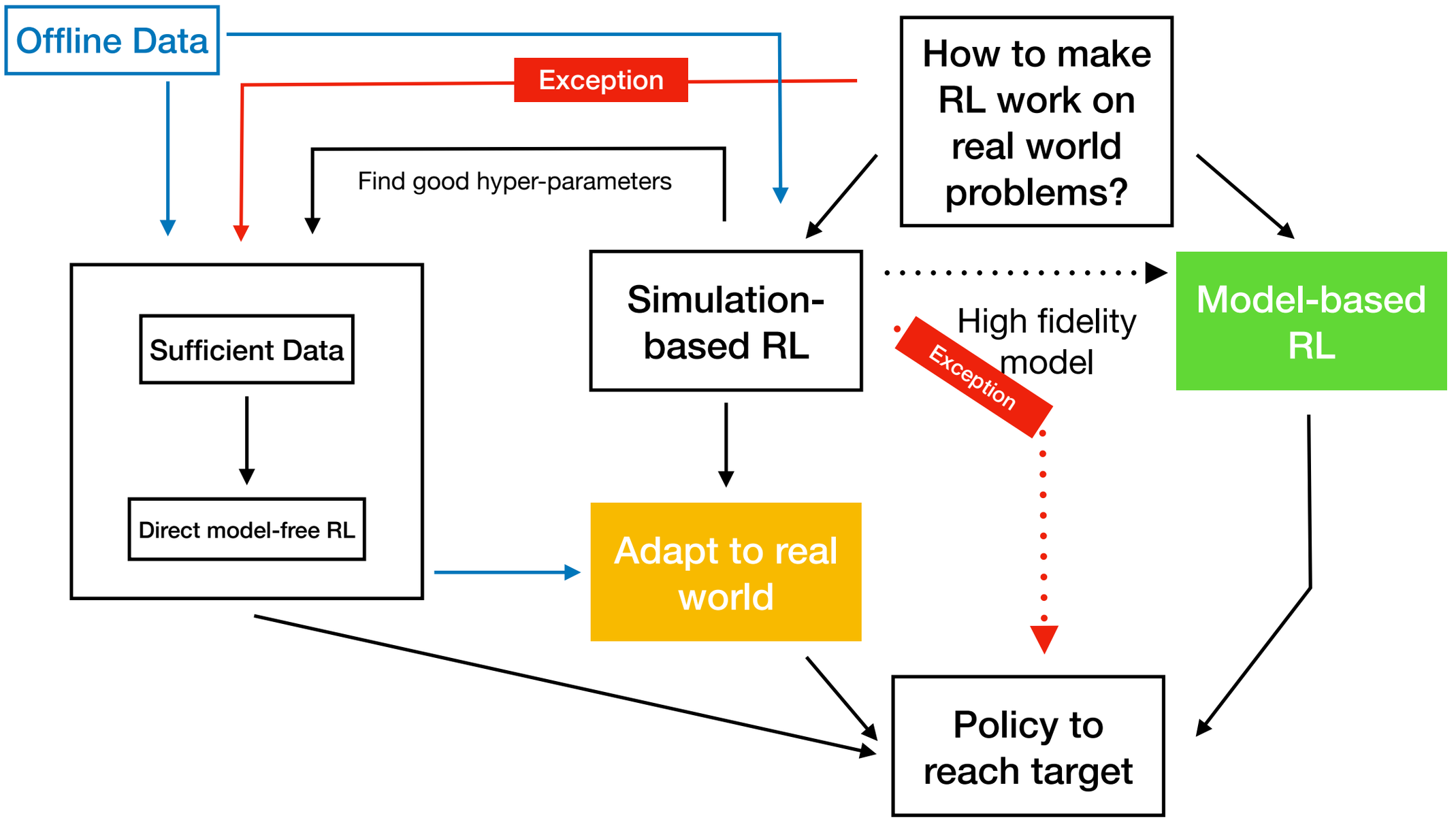
Hyper-parameters

- You can't do hyper-parameter sweeps in real applications
  - ➡ How representative is a simulator? (Usually not very)
- Sample complexity = time to execute algorithm times number of iterations for sweeps
  - ➡ Stochastic search and gradient-based optimisation
- Can we develop more stable algorithms that are not so sensitive to the hyper-parameters?





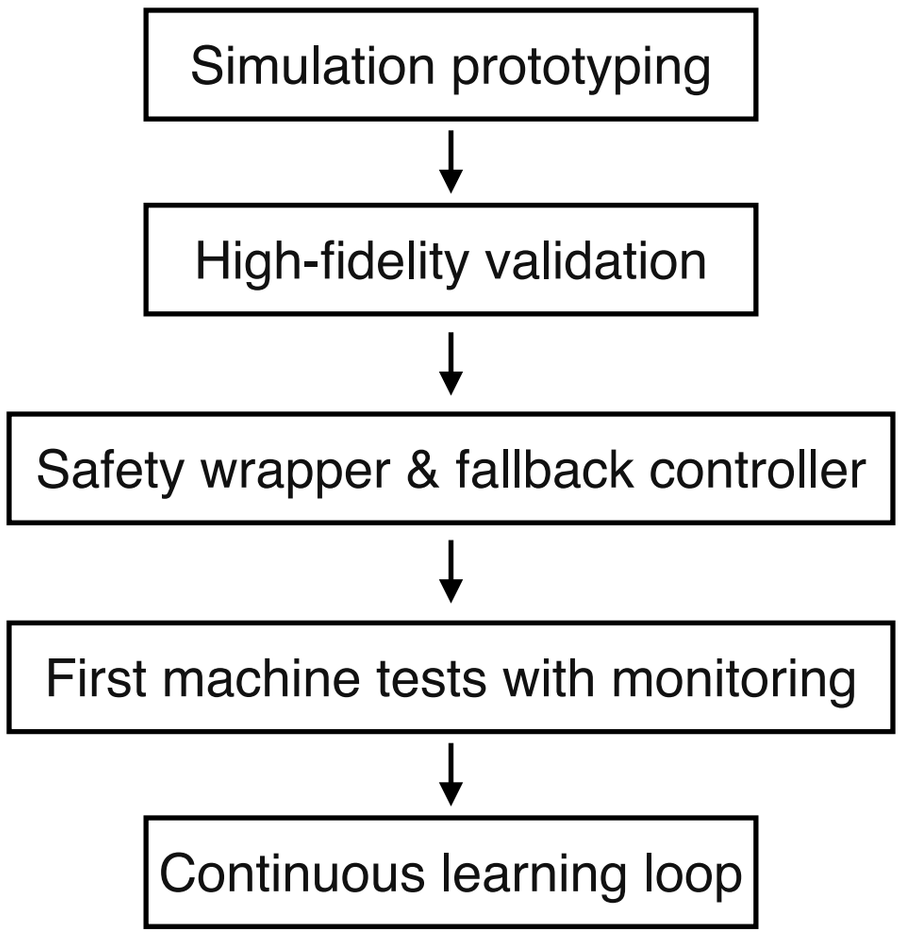
Common real world scenarios



Challenges in Technology Transfer

- Explainability, reliability, and implementation
- Scaling from research prototypes to robust systems
- Developing models
- When not to use RL: task complexity, interpretability, data

From Simulation to Deployment



Deployment requires process, not just algorithms.

Summary & Takeaways

Summary & Takeaways

- RL Landscape
  - ➡ Value-based, policy-based, and actor-critic methods
  - ➡ Policy gradients: powerful but inefficient & noisy
- Stabilising Policy Gradients - PPO
  - ➡ Step-size sensitivity → risk of collapse
  - ➡ Surrogate objectives & KL control (TRPO, PPO)
- From Theory to Practice
  - ➡ Real-world ≠ clean MDPs → often POMDPs
  - ➡ Approximation, filtering, robustness required
  - ➡ Safety wrappers & fallback controllers essential
- Core Challenges
  - ➡ Sample efficiency
  - ➡ Stability & runtime
  - ➡ Hyperparameter tuning
  - ➡ Exploration vs. safety
- Takeaway:
  - ➡ Progress in RL needs theoretical guarantees + practical engineering to succeed in real-world deployment.