UNIVERSITY OF LIVERPOOL

INTRODUCTION TO REINFORCEMENT LEARNING II

Dr. Andrea Santamaria Garcia
Lecturer

CI lectures CI-ACC-226
Lecture on 02/03/2026

OUTLINE

- **Sampling and approximation methods** (9 slides)
  - *Monte Carlo learning, temporal difference learning, off-policy learning*

- **Deep reinforcement learning** (3 slides)

- **Reinforcement learning for optimisation** (8 slides)
  - *Sequential decision making, when to use BO?, when to use RL?, RL as a learned optimiser, automatic beam steering and focusing*

- **Challenges in RL for particle accelerators** (11 slides)
  - *Sample efficiency, partial observability, safety, real-time inference*

SAMPLING AND APPROXIMATION METHODS

The expanded Bellman equation

How to solve it?

$$\boxed{\mathcal{V}^{\pi}(s) = \sum_{a \in \mathcal{A}} \pi\,(a|s) \sum_{s\prime \in \mathcal{S}} \mathcal{P}^{a}_{s,s\prime}\left(\mathcal{R}^{a}_{s} + \gamma \mathcal{V}^{\pi}(s')\right)}$$

Approximation ❓
Temporal-difference learning

Sampling ❓
Monte Carlo methods

Iteration ✅
Dynamic programming

Directly ✅
System of $\mathcal{S}$ simultaneous linear equations with $\mathcal{S}$ unknowns

Computational complexity →

Transitioning to “modern” RL

Ideal setting

State fully observable

- MDP (finite, discrete)
- Model known and tractable
- Value function computable
- Optimal policy computable

vs

Real world

State partially observable

- POMDP
- Model unknown or learned
- Value function approximated
- Policy approximated $\pi \approx \pi^*$

Classical dynamic programming

- The Bellman operator is evaluated exactly using the known transition model
- Expectations are computed analytically from $\mathcal{P}(s|s,a)$
- Value functions are obtained via deterministic fixed-point iteration
- Convergence is algebraic and guaranteed under contraction of the Bellman operator

Modern RL (model free!)

- The Bellman operator is approximated via stochastic samples
- Expectations are replaced by sample-based estimators
- Value functions are learned through stochastic approximation
- Convergence is statistical and depends on sampling, noise, and step sizes

Monte Carlo learning

- We have access to a black box model that we query sequentially (simulation or real-world)
- We get samples of trajectories
- We don't know $\mathcal{P}$

The experience is organised in episodes:

$s_0, a_0, r_1, s_1, a_1, r_2, \dots, s_{T-1}, a_{T-1}, r_T$

Episode 1

$\vdots$

$s_0, a_0, r_1, s_1, a_1, r_2, \dots, s_{T-1}, a_{T-1}, r_T$

Under policy π

Episode M

Value estimation $\mathcal{V}^{\pi}(s)$

Every visit MC

- Loop through each episode $t = T, T-1, \dots, 0$ to see when each state s was visited
- Each time a particular s is visited update the return

$$\mathcal{G} \leftarrow \gamma \mathcal{G} + \mathcal{R}_{t+1}$$

- Average the returns to estimate $\mathcal{V}^{\pi}(s)$:

$$\boxed{\mathcal{V}(s) \approx \frac{1}{N(s)} \sum_{i=1}^{N(s)} \mathcal{G}_t^{(i)}}$$

N = # of times s was visited across episodes

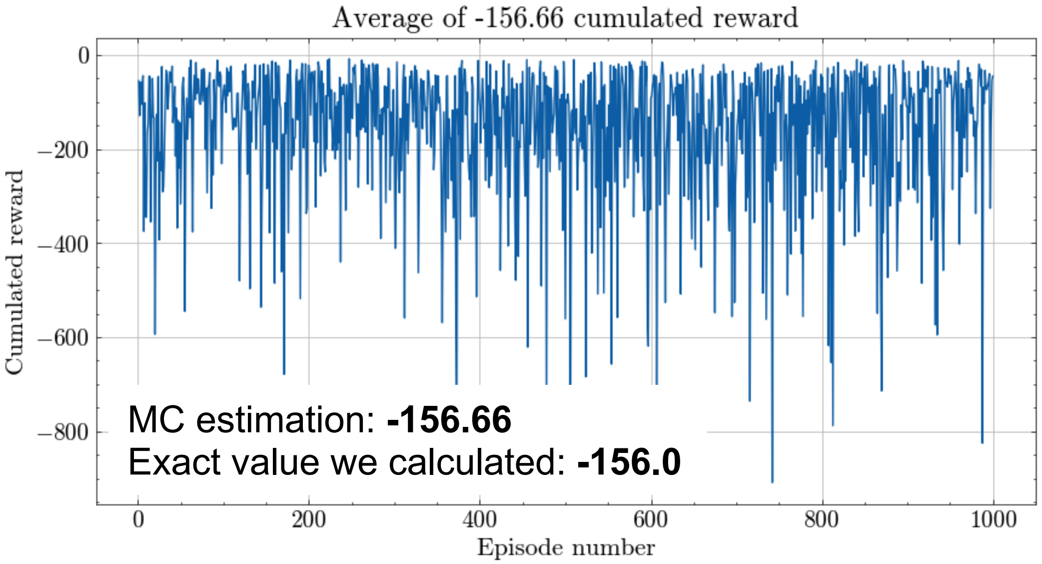
- The first-visit MC variant averages returns only from the first occurrence of each state per episode
- As the number of visits tends to infinity, the **estimator converges** by the law of large numbers:

$$\lim_{t \to \infty} \mathcal{V}_t(s) = v^{\pi}(s)$$

Monte Carlo estimates the expectation in the Bellman equation by empirical averages of full returns

Monte Carlo learning

Let's try this by considering 1000 episodes from our gridworld example for state $s = 0$ (initial state) and a random policy:



Can we update before the episode ends?

😏 *well, yes, we can*

- Conceptually simple: **direct empirical estimation of expected return**
- No model $\mathcal{P}$ required: **uses only observed trajectories**
- **Unbiased estimator** of $v^{\pi}(s)$
- Foundation for many **modern policy gradient methods**

- Requires **full episodes before updating** (slow learning, expensive simulation or experiment)
- **High variance**: returns depend on long stochastic trajectories
- **Sample inefficient**: many visits needed for accurate estimates
- **No bootstrapping**: slow propagation of value information

Temporal difference learning

How to compute the averages of action-value methods with **constant memory** and **constant computation step**, i.e., without storing and averaging a lot of data in tables?

Dynamic programming *exact expectation*

- **Memory**: must store full transition model $\mathcal{P}(s|s,a)$ and value function over all states
- **Computation per update**: requires summing over all possible next states (full expectation)

Monte Carlo *empirical full return*

- **Memory**: must store complete trajectories (or future rewards) until episode termination
- **Computation per update**: computes full returns by summing over the entire remaining trajectory

Temporal difference *stochastic fixed-point bootstrapping*

- **Memory**: stores only current value estimates, no model and no trajectory buffering required
- **Computation per update**: constant per step, uses a single transition (s, r, s')

- TD replaces exact expectation with a **stochastic one-step update** using the current estimate itself (bootstrapping)
- It trades exactness for incremental, constant-memory updates that converge to the Bellman fixed point

Bootstrapping means updating an estimate using another estimate in place of an exact quantity

Temporal difference learning

Bellman equation: $\mathcal{V}^{\pi}(s) = \mathbb{E}[r + \gamma \mathcal{V}^{\pi}(s')] = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} \mathcal{P}^{a}_{s,s'} \left(\mathcal{R}^{a}_{s} + \gamma \mathcal{V}^{\pi}(s')\right)$

Full expectation, sum over all states and actions required

TD update:

$$\mathcal{V}(s) \leftarrow \mathcal{V}(s) + \alpha[\underbrace{r' + \gamma \mathcal{V}(s')}_{\text{Target}} - \mathcal{V}(s)]$$

Target
No expectation, one sample only

New estimate $\leftarrow$ Old estimate + Step size $\underbrace{\text{(Target - Old estimate)}}_{\text{Temporal difference error}}$

Temporal difference error

The target itself depends on the current value estimate (bootstrapping)

- **Online updates:** learns after each transition, no need to wait for episode termination
- **Constant memory and computation per step:** uses only (s, r, s')
- **Lower variance than Monte Carlo:** bootstrapping reduces trajectory-level noise
- **Scales better to continuing tasks:** naturally handles infinite-horizon settings.

- **Bootstrapping introduces bias:** the target depends on current estimates
- **Convergence requires conditions:** appropriate step sizes and sufficient exploration
- **Requires explicit representation of each state,** impractical for very large or continuous state spaces

Off-policy learning

Exploration vs exploitation dilemma appears again

We want to learn the optimal policy (behaviour) and for that we need to behave non-optimally to explore all state-action pairs

Off-policy learning decouples acting from learning:

🤖 **Behaviour policy $\boldsymbol{b(a|s)}$**
Policy used to generate data

Exploration
ε-greedy, soft policy

🎯 **Target policy $\boldsymbol{\pi(a|s)}$**
Policy being evaluated $\pi \approx \pi^*$

Exploitation
greedy or near-greedy

Example of off-policy TD control: Q-learning learns a greedy target policy while behaving ε-greedily

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \operatorname*{argmax}_a Q(s',a) - Q(s,a)]$$

- **Behaviour**: take exploratory action from behaviour policy $a \sim b(\cdot\,|s)$ and generate transition (s, a, r', s')
- **Update target**: Use target policy to define the bootstrap target $\pi(s) = \operatorname*{argmax}_a Q(s',a)$ *Look at all possible actions in s' and take the largest stored Q-value*

Summary

Tabular solution methods for finite MDPs

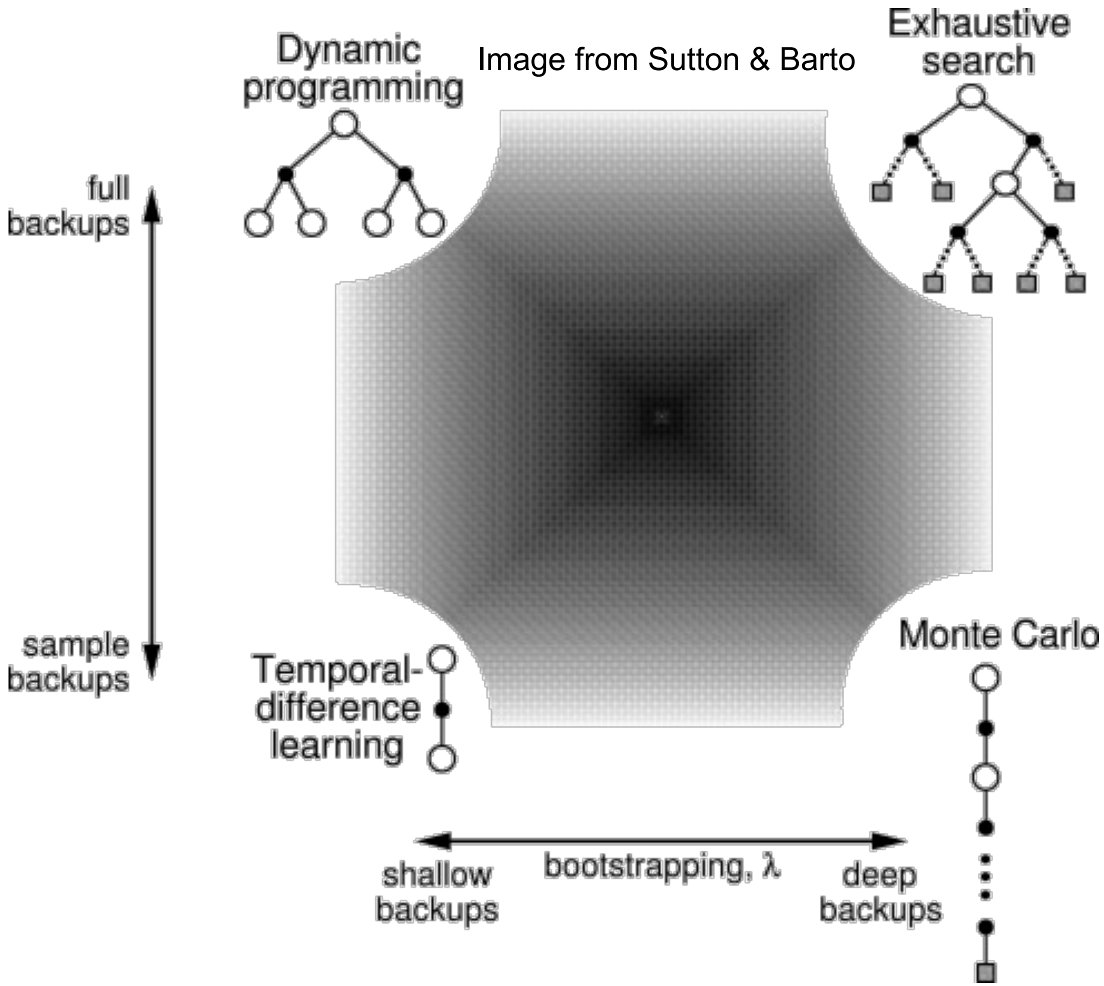
Methods	Techniques	Model-based	Bootstrapping	Algorithms
Dynamic programming	Iterative	Yes	Yes	Policy evaluation Policy iteration Value iteration
Monte Carlo	Sampling (episode-based estimation)	No	No	First-visit MC Every-visit MC
Temporal difference	Approximation (sampling + approximation)	No	Yes	TD(0) Q-learning SARSA

Model-based = we know the transition dynamics $\mathcal{P}$ of the problem

Summary

Tabular solution methods for finite discrete MDPs

Image from Sutton & Barto



$\mathcal{V}$ or $\mathcal{Q}$ and π are stored as arrays

- What happens to infinite or continuous MDPs? $\mathcal{V}(s)$?
- Can we identify and enumerate all states and actions?

Model-free deep RL

Function approximation of $\mathcal{V}$ / $\mathcal{Q}$ and π

→ Opens the door to high dimensional continuous problems (tractable)
→ Can learn abstract features
→ Introduces bias, variance, and stability challenges
→ Fewer convergence guarantees

The function we learn can generalise to states never seen before

→ Parameters θ are shared over all states
→ Generalisation only as good as data

DEEP REINFORCEMENT LEARNING

Deep reinforcement learning

Value-based

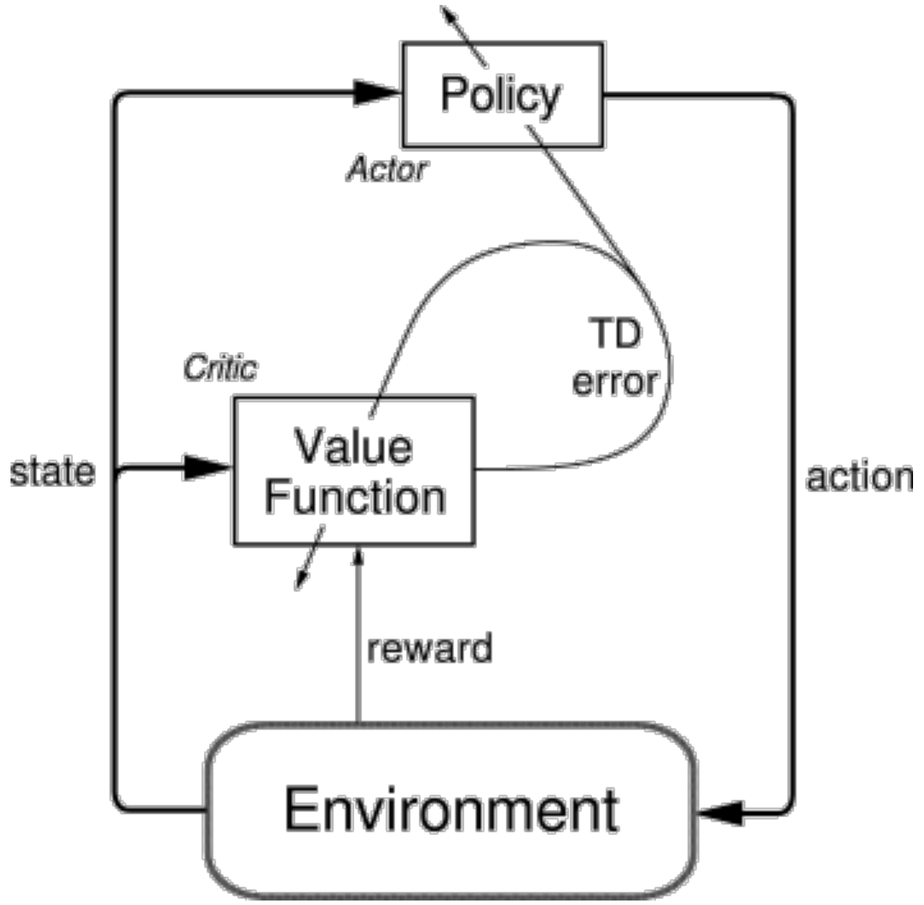
- Learn a value function $V(s)$ or $Q(s, a)$ with an NN
- Policy is implicit: choose action with highest Q-value
- Action selection via $\text{argmax}_a Q(s, a)$
- Exploration added separately
- Well suited to discrete action spaces

Policy-based methods

- Learn the policy $\pi_\theta(a|s)$ directly as an NN
- Optimise expected return via gradient ascent
- Naturally handle stochastic and continuous actions
- Higher variance in basic form

Actor-critic methods

- **Actor**: learns policy $\pi_\theta(a|s)$
- **Critic**: learns $V(s)$, $Q(s, a)$, or advantage $A(s, a)$
- Critic trained with TD
- Actor updated using critic's estimate



Deep reinforcement learning

Policy gradient

Policies are parametrised with parameters θ and the goal is always to maximise the cumulated expected reward

$$\max_{\theta} J(\pi_\theta) = \mathbb{E}_{\pi_\theta}[\mathcal{G}_t]$$

If the policy is parametrised with a neural network that we can optimise with gradient descent:

$$\theta \leftarrow \theta + \alpha \nabla J(\pi_\theta)|_\theta$$

How to calculate $\nabla J(\pi_\theta)$
→ Policy gradient theorem

- Poor sample efficiency (needs many interactions).
- Sensitive to learning rate α and initialisation parameters.
- In its basic form has high variance due to MC return estimations.
- Used in REINFORCE, A2C, A3C, TRPO, PPO, SAC.

Deep reinforcement learning

Common model-free algorithms

	Description	Policy	Action space	State space	Operator
DQN	Deep Q Network	Off-policy	Discrete	Continuous	Q-value
DDPG	Deep Deterministic Policy Gradient	Off-policy	Continuous	Continuous	Q-value
A3C	Asynchronous Advantage Actor-Critic Algorithm	On-policy	Continuous	Continuous	Advantage
TRPO	Trust Region Policy Optimization	On-policy	Continuous	Continuous	Advantage
PPO	Proximal Policy Optimization	On-policy	Continuous	Continuous	Advantage
TD3	Twin Delayed Deep Deterministic Policy Gradient	Off-policy	Continuous	Continuous	Q-value
SAC	Soft Actor Critic	Off-policy	Continuous	Continuous	Advantage

REINFORCEMENT LEARNING FOR OPTIMISATION

Sequential decision making

Two different abstractions: BO and RL

What question we are asking and **what information is retained vs discarded** when making decisions?

What parameters maximise a performance metric/function?

Bayesian optimisation

Sequential decision making over a static objective

- Decisions select what to evaluate next
- Assumes the system can be represented as a static input-output map
- Retains information only at the level of (input, outcome) pairs
- Temporal structure is discarded
- Time is an index, not a state

input → system → outcome

Information is compressed into parameter-outcome pairs

Sequential decision making

Two different abstractions: BO and RL

What question we are asking and **what information is retained vs discarded** when making decisions?

What action should I take now given the state of the system?

Reinforcement learning

Sequential decision making over evolving system state

- Decisions select actions given the current state
- Assumes the system evolves over time under action
- Information includes state transitions
- Actions influence future system behaviour
- Time is part of the problem

state → action → reward → next state

Information is preserved at the level of state transitions

When to use Bayesian optimisation?

→ When the tuning problem can still be treated as a **static optimisation problem** and **data collection is costly**

e.g., clinical trials and dose finding, product formulation and design, general parameter tuning and calibration

In Bayesian optimisation:

- Each experimental run can be treated as an **independent evaluation**
- Control parameters **do not meaningfully alter future system dynamics,** or the system can be reset
- Evaluations are expensive enough that **sample efficiency dominates**

Limited when:

- Actions influence future system behaviour
- Performance depends on temporal structure or trajectories
- Delayed effects dominate single-run outcomes
- Non-stationarity is induced by the tuning process itself

When to use reinforcement learning?

→ When ignoring temporal structure throws away relevant information and learning-induced suboptimal behaviour is tolerable within a safety envelope

e.g., games, robotics, autonomous driving, adaptive trading strategies in finance

In reinforcement learning:

- Control actions **affect future system behaviour**
- System performance depends on **trajectories, not single outcomes**
- Relevant information is contained in **state transitions over time**
- Adaptation to **non-stationarity or disturbances** is required
- Policies must act in **real time** or **near real time** once learned

Limited when:

- Online/offline interaction is extremely costly or impossible
- Exploration-induced behaviour is unsafe or unacceptable
- Safety or stability guarantees must always hold

RL as a learned optimiser

- An RL policy is trained by interacted with a simulated environment
- The policy is deployed as a fast optimiser (mapping from observation to actions)
- Sequential structure is exploited during training, not deployment (fast inference)

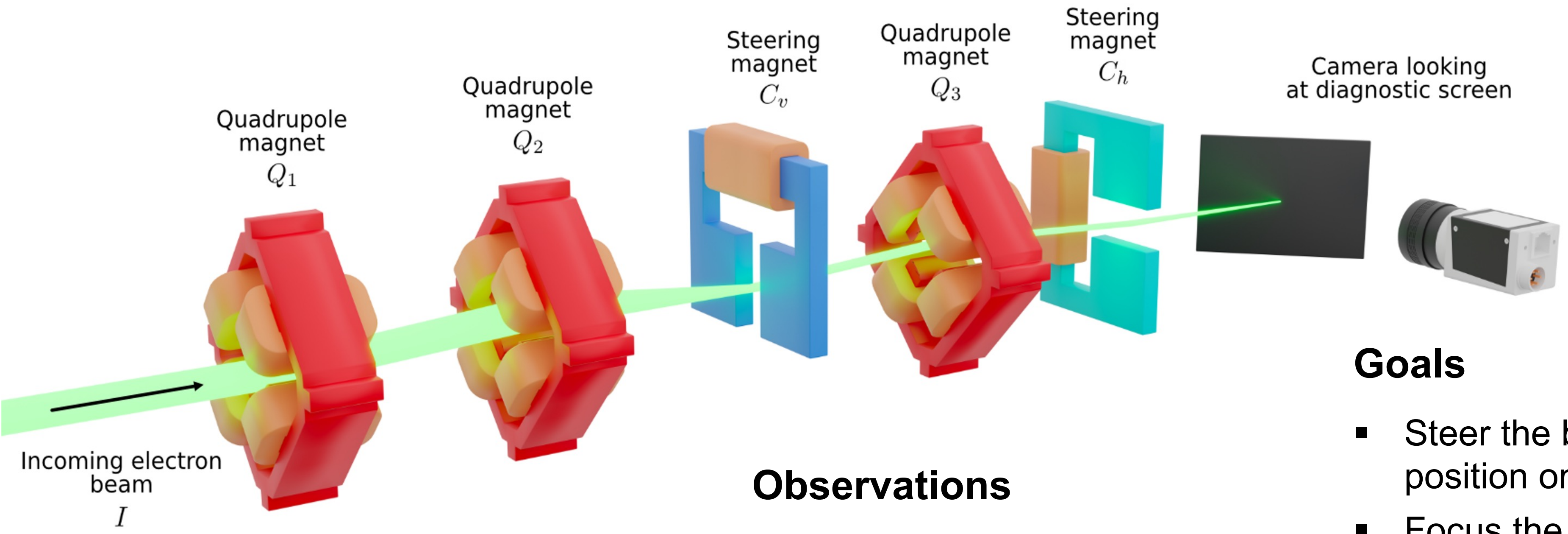
At runtime, the policy is **not acting as a controller**, but as a **learned optimiser**

Advantages
- **Faster convergence** than BO at deployment
- **Robustness** to slow and fast environment changes if trained across variations
- **Constant time inference**

Disadvantages
- Requires **many interactions** (10^3-10^6)
  → fast simulation
- Careful **problem formulation**
- Generally, **more engineering effort**

Automatic beam steering and focusing

BO vs RL at the ARES linear accelerator



Goals

- Steer the beam to a certain position on the screen
- Focus the beam to desired beam size

Observations

- Beam parameters
- Magnet settings

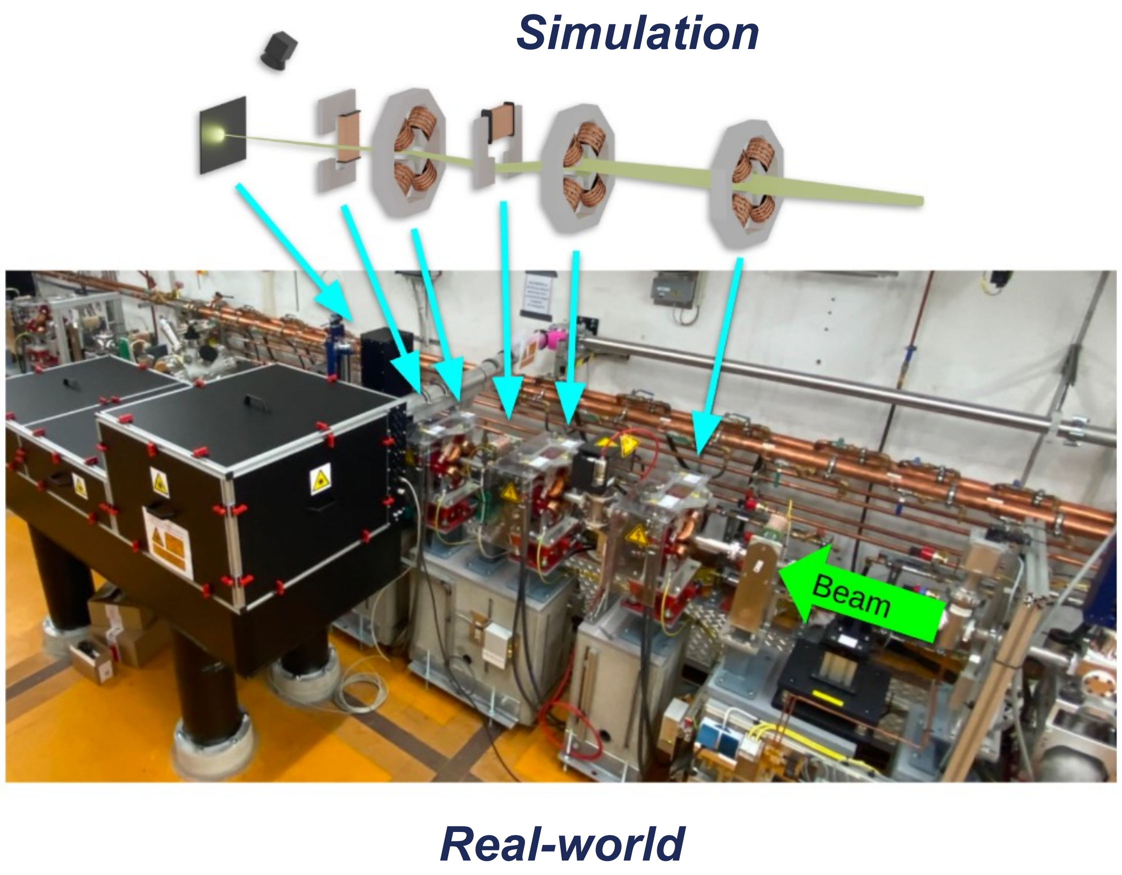
Actions

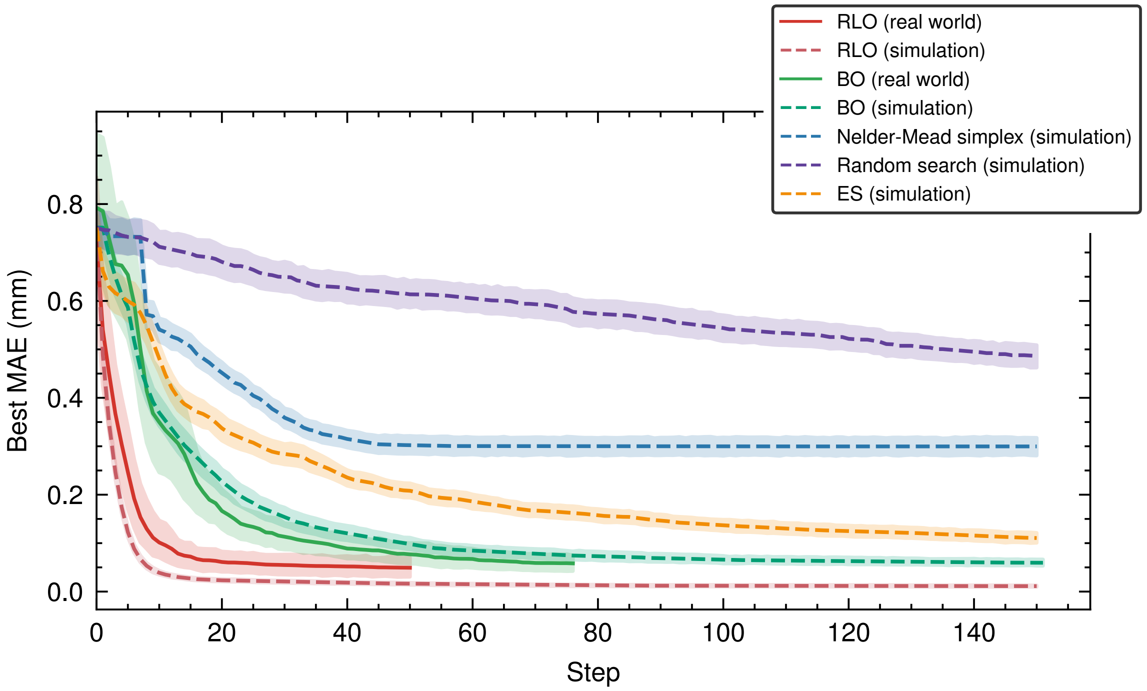
- Quadrupole magnet strengths
- Steerer magnet deflecting angle

Sci.Rep. 14 (2024) 1, 15733

Automatic beam steering and focusing

BO vs RL at the ARES linear accelerator

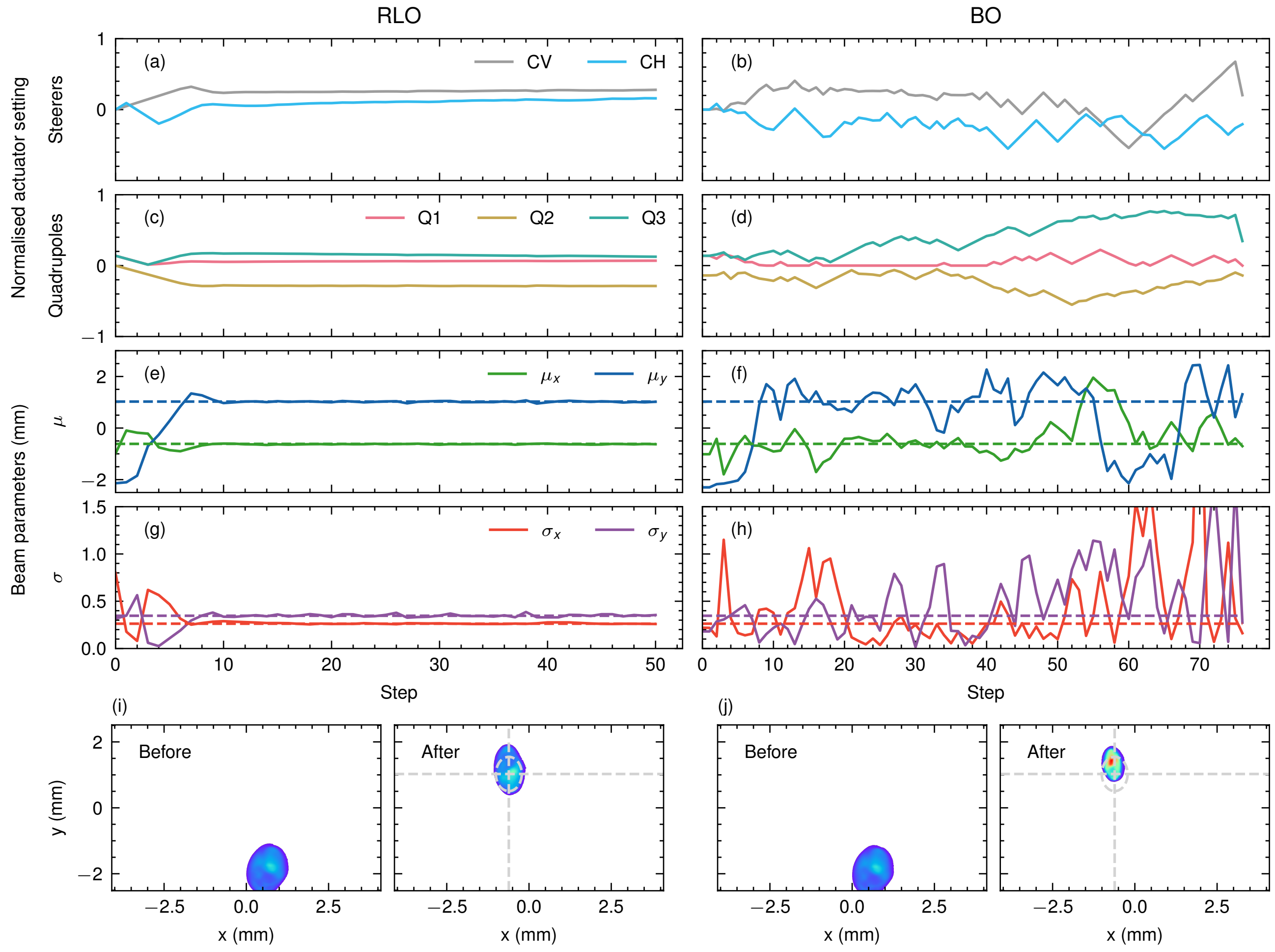


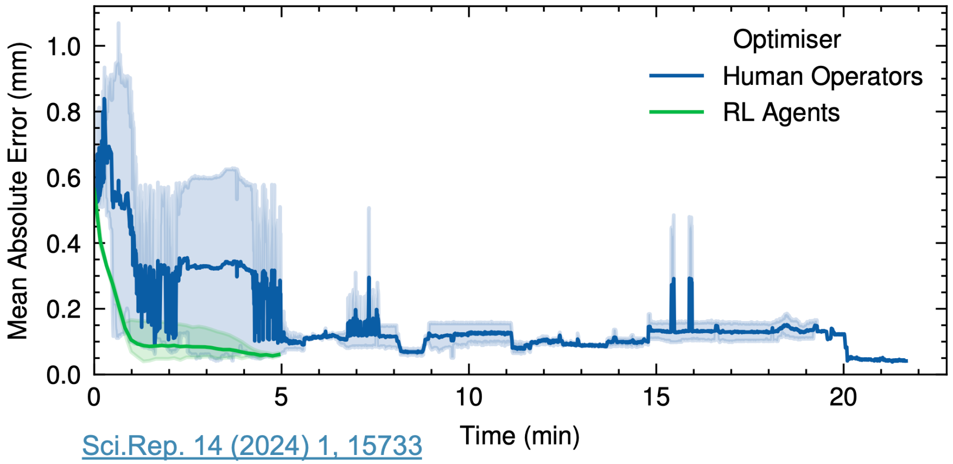


Sci.Rep. 14 (2024) 1, 15733

Automatic beam steering and focusing

BO vs RL at the ARES linear accelerator





[Sci.Rep. 14 (2024) 1, 15733]

CHALLENGES IN RL FOR PARTICLE ACCELERATORS

Main challenges of RL deployment

Policy and value functions are approximated by deep neural networks (DNNs)

$$\max_{\theta} J(\pi_\theta) = \max_{\theta} \mathbb{E}_{\pi_\theta}[\mathcal{G}_t] \qquad \theta \leftarrow \theta + \alpha \nabla J(\pi_\theta)|_\theta$$

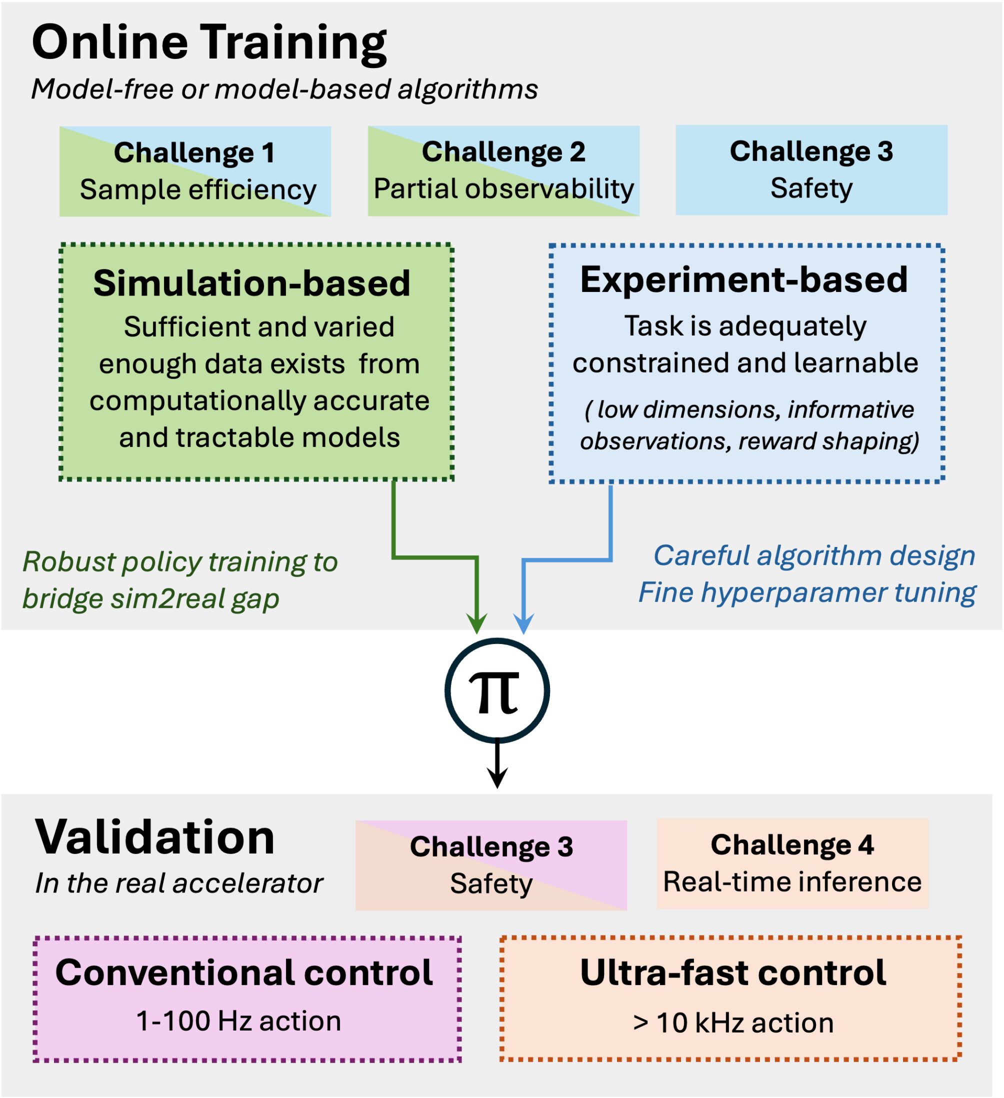
Generalisation capabilities

→ quantity and quality of data

No real **convergence** guarantees

Training instability due to:

- Bootstrapped value targets
- Function approximation bias (net. architecture, weight initialisation, training dynamics)
- Hyperparameter sensitivity (high variance in performance across random seeds)



Challenge 1: sample efficiency

Sample efficiency ↗
Training cost ↘

Sample efficiency: number of interactions with the environment required to achieve a certain level of performance during the decision-making process

	✅	❌
Model-free, on-policy *Policy gradient: REINFORCE* *Actor-critic: PPO, A2C*	▪ *Simple implementation* ▪ *Good for continuous action*	▪ *Poor sample efficiency* ▪ *Large variance if unclipped*
Model-free, off-policy, value based *DQN*	▪ *Sample efficient* ▪ *Efficient in discrete envs*	▪ *Unstable (function appr.)* ▪ *Limited to discrete or low-dimensions*
Model-free, off-policy, actor-critic *DDPG, TD3, SAC*	▪ *Sample efficient* ▪ *Good for continuous action* ▪ *Stable*	▪ *Hard to tune* ▪ *Hyperparameter sensitivity* ▪ *Overestimation bias*
Model-based RL	*Very high sample efficiency*	*Model is hard to train, complex to tune, brittle & sensitive*

?

Challenge 1: sample efficiency

How does this play in practice?

Idealised setting ---------------------------→ ***Noisy, unpredictable dynamics***

Training

Simulation-based

Sufficient and varied enough data exists from computationally accurate and tractable models

Diff. simulations (Cheetah), DNN surrogates, GPU-accelerated

$\sim 10^{3}$-10^{6} interactions

sim2real gap

π

Need ***robust*** policies!

Validation

In the real accelerator

Domain randomization: train on "perturbed" environments

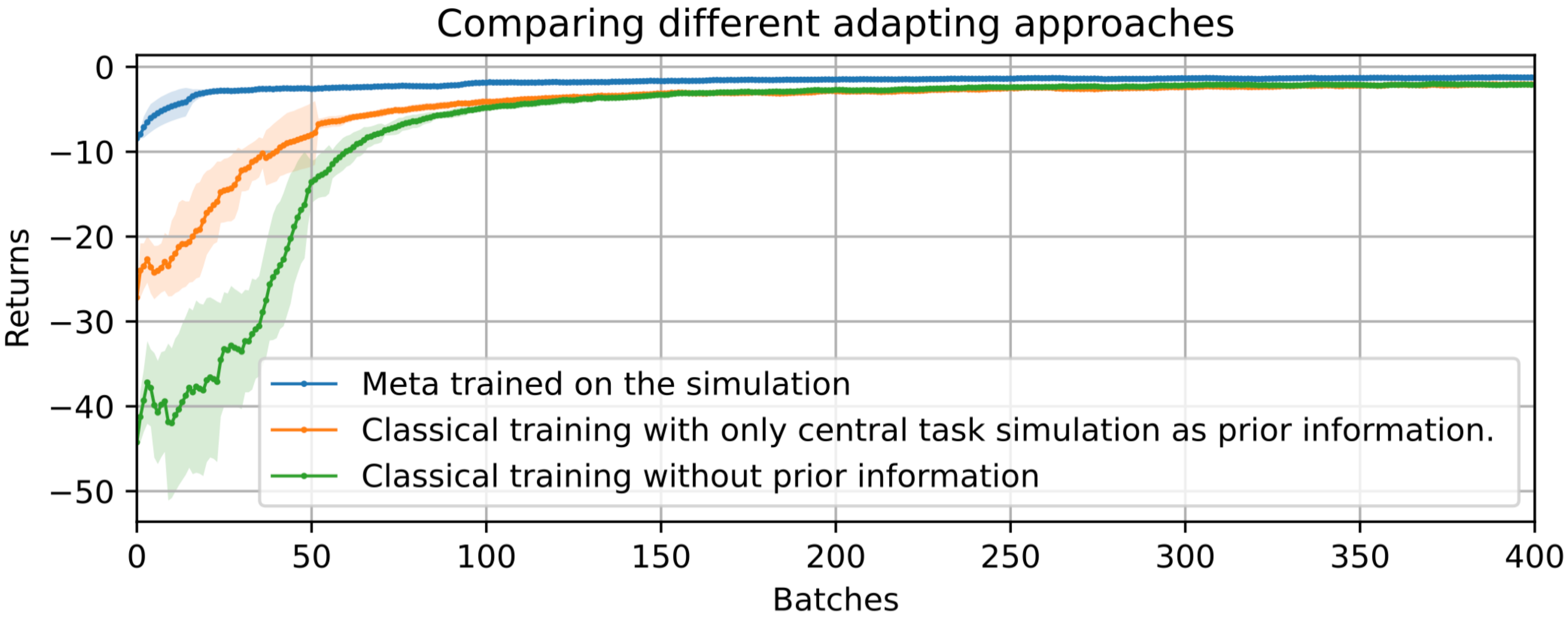
Meta-RL: learn an adaptable policy that can quickly specialize with minimal fine tuning

More robust and sample efficient in validation (real machine) but requires more samples (simulation)

Challenge 1: sample efficiency

How does this play in practice?

Beam steering task at AWAKE beamline
10 H dipoles, 10 V dipoles, 10 BPMs → ideal trajectory

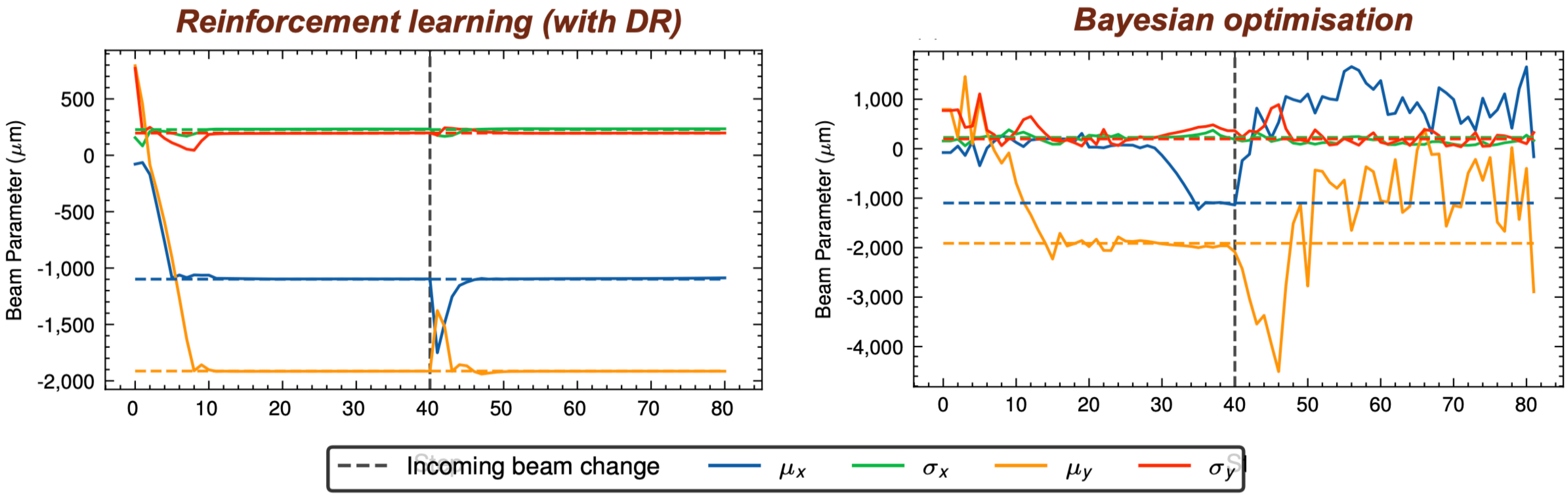


[Towards few-shot reinforcement learning in particle accelerator control, *JACoW* IPAC2024 (2024) TUPS60]

Robustness & sample efficiency

How does this play in practice?

Beam steering and focusing task at ARES linear accelerator
3 quadrupoles, 2 correctors → target beam size and position on a screen
Recovery from sudden change in incoming beam



[Reinforcement learning-trained optimisers and Bayesian optimisation for online particle accelerator tuning, *Sci.Rep.* 14 (2024) 1, 15733]()

Challenge 1: sample efficiency

How does this play in practice?

Training

Experiment-based

Task is adequately constrained and learnable

(low dimensions, informative observations, reward shaping)

Very rare! Only a handful of cases

FERMI, AWAKE, Linac4, KARA

- ~10^3 real-world interactions required for training
- Low-dimensional action and observation spaces
- Dense reward
- Very sensitive to hyperparameter choices
- Hard to find dedicated beamtime
- Safety concerns

“Basic reinforcement learning techniques to control the intensity of a seeded free-electron laser”, Electronics, vol. 9, no. 5, 2020
“Model-free and Bayesian ensembling model-based deep reinforcement learning for particle accelerator control demonstrated on the FERMI FEL”, arXiv:2012.09737, 2022.
“Sample-efficient reinforcement learning for CERN accelerator control”, Phys. Rev. Accel. Beams, vol. 23, no. 12, p. 124 801, 2020.
“Preliminary results on the reinforcement learning-based control of the microbunching instability” IPAC2024-TUPS61

Challenge 2: partial observability

Ideal setting

State fully observable

- MDP (finite, discrete)
- Model known
- Value function computable
- Optimal policy computable

vs

Real world

State partially observable

- POMDP (infinite, continuous)
- Model unknown or learned
- Value function approximated
- Policy approximated

Partial observability will always be a challenge in particle accelerator deployment, but can be mitigated with:

- Frequent and informative observations
- Memory (e.g., recurrent architectures) or a learned model
- Well-structured state representation
- Low-frequency decision making

Challenge 3: safety

Exploration vs exploitation dilemma:

We want to learn the **optimal behaviour** and for that we need to behave non-optimally to **explore** the state-action space.

→ ***Hard safety*** cannot be ensured in high-dimensional continuous state spaces!

Hard safety in RL, especially during exploration, is an active area of research

Soft safety can be implemented:

- Shielding
- Reward shaping
- Uncertainty-aware planning

Trade-offs between safety, optimality, and sample efficiency.

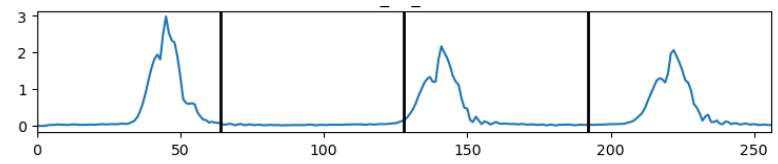
My recommendation: do experiment-based training only in safe machines (low energy, electrons) or have an excellent interlock system.

Challenge 4: real-time inference

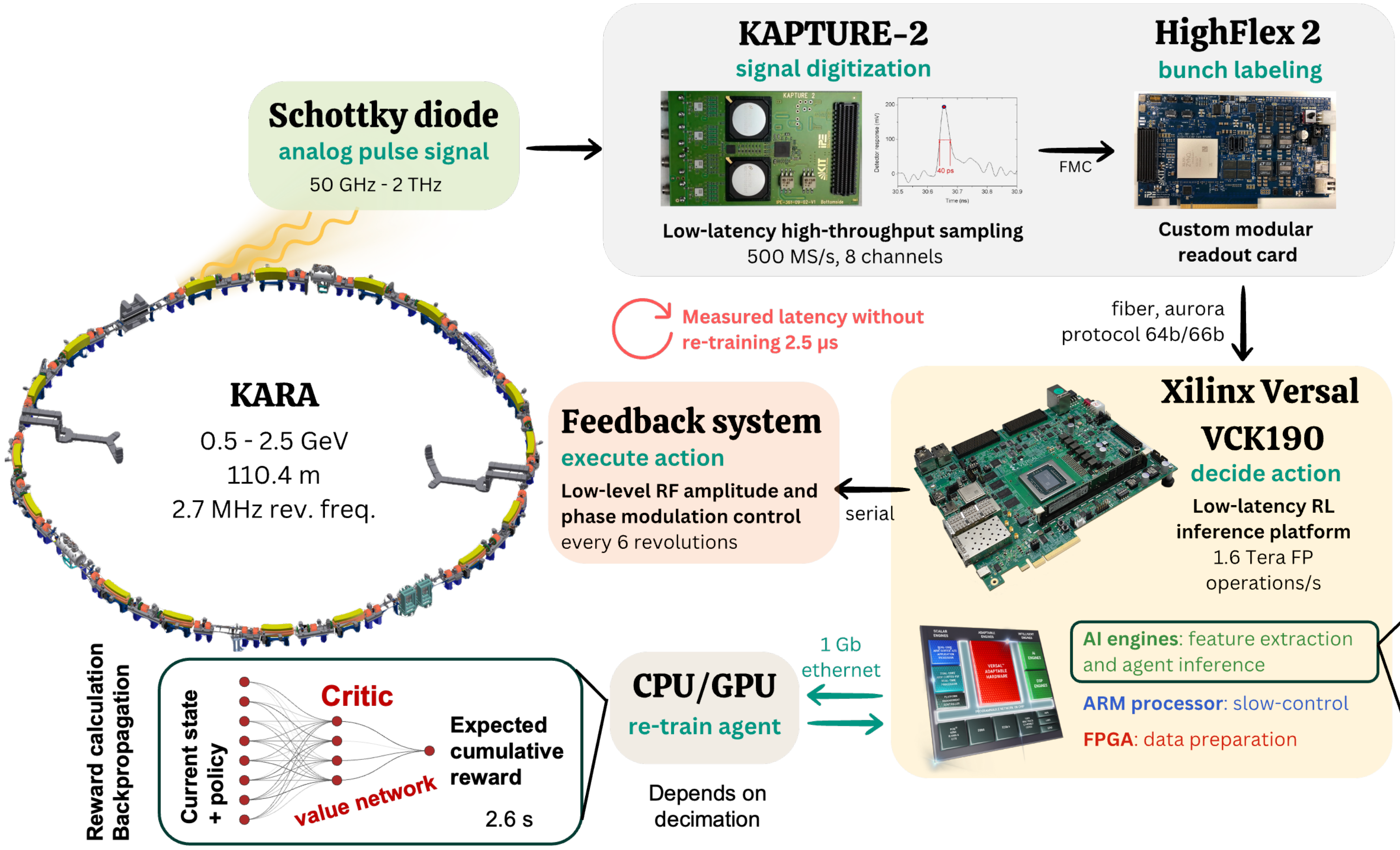
Control of the microbunching instability

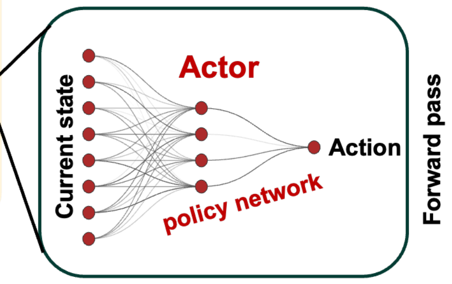
- Revolution frequency: 2.7 MHz (T_{rev} = 370 ns)
- Synchrotron frequency: 7-9 kHz (T_{sync} = 110-143 μs)
- → 300-400 T_{rev} ~ T_{sync}

Circular buffer of last 64 THz signal samples (decimated)



1. Agent acts every:
   96 x T_{rev} ~ 28 kHz ~ 0.25 x T_{sync} ~ 36 μs during 2048 steps (samples of decimated signal)
2. Agent stops and is re-trained in a CPU (~2.6 s)
   - We train every (2048 x 96) T_{rev} = 509 T_{sync}
3. New weights are sent to Versal board and agent starts again





[Doctoral thesis L. Scomparin]

Main challenges of RL deployment

In particle accelerators

Online Training

Model-free or model-based algorithms

Challenge 1
Sample efficiency

Challenge 2
Partial observability

Challenge 3
Safety

Simulation-based
Sufficient and varied enough data exists from computationally accurate and tractable models

Experiment-based
Task is adequately constrained and learnable

(low dimensions, informative observations, reward shaping)

Robust policy training to bridge sim2real gap

Careful algorithm design
Fine hyperparamer tuning

π

Validation

In the real accelerator

Challenge 3
Safety

Challenge 4
Real-time inference

Conventional control
1-100 Hz action

Ultra-fast control
> 10 kHz action

RL is a promising and powerful framework for adaptive, goal-directed behaviour in complex environments…

…that requires careful design!

[Reinforcement learning in particle accelerators, A. Santamaria Garcia, 2025]

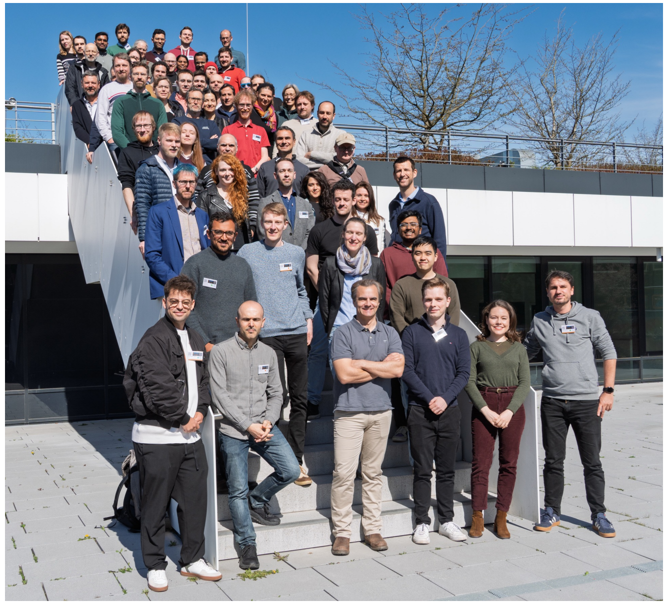
If you want to learn more about RL

Reinforcement Learning for Autonomous Accelerators collaboration

Yearly targeted workshop

https://indico.ph.liv.ac.uk/event/2025/

- Free registrations
- Beginner friendly
- Group coding challenge
- Keynote speaker
- Poster session
- Visit to Cockcroft Institute



RL4AA'25 @DESY

UNIVERSITY OF LIVERPOOL

THANK YOU FOR YOUR ATTENTION!

What questions do you have for me?

RESOURCES

- Sutton & Barto book
- https://arxiv.org/pdf/cs/9605103.pdf
- Reinforcement learning lectures by David Silver
- https://spinningup.openai.com/en/latest/
- Coursera RL specialization
- https://arxiv.org/pdf/1810.06339.pdf

Dr. Andrea Santamaria Garcia

Lecturer at University of Liverpool

Cockcroft Institute

ansantam@liverpol.ac.uk

https://www.linkedin.com/in/ansantam/

https://github.com/ansantam

https://instagram.com/ansantam

https://www.liverpool.ac.uk/people/andrea-santamaria-garcia