Cheetah – A High-speed Differentiable Beam Dynamics Simulation for Machine Learning Applications

4th ICFA Machine Learning Workshop

Jan Kaiser, Chenran Xu, Annika Eichler and Andrea Santamaria Garcia
Gyeongju, 8 March 2024

This Talk

Questions

What is *Cheetah*? 🐆

What can you do with it? 🚀

What is *Cheetah*?

Cheetah

Linear Beam Dynamics Simulation Python Package

- **Python package for beam dynamics simulations based on PyTorch for use with machine learning applications.**

- Two main features in support of ML applications:
  - **Ultra-fast compute**: (at the cost of fidelity) Cheetah can run order of magnitude faster than some other codes.
  - **Differentiability**: Based on PyTorch, Cheetah supports automatic differentiation for all its computations.

- Incidentally, Cheetah provides full **GPU support** and **integrates seamlessly with ML** models built in PyTorch.

- Designed to be **easy to use** and **easy to extend**.
  - We generally aim for high **code quality**!
  - **Black** / **isort** code formatting + **flake8** conformity enforced.
  - Encourage proper procedures in GitHub repository (automatic tests / PR templates, good **documentation** etc.)

https://github.com/desy-ml/cheetah

```
pip install cheetah-accelerator
```

```python
# Load initial beam distribution from ASTRA tracking
beam_in = ParticleBeam.from_astra("beam_in.ini")

# Create a FODO lattice
segment = Segment(
    [
        Drift(length=torch.tensor(0.2)),
        Quadrupole(length=torch.tensor(0.2), name="Q1"),
        Drift(length=torch.tensor(0.4)),
        Quadrupole(length=torch.tensor(0.2), name="Q2"),
        Drift(length=torch.tensor(0.2)),
    ]
)

# Change the magnet strengths
segment.Q1.k1 = torch.tensor(10.0)
segment.Q2.k1 = torch.tensor(-9.0)

# Tracking through the segment
beam_out = segment.track(beam_in)
```

Elements and Beams

The basic structure of Cheetah

Element (`accelerator.py`)

- Subclasses represent **accelerator components** like drifts, quadrupoles, steerers etc.
  - Currently Cheetah supports 14 different element types
- Special element `Segment` represents lattices (sequences) of elements.
  - Supports loading from LatticeJSON, Ocelot and Bmad

```
class Element(nn.Module):

    def track(self, incoming: Beam) -> Beam:
        # Implement your own beam tracking here
        # ...
        return outgoing
```

Beam (`particles.py`)

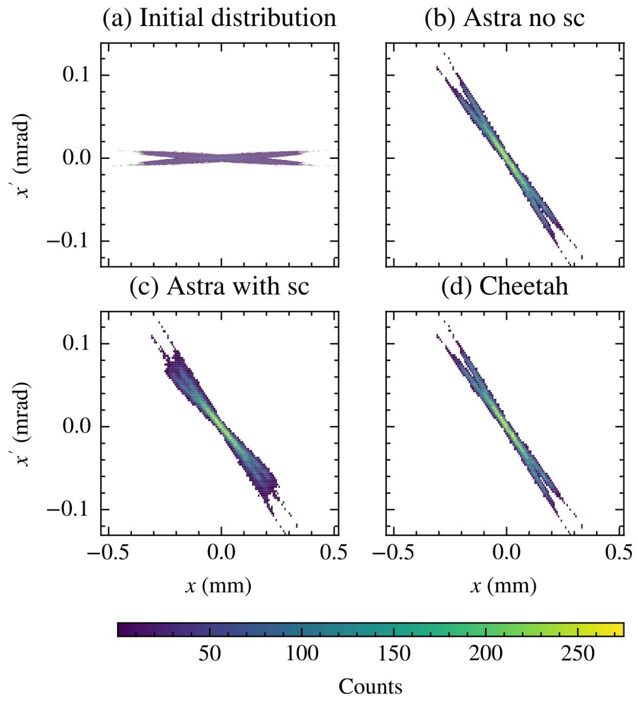
- Subclasses implement different ways of representing charged particle beams
  - `ParameterBeam` for fast compute:
    $\mu \in \mathbb{R}^7, \Sigma \in \mathbb{R}^{7\times 7}$
  - `ParticleBeam` for more precision:
    $P \in \mathbb{R}^{N\times 7}$

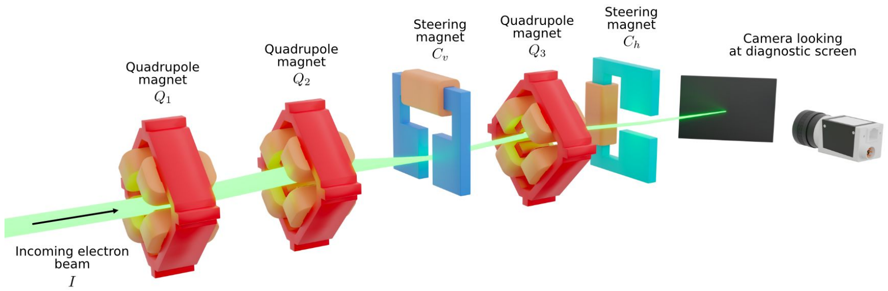
```
class Beam(nn.Module):
    # Has some representation of the beam

    @property
    def emittance_x(self) -> torch.Tensor:
        # Compute emittance from representation
        return result

    # ... etc.
```

Results

Does Cheetah work?

Phase space through the ARES Experimental Area





Step compute times through the ARES Experimental Area

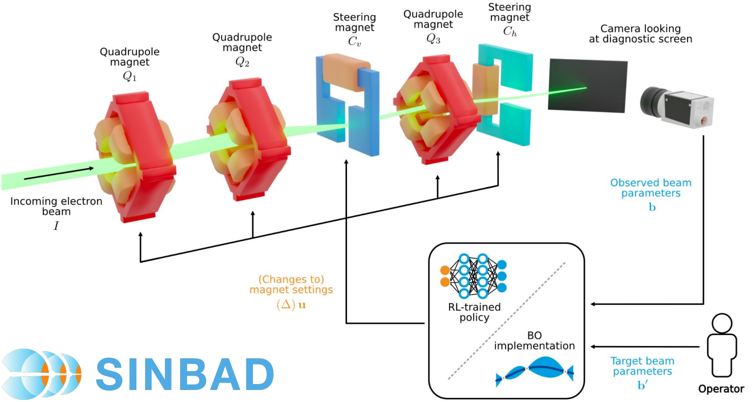
Code	Comment	Laptop	HPC node
ASTRA	space charge	264 000.00	3 605 000.00
	no space charge	109 000.00	183 000.00
Parallel ASTRA	space charge	39 000.00	17 300.00
	no space charge	16 900.00	12 600.00
Ocelot	space charge	22 100.00	21 700.00
	no space charge	182.00	119.00
Bmad-X		40.50	74.30
Xsuite	CPU	0.81	2.82
	GPU	-	0.57
Cheetah	`ParticleBeam`	1.60	2.95
	`ParticleBeam` + optimisation	0.79	0.72
	`ParticleBeam` + GPU	-	4.63
	`ParticleBeam` + optimisation + GPU	-	0.09
	`ParameterBeam`	0.76	1.29
	`ParameterBeam` + optimisation	0.02	0.04

What can you do with it?

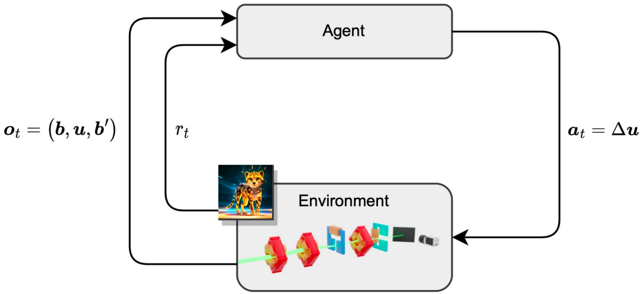
Fast Reinforcement Learning

Transverse beam tuning at ARES

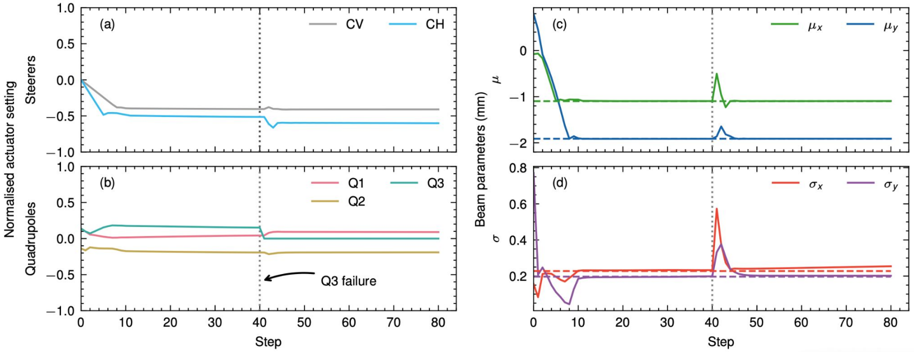
- Train a neural network policy to **tune transverse beam parameters** on a diagnostic screen using **five magnets** (3 quadrupoles, 2 dipoles).
- Would require **3 years of beam time** one the real machine, training would take **11 days with Ocelot**, takes ca. **1 hour with Cheetah**.
- Deploy a RL-trained optimisation algorithm to the **real-world** with **zero-shot learning** thanks to **domain randomisation**
- The trained policy outperforms other optimisation algorithms and expert human operators.



SINBAD



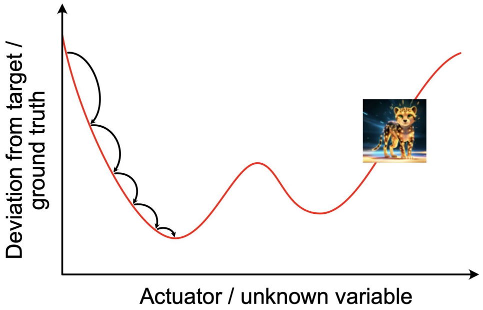
RLO in ARES EA (including feedback)



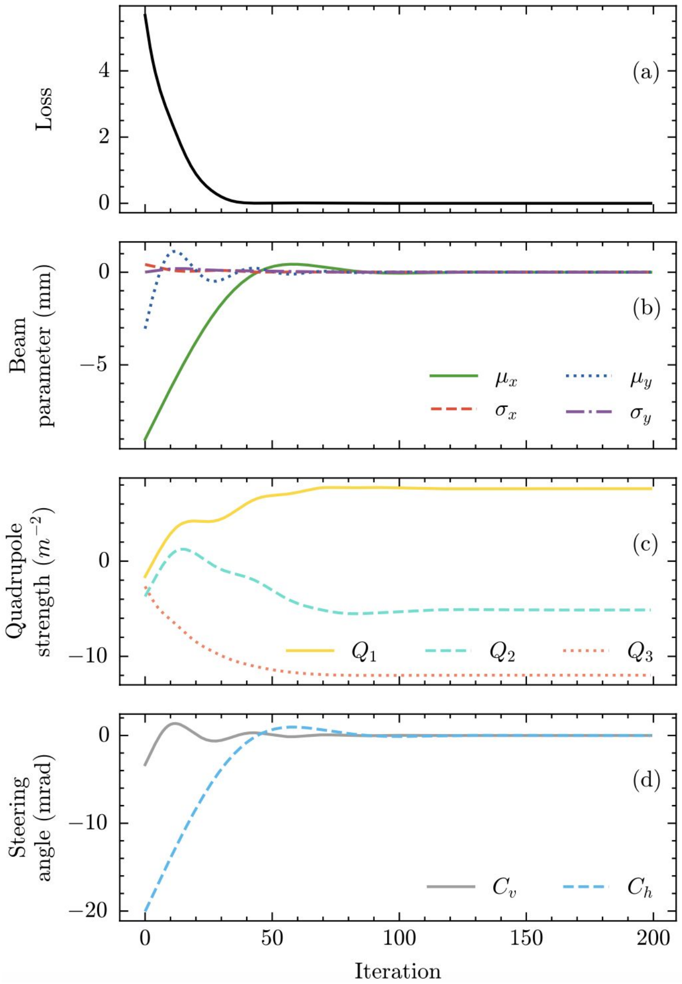
Gradient-based Tuning

Transverse beam tuning at ARES

- Tune magnet settings or lattice parameters using the **gradient of the beam dynamics model** computed through **automatic differentiation**.
- Seamless **integration with PyTorch** tools tuning neural networks.
- Becomes very useful for **high-dimensional tuning tasks** (see neural network training).



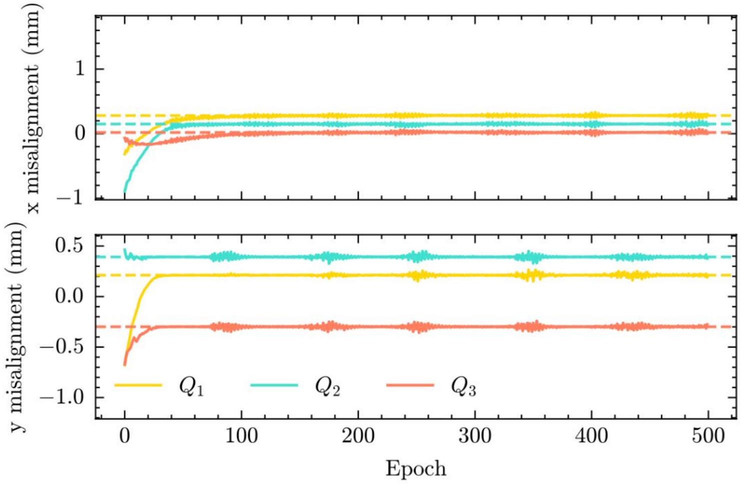

```
ares_ea.AREAMQZM1.k1 = nn.Parameter(0.0)
ares_ea.AREAMQZM2.k1 = nn.Parameter(0.0)
ares_ea.AREAMCVM1.angle = nn.Parameter(0.0)
ares_ea.AREAMQZM3.k1 = nn.Parameter(0.0)
ares_ea.AREAMCHM1.angle = nn.Parameter(0.0)

optimizer = Adam(ares_ea.parameters())

for _ in range(42):
    ougoing = ares_ea.track(incoming)
    loss = loss_fn(outgoing)

    loss.backward()
    optimizer.step()
    optimizer.zero_grad()
```

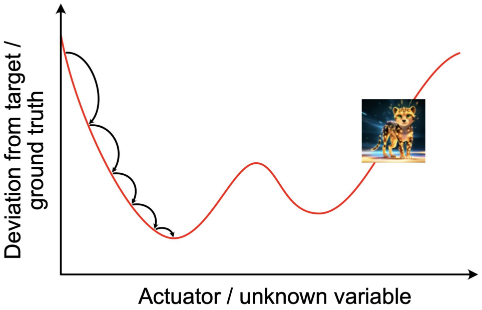


Gradient-based System Identification

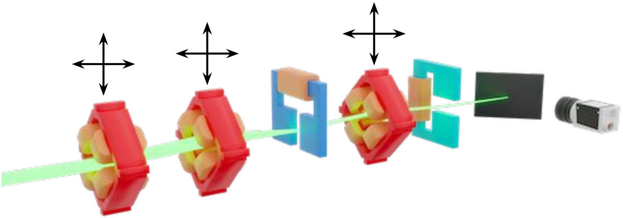
Quadrupole misalignments in the ARES Experimental Area

- Determine **hidden system properties** using the **gradient of the beam dynamics model** computed through **automatic differentiation**.
- Seamless **integration with PyTorch** tools tuning neural networks.
- Can be used in **combination with gradient-based tuning**.





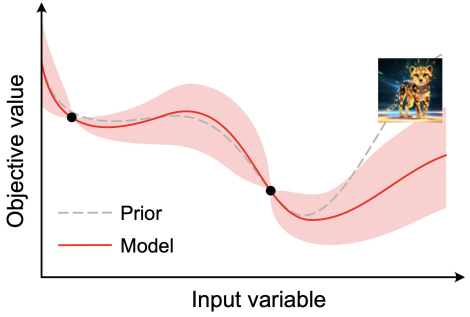

```python
ares_ea.AREAMQZM1.misalignment = nn.Parameter([0.0, 0.0])
ares_ea.AREAMQZM2.misalignment = nn.Parameter([0.0, 0.0])
ares_ea.AREAMQZM3.misalignment = nn.Parameter([0.0, 0.0])

optimizer = Adam(ares_ea.parameters())

for sample in dataset:
    set_magnets(ares_ea, sample.magnets)
    ougoing = ares_ea.track(incoming)
    loss = loss_fn(outgoing, sample.measurement)

    loss.backward()
    optimizer.step()
    optimizer.zero_grad()
```



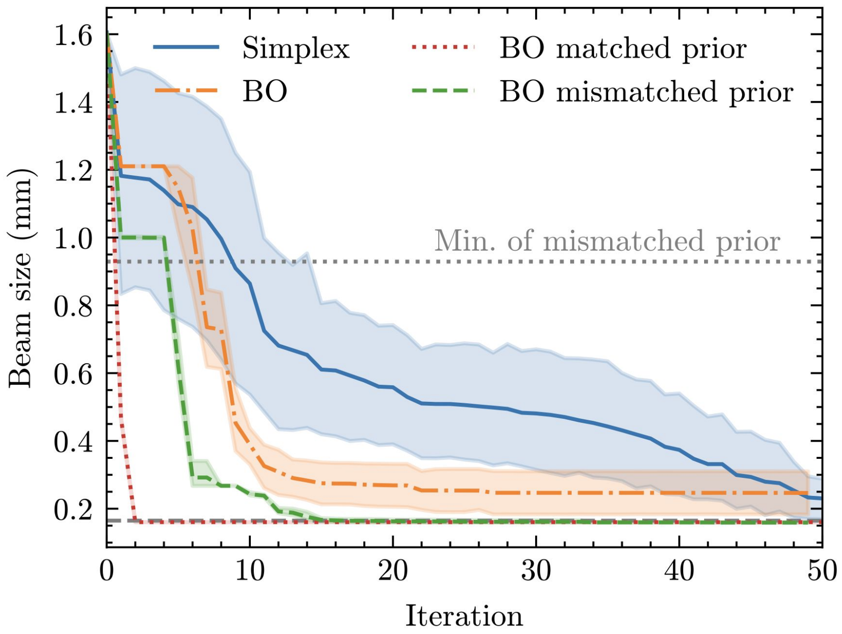
Physics-based Prior Mean for Bayesian Optimisation

Combine Cheetah with BO

- A physics-informed prior can help **improve the performance of BO** by preventing over-exploitation.
- Cheetah's differentiability allows **efficient acquisition function optimisation** using gradient descent methods in modern BO packages like BoTorch.
- Has well-defined behaviour and **does not need data** to train like neural network priors.
- Can be used in **combination with gradient-based system identification** to overcome model inaccuracies.



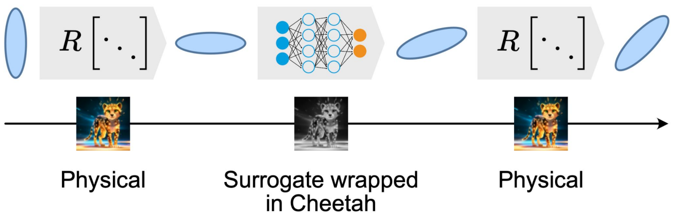
FODO cell beam focusing example

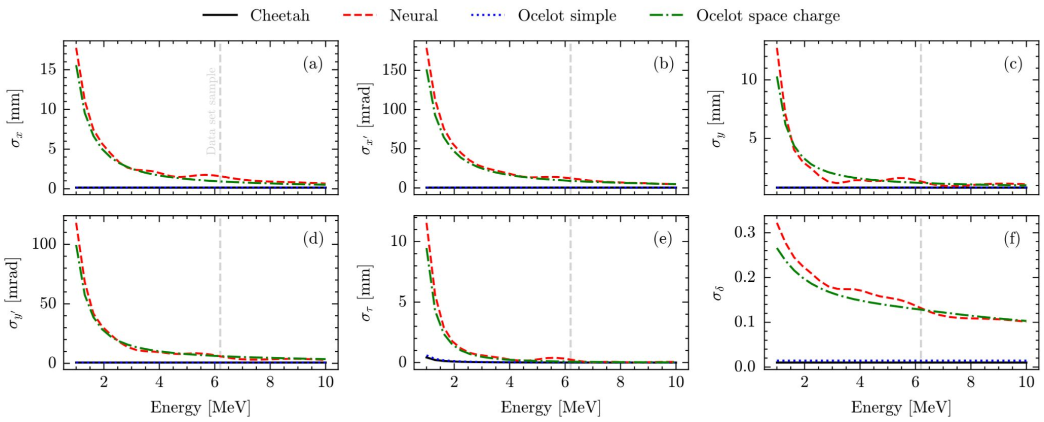


Integrate Modular Neural Network Surrogate Models

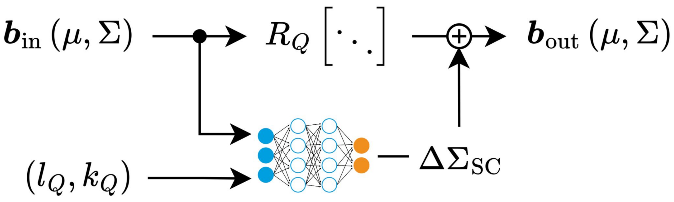
Increasing Cheetah's fidelity with surrogate models than can be reused

- Replace / **augment Cheetah** elements with neural network surrogates trained on high-fidelity simulations or real data.
- Neural networks implemented in PyTorch are effectively **native** to Cheetah. **Differentiability** is preserved. Integration is **easy**.
- Example: **Tracking with space charge** through quadrupole 3 orders of magnitude faster than Ocelot (**370 microseconds**).






```python
class SCQuadrupole(Element):
    net = SCNet().load_state_dict(torch.load("weights.pth"))

    def track(self, incoming: Beam) -> Beam:
        return self.net(incoming)
```

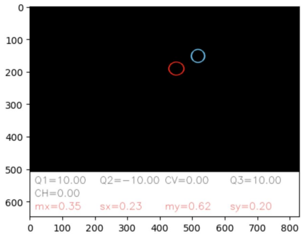


ICFA ML Contributions Using Cheetah

Many other utilities

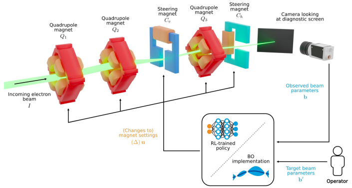
Applying Reinforcement Learning to Particle Accelerators: An Introduction

Environment has Cheetah backend, enabling us to see results quickly.



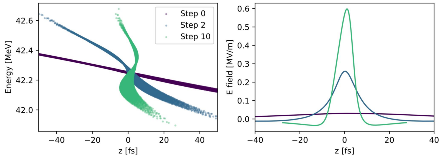
Learning to Do or Learning While Doing: Reinforcement Learning and Bayesian Optimisation for Online Continuous Tuning

Cheetah-based environment enabled RLO policy training and large scale evaluation.



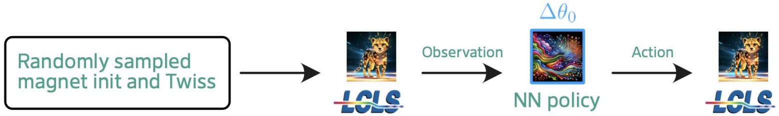
Reinforcement Learning Based Radiation Optimization at a Linear Accelerator

Another RL environment based on Cheetah enables fast training for CSR radiation optimisation.



Reinforcement Learning for Intensity Tuning at Large FEL Facilities

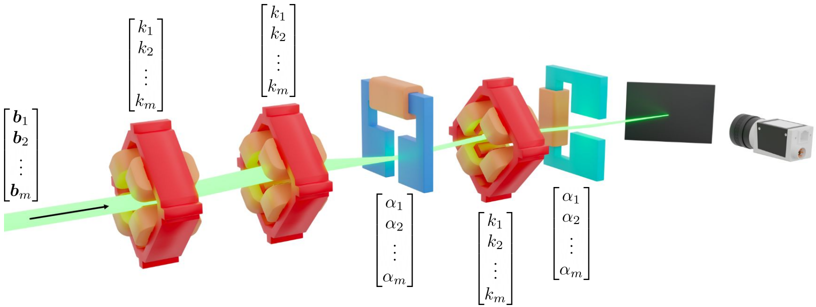
Cheetah enables gradient-based RL and 45x more sample-efficient training for FEL tuning.



Outlook

What's next for Cheetah?

- The next big thing → **Vectorised Cheetah**
  - **Concurrent simulation** of different actuator settings and beams
  - About **50x faster** on CPU, expected to be **even faster on GPU**
  - **Try it TODAY** with PR [Batched execution #116] on GitHub

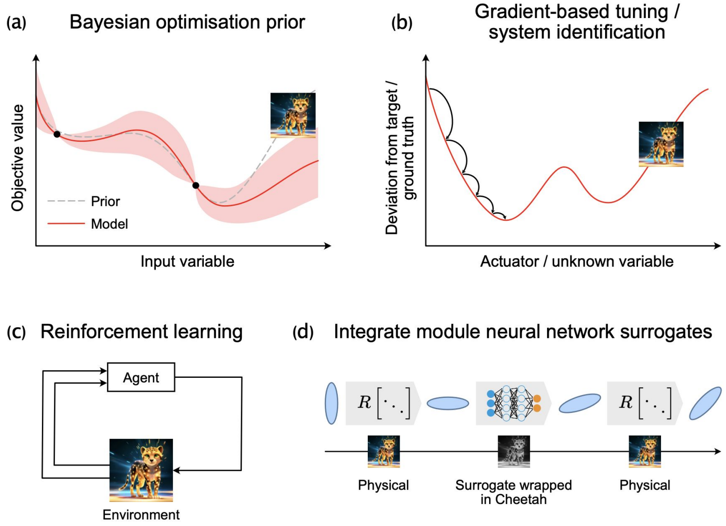


- We will continue to implement **further elements and adapters**, while applying Cheetah to **new applications**.
  - Contributions from the **community** are welcome!
- Explore **Cheetah with JAX** for further speed gains.

Conclusion

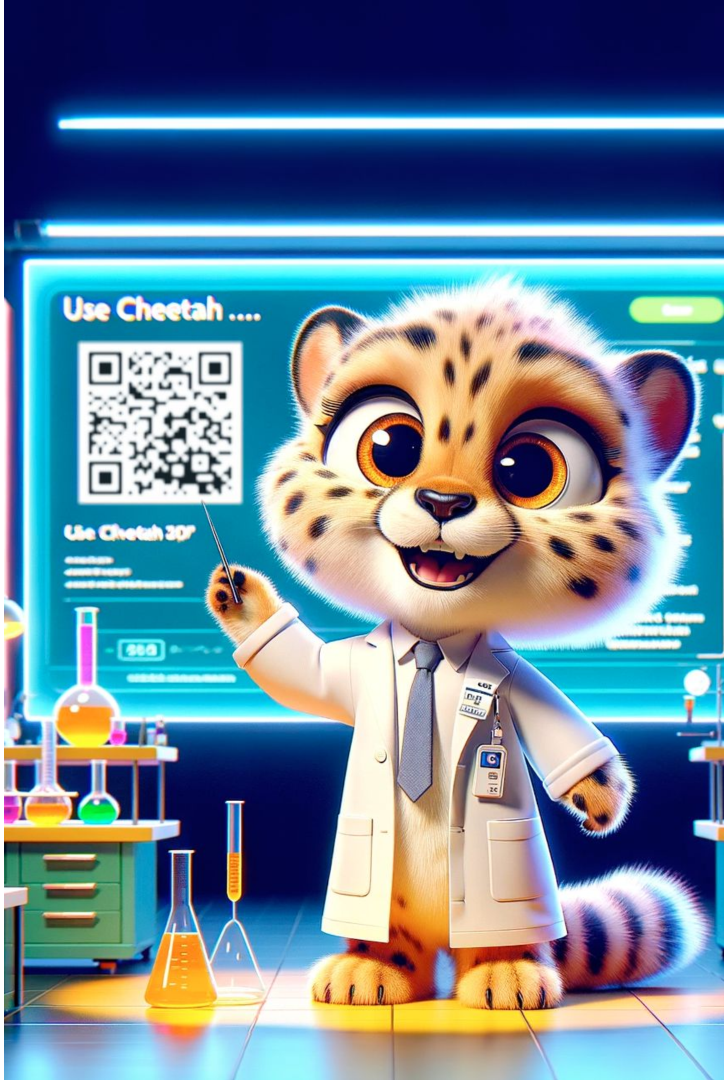
Answers to our questions

- **What is Cheetah? 🐆**
  - An easy-to-use Python package for fast and differentiable beam dynamics simulations.
  - Specifically designed for machine learning applications.
- **What can you do with it? 🚀**



\+ Gradient-based reinforcement learning

\+ **All the things you come up with!**



Contact

DESY. Deutsches
Elektronen-Synchrotron

www.desy.de

Jan Kaiser
Machine Beam Controls (MSK)
jan.kaiser@desy.de