ISBN: 978-3-95450-227-1 ISSN: 2673-5490 doi:10.18429/JACoW-IPAC2022-WEPOMS036

ACCELERATING LINEAR BEAM DYNAMICS SIMULATIONS FOR MACHINE LEARNING APPLICATIONS

O. Stein*, J. Kaiser, A. Eichler, I. Agapov
Deutsches Elektronen-Synchrotron DESY, Germany

Abstract

Machine learning has proven to be a powerful tool with many applications in the field of accelerator physics. Training machine learning models is a highly iterative process that requires large numbers of samples. However, beam time is often limited and many of the available simulation frameworks are not optimized for fast computation. As a result, training complex models can be infeasible. In this contribution, we introduce *Cheetah*, a linear beam dynamics framework optimized for fast computations. We show that Cheetah outperforms existing simulation codes in terms of speed and furthermore demonstrate the application of Cheetah to a reinforcement-learning problem as well as the successful transfer of the Cheetah-trained model to the real world. We anticipate that Cheetah will allow for faster development of more capable machine learning solutions in the field, one day enabling the development of autonomous accelerators.

INTRODUCTION

In recent years, machine learning (ML) has proven to be a powerful solution to many problems in a variety of fields, including accelerator physics. The subfield of reinforcement learning (RL), in particular, promises solutions to many control and optimization problems encountered during accelerator operations, with previous works demonstrating the ability of RL to solve difficult high-stakes physics-based control problems [1].

In RL, intelligent agents – implemented for example as an artificial neural network (ANN) – are trained through experience to iteratively solve a problem by observing an environment $\mathscr{E}$ – for example representing an accelerator control problem – through observations o_t and applying actions a_t to it in order to maximize a *cumulative reward*, the sum of step-wise rewards r_t.

A major challenge in the field of RL remains the large amount of experience required to successfully train agents. While there is active research ongoing on improving the sample-efficiency of RL [2], solving problems of sufficient complexity continues to require in the order of 10^5 up to 10^9 samples. Due to physical limitations, such large amounts of experience are usually infeasible to acquire in the real world, as taking just a single sample can sometimes take from a second to a multiple minutes depending on the physical system one is hoping to solve. Accelerator physics are particularity constrained in terms of taking large numbers of real-word samples as beam time is a notoriously scarce resource.

The issue of gathering large numbers of samples in the real-world has given rise to a research direction in RL called *sim2real*, where agents are trained in a simulation of the real-world system and then transferred to the latter. With ever more efficient simulation codes and ever faster and increasingly parallel compute resources available, especially thanks to the rise of general purpose GPUs (GPGPUs), this approach has enabled the gathering of large amounts of samples in feasible time frames. A similar approach can be pursued when training RL agents for accelerator control, as a variety of capable simulation codes have been implemented for particle accelerators over the years. These include *ASTRA* [3], *elegant* [4], *MAD-X* [5] and *Ocelot* [6], just to name a few. Existing accelerator simulation codes have however been developed to be used during the design phase of accelerators and for finding new working points – applications where physical accuracy is often crucial and compute times of multiple seconds, minutes and sometimes even hours are acceptable. As a result, these simulation codes usually require infeasible amounts of compute time and resources when used to train RL agents. At the same time, recent work in the field of sim2real [7] has found that *dynamics randomization* – adding random disturbances to the dynamics of a simulation – during the training of an RL agent yields improved results over perfectly accurate simulations.

In this paper, we propose a new accelerator simulation code *Cheetah*[1] that trades accuracy for speed to achieve simulation compute times in the order of a few hundred microseconds on off-the-shelf PC hardware. To this end, we introduce Cheetah and its features, present benchmarks on its speed and accuracy as well as demonstrate the application of Cheetah to train an RL agent to solve an accelerator control problem.

FAST PYTORCH-BASED PARTICLE ACCELERATOR SIMULATION

The goal of Cheetah is to achieve fast iterations in simulation, if necessary at the cost of accuracy. To this end, we implement a linear beam transfer based on matrix multiplication with first-order transport matrices in six-dimensional phase space. Cheetah is implemented in *Python* and employs

* oliver.stein@desy.de

This work has in part been funded by the IVF project InternLabs-0011 (HIR3X) and the Initiative and Networking Fund by the Helmholtz Association (Autonomous Accelerator, ZT-I-PF-5-6).

[1] Source code available at https://github.com/desy-ml/cheetah

ISBN: 978-3-95450-227-1 ISSN: 2673-5490 doi:10.18429/JACoW-IPAC2022-WEPOMS036

the *PyTorch* [8] framework. While the primary purpose of PyTorch is the implementation of ML algorithms, its fast tensor compute capabilities, strong GPU support and interface similar to that of *NumPy* [9] make it an ideal fit for fast parallel scientific computation.

Cheetah represents beams in either one of two ways. The first, called a *Particle Beam*, represents every particle in a beam as a seven-dimensional vector, tracking each particle's phase space position. This is similar to other accelerator simulation codes and allows for relatively precise beam tracking. The second, called a *Parameter Beam*, assumes a Gaussian beam and represents the entire beam by a seven-dimensional vector μ of the mean position in each of the phase space's dimensions and a covariance matrix Σ. Note that the seventh dimension is only maintained by Cheetah internally in order to track beams off-axis, but that the external interface of Cheetah assumes a standard 6-dimensional phase space. Cheetah can generate beams itself, but also supports loading Ocelot *Particle Arrays*, thereby adding support to load from other formats such as ASTRA as well.

Either type of beam can be tracked through an accelerator lattice, in Cheetah called a *Segment*. A Segment represents a sequence of *Elements* that make up its lattice. Elements may be magnets, drift sections, beam position monitors (BPMs), other accelerator components as well as nested Segments. Segments may be defined directly in Cheetah or loaded from existing Ocelot lattice files. To track a beam through an element with a transfer map R, Cheetah either computes $P_{\text{out}} = P_{\text{in}}R^T$ for a Particle Beam with particle matrix P_{in}, or $\mu_{\text{out}} = R\mu_{\text{in}}$ and $\Sigma_{\text{out}} = R\Sigma_{\text{in}}R^T$ for a Parameter Beam.

Cheetah currently supports drift sections, quadrupoles, horizontal correctors, vertical correctors, BPMs and diagnostic screens. Cheetah simulates the readings of BPMs and diagnostic screens for convenient use in RL environments. For quadrupoles, diagnostic screens and BPMs, Cheetah can simulate these elements being misaligned with respect to the golden orbit. Support for further elements is planned.

As the key focus of Cheetah is speed, a number of steps have been taken to optimize its performance: Firstly, **GPU acceleration** enables Cheetah to make use of the parallel compute capabilities of modern GPGPUs for tracking large numbers of particles in parallel with the help of PyTorch's proven interface for GPU acceleration. This includes automatic dispatching of computations to CPU or GPU based on hardware availability. Secondly, **Parameter Beams** reduce the amount of required computations significantly by considering only the particle beam's parameters instead of every single particle when accuracy is not needed. Thirdly, **dynamic transfer map combination** helps reduce the number of matrix multiplications required for particle tracking. Cheetah will dynamically determine the positions at which the beam must be known, for example where there is a BPM or a diagnostic screen, and combine all other transfer matrices into exactly as many as are strictly needed.

In addition, Cheetah offers an interface that makes it well suited for use in RL environments written in the *OpenAI Gym* framework. A simple example of using Cheetah to track an ASTRA beam through a FODO cell is given in Listing 1.

```
beam_in = ParticleBeam.from_astra("beam.astra")
segment = Segment([
    Drift(length=0.2),
    Quadrupole(length=0.2, name="ACCQ1"),
    Drift(length=0.4),
    Quadrupole(length=0.2, name="ACCQ2"),
    Drift(length=0.2)
])

segment.ACCQ1.k1 = 10.0
segment.ACCQ2.k1 = -9.0

beam_out = segment(beam_in)
```

Listing 1: Example Python code for tracking an ASTRA beam through a FODO cell using Cheetah.

BENCHMARKS

In this section, we compare the speed of Cheetah to that of other simulation codes and briefly show that Cheetah's computations remain relatively accurate.

For the speed benchmark, we track a beam of 100 000 particles through a 2.05 m lattice of the *Experimental Area* section of the *ARES* accelerator at DESY. The simulations were run on a machine with an AMD Ryzen 5 2600 CPU, an NVIDIA GeForce RTX 2070 and 16 GB of RAM. Simulation times were averaged over multiple runs using Python's *timeit* package. To enable a comprehensive judgement of the relative speeds, Cheetah was run in three different configurations: Tracking a Parameter Beam, tracking a Particle Beam on CPU and tracking a Particle Beam on GPU. Furthermore, we compared Ocelot [6] once without space charge and a step size the same as the entire section, and once with space charge and a step size of 2 cm; elegant [4] with space charge off; and ASTRA [3] both with space charge off and with space charge on. The results of the speed benchmark are listed in Table 1.

Thanks to the optimizations listed in the previous Section, Cheetah can compute simple simulation setups between 2 and 6 orders of magnitude faster than existing simulation codes. One must however keep in mind that this comes at the cost of accuracy, where higher-order effects, collective effects and others are left out in order to achieve the reported speeds. This loss of accuracy is not desirable for applications in accelerator and working point development, where existing simulation codes are used, but it may actually even be advantageous when used in the training of RL agents [7].

The correctness of Cheetah's computations was briefly evaluated by comparing the tracking results of Cheetah vs. Ocelot in the ARES Experimental Area (EA) for 125 different magnet settings. Ocelot had space charge simulation activated in order to provide a more reliable reference on Cheetah's accuracy compared to the real world. We observe that the results computed by Cheetah at 0.1 pC and 150 MeV deviate only 0.5 % to 5 % from those computed by Ocelot.

ISBN: 978-3-95450-227-1 ISSN: 2673-5490 doi:10.18429/JACoW-IPAC2022-WEPOMS036

Table 1: Step Computation Times of Simulation Codes

Simulation	Comment	Time (ms)
ASTRA	space charge	609 000.00
	no space charge	234 000.00
elegant		300.00
Ocelot	space charge	24 600.00
	no space charge	247.00
Cheetah	Particle Beam (CPU)	1.33
	Particle Beam (GPU)	0.84
	Parameter Beam	0.27

ACCELERATING REINFORCEMENT LEARNING

We demonstrate an example where Cheetah enabled the successful implementation of an RL agent to a particle accelerator control task [10, 11]. Specifically, we look at autonomously optimising the transverse beam properties on a diagnostic screen in the Experimental Area (EA) at *ARES* [12], an s-band linear electron accelerator at the DESY SINBAD facility in Hamburg, Germany. The EA is followed by an experimental vacuum chamber, which allows the installation of experimental setups that require specific transverse beam properties – beam position and size. These properties are measured with a diagnostic station upstream of the experimental chamber, where a camera is pointed toward a scintillating screen that can be moved into the electron beam. A sequence of three quadrupoles and two dipoles enable the tuning of the beam properties in order to realize desired beam parameters. Up to now, this task is mostly solved manually by experienced operators, which requires a lot of time and makes it difficult to reproduce results.

In order to apply RL to this problem, we define the task-specific RL loop. For the given task, the RL agent can observe target beam parameters b' provided by the human operator, the current beam parameters b_t measured on the screen as well as strength and deflection angle readbacks x_t of the five considered magnets. Based on the observation $o_t = (x_t, b', b_t)$, the agent may then compute an action $a_t = \left(\Delta k_{Q_1}, \Delta k_{Q_2}, \Delta k_{Q_3}, \Delta \alpha_{C_v}, \Delta \alpha_{C_h}\right)$ – the desired changes to the magnet settings – which is then sent to the accelerator's control system. On each step, the agent is rewarded by a reward r_t based on how much closer it got the measured beam parameters to the target beam parameters.

Training an RL agent on the given task takes 6 000 000 steps, i.e. 6 000 000 iterations of setting the magnets, reading the beam from the screen and doing a background subtraction. Due to limitations of the accelerator's hardware, such as relatively slow magnet power supplies and network delays of the distributed control system, one such step takes ca. 10 s to 20 s. A full RL training on the real accelerator would therefore require at least two years of continuous beam time – an infeasible amount.

Even training the RL agent in simulation for the same number of steps using existing simulation codes is impractical at best. An ASTRA simulation or an Ocelot simulation including space charge would require about between 4 and 115 years of compute time. Even a training with the simplest and therefore fastest configuration of an Ocelot simulation would require over 17 days of compute time. While this is in the realm of some other RL work [13] and can be sped up in terms of wall-time using parallel computation on high-performance compute clusters, it is still an expensive proposition and would slow down the design process when developing an RL solution such as the one presented here. Using a Cheetah simulation for the training, the simulation compute time can be reduced to just 27 minutes, at which point the compute requirements of the actual RL algorithm start dominating. A full training of the presented agent takes around 3 hours on the machine listed in the *Benchmark* Section when using the *Stable Baselines3* RL library [14].

We observe that despite having been trained in a highly simplified simulation, our trained RL agents perform very well on the real accelerator, achieving better results faster than alternative black-box optimization algorithms. Furthermore, the trained agents manage to optimize the beam faster than even experienced human operators. If given enough time, human operators can, however, still achieve a slightly better beam than the RL agents.

SUMMARY AND OUTLOOK

In this work we introduced Cheetah, a high-speed particle accelerator simulation package for Python that trades accuracy for speed in order to achieve the speed required for training RL agents on accelerator control and optimization tasks. Cheetah offers an interface well-suited to writing RL environments and achieves simulations speeds 2 to 6 orders of magnitude faster than existing accelerator simulation codes. We further demonstrated the successful application of Cheetah to the training of an RL agent on a beam optimization task on the ARES accelerator. Despite, or possibly even because of training in a slightly less accurate simulation, the trained agent outperformed other algorithms and kept up with human performance on the real accelerator.

As part of our ongoing research into RL for accelerator control and optimization toward autonomous accelerators, we intend to continue to maintain and extend Cheetah for training agents on future RL applications.

ACKNOWLEDGEMENTS

The authors thank the ARES team Florian Burkart, Willi Kuropka, Hannes Dinter, Frank Mayet, Sonja Jaster-Merz and Thomas Vinatier for their great support during shifts as well as always insightful brainstorms. Furthermore, the authors acknowledge the support from their Helmholtz AI Autonomous Accelerator project partners at the Karlsruhe Institute of Technology (KIT) Andrea Santamaria Garcia, Chenran Xu and Erik Bründermann. In addition, the authors acknowledge support from DESY (Hamburg, Germany), a member of the Helmholtz Association HGF.

ISBN: 978-3-95450-227-1 ISSN: 2673-5490 doi:10.18429/JACoW-IPAC2022-WEPOMS036

REFERENCES

[1] J. Degrave *et al.*, "Magnetic control of tokamak plasmas through deep reinforcement learning", *Nature*, vol. 602, pp. 414-419, 2022. doi:10.1038/s41586-021-04301-9

[2] D. Yarats *et al.*, "Improving sample efficiency in model-free reinforcement learning from images", *Computing Research Repository*, vol. 1910.01741, 2019. doi:10.48550/arXiv.1910.01741

[3] ASTRA – A space charge tracking algorithm, https://www.desy.de/~mpyflo/.

[4] M. Borland, "elegant: A flexible SDDS-compliant code for accelerator simulation", in *Proc. ICAP'00*, Sep. 2000, Darmstadt, Germany. doi:10.2172/761286

[5] MAD-X – Methodical accelerator design, https://madx.web.cern.ch/madx/.

[6] I. Agapov *et al.*, "OCELOT: A software framework for synchrotron light source and FEL studies", *Nucl. Instr. Meth.*, vol. 768, 2014. doi:10.1016/j.nima.2014.09.057

[7] OpenAI *et al.*, "Solving Rubik's Cube with a robot hand", *Computing Research Repository*, vol. 1910.07113, 2019. doi:10.48550/arXiv.1910.07113

[8] A. Paszke *et al.*, "PyTorch: An imperative style, high-performance deep learning library", in *Proc. NeurIPS'19*, Vancouver, BC, Canada, Dec. 2019, pp. 8024-8035, 2019. doi:10.5555/3454287.3455008

[9] C. R. Harris *et al.*, "Array programming with NumPy", *Nature*, vol. 585, 2020. doi:10.1038/s41586-020-2649-2

[10] A. Eichler *et al.*, "First steps toward an autonomous accelerator, a common project between DESY and KIT", in *Proc. IPAC'21*, Campinas, Brazil, May 2021, pp. 2182–2185. doi:10.18429/JACoW-IPAC2021-TUPAB298

[11] J. Kaiser, O. Stein and A. Eichler, "Learning-based optimisation of particle accelerators under partial observability without real-world training", in *Proc. ICML'22*, Baltimore, USA, Jul. 2022, to be published.

[12] E. Panofski *et al.*, "Commissioning results and electron beam characterization with the s-band photoinjector at SINBAD-ARES", *Instruments*, vol. 5(3), p. 28, 2021. doi:10.3390/instruments5030028

[13] O. Vinayals *et al.*, "Grandmaster level in StarCraft II using multi-agent reinforcement learning", *Nature*, vol. 575, 2019. doi:10.1038/s41586-019-1724-z

[14] Stable Baselines3, https://github.com/DLR-RM/stable-baselines3/.