Batch spacing optimization by reinforcement learning

Matthias Remta and Francesco Velotti
CERN, Geneva 1211, Switzerland

Sharwin Rezagholi
UAS Technikum Wien, Vienna 1200, Austria

(Received 21 January 2025; accepted 3 September 2025; published 25 September 2025)

Beams designated for the LHC are injected into the SPS in multiple batches. Given the tight spacing of 200 ns between these batches, the injection kickers have to be precisely synchronized with the injected beam to minimize injection oscillations. Due to machine drift, the optimal settings for the kickers vary. This paper presents an active controller trained by reinforcement learning that counteracts the machine drifts by adjusting the settings. The agent was exclusively trained in a simulation environment and directly transferred to the accelerator. Although its results are slightly worse than those obtained by an explicit numerical optimizer, the BOBYQA algorithm, the agent attains these results much faster since it requires far less computation.

DOI: 10.1103/g9wr-197z

I. INTRODUCTION

At CERN, particles are accelerated through a chain of accelerators, each suitable for a specific energy range. The injected particles are guided toward the circular orbit of the receiving accelerator, and at extraction, particles are diverted from that trajectory into a transfer line. In both processes, fast-pulsed magnets, known as kickers, are used to guide the motion of particles. The Super Proton Synchrotron (SPS) is a circular accelerator that provides proton beams for the Large Hadron Collider (LHC). To do so, batches of particles are sequentially injected into the SPS from the Proton Synchrotron. The luminosity of the LHC is directly proportional to the number of particles stored; hence, reducing the batch spacing may allow for more total intensity [1]. If the batch spacing is smaller than the time it takes for the injection kickers of the SPS to reach their design voltage, either the circulating beam is perturbed or the injected beam is not placed on a suitable orbit. This causes excessive oscillations of the circulating beam, known as injection oscillations, which may degrade the beam quality or cause the loss of the beam. Therefore, the batch spacing must trade off oscillations of the injected and circulating beam to optimize the beam quality delivered to the LHC (Sec. III H). Currently, numerical optimization is employed for this purpose [2], specifically BOBYQA (Bound Optimization By Quadratic Approximation) [3]. Numerical approaches to batch spacing have three disadvantages. First, the optimization problem itself is subject to drift due to gradual changes in the injection system and the accelerator. In such cases, the optimizer has to be rerun or losses must be accepted. Second, the instability or nonuniqueness of solutions may lead to relatively large changes in the settings, potentially causing large changes in beam quality between injections. Third, each time the numerical optimizer is applied, it must explore the surface of the loss function anew [3], sometimes requiring several hundred evaluations. These evaluations are costly since they can only be undertaken once per supercycle of the accelerator, that is, roughly twice per minute. One alternative is time-varying Bayesian optimization [4–7] and another is reinforcement learning (RL). We investigate the latter. Our aim is to develop an agent capable of optimizing the injection system even after an extended shutdown and of serving as an active controller during operation, that is, counteracting drifts of the accelerator during its active use. A partially observable simulation environment of the accelerator was used to train such agents via RL. The agents were then directly deployed on the accelerator without any training in the real environment. The best performance was achieved with a recurrent version of proximal policy optimization (PPO) [8]. This agent consistently maintained a valid working point under perturbations. To the best of our knowledge, this constitutes the first successful application of RL with recurrent policy networks in accelerator physics. The simulation environment, data, and program code are publicly available [9].

II. RELATED WORK

Since training on real accelerators is costly [10–14], the application of machine learning in this domain requires

either extremely data-efficient algorithms, a description that does not apply to deep RL methods, or sufficiently valid simulation environments [13,15–17]. The transfer of an agent from a simulation environment to the real hardware is complicated by the noisiness of observations and the partial observability of the environment [14,18,19]. Kain *et al.* evaluated RL agents on the task of trajectory correction at two beamlines at CERN: the electron line of the Advanced Proton-Driven Plasma Wakefield Acceleration Experiment (AWAKE) and the Linear Accelerator 4 (LINAC4) [11]. Their experiments used deep Q-Learning with normalized advantage functions (NAFs) [20]. At AWAKE, the agent was usually able to correct the trajectory below a root mean squared deviation (RMSD) from the target trajectory of 1 mm within one or two iterations; at LINAC4, the agent needed at most three iterations to achieve this, in both cases outperforming numerical optimizers [11]. Although RL was deemed sufficiently sample efficient to train on the actual accelerator, the authors proposed using simulation environments for off-line training [11].

Velotti *et al.* [13] compared RL (TD3 [21]) and numerical optimization on the task of setting up the electron beamline for the AWAKE experiment. The beamline requires frequent optimization to maintain the brightness of the beam. Agents were pretrained on a simulation of the AWAKE machinery and fine-tuned and tested on the actual hardware. The final agents were able to reach their target in less than ten iterations and performed significantly better than numerical optimizers, which needed up to several hundred iterations. However, after a few days, the performance of the RL agents declined, possibly due to environment drift. Velotti *et al*. conclude that the operational deployment of RL agents is impossible unless this problem is solved [13].

Hirlaender *et al.* evaluated Meta-RL for tuning the AWAKE beamline [22]. They employed Model Agnostic Meta Learning (MAML) [23] to tackle the problem of partial observability. The technique operates on a distribution of environments, induced by the distribution of the unobserved state components. During pretraining, MAML aims to find model weights that can be quickly fine-tuned for a specific environment drawn from the distribution. The authors observed fast adaptation capabilities on the accelerator [22].

Meier *et al.* optimized a radio-frequency gun with RL [24]. Starting from physical simulations, they first trained a surrogate model represented by a neural network. Using this surrogate model, they trained and evaluated an RL agent, which outperformed numerical optimizers [24]. St. John *et al.* developed a controller for the gradient magnet power supply of the booster synchrotron at Fermilab [25]. On a surrogate model of the accelerator, the RL agent, based on a double deep Q-network (DQN, [26]), showed significant improvement compared to a numerical proportional-integral-derivative controller [25].

Bruchon *et al.* leveraged RL to perform two tasks at a free-electron laser (FEL): reaching a target intensity and recovering the target intensity after machine perturbations [27]. For the first task, deep Q-Learning [28] was used, and for the second, REINFORCE [29] with natural policy gradients [30].

Bruchon *et al.* proposed an online iterative linear quadratic regulator (iLQR) to align the seed laser of the FERMI FEL with the electron beam [31]. They modeled the system response using a neural network trained on experimental data and used the iLQR algorithm to compute an optimal control sequence. Only the first control input was applied at each step, followed by reoptimization based on the updated system state, thereby mitigating model-plant mismatches. The approach successfully resolved the alignment problem and outperformed gradient ascent by requiring fewer iterations [31].

O'Shea *et al*. used RL based on policy gradients for three optimization problems arising for the FEL at Elettra: optimizing the signal at a beamline, maximizing the FEL energy after the seed laser energy is reduced, and recovering the output energy of the FEL after perturbations [32].

Chen *et al.* used RL for trajectory correction at the China Accelerator Facility for Superheavy Elements (CAFe II) [15]. Their TD3-based agent employs a convolutional policy network. It was trained in a simulation environment and then directly applied to the actual hardware. The agent was able to correct the trajectory of the $^{40}Ca^{13+}$ particle to less than 1 mm RMSD in under 15 s. They also trained an agent on $^{55}Mn^{18+}$ particles and applied it to $^{40}Ca^{13+}$ particles and protons without retraining. The agent was able to correct both orbits to less than 1 mm RMSD [15]. The authors provided the trend from the beam position monitor to the agent, thereby equipping the agent with a form of "memory."

Kaiser *et al.* developed a TD3-agent for the control of the transverse electron beam of a linear accelerator at the Accelerator Research Experiment at Sinbad [18]. The agent was entirely trained via simulation. The authors randomized the nonobservable components of the state during training. Their agent achieves similar losses as numerical optimizers but attains them much faster [18].

Wang *et al.* investigated RL for automated tuning of the off-line ion source beamline at TRIUMF [14]. The internal state of their beamline is not fully measurable, hence their agents' environment is only partially observable. They employed the deep deterministic policy gradient algorithm (DDPG) with a recurrent policy network, namely a long short-term memory (LSTM) [33] network.

In contrast to the studies reviewed above, our approach differs in several key aspects. While several prior works used deep RL algorithms such as deep deterministic policy gradient (DDPG) [14], twin delayed DDPG (TD3) [13,15,18], or normalized advantage functions (NAF) [11], we employ proximal policy optimization (PPO) [8]

with an LSTM, which is particularly suited to partially observable environments. To mitigate the effects of environment drift, we expose the agent to diverse training scenarios with stochastic noise, following the strategy of domain randomization [18], but in combination with a memory-based agent architecture (PPO with LSTM). Unlike approaches relying on surrogate models [24,25,31], our method is based on a fast transfer map model. In contrast to agents using fixed-length observation histories, such as convolutional encoders [15] or finite-window LSTM architectures [14], our recurrent PPO agent maintains an internal memory state, allowing it to leverage information from the entire trajectory. Finally, for the first time, we successfully applied a recurrent agent to a particle accelerator.

III. METHODS

A. The SPS injection system

Preaccelerated particles are delivered via a transfer line (TT10) to the SPS. The SPS injection system is located at the end of the transfer line. The main electromagnetic elements are septa, followed by a defocusing quadrupole and the injection kickers. Figure 1 shows a schematic of the injection system.

The quadrupole and the septa maintain a static magnetic field. The injection kickers are only powered at injection to deflect the incoming beam onto the design orbit [34]. These kickers are referred to as MKPs. They are fast-pulsed, that is, their voltage rises within tens of nanoseconds to their design voltage. There are 16 kicker magnets in four vacuum tanks. The kickers are grouped in pairs, and each pair shares an electric circuit which includes an electrical switch that controls whether the respective kickers are active or not [35]. The start of the injection procedure is marked by an electrical signal that is synchronized with the injected beam. After roughly 5000 ns, the injected beam reaches the kickers. The times at which the electrical switches trigger can be controlled individually for each switch by setting a delay relative to the start of the injection.

The time that passes between the kicker's current reaching $\epsilon\%$ and $(100-\epsilon)\%$ of the respective design current is referred to as rise time. A common choice is $\epsilon = 2$ [35]. The rise times impose a lower boundary on the feasible choices for the batch spacing. A change in rise time might necessitate a correction in the corresponding delay to ensure optimal alignment with the beams. Analysis of historical rise times has revealed frequent changes in rise time for some of the MKPs (Appendix A).

B. Reinforcement learning approach

As commonly done in RL, we conceptualize the agent's environment as a partially observable Markov decision process (POMDP). Let $\Delta(X)$ denote the set of probability distributions over the set X. We describe the POMDP by the state space S, the action space A, the observation space O, the observation map $S \to \Delta(O)$, a stationary Markovian conditional probability system $\mathbb{P}(s_{t+1}|s_t, a_t)$ where $a_t \in A$ and $s_t, s_{t+1} \in S$, and a reward function $r\colon S \times A \to \Delta(\mathbb{R})$. The aim of the agent is to employ a policy $O \to \Delta(A)$ that maximizes the discounted expected continuation reward $\sum_{\tau=t}^{\infty} \gamma^{k-t} r(s_k, a_k)$ where $\gamma \in (0,1)$ denotes the discount factor. Conceptually γ is a part of the definition of the POMDP, but in practice, the discount factor is rather akin to a hyperparameter. Due to partial observability, the agent must learn to extract the relevant information on the true state from the history $(o_0, a_0, r_0, \ldots, o_{t-1}, a_{t-1}, r_{t-1})$. We refer to [36] for a detailed discussion of POMDPs in the context of RL.

C. Algorithm

We employ proximal policy optimization (PPO [8]). The agent's policy is determined by an artificial neural network (ANN), the policy network, with a parameter vector w. Its output is a vector of Gauss-distributed random variables, the parametrization of which is determined by the ANN. In the following, $\pi_w(.|s)$ denotes the stochastic policy induced by the weights of the policy network and conditioned on the state $s \in S$, which the policy network takes as input. Furthermore, let $a \in A$ be an action, $\epsilon \in (0,1)$ and w_k

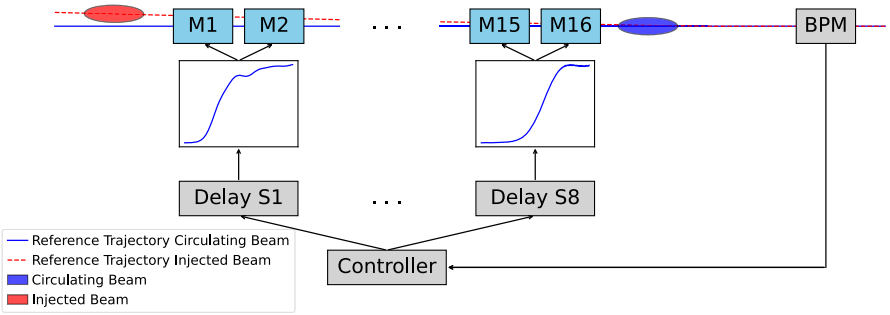

FIG. 1. Schematic of the control system for the SPS injection kickers. The 16 magnet modules (M1–M16) are powered in pairs of two. Each circuit is controlled by the delay of an electrical switch (S1–S8). The objective is to minimize deviations from the reference trajectory at a beam position monitor (BPM) downstream.

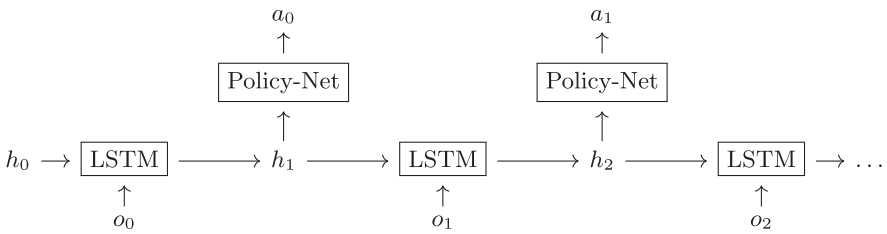

FIG. 2. Schematic of recurrent neural architecture for PPO. At each time step i, the LSTM network takes an observation o_i and the current internal state h_i as input and outputs the next internal state h_{i+1}. The policy network maps h_{i+1} to the parameters of the distribution from which the action a_i is drawn. The initial internal state of the LSTM (h_0) is a zero vector.

the weights at the kth gradient step. During training, ANNs are used to approximate the Q-function $Q^{\pi_w}: S \times A \to \mathbb{R}$ that approximates the expected on-policy continuation reward of a state-action combination and the value function $V^{\pi_w}: S \to \mathbb{R}$ that approximates the expected on-policy continuation reward from a state. The objective function for policy updates, stated in Eq. (1), relies on the advantage function $A^{\pi_w}(s,a) = Q^{\pi_w}(s,a) - V^{\pi_w}(s)$.

$$L_{w_k}(w) = \mathbb{E}\left[\min\left\{\frac{\pi_w(a|s)}{\pi_{w_k}(a|s)}A^{\pi_{w_k}}(s,a), g(\epsilon, A^{\pi_{w_k}}(s,a))\right\}\right],$$
$$g(\epsilon,A) = \begin{cases}(1+\epsilon)A & \text{if } A \geq 0\\ (1-\epsilon)A & \text{otherwise.}\end{cases} \quad (1)$$

Equation (1) increases the likelihood of actions with positive advantage and decreases the likelihood of actions with negative advantage. Taking the minimum ensures that the updated policy stays in the proximity of the previous policy [37].

D. PPO with history

The agent's observations in a POMDP may lack crucial information on the state that may be deduced from the history of observations, actions, and rewards (Sec. III B). An LSTM network can be integrated into the policy network of PPO, enabling it to leverage the information contained in the history. At each step, the LSTM network takes the current observation and its current internal state as input and outputs the next internal state. The policy network itself is fully connected and maps the internal state of the LSTM network to the distribution parameters [38]. Figure 2 illustrates this concept. In the present study, we employ a reference implementation of LSTM-PPO (stable-baselines3 [39,40], stable-baselines3-contrib [39]).

E. Data

To set up the simulation environment, we used 100 waveform measurements from the SPS injection system gathered between September 2021 and August 2022. Each set contains 16 waveforms, one per kicker magnet. Each waveform is measured over 40,000 ns with a spacing of 2 ns. Consequently, each waveform is a time series of 20,000 datapoints. The data used in this study are publicly available [9].

F. Action space

The action space corresponds to adjustments to the eight kicker delays (see Sec. III A for details). Actions are encoded as $a_t \in [-1,1]^8$. Actions are converted to nanoseconds by multiplication with a suitably chosen scaling factor $\delta_{\max}$, which we refer to as step size, and the delays $\tau \in [0,\infty)^8$ are updated according to $\tau_{t+1} = \tau_t + \delta_{\max} a_t$. Figure 3 displays a waveform, the corresponding kicks with a batch spacing of 200 ns, and illustrates the effect of taking an action.

G. Observation space

The observation space consists of the values of the 8 waveforms on an equidistant grid of size 33 covering a time window of 256 ns (see Fig. 4). There are thus 264 numbers describing the waveforms to the agent. We additionally provide the agent with the horizontal deviations of the injected and the circulating beam, making for an observed state of 266 dimensions, which we represent in the hypercube $[-1,1]^{266}$. Our preprocessing of the waveforms consists of setting any negative values in the waveforms

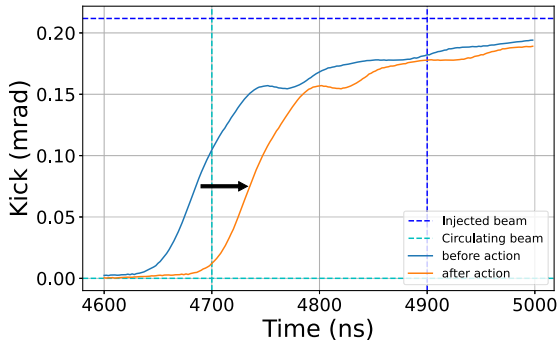

FIG. 3. The effect of an action on a single waveform. The vertical dashed lines indicate the timestamps of the circulating and the injected beam which are spaced by 200 ns. The horizontal dashed lines correspond to the design strength of the kicks.

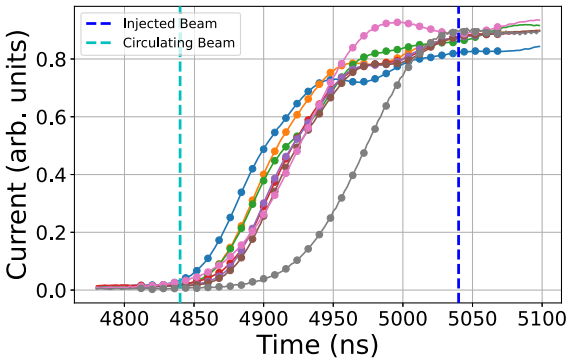

FIG. 4. Visualization of the observation space. Each waveform is presented to the agent as 33 discrete points.

to zero and then using the minimum and the maximum of each waveform to scale the waveforms' values to the interval $[-1, 1]$. The 17th waveform value in our grid corresponds to the timestamp exactly halfway between the circulating and injected beam. The time window is chosen relative to the current position of the injected beam. In the real hardware, the corresponding time has to be estimated.

H. Rewards

Let x_i and x_c be the horizontal beam deviations of the injected and the circulating beam, respectively. We want to minimize the loss function

$$L(x_i, x_c) = x_i^2 + x_c^2 + (x_i - x_c)^2, \tag{2}$$

that is the sum of the squared deviations of the injected and the circulating beam from their design trajectories and the squared deviation of the injected and the circulating beam from each other. The reward function of the agent is a negative affine transformation of this loss function, namely $R(x_i, x_c) = \theta_1 - \theta_2 L(x_i, x_c)$ for conveniently chosen $\theta_1, \theta_2 > 0$.

I. Starting state

The initial delays $\tau_0 \in \mathbb{R}^8$ are randomly drawn at the beginning of each simulated episode according to $(\tau_0)_i = u + v_i, i = 1, \ldots, 8$. Here, $u \sim N(4750\ \text{ns}, 10\ \text{ns})$ represents a time difference from the start of the injection sequence, which is applied as a delay to all eight switches. Furthermore, $v_i \sim N(0\ \text{ns}, 5\ \text{ns})$ $i = 1, \ldots, 8$ is an individual delay for each switch, applied additionally. For each episode, one of 80 sets of waveforms is randomly chosen and augmented. For each switch, a stretching factor $f_i \sim U(0, 0.2)$ is independently drawn. New data points are generated by linear interpolation of adjacent points to obtain a waveform of the required length. As data points are still considered to be separated by 2 ns, the waveform is effectively stretched by the factor $(1 + f_i)$. We do not stretch the entire waveform because doing so might shift the rising edge by an unrealistic amount.

J. Episode end

The control problem is conceptualized as a continuing problem without terminal states. During training, we truncate an episode in two cases: First, if there have been 1000 adjustment steps, or, second, if at least one of the delays differs by more than 1000 ns from its starting value. Unlike terminal states, which imply a zero continuation value, the agent is expected to continue collecting rewards beyond the truncation point. An estimate of this continuation value is then taken into account when updating the policy network. The intention of this procedure is to expose agents to diverse waveforms, which are resampled at the start of each episode, and to avoid wasteful exploration extremely far from the optimum.

K. Simulation of the SPS injection system

Using the current delays, the kicks imparted to the circulating and the injected beam by the magnets are calculated. This is illustrated in Fig. 3, where the kicks correspond to the amplitudes of the waveform at the intersections with the dashed vertical lines, representing the timestamps of the circulating and the injected beam. To further encourage the learning of a noise-robust strategy, time shifts are applied to the injected and the circulating beam during our simulations. These shifts, which we call hidden delays, do not affect the waveforms provided to the agent. For each switch, the hidden delay τ_i^* is randomly determined at the start of each episode and equals

$$\tau_i^* = c + d_i, i, \ldots, 8, \tag{3}$$

where $i \in \{1, \ldots, 8\}$, $c \sim U(-20\ \text{ns}, 20\ \text{ns})$, and $d_i \sim U(-15\ \text{ns}, 15\ \text{ns})$. The common shift c simulates an error in the estimation of the injected beam's detection time. The individual shifts d_i represent errors in the measurements of the waveforms. Due to these unobserved hidden delays, the environment is partially observable. We also performed experiments on a version of the environment with these hidden delays set to zero, which we refer to as the fully observable environment.

Starting from the center of the first vacuum tank, we simulate the transport of the beam centroids of the two beams to a beam position monitor (BPM). Initially, both centroids are assumed to be on the reference trajectory; hence, their initial condition equals zero. To remain on the reference trajectory, the circulating beam would have to receive zero deflection. On the other hand, the injected beam would have to receive a specific total deflection (see Fig. 3). The differences between the occurred kicks and these ideal kicks are calculated. Then, the simulation aggregates the kick differences per vacuum tank and applies them as kicks to the beam at the center of the respective tank (thin kick approximation). Hence, four kicks are applied to each beam. Finally, the beam centroids

are propagated from the last vacuum tank to the BPM via a second-order Taylor approximation of the transfer map, which was calculated with MAD-X [41].

IV. RESULTS

A. Performance in the simulation environment

The maximal step size (Sec. III F) of the environment can be chosen freely. The training complexity seems to increase with the maximal step size and the simulation-reality gap might widen due to the absence of highly nonlinear effects in our simulation environment. Four different step sizes (5, 10, 15, and 20 ns) were considered. Hyperparamter tuning was conducted for each step size. Due to the results of these experiments (Appendix B), we settled on the step size of 15 ns. The obtained agents were evaluated against the numerical optimization algorithm that is currently employed at CERN, BOBYQA, with a global optimization heuristic [42,43]. The agents were evaluated over 10,000 seeded episodes with a length of 1000 steps each. For each of those episodes, the median loss was calculated. Afterward, the same seeds were used to run BOBYQA 10,000 times with a budget of 1000 steps for each optimization. The smallest obtained loss and the number of steps required to reach this minimum were recorded for each of these optimizations. We report the percentage relative difference in loss. The relative loss difference is appropriate since the minimum loss varies with the waveforms, which are randomized each episode. The results of LSTM-PPO (see Appendix C for hyperparameters) for fully and partially observable environments are shown in Fig. 5. In the partially observable case, hidden delays are enabled, while in the fully observable case, the hidden delays are set to zero (see Sec. III K). The unrealistic case of a fully observable training and testing environment is provided for reference (Fig. 5, left). While the agent trained on the fully observable environment performed very well when the evaluation environment was also fully observable, its performance degraded severely when evaluated on the partially observable environment (Fig. 5, middle). This mimics transferring an agent from a fully observable simulation to the partially observable real accelerator and highlights the importance of including uncertainties in the training environment. The practically relevant case corresponds to training and testing on the partially observable environment (Fig. 5, right). Here, LSTM-PPO performed on average 2.73% worse than BOBYQA. During experiments on the accelerator, deviations less than 5% from the optimum found through numerical optimization yielded satisfying injection performance. Hence, the agents performance is well within the acceptable range. Furthermore, the median number of steps required by LSTM-PPO to reach stable behavior is 7, while the analogous number for BOBYQA was 246. Therefore, the use of RL allows a significant speed-up.

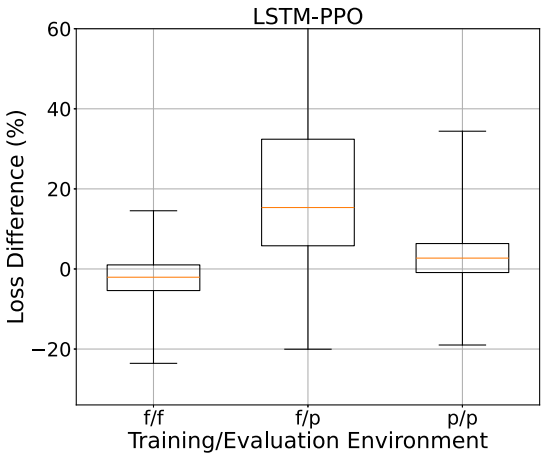

FIG. 5. Boxplots of relative differences in loss between LSTM-PPO and BOBYQA over 10,000 episodes. Left: training and evaluation on a fully observable environment (f). Middle: training on a fully observable environment and evaluation on a partially observable environment (p). The whisker was cut at 60% but extends to 184.12%. Right: training and evaluation on the partially observable environment. For the RL agents, the median loss of each episode was calculated, for BOBYQA, the minimal loss found in each episode was taken. Negative relative loss differences indicate that the agents outperformed BOBYQA.

B. Performance on the accelerator

An agent was trained in the simulated environment and directly deployed to the accelerator. There were two key steps during this transfer: determining the timestamp of the injected beam and shifting the waveform data such that the time of flight is taken into account. The first step provides a reference point to determine which points from the waveform data should be extracted for the state information. The second step removes a gap between simulation and reality: in the simulation environment, the entire waveform is processed at once. This allowed a convenient implementation and does not impact the optimization problem. However, in reality, it takes some time for the beam to travel between the magnets. Therefore, the raw waveform data have to be shifted accordingly.

LSTM-PPO was evaluated in two trials. Both trials were started with the optimized delays found by BOBYQA. After the agent took four steps, a random switch was selected and the corresponding delay was randomly changed by $\delta \in [-10, -5] \cup [5, 10]$. Thereafter, this procedure was repeated every seventh supercycle, so the agent had six opportunities to correct the settings after each perturbation. The size of the perturbations was rather small as the impact on other cycles during the experiments had to be small. The results of these trials are shown in Fig. 6. The agent purposefully moved toward a state that it deemed desirable and counteracted perturbations to recover that state. In both trials, the agent seemed to prefer the same settings. This stable state was not the optimum of the optimization problem, as some random perturbations yielded an improvement in

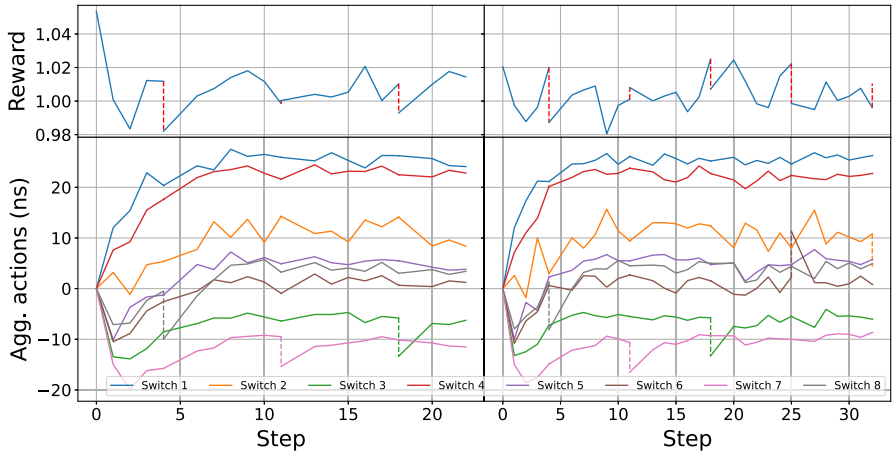

FIG. 6. Performance of LSTM-PPO during the two trials on the accelerator. Up: rewards during the trial. Down: cumulative changes in delays. The dashed vertical lines indicate a random perturbation.

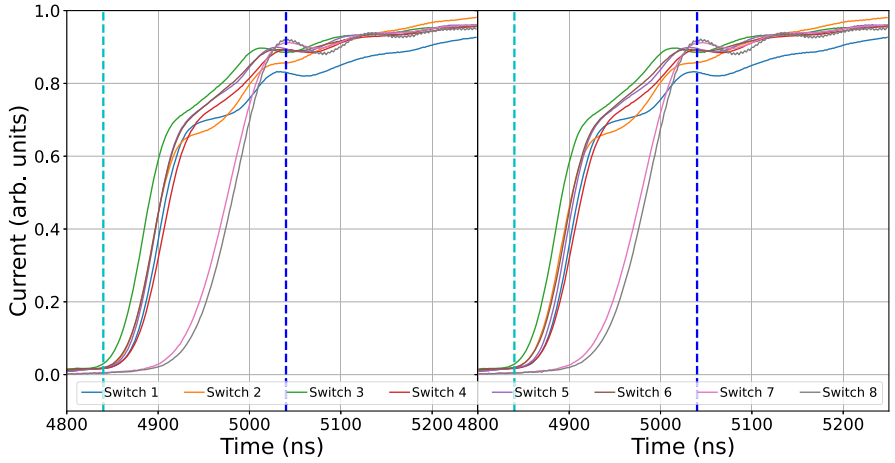

FIG. 7. The final waveforms of the two trials on the accelerator. Both configurations are valid working points, with a small perturbation to the circulating beam and large kicks to the injected beam.

reward (e.g., the second random action in the bottom-right panel of Fig. 6). Comparing the initial reward of each episode with later rewards, one can conclude that the agent performed slightly worse than BOBYQA. Nonetheless, the agent maintained adequate operational settings (see Fig. 7).

V. DISCUSSION

This paper has shown that a robust simulation environment, incorporating variations, such as different waveforms, random changes in rise time, and variability in initial delays, is essential to obtain effective agents. Hidden delays and simulated measurement errors lead to a formulation as a partially observable Markov decision problem. LSTM-PPO, designed for partially observable environments, proved effective. The agent's actions clearly showed the intention of reaching and maintaining a specific state close to the optimum found by numerical optimization (BOBYQA).

The present study seems to be the first successful application of RL with recurrent policy networks to accelerator physics. While one prior study evaluated a recurrent version of DDPG, it was limited to simulations [14]. Since many problems in accelerator control are partially observable, recurrent neural networks may be well suited for the domain. The present study demonstrates the feasibility of training RL agents in a simulation environment and successfully transferring them to real-world accelerator operations.

This paper trains its agent in a simulated environment. Even sample-efficient algorithms for online training are not applicable to our use case since variations of the system are sparse and infrequent (see Appendix A). Hence, one would have to collect training data over months, which is incompatible with accelerator operations.

Meta-RL mitigates the discrepancies between simulated and real-world environments by pretraining agents in

simulation, enabling them to rapidly adapt to new real-world conditions with minimal interaction. While this framework could be applicable to our use case, it inherently relies on an adaptation phase for each realization of the stochastic environment. In contrast, our agent is designed to act without requiring such an adaptation period, truly functioning as an active controller.

VI. CONCLUSION

When RL is applied to particle accelerator control, the challenge of transferring the agent from a simulated environment to the actual machine arises. This paper highlighted the importance of incorporating adequate variation and partial observability into the training environment. Recurrent policies, which are designed for partially observable environments, proved to be effective. During simulation, LSTM-PPO required an order of magnitude fewer steps than the numerical optimizer BOBYQA. The RL agent performed 2.73% worse than BOBYQA on average. Trials on the accelerator are consistent with the simulations: the agent optimizes much faster but attains slightly worse solutions.

Further research is required to completely replace BOBYQA with an RL agent and to deploy such an agent as an active controller during operation. Future research should also consider a representation of the waveforms that is invariant under a common temporal shift applied to all signals, thereby removing the need to estimate the time of injection. Adjustments to the simulation environment could reduce the simulation-reality gap (Appendix D). The results of this paper encourage further research on memory-based deep RL policies in accelerator physics.

DATA AVAILABILITY

The data that support the findings of this article are openly available [9].

APPENDIX A: RISE TIME ANALYSIS

Historical rise time data from May 26 until October 10, 2023 were analyzed for all eight magnet pairs: on the first switch of the MKP, there seem to be frequent changes in rise time. Such changes are detrimental to kicker performance. The second magnet exhibits similar behavior at a lower amplitude, as shown in Fig. 8. All other magnets appear to be rather stable within the observed time frame. Generally, significant changes of rise times are rare and might not occur at all over the course of months. To expose an agent to diverse experiences using the actual accelerator, one would have to train it for multiple months.

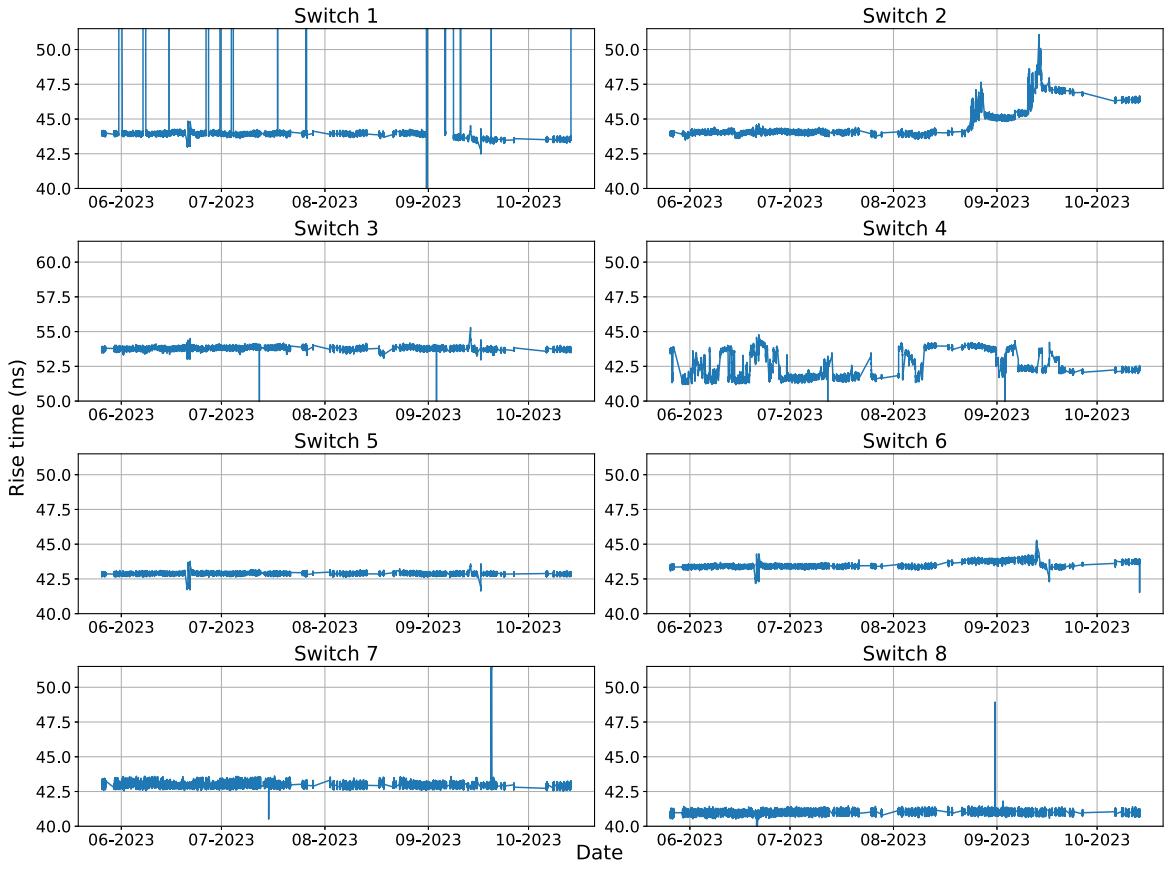

FIG. 8. The rise times of the eight magnet pairs between May 26, 2023 and October 10, 2023.

TABLE I. Descriptive statistics for the historical rise times of the eight magnet pairs.

	S1	S2	S3	S4	S5	S6	S7	S8
Mean	51.16	44.74	53.77	42.60	42.87	43.53	42.99	41.00
Std	19.51	1.11	0.30	0.94	0.12	0.24	0.31	0.12
Min	28.75	43.48	0.22	0.27	41.63	41.52	40.51	39.12
25%	43.84	43.98	53.71	41.75	42.84	43.38	42.89	40.93
50%	43.95	44.10	53.78	42.30	42.88	43.45	42.96	41.01
75%	44.03	45.30	53.85	43.66	42.93	43.72	43.05	41.09
Max	104.28	51.08	55.30	44.78	43.77	45.28	51.93	48.93

Disregarding the extreme variability of switch 1, the largest deviation from the median was 20.88%, exhibited by switch 7. The median rise time of switch 3 was up to 25.42% higher than the median rise times of switches 1, 2, 4, 5, and 6, which are of the same type. Descriptive statistics for all eight switches are provided in Table I. Based on these numbers, it was decided to increase the rise times of historical waveforms by up to 20% during agent training. Furthermore, the sets of waveforms collected for agent training were studied. The flattops, which are assumed to correspond to the design kick, were estimated using a method proposed by Waagard [2]: for each waveform, the data points less than 96% of its maximum were set to zero. Then we obtained, via differential evolution, the boxcar function that minimizes the sum of squared errors. The flattop estimate equals the height of the boxcar function. The waveforms are vaguely sigmoidal, with initially steep rising edges that reach an inflection point well before the flattop. Consequently, a nonmitigable error remains for the injected beam.

APPENDIX B: COMPARISON OF STEP SIZES

The agent changes the delays by $\delta \in [-\delta_{\max}, \delta_{\max}]^8$ each step, as outlined in Sec. III F. The maximum step size, $\delta_{\max}$, can be freely chosen; however, practical aspects should be considered. Given the median rise times of less than 50 ns, $\delta_{\max}$ can be set smaller than the batch spacing of 200 ns, and the optimal solution will still reachable within one step for all sensible initial delays. Another aspect is training complexity: given two environments with step sizes δ_1 and δ_2, where $\delta_1 < \delta_2$, every policy applicable to the first environment can also be applied to the second environment; however, the reverse does not hold true. Therefore, the training complexity increases with the step size. Furthermore, the simulation-reality gap might widen with increasing step size due to nonlinear effects or random variations in the accelerator.

Four different step sizes (5, 10, 15, and 20 ns) were evaluated empirically with simulated environments in order to find the optimal step size for the real environment. Those environments were fully observable as the option to include hidden delays was added in a later iteration. Hyperparamter tuning was conducted for each step size using the Optuna framework [44]: first, one has to define a search space by specifying an interval or a list of eligible values for each hyperparameter. Then, Optuna uses a sampler and a pruner for the hyperparameter optimization. The sampler suggests which combinations of hyperparameters to evaluate, and the pruner stops a trial early if certain termination conditions are met. Optuna readily provides different samplers and pruners. For pruning, MedianPruner was chosen, which cuts a trial short if it performed worse than the median performance of previous trials at intermediate evaluations [44]. For sampling, a simple grid search was performed. The evaluated hyperparameters and their

TABLE II. Optimal hyperparameter combinations and performance metrics for PPO, identified individually for four step sizes. The eligible values for each hyperparameter are shown in the column "Values." The loss of an episode is calculated as the median of the losses of all steps of that episode. Performance metrics were calculated with 100 evaluation episodes.

Hyperparameter	5 ns	10 ns	15 ns	20 ns	Values
Gamma	0.85	0.9	0.85	0.85	$\{0.85, 0.9, 0.95, 0.99\}$
Learning rate	3×10^{-5}	3×10^{-4}	3×10^{-4}	3×10^{-4}	$3 \cdot \{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}\}$
Batch size	16	64	8	16	$\{8, 16, 32, 64, 128\}$
No. of hidden layers	2	1	1	2	$\{1, 2\}$
No. of nodes per layer	64	128	32	16	$\{16, 32, 64, 128\}$
Performance metric					
Mean loss	2.419×10^{-4}	2.435×10^{-4}	2.431×10^{-4}	2.433×10^{-4}	
Median steps until stable	10	7	4	7	
Mean reward sum	683.27	680.19	681.25	680.52	

respective values for the grid search are given in Table II. All other available hyperparameters were set to the default chosen in the stable-baselines3 implementation of PPO (see [40] for the default values). The maximum budget per trial was 10^6 steps. Every 2.5×10^5 steps, the agents were evaluated on a differently seeded environment, and unpromising trials were pruned. The vast majority of the hyperparameters performed similarly, only the largest evaluated learning rate of 3×10^{-2} was clearly worse.

The stability of the behavior close to a minimum is another important aspect, especially for an active controller. In order to quantify stability, a formal definition is needed. Given the rewards $R_0, R_1, \ldots$ of an episode, the agent's behavior became stable after n steps if $R_m \geq 0.95 \max(R), \ \forall\, m \geq n$. The agents were compared based on the reward sum, the achieved loss and the amount of steps it took to become stable, which are all shown in Table II. The reward sums and losses were very close between all step sizes. The smallest step size performed the best in that regard. Stable behavior was reached the fastest with a step size of 15 ns. Overall, a step size of 15 ns seems to be the best trade-off between stability, speed, and achieved loss.

APPENDIX C: HYPERPARAMTER-TUNING FOR LSTM-PPO

The eligible values and the best combination of hyperparameters for LSTM-PPO are shown in Table III. Again, the Optuna framework [44] was used for hyperparameter tuning. For pruning, MedianPruner was chosen as for PPO. For sampling, TPESampler was used, which relies on the Tree of Parzen Estimators (TPE) algorithm [44]. For each hyperparameter, the TPE algorithm splits the values used in previous trials into two groups according to the loss that was achieved in their respective trials. Then, the algorithm fits a Gaussian mixture model (GMM) $l(x)$ to the well-performing hyperparameter values and another GMM $g(x)$ to the remaining hyperparameter values before each trial. Finally, it chooses the hyperparameter value for the trial by maximizing $l(x)/g(x)$ [45,46].

APPENDIX D: SIMULATION-REALITY GAP

The logged waveforms and the estimated timestamp of the injected beam (5040 ns) from the accelerator trials were used to parametrize the simulation environment, thereby matching the simulation to the trials. We retrieved the simulated BPM data and the corresponding reward. By comparing this data to the real data collected during the trials, one can elucidate the simulation-reality gap. Figure 9 (left panel) reveals that the horizontal deviations of the injected beam are significantly smaller in reality than in the simulation. For the circulating beam, the reverse holds true, the horizontal deviations in the accelerator are much larger than expected (Fig. 9, middle panel). Both effects could be caused by an incorrect estimation of the flat top. As the linear conversion from voltage to kick depends on the flattop (which was assumed to correspond to the design kick for the injected beam), a lower estimate of the flattop would result in larger simulated kicks. In turn, the deviations of the injected beam would decrease, and the deviations of the circulating beam would increase, closing the gap between simulation and accelerator. The difference in scale between the injected and circulating beam, in conjunction with the quadratic loss function, causes the loss to be more sensitive toward the deviation of the injected beam. Also, the role of the term in the loss function that penalizes differences between the deviations of the beams (Sec. III H) is overemphasized. This can be seen in Fig. 9 (Right panel) by comparing the simulated deviations and reward of step 2 and step 4. Both the deviations of the injected and circulating beam decreased from step 2 to step 4. Yet the reward of step 2 is larger than the reward of step 4 because the improvement of the deviations of the

TABLE III. Hyperparameter tuning for LSTM-PPO. The eligible values for the TPE algorithm were either intervals or sets and are displayed in the second column. The third column shows the best combination of hyperparameters found during the study.

Hyperparameter	Values	Best
Gamma	$[0.5, 1]$	0.63
Learning rate	$[10^{-5}, 1]$	5.47×10^{-5}
n_steps	$\{2^3, 2^4, 2^5, 2^6, 2^7, 2^8, 2^9, 2^{10}, 2^{11}\}$	2^{11}
Batch size	$\{2^3, 2^4, 2^5, 2^6, 2^7, 2^8, 2^9\}$	2^8
n_epochs	$\{1, 5, 10, 20\}$	20
ent_coef	$[10^{-8}, 10^{-1}]$	2×10^{-3}
gae_lambda	$\{0.8, 0.9, 0.92, 0.95, 0.98, 0.99, 1\}$	0.92
max_grad_norm	$\{0.3, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 2, 5\}$	2
clip_range	$\{0.1, 0.2, 0.3, 0.4\}$	0.1
vf_coef	$[0, 1]$	0.95
No. of hidden layers	$\{1, 2\}$	1
No. of nodes per layer	$\{16, 32, 64, 128\}$	32
lstm hidden size	$\{8, 16, 32, 64\}$	32

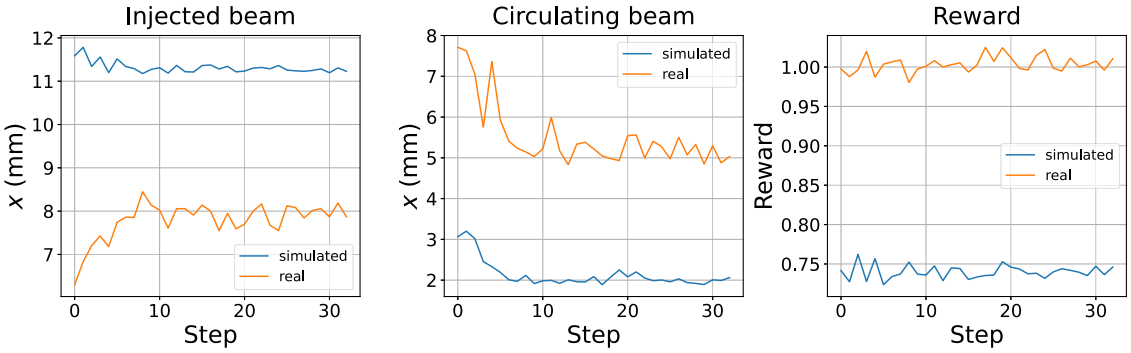

FIG. 9. Comparison of simulation and accelerator with respect to the horizontal deviations of the injected beam (left panel), the horizontal deviations of the circulating beam (central panel) and reward (right panel). The waveforms and BPM data are taken from the second evaluation episode on the accelerator (see Fig. 6).

circulating beam is overpowered by the third term of the loss function.

The general trend of the deviations is similar for the circulating beam but opposite for the injected beam. An incorrect estimation of the timestamp of the injected beam (general shift) and errors in the waveform measurements (individual shifts) could explain this effect: the true values of the shifts are unknown on the accelerator and cannot be taken into account before feeding the waveforms into the simulation. Due to the ripple after the rising edge of the waveforms, the injected beam is more sensitive toward these shifts. The training environment includes random general and individual shifts, emulating this mismatch between waveforms and BPM data (Sec. III K). Even though the agents have been exposed to such shifts during training, their performance could still have been negatively impacted (see Fig. 5).

In the simulated environment, the shape of the waveforms is determined at the start of each episode and does not change throughout. On the accelerator, many small changes might occur during an episode. Figure 6 shows two evaluation episodes on the accelerator. The second episode was started less than 30 min after the first episode and with the same initial delays. A single episode on the simulation environment would cover multiple hours of accelerator operation. Therefore, it is reasonable to compare the initial rewards and waveforms of the two episodes on the accelerator to estimate intraepisode variation. The initial rewards of these episodes differ by 3.2%. The respective waveforms differ by up to 13.3%. The largest deviations occur at the start of the rising edge, likely since the steep slope renders the system more sensitive. After the halfway point toward the flattop, the deviations are much smaller, at most 3.6%.

[1] W. Bartmann, M. Barnes, J. Boyd, E. Carlier, A. Chmielinska, B. Goddard, G. Kotzian, C. Schwick, L. Stoel, D. Valuch, F. Velotti, V. Vlachodimitropoulos, and C. Wiesner, Impact of LHC and SPS injection kicker rise times on LHC filling schemes and luminosity reach, in *Proceedings of the International Particle Accelerator Conference (IPAC-2017)* (JACoW, Geneva, Switzerland, 2017), pp. 2043–2046.

[2] E. Waagaard, Optimising the present and designing the future: A novel SPS injection system, Master's thesis, Uppsala University, Uppsala, Sweden, 2022, https://cds.cern.ch/record/2813209.

[3] M. Powell, The BOBYQA algorithm for bound constrained optimization without derivatives, Technical Report No. 2009/NA06, University of Cambridge, Department of Applied Mathematics and Theoretical Physics, 2009.

[4] I. Bogunovic, J. Scarlett, and V. Cevher, Time-varying Gaussian process bandit optimization, in *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, Volume 51 of Proceedings of Machine Learning Research, Cadiz, Spain*, edited by A. Gretton and C. C. Robert (2016), pp. 314–323, https://proceedings.mlr.press/v51/bogunovic16.html.

[5] N. Kuklev *et al.*, Robust adaptive Bayesian optimization, in *Proceedings of the 14th International Particle Accelerator Conference, IPAC-2023* (JACoW, Geneva, Switzerland, 2023), pp. 4428–4431, 10.18429/JACoW-IPAC2023-THPL007.

[6] N. Kuklev, M. Borland, G. Fystro, H. Shang, and Y. Sun, Online accelerator tuning with adaptive Bayesian optimization, in *Proceedings of the 5th International Particle Accelerator Conference (NAPAC'22), Number 5 in International Particle Accelerator Conference* (JACoW, Geneva, Switzerland, 2022), Vol. 10, pp. 842–845, 10.18429/JACoW-NAPAC2022-THXD4.

[7] R. Roussel *et al.*, Bayesian optimization algorithms for accelerator physics, Phys. Rev. Accel. Beams **27**, 084801 (2024).

[8] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, Proximal policy optimization algorithms, arXiv:1707.06347.

[9] M. Remta, mkp-delays-rl, GitLab, 2024, https://gitlab.cern.ch/mremta/mkp-delays-rl [accessed January 8, 2025].

[10] L. Grech, G. Valentino, D. Alves, and S. Hirlaender, Application of reinforcement learning in the LHC tune feedback, Front. Phys. **10**, 929064 (2022).

[11] V. Kain, S. Hirlander, B. Goddard, F. M. Velotti, G. Z. Della Porta, N. Bruchon, and G. Valentino, Sample-efficient reinforcement learning for CERN accelerator control, Phys. Rev. Accel. Beams **23**, 124801 (2020).

[12] M. Schenk, E. F. Combarro, M. Grossi, V. Kain, K. S. B. Li, M.-M. Popa, and S. Vallecorsa, Hybrid actor-critic algorithm for quantum reinforcement learning at CERN beam lines, arXiv:2209.11044.

[13] F. Velotti, B. Goddard, V. Kain, R. Ramjiawan, G. Porta, and S. Hirlander, Towards automatic setup of 18 MeV electron beamline using machine learning, Mach. Learn. **4**, 025016 (2023).

[14] D. Wang, H. Bagri, C. Macdonald, S. Kiy, P. Jung, O. Shelbaya, T. Planche, W. Fedorko, R. Baartman, and O. Kester, Accelerator tuning with deep reinforcement learning, in *Proceedings of the 4th Workshop on Machine Learning and the Physical Sciences* (2021), https://ml4physicalsciences.github.io/2021/files/NeurIPS_ML4PS_2021_125.pdf.

[15] X. Chen, Y. Jia, X. Qi, Z. Wang, and Y. He, Orbit correction based on improved reinforcement learning algorithm, Phys. Rev. Accel. Beams **26**, 044601 (2023).

[16] N. Madysa, R. Alemany-Fernández, N. Biancacci, B. Goddard, V. Kain, and F. Velotti, Automated intensity optimisation using reinforcement learning at LEIR, in *Proceedings of the13th International Particle Accelerator Conference, IPAC-2022, Bangkok, Thailand* (JACoW, Geneva, Switzerland, 2022), pp. 941–944.

[17] M. Schram, K. Rajput, N. S. K. S., P. Li, J. St. John, and H. Sharma, Uncertainty aware machine-learning-based surrogate models for particle accelerators: Study at the Fermilab booster accelerator complex, Phys. Rev. Accel. Beams **26**, 044602 (2023).

[18] J. Kaiser, O. Stein, and A. Eichler, Learning-based optimisation of particle accelerators under partial observability without real-world training, in *Proceedings of the 39th International Conference on Machine Learning* (2022), pp. 10575–10585, https://proceedings.mlr.press/v162/kaiser22a.html.

[19] J. Kaiser, C. Xu, A. Eichler, A. S. Garcia, O. Stein, E. Bründermann, W. Kuropka, H. Dinter, F. Mayet, T. Vinatier, F. Burkart, and H. Schlarb, Learning to do or learning while doing: Reinforcement learning and Bayesian optimisation for online continuous tuning, arXiv:2306.03739.

[20] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, Continuous deep Q-learning with model-based acceleration, in *Proceedings of the 33rd International Conference on Machine Learning* (2016), pp. 2829–2838, https://proceedings.mlr.press/v48/gu16.html.

[21] S. Fujimoto, H. van Hoof, and D. Meger, Addressing function approximation error in actor-critic methods, in *Proceedings of the 35th International Conference on Machine Learning* (2018), pp. 1587–1596, https://proceedings.mlr.press/v80/fujimoto18a.html.

[22] S. Hirlaender, S. Pochaba, L. Lamminger, A. Santamaria Garcia, C. Xu, J. Kaiser, A. Eichler, and V. Kain, Deep meta reinforcement learning for rapid adaptation in linear Markov decision processes: Applications to CERN's AWAKE project, in *Volume 1458 of Advances in Intelligent Systems and Computing* (Springer Nature, Switzerland, 2024), 10.1007/978-3-031-65993-5_21.

[23] C. Finn, P. Abbeel, and S. Levine, Model-agnostic meta-learning for fast adaptation of deep networks, in *Proceedings of the 34th International Conference on Machine Learning, Volume 70 of Proceedings of Machine Learning Research*, edited by D. Precup and Y. W. Teh (2017), pp. 1126–1135, https://proceedings.mlr.press/v70/finn17a.html.

[24] D. Meier, L. V. Ramirez, J. Völker, J. Viefhaus, B. Sick, and G. Hartmann, Optimizing a superconducting radio-frequency gun using deep reinforcement learning, Phys. Rev. Accel. Beams **25**, 104604 (2022).

[25] J. St. John, C. Herwig, D. Kafkes, J. Mitrevski, W. A. Pellico, G. N. Perdue, A. Quintero-Parra, B. A. Schupbach, K. Seiya, N. Tran, M. Schram, J. M. Duarte, Y. Huang, and R. Keller, Real-time artificial intelligence for accelerator control: A study at the Fermilab booster, Phys. Rev. Accel. Beams **24**, 104601 (2021).

[26] H. Hasselt, Double q-learning, in *Advances in Neural Information Processing Systems*, edited by J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta (Curran Associates, Inc., Red Hook, NY, 2010), Vol. 23, https://proceedings.neurips.cc/paper_files/paper/2010/file/091d584fced301b442654dd8c23b3fc9-Paper.pdf.

[27] N. Bruchon, G. Fenu, G. Gaio, M. Lonza, F. H. O'Shea, F. A. Pellegrino, and E. Salvato, Basic reinforcement learning techniques to control the intensity of a seeded free-electron laser, Electronics **9**, 781 (2020).

[28] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, Playing Atari with deep reinforcement learning, arXiv:1312.5602.

[29] R. J. Williams, Simple statistical gradient-following algorithms for connectionist reinforcement learning, Mach. Learn. **8**, 229 (1992).

[30] S. M. Kakade, A natural policy gradient, in *Proceedings of the Advances in Neural Information Processing Systems 14, NIPS 2001* (2001), https://proceedings.neurips.cc/paper_files/paper/2001/file/4b86abe48d358ecf194c56c69108433e-Paper.pdf.

[31] N. Bruchon, G. Fenu, G. Gaio, S. Hirlaender, M. Lonza, F. A. Pellegrino, and E. Salvato, An online iterative linear quadratic approach for a satisfactory working point attainment at Fermi, Information **12**, 262 (2021).

[32] F. H. O'Shea, N. Bruchon, and G. Gaio, Policy gradient methods for free-electron laser and terahertz source optimization and stabilization at the Fermi free-electron laser at Elettra, Phys. Rev. Accel. Beams **23**, 122802 (2020).

[33] S. Hochreiter and J. Schmidhuber, Long short-term memory, Neural Comput. **9**, 1735 (1997).

[34] E. Waagaard, M. Barnes, W. Bartmann, V. Bencini, J. Borburgh, L. Ducimetière, T. Kramer, T. Stadlbauer, P. Trubacova, and F. Velotti, Design of a new CERN SPS injection system via numerical optimisation, in *Proceedings of the 14th International Particle Accelerator Conference (IPAC-2023)* (JACoW, Geneva, Switzerland, 2023), pp. 267–270.

[35] J. Uythoven, The new SPS injection channel, in *Proceedings of the 9th LEP Performance Workshop* (1999), pp. 120–125, https://cds.cern.ch/record/398068.

[36] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. (MIT Press, Cambridge, MA, 2018).

[37] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, Proximal policy optimization algorithms, arXiv:1707.06347.

[38] StableBaselines3, Recurrent PPO, https://sb3-contrib.readthedocs.io/en/master/modules/ppo_recurrent.html (2024) [accessed April 26, 2024].

[39] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, Stable-baselines3: Reliable reinforcement learning implementations, J. Mach. Learn. Res. **22**, 1 (2021), http://jmlr.org/papers/v22/20-1364.html.

[40] StableBaselines3, PPO, https://stable-baselines3.readthedocs.io/en/master/modules/ppo.html (2024) [accessed February 12, 2024].

[41] MAD-X project, MAD-X: Methodical accelerator design, https://cern.ch/madx (2025) [accessed June 2, 2025].

[42] C. Cartis, J. Fiala, B. Marteau, and L. Roberts, Improving the flexibility and robustness of model-based derivative-free optimization solvers, ACM Trans. Math. Softw. **45**, 1 (2019).

[43] C. Cartis, L. Roberts, and O. Sheridan-Methven, Escaping local minima with local derivative-free methods: A numerical investigation, Optimization **71**, 2343 (2021).

[44] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, Optuna: A next-generation hyperparameter optimization framework, in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (2019), pp. 2623–2631, 10.1145/3292500.3330701.

[45] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, Algorithms for hyper-parameter optimization, in *Proceedings of the 24th International Conference on Neural Information Processing Systems* (2011), pp. 2546–2554, https://proceedings.neurips.cc/paper_files/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf.

[46] J. Bergstra, D. Yamins, and D. Cox, Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures, in *Proceedings of the 30th International Conference on Machine Learning, Atlanta, Georgia* (2013), pp. 115–123, https://proceedings.mlr.press/v28/bergstra13.html.