












Real-time Artificial Intelligence for Accelerator Control: A Study at the Fermilab Booster

Jason St. John ^{*}, Christian Herwig , Diana Kafkes ^s, Jovan Mitrevski , William A. Pellico ,
Gabriel N. Perdue , Andres Quintero-Parra , Brian A. Schupbach , Kiyomi Seiya , and Nhan Tran 
Fermi National Accelerator Laboratory, Batavia, Illinois 60510, USA

Malachi Schram 

Thomas Jefferson National Accelerator Laboratory, Newport News, VA 23606, USA

Javier M. Duarte 

University of California San Diego, La Jolla, California 92093, USA

Yunzhi Huang 

Pacific Northwest National Laboratory, Richland, Washington 99352, USA

Rachael Keller

*Department of Applied Physics and Applied Mathematics,
Columbia University, New York, New York 10027, USA*

(Received 5 January 2021; Accepted 16 August 2021; Published 18 October 2021)

We describe a method for precisely regulating the Gradient Magnet Power Supply (GMPS) at the Fermilab Booster accelerator complex using a neural network trained via reinforcement learning. We demonstrate preliminary results by training a surrogate machine-learning model on real accelerator data to emulate the GMPS, and using this surrogate model in turn to train the neural network for its regulation task. We additionally show how the neural networks to be deployed for control purposes may be compiled to execute on field-programmable gate arrays (FPGAs), and show the first machine-learning based control algorithm implemented on an FPGA for controls at the Fermilab accelerator complex. As there are no surprise latencies on an FPGA, this capability is important for operational stability in complicated environments such as an accelerator facility.

DOI: [10.1103/PhysRevAccelBeams.24.104601](https://doi.org/10.1103/PhysRevAccelBeams.24.104601)

I. INTRODUCTION

Particle accelerators are among the most complex engineering systems in the world. They are crucially important to the study of the elementary constituents of matter and the forces governing their interactions. Tuning and controlling accelerators is challenging and time consuming, but even marginal improvements can translate very efficiently into improved scientific yield for an experimental particle physics program, where integrated accelerator run time imposes a substantial cost.

To date, the most common approach to accelerator systems control has largely consisted of hand tuning by experts, and has been guided by physical principles whenever possible. However, accelerator physics is complex and highly nonlinear. While we may model accelerator beams with impressive and improving precision [1], building fully comprehensive Monte Carlo-based models of entire facilities is challenging, often leaving optimization and control to the intuition of experienced experts. Even though this process has been successful to date, it

is arduous and likely contains hidden inefficiencies. Recently, deep learning [2–4], the training of neural networks consisting of many hidden layers, has proven itself useful for complex control problems [5–8]. Notably, processes within accelerator complexes occur between microsecond and millisecond timescales, much faster than human operators can react.

In this study, we present a real-time artificial intelligence (AI) control system for precisely regulating an important subsystem of the Fermilab Booster accelerator complex [9]. Our ultimate goal is to achieve a ten-fold improvement in precision for the regulation system we describe in this paper. To realize this system, we use machine learning (ML) for two key features: to develop *surrogate models* [10] that reproduce the behaviors of the real-world system, and to train an *online agent* to take control actions in the system. The online agent is developed using reinforcement learning (RL) [11, 12], a framework in which an artificial agent learns by interacting with its environment. This online agent will ultimately control the actual accelerator system. The surrogate model provides an initial “safe” environment in which to train and evaluate control algorithms, and does so at higher rates and in a more controlled fashion than real-time training could allow. In this work, to

^{*} stjohn@fnal.gov

demonstrate the complete methodology, we show how the control algorithms perform within the surrogate model.

Additionally, we propose a scheme for the first ML control system implemented in FPGA firmware at the Fermilab accelerator complex. Our approach utilizes field-programmable gate arrays (FPGAs) for real-time responses and includes full integration into the Booster controls system for high-speed data ingestion. On-board implementation, with dedicated hardware, is important for operational stability and for reliable low-latency response times. In advance of commissioning the full system during online operations, we present important results from running a pretrained, static RL model in FPGA test benches.

Fully realizing this system will inevitably create an important and versatile set of tools that could find application in many areas in accelerator controls and monitoring, thereby paving the way for more ambitious control schemes. These tools may allow for more complex agents to work in tandem to manage ever-larger portions of the accelerator complex. While open questions remain as to whether data-intensive RL algorithms can be effectively used in this context without an independent, high-quality simulator, the results presented in this paper are encouraging.

In Sec. II we briefly highlight some previous publications at the intersection of machine learning and accelerator control. Section III describes the Fermilab Booster accelerator and surrounding complex in sufficient detail to understand the specific controls problem we address. In Sec. IV, we describe the accelerator data used to train a surrogate model, as well as the data processing used to support our ML workflow. Next, in Sec. V we describe the ML algorithms for the surrogate model and candidate agents. Then, in Sec. VI we briefly detail the process of deploying ML control algorithms to an FPGA. Section VII concludes with a discussion of our plans for full RL-based control.

II. PREVIOUS WORK

The field of AI for accelerators is quite active in the development of new applications. AI algorithms are poised to play a salient role in accelerator control, tuning, diagnostics, and modeling. Here we provide a brief overview with a focus on recent work.

Predictive diagnostics are important precursors for control networks because they enable modeling of accelerator functions to a high degree of precision. The authors of Ref. [13] use ML-based diagnostics to predict the longitudinal phase space distribution in a particle accelerator. Reference [14] offers an example of AI applications for fault classification in superconducting radio frequency cavities. Furthermore, in groundbreaking conceptual studies [15–20], the authors built predictive networks for accelerator modeling that could serve as predecessors to control networks. See also [21].

In Ref. [22] the authors use a neural network to demonstrate control of the longitudinal phase space of relativistic electron beams with very fine time resolution. However, as they and others have found, methods based on *a priori* models face serious challenges in regard to managing complexity. A very recent study, found in Ref. [23], explores some of the trade-offs between model-based and model-free training methods in an RL-based context and find some interesting advantages for each approach. Here we seek to train, test, and deploy model-free RL control algorithms, but nevertheless must initiate that training using a surrogate model that provides a safe environment in which to evaluate algorithms before deployment. Surrogate models are an important ML-based tool for understanding other ML models (among other things) at accelerators where it is impractical to train from scratch using real accelerator hardware due to the risk of downtime and the absence of precisely repeatable history. The authors of Ref. [24] use neural network (NN) based surrogate models for the Compact Linear Collider (CLIC) final-focus system. See also [15, 25, 26].

Additionally, ML algorithms for tuning have been studied at the Linac Coherent Light Source (LCLS) at SLAC National Accelerator Laboratory, and at the Swiss Free Electron Laser (FEL). Bayesian optimization of the Swiss FEL is explored in Ref. [27]. Likewise, in Ref. [28], the authors use Bayesian optimization via Gaussian processes for fast tuning in response to the need to change beam configurations frequently. Gaussian process models are sample efficient for producing accurate representations and uncertainties with limited data, but inefficient with respect to the number of samples in the dataset. Furthermore, in Ref. [29], the authors extend the Gaussian process-based methods to employ physics-based simulations in cases where archival data is not available. The challenge addressed in this paper does not involve an explicit physical model for use by such a simulation.

Reference [30] summarizes a workshop that functions as a partial review of ML for accelerator controls and diagnostics and Ref. [31] provides an overview of beam dynamics studies using AI at CERN. See also Ref. [32]. In the most closely related previous work [33], the authors approach the controls problem of maximization of the CERN Low Energy Ion Ring multi-turn injected intensity in a similar way to how we approach rapidly cycling magnet power supply stabilization here. Some of the same authors very recently published Ref. [34], which studies continuous, model-free RL training algorithms at some different parts of the CERN accelerator complex. Reference [35] similarly studies RL to control the RF system of the KIT Karlsruhe Research Accelerator (KARA) storage ring and improve microbunching instability. The authors of Refs. [36, 37] deploy model-free RL training algorithms at the FERMI free-electron laser at Elettra with promising results for alignment in the first paper and for optimization and stabilization in the second. As a proof of concept, the authors deploy a RL environment on an FPGA-ARM system for solving a classic cart-pole

control problem [38].

Fast NN inference has been explored on various edge devices, including FPGAs, for a variety of applications ranging from internet of things [39] to high energy physics [40]. Surveys of some existing toolflows can be found in Refs. [41–44], and include the FPGAConvNet library [45–48], FP-DNN [49], DNNWEAVER [50], CAF-FEINE [51], VITIS AI [52], the FINN project [53–55], FIXYNN and DEEPFREEZE [56, 57], HLS4ML [58–66], and others [67–70]. Many of these toolflows are specific to Xilinx FPGAs. In this work, we build our implementation using the HLS4ML library as it has been extended to Intel FPGAs and allows for an easy exploration of the NN design space, as explained in Sec. VI B.

Relative to the body of prior work on AI for accelerators, we offer new developments in the integration of realistic RL algorithms into embedded systems (FPGAs). By embedding the algorithm in an FPGA, we may take advantage of the very stable, low-latency performance offered by that platform, which is critical for controls in a particle accelerator. Further, we indicate how this result may be extended through the use of a statistical ensemble of agent models, enabling increased learning rates, prediction stability, and robustness against potential mode collapse in individual agents. Given models of sufficiently small size, FPGA implementation of the models enables us to run an ensemble of models in parallel in order to optimize decision stability at no additional cost to overall latency.

III. FERMILAB BOOSTER ACCELERATOR COMPLEX

A. Accelerator Environment of the GMPS Regulator

The Booster rapid-cycling synchrotron receives the 400 MeV (kinetic energy) beam from the Fermilab Linear Accelerator (Linac) via charge-exchange (H^- to H^+). This beam is accelerated to 8 GeV by synchronously raising (“ramping”) the Booster accelerator cavities’ frequency and the magnetic field of the combined-function bending and focusing electromagnets known as gradient magnets, which are powered by the gradient magnet power supply (GMPS) [71, 72]. The beam is extracted at peak kinetic energy, after which the system is returned to the injection state. This complete cycle repeats, sinusoidally varying the GMPS magnet current between programmed current minimum and maximum at 15 Hz.

Meanwhile, other nearby high-current, high-power electrical loads are varying in time, causing unwanted fluctuations of the actual GMPS electrical current, and thus fluctuations of the magnetic field in the Booster gradient magnets. The role of the GMPS regulator is to calculate and apply small compensating offsets in the GMPS driving signal, improving the agreement of the resulting minimum and maximum currents with their set

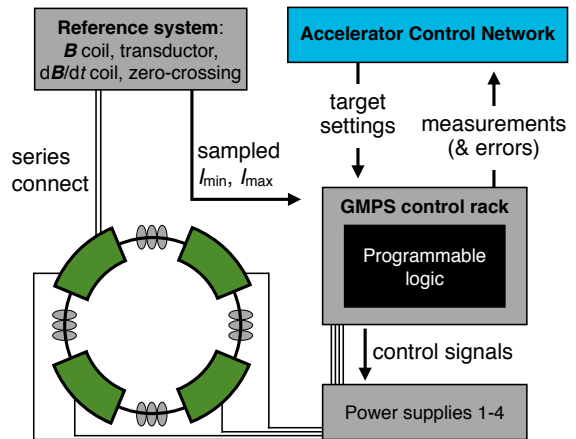


FIG. 1. Schematic view of the GMPS control environment. The human operator specifies a target program via the Accelerator Control Network that is transmitted to the GMPS control board. The FPGA-based control logic utilizes these settings together with readings from a reference magnet to prescribe a driving signal to the GMPS. The effect of this prescribed signal on the bending magnets is measured by an in-series reference magnet, with sampled readings transmitted back to the GMPS control board. Reference measurements and prescribed signals may be logged and transmitted over network for later analysis.

points. The present GMPS regulator system is a proportional–integral–derivative (PID) controller [73, 74]. Figure 1 shows a schematic overview of the GMPS control environment.

The power supplies that provide the combined DC and AC components of the desired gradient magnet current consist of a pair of three-phase series-connected silicon controlled rectifier [71] bridges fired at 720 Hz. An inductor-capacitor (LC) filter network at the output greatly reduces 720 Hz ripple, resulting in an output voltage that is proportional to the sinusoidal program provided by the GMPS regulator system. The series-connected electromagnet circuits and their cell capacitor banks are driven at resonance at 15 Hz and coupled via a distributed choke system for bypassing DC current. Figure 2 shows the power supply output voltage due to the cosine program.

For monitoring purposes, a special series-connected half-cell reference magnet located in the equipment gallery includes a pickup coil located between its poles to measure the time rate of change of the magnetic field \dot{B} , which is an important input to the GMPS regulator. Powered with the other gradient magnets and housed in a low-radiation environment without charged particle beam passing through it, this reference magnet provides an accurate representation of the magnetic field under control throughout the accelerator.

Timing information derived from $\dot{B} = 0$ synchronizes the GMPS regulator system to the minimum and maximum values of the magnetic field and provides a

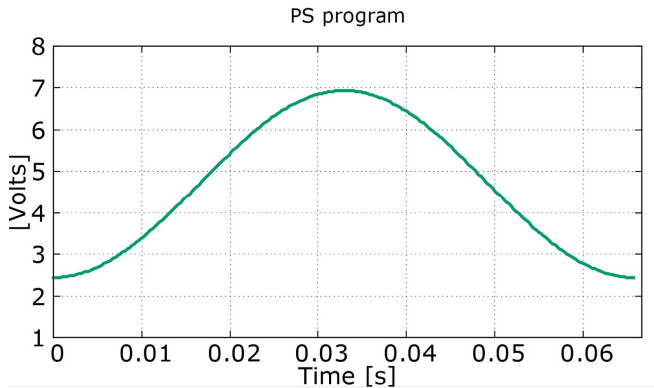


FIG. 2. A single 15 Hz cycle of the power supply program voltage, from one minimum to the next.

transistor-transistor-logic (TTL) based 15 Hz master clock signal that drives the timing system for the GMPS regulator and indeed the rest of the accelerator complex. The high-frequency sampled measurements near the minimum and maximum values of the magnetic field are automatically fitted each cycle, and the finite impulse response (FIR) parameters are used as the primary feedback mechanism for the GMPS regulator system. Reducing the errors (the difference between the target and realized GMPS current especially at injection) of the GMPS system is of primary concern in the operational performance and efficiency of the Booster. The following Sections discuss the details of the present and proposed regulation systems.

B. GMPS Regulation

The GMPS regulation system seeks to minimize the impact of disturbances due to environmental factors such as ambient temperature, nearby high-power pulsed RF systems, and ramping power supplies with inductive loads. Variations in the AC line frequency and amplitude are also significant sources of error, and are due in part to other particle accelerators in the complex changing currents in their own high-current electromagnets as part of their normal operations. Without regulation, the fitted minimum of the magnetic field may vary from the set point by as much as a few percent.

The existing regulator reduces GMPS regulation errors to roughly 0.1% of the set value by implementing a proportional-integral-derivative (PID) control scheme. Each cycle, the fitted minimum and maximum of the magnetic field reflect the combined influences of the set points, any compensation applied by the regulator for that cycle, and any new influence of other nearby electrical loads. Calculated estimates for the minimum and maximum values of the changing magnetic field of the previous 15 Hz cycle are used to adjust the power supply program to decrease the errors of the system. The cal-

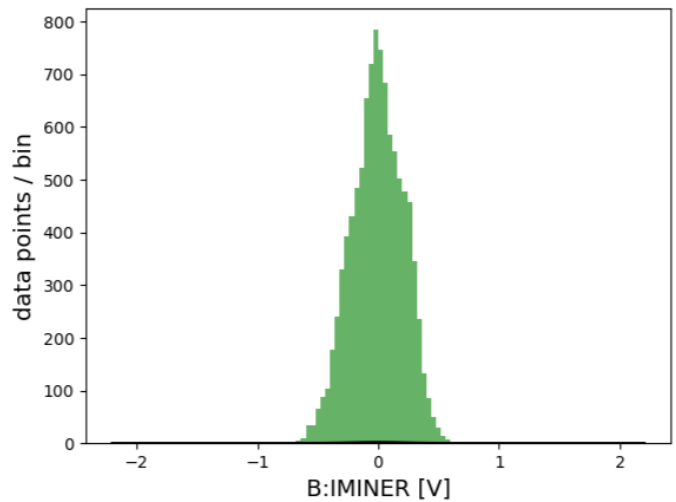


FIG. 3. Distribution of fractional measured error in the GMPS current at the minimum value of the magnet current, with the non-ML PID regulator discussed in the text.

culuation of the compensated minimum current $B:VIMIN$, used in the next cycle of the power supply program voltage, proceeds as follows at each time step t :

$$B:IMINER(t) = 10[I_{fit}^{min}(t) - I_{set}^{min}(t)], \quad (1)$$

$$\beta(t) = \Gamma(t)B:IMINER(t) + \beta(t-1), \quad (2)$$

$$B:VIMIN(t) = I_{set}^{min}(t) - \alpha(t)B:IMINER(t) - \beta(t), \quad (3)$$

where $B:IMINER(t)$ and $B:VIMIN(t)$ are the corresponding control system readings at time t ; I_{fit}^{min} is the measured (fitted) current minimum of the present cycle; I_{set}^{min} is the nominal set value of the current minimum; Γ is the integral gain; and α is the proportional gain. The running parameter β is effectively an integral of the error which rapidly downweights past measurements of the error. Typical values of the gain constants are $\Gamma = 7.535 \times 10^{-5}$ and $\alpha = 8.5 \times 10^{-2}$, adjustable by system experts. $B:IMINER$ is proportional to the error in the system, and for our purposes they are synonymous.

The environmental perturbations, discussed above, increase the distributed long-term steady-state errors of the GMPS system. A traditional PID regulation loop, given a sufficient amount of time and unchanging perturbations, will decrease the steady-state error of a system to zero. In reality, and within the timescale of the Booster beam cycle, the PID loop decreases the steady-state error to some nonzero error, typically of order 0.1% for the cycle minimum. See Figure 3 for a sample distribution of measured errors for the minimum value of the sinusoidally varying magnet current. Efforts to decrease the steady-state error further by adjusting the closed-loop gains to be more responsive would come at the cost of reduced overall system stability. Therefore, a balance between the steady-state error and stability of the system has been struck.

We seek to improve upon this regulation performance in order to decrease injection losses and improve Booster

efficiency. An RL model, whose inputs include all of the most important outside influences on the GMPS system, can infer more accurate cycle-by-cycle compensations on an FPGA, reliably producing better regulation against those influences. Creating such an algorithm begins with collecting time-series data from the accelerator complex in operation.

IV. DATASET

We collected a dataset for Booster GMPS regulation to provide cycle-by-cycle time series of readings and settings from the most relevant devices available in the Fermilab control system [75]. This data was drawn from the time series of a select subset of the roughly 200,000 entries that populate the device database of the accelerator control network [76]. Data was sampled at 15 Hz for 54 devices that pertain to the regulation of the GMPS during two separate periods of time: Period 0 (June 3, 2019 to July 11, 2019) was ended by the annual Summer Shutdown and Maintenance. Period 1 (December 3, 2019 to April 13, 2020) ended when the accelerator operations were suspended in response to the COVID-19 pandemic. In this paper, we use a subset of this dataset’s devices and a single day—March 10, 2020—for development and demonstration, as detailed in Sec V A. For a more detailed overview, please see the corresponding Data Descriptor available online [77]. To our knowledge, this is one of the first well-documented datasets of accelerator device parameters made publicly available. We strongly encourage our colleagues on other accelerator ML application projects to do likewise; reproducible results are the foundations of science.

In this accelerator complex data-logging nomenclature, device parameters with the **B:** prefix are related to the Booster, whereas device parameters beginning with **I:** are related to the Main Injector. Additionally, “MDAT” denotes the accelerator (machine) data communication broadcast.

A. Variable Selection

We chose a subset of the 54 device time series available to facilitate initial studies of **B:IMINER**, the measure of regulation error at each GMPS cycle minimum. These first studies (which are presented here) were conducted using five sets of time-series data suggested by accelerator domain experts to be the most important for regulating **B:IMINER**. In addition to **B:IMINER**, these include **B:LINFRQ**, **B:VIMIN**, **I:IB**, and **I:MDAT40**. Further optimization will be pursued prior to deployment of the production system.

Here, **B:VIMIN** is the compensating recommendation for the minimum value of the offset-sinusoidal GMPS current, issued by the GMPS regulator in order to reduce the magnitude of **B:IMINER**. **B:LINFRQ** is the measured off-

set from the expected 60 Hz line frequency powering the GMPS. **I:IB** and **I:MDAT40** provide measurements of the main injector bending dipole current at different points in the circuit and through different communication channels.

This expert-chosen set of just five parameters was used to train the surrogate model with the RL agent described in Sec. V, and to characterize RL agent models on an FPGA in Sec. VI.

As a check on this selection, a Granger causality study [78] was performed using those variables correlated or anticorrelated with **B:IMINER**, with absolute Pearson correlation coefficient $|r| > 0.20$. Additionally, **B:LINFRQ** and **I:IB** were included in the study on the advice of system experts. The Granger causality study, described here, allows us to explore the utility of the full set of logged signals for future studies, with minimal human bias.

For a given pair of concurrent time series—one potentially causal and the other responsive (one affects the other)—Granger causality does not prove pure causation. Instead, Granger causality suggests the response variable could be better predicted using both time series jointly rather than using the response variable’s self-history alone. This test consists of creating and comparing two linear regression models: a self-model and a joint-model for both the response variable and the potentially causal variable, and calculating coefficients for each lag value (time difference) being tested, from one up to some predetermined maximum number of time offsets [78]. If at least one coefficient is not zero in the joint model, then the other variable is said to be “Granger causal” with respect to the response variable, at the lag value being tested. We compared p-values to test statistical significance at each lag value up to 50 lags (approximately 3.33 s) as well as looked at the difference between the Bayesian information criterion of the self and joint-models. As a result of these iterative calculations, we identified three additional variables—**B:VIMAX**, the compensated maximum GMPS current, **B:VIPHAS**, the GMPS ramp phase with respect to line voltage, and **I:MXIB**, the main injector dipole bend current—that will be considered in the next iteration of the surrogate model.

The expert-selected devices **B:IMINER**, **B:LINFRQ**, **B:VIMIN**, **I:IB**, and **I:MDAT40** were indeed a subset of the top eight “causal” variables identified through the causality study, boosting confidence in their utility. Table I briefly summarizes the parameters of interest used in this iteration of the surrogate model.

B. Data Processing for ML

Although the devices were configured to write out reading and setting data at 15 Hz, actual timestamp intervals varied from this nominal frequency, and timestamps were not well synchronized across devices. Thus, for time alignment purposes, we made use of the recorded

these actions and presenting new situations to the agent.¹ The environment also gives rise to rewards, special numerical values that the agent seeks to maximize over time through its choice of actions.

TABLE I. Description of dataset parameters chosen by experts and later validated with a causality study. Here, “MI” means Main Injector, “MDAT” means accelerator (machine) data communication, and device parameters that begin with B are related to the Booster, whereas device parameters that begin with I are related to the Main Injector.

Parameter	Details [Units]
B:IMINER	Setting-error discrepancy at injection [A]
B:LINFRQ	60 Hz line frequency deviation [mHz]
B:VIMIN	Compensated minimum GMPS current [A]
I:IB	MI lower bend current [A]
I:MDAT40	MDAT measured MI current [A]

timestamps for `Event0C`, a broadcast accelerator control event which is synchronized to the fitted minimum of the periodically varying magnetic field in the gradient magnets described in Sec. III. This control event serves as a logical choice of reference time for GMPS-related parameters. Using Apache Spark-based [79] algorithms to distribute data processing in parallel, we first calculated the maximum interval between successive timestamps for each device across all 176 days (necessarily excluding the five-month gap between our two data-taking periods). We then used the corresponding largest observed lag between recorded values within each device to place the upper limit on a look-back window from an `Event0C` timestamp, and took for every `Event0C` the most recent device datum, whose timestamp was within that window for that device. For more details on the data processing decisions made in the creation of this dataset please see our Data Descriptor [77].

V. MACHINE LEARNING METHODS

Machine Learning (ML) refers to the process by which we adjust the randomly initialized parameters of generic function approximators, termed “models,” so as to minimize an appropriately chosen loss function, or conversely to maximize a reward function. As used in ML, feed-forward neural network model architectures specify arrangements of “nodes,” usually co-evaluated in layers, where each node calculates the weighted sum of the inputs and a bias term, and outputs the value of a non-linear “activation function.” In the simple multilayer perceptron (MLP) architecture, all nodes in each layer send copies of their output values to all nodes in the next layer. Layers not at the input or output are termed “hidden” layers. There are useful variations on this simple layer stack architecture such as recurrent neural networks (RNN), wherein some outputs from a previous forward inference are taken as inputs. Our work makes use of both MLP and RNN architectures.

For a given architecture with its activation functions, the weights and biases are the parameters being adjusted in the optimization or “training.” The “learning rate” sets the proportionality of parameter adjustment to gra-

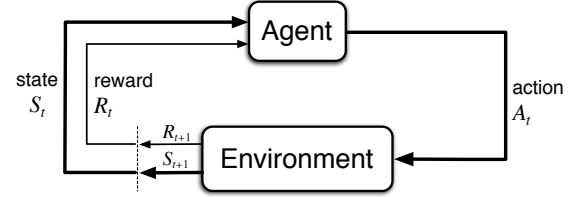


Figure 3.1: The agent–environment interaction in a Markov decision process.

FIG. 4. The agent–environment interaction in a Markov decision process [12]. The agent executes a policy that selects an action A_t given the current state S_t and on that basis selects a new state S_{t+1} of the environment. One time step later, in part as a consequence of its action, the agent receives a numerical reward, $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$, and finds itself in a new state, S_{t+1} .⁴ The MDP and agent together thereby give rise to a sequence or *trajectory* that begins like this:

$(S_0, A_0, R_1, S_1, A_1, R_2, S_2, \dots)$ and the development of sophisticated optimization schemes is an active research area.

Reinforcement Learning (RL) is the study of AI aimed at optimizing or planning of complex tasks based on iterative feedback inputs from an environment, as explained below. The main components of RL are the environment and the agent, an ML model, as illustrated in Fig. 4. RL trains an agent model over many time steps, and the resulting agent model may then be taken as fixed to be deployed on new data. The agent model may be expected to perform similarly on new data as it did in training, provided that the dynamics of the environment function $a: \mathcal{S} \times \mathcal{R} \times \mathcal{S} \rightarrow \mathcal{S}$ is an ordinary deterministic function of four arguments. The “if” in the middle of it comes from the notation for conditional probability, which would allow our regulator to adapt to changing environmental dynamics such as seasonality with new modes of accelerator complex operation, even though we would initially deploy a static or infrequently updated model out of prudence. We set out to use RL to train an optimal regulation policy, which dictates an appropriate action that the GMPS regulator should take for any given state of the system. Of course, as was highlighted in Sec II, methods other than neural network-based RL have had success in control problems and practitioners should assess and compare these approaches as well.

The environment, usually formulated as a Markov decision process, is represented by a time-independent, discrete system with which the RL agent interacts (e.g. the accelerator complex). For the regulation of the GMPS current minimum, the environment includes the time-varying power demands and outside electrical influences for which the GMPS regulator makes compensating changes. At each time step t , the environment takes in the control action A_t determined by the RL agent based on the current state S_t , and provides the new system state S_{t+1} (e.g. settings and measured quantities) along with an associated reward R_{t+1} . Optimizing the agent’s policy actions is defined to mean maximizing the long-term integrated reward, which is calculated over each fixed episode. In this study, the reward is calculated from the error in the minimum value of the GMPS current, B:IMINER:

$$R_t = -|\text{B:IMINER}(t)|. \quad (4)$$

The larger the magnitude of **B:IMINER**, the lower the reward. The possible actions A_t we consider correspond to overriding the fixed setting of **B:VIMIN**, the lone control variable, with small, compensating adjustments. This enables the agent’s policy model to control **B:IMINER** without the PID regulator’s input.

Recently, significant progress has been made in RL by combining it with advances in deep learning. Deep learning models are well suited to representing complex policies for high-dimensional problems such as regulation in a dynamically variable environment. The deep Q -network (DQN) [80, 81] approach, which we adopt for this study, involves using a deep neural network to learn the action-value function, or Q -value, and is usually deployed in environments that take discrete control actions. The optimal policy can then be derived by choosing the action that maximizes the expected Q -value.

More formally, a policy π is used by an agent to decide what actions $A_t = \pi(S_t)$ to take given a state S_t at time t . An optimal policy π^* maximizes the Q -value,

$$Q(S_t, A_t) = \sum_{t'=t}^T \mathbb{E} \left[\gamma^{t'-t} R(S_{t'}, A_{t'}) | S_t, A_t \right], \quad (5)$$

where \mathbb{E} is the expectation value operator, $R_t \equiv R(S_t, A_t)$ is the reward at time t , and γ is the discount factor that de-emphasizes future rewards relative to present ones. For this study, we used a value of $\gamma = 0.85$. The Q -value is the sum of the expected discounted rewards from the current time t up to the horizon T . In practice, the optimal action-value function Q^* is not known a priori, but it can be approximated iteratively because it satisfies the Bellman equation [82],

$$Q^*(S_t, A_t) = \mathbb{E} \left[R_t + \gamma \max_{A_{t+1}} Q^*(S_{t+1}, A_{t+1}) | S_t, A_t \right]. \quad (6)$$

In a DQN, the Q -value is approximated using a deep neural network, or policy model, with parameters θ . In particular, the loss function at a time t is given by the mean squared error (MSE) in the Bellman equation,

$$L_t(\theta_t) = \mathbb{E} \left[(y_t - Q(S_t, A_t; \theta_t))^2 \right], \quad (7)$$

where the (unknown) optimal target values are replaced by the approximate target values $y_t = R_t + \gamma \max_{A_{t+1}} Q(S_{t+1}, A_{t+1}; \theta_t^-)$ using parameters θ_t^- derived from previous iterations.

Continuous action space environments, such as the compensating adjustments of our GMPS current regulator, can adopt the DQN algorithm by discretizing the action space. To keep the DQN action space finite, we discretize the *change* of control signal **B:VIMIN** using steps of just a few different sizes, including the option for zero-size change. By doing so, we explicitly limit the control signal variation between time steps, while helping to minimize the tuning and possible resonance of the overall surrogate-RL training loop. Other RL algorithms, such as deep deterministic policy gradient

(DDPG) [83], proximal policy optimization [84], and twin delayed DDPG [85] can provide continuous control suggestions which fit the GMPS regulator more naturally. However, the sample efficiency and tuning stability of these algorithms presents difficulties as discussed in various studies [86, 87].

The trial and error nature of RL training requires an environment that accommodates offline iteration. As mentioned in Sec. II, most real-world, complex systems cannot afford such an approach due to the risk of system failure. Therefore some level of offline training is essential. To facilitate offline training of a control agent, here we develop a surrogate model using the select collection of historical measurements described in Sec. IV. This surrogate captures the subset of the Booster accelerator complex necessary to model the dynamics of GMPS regulation in its environment.

In the following subsections, we discuss the workflow employed to develop and use an offline policy optimization, a prudent step prior to deploying the agent model in the real system. Sec. V A details the surrogate model that is developed to test the deep RL agent. Then, in Sec. V B, we describe how the agent itself is trained.

A. Virtual Accelerator Complex Model

In order to train the RL policy model, we adopted a long short-term memory (LSTM) [88] architecture for a surrogate of the GMPS regulator in its operating environment using data which describe the GMPS current and the Main Injector operation cycle. Its purpose is to predict the impact of changing the control variable **B:VIMIN** on **B:IMINER** over the course of the RL episodes.

An LSTM model is a specific type of recurrent neural network (RNN), which is appropriate for modeling sequential data such as our accelerator data. The key feature of an RNN is the network recursion, which enables it to describe the dynamic performance of systems over sequential time steps. One difficulty of training RNNs is the vanishing gradient problem [89, 90], which is a result of the gradient at a recursion step k being dominated by the product of partial derivatives of hidden-state vectors h , $\prod_{j=k}^{R-1} \partial h_{j+1} / \partial h_j \propto w^{R-k-1}$, where R is the number of recursions and w is a recurring weight. If $w < 1$, then this product tends to zero for large R . The design of the LSTM includes an internal memory state, which effectively mitigates the vanishing gradient problem, and allows the network to retain a longer memory of past inputs [88]. Given the complexity of the GMPS regulation environment, we selected the LSTM architecture in order to capture the multiple frequency modalities observed in the data.

The details of the surrogate model’s architecture are given in Table V A. At each time step, the input to the Booster ML surrogate model is the most recent 150 time steps (equivalent to 10 seconds of data) from the five input variables selected in Sec. IV A: **B:VIMIN**, **B:IMINER**,

TABLE II. Fermilab Booster surrogate model, which learns to reproduce the environment in terms of the three time-series variables, one of which determines the reward as given in Eq. 4. The input LSTM layer receives five values, describing the current state **B:IMINER**, **B:LINFRQ**, **B:VIMIN**, **I:IB**, and **I:MDAT40**. The output layer is a prediction of **B:IMINER**, **B:LINFRQ**, **B:VIMIN**.

Layer	Layer Type	Outputs	Activation	Parameters
1	LSTM	256	tanh	416,768
2	LSTM	256	tanh	525,312
3	LSTM	256	tanh	525,312
4	dense	3	linear	771
Total	1,468,163

B:LINFRQ, **I:IB**, and **I:MDAT40**, and the output is a prediction for the values of **B:IMINER**, **B:LINFRQ**, and **B:VIMIN** at the next time step. While the latter two values are not included in the environment state propagated to the RL agent, the additional output structure helps to regularize the training process and validate performance. The LSTM model uses this 150-step look-back window to “recall” the historical patterns of the time-series data, and thus achieve high accuracy in prediction.

The LSTM surrogate model was developed and implemented using the KERAS library [91]. We used the Adam optimizer [92] and a cost function of the mean squared error (MSE) of the predictions. The total number of data samples used for the analysis of the LSTM surrogate was 250,000 time steps from March 10, 2020, which we split into two non-overlapping data sets composed of 175,000 time steps for training and 75,000 time steps for testing. These data sets were then processed to allow 150 time step look-back, 1 time step look-forward as mentioned above.

The training samples were then further split using a K-fold cross-validation method: we defined five cross-validation folds that split the training and validation in a 80%/20% split. This technique was used in order to estimate how the surrogate model is expected to perform in general as well as to monitor over-fitting. While this sort of cross-validation was performed on the same segment of data in this implementation, we plan to cross-validate on different data samples when training the surrogate model in the future, at a larger scale. The loss values from the validation sample were used to determine if the learning rate should be reduced or if the surrogate model had stopped learning, as shown in Figure 5. After more than 300 training epochs, the figure shows a bifurcation between the values of training loss and validation loss, suggesting some over-fitting. Therefore, we used the values of model parameters prior to this bifurcation as the parameters of our surrogate model. On separate test data, the loss value for this surrogate model was determined to be 9×10^{-4} which is consistent with the training data set prior to the bifurcation.

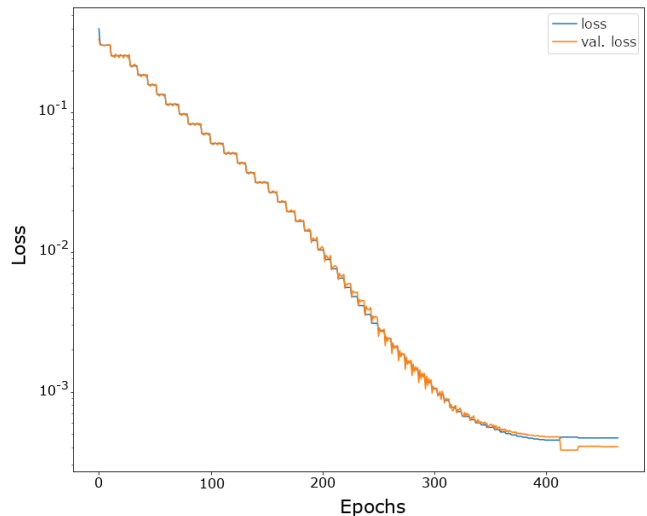


FIG. 5. Loss function as a function of the number of training epochs for the Booster LSTM surrogate model. The blue line gives the loss values for training sample and the orange line is the calculated loss using validation samples.

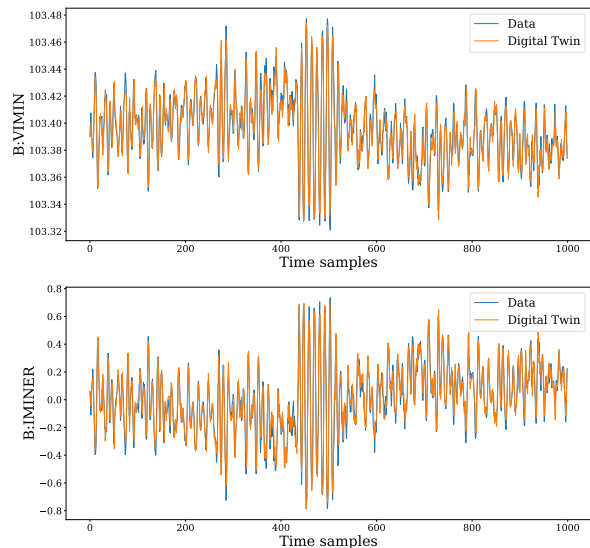


FIG. 6. Selected test data (blue) versus prediction values (orange) from the Booster LSTM surrogate model. New data is fed into the trained surrogate model at each time step.

Overlaid time series from the data and from LSTM predictions for a selected time window are shown in Fig. 6. Based on the great similarity of these results, the surrogate model was deemed adequate to use for initial training of the RL agent policy model.

B. Reinforcement Learning for GMPS Control

For this study, we formulated the problem as an episodic Markov decision process, where every episode contains 50 time steps. As in all Q -learning, the agent learns to maximize the reward within the time horizon of an episode. We developed our RL workflow based on a variant of DQN, the double DQN algorithm [80, 93, 94], using KERAS [91] to optimize **B:VIMIN** settings dynamically to minimize GMPS error **B:IMINER**. The double DQN explicitly decouples the *target model*, which is used to evaluate actions, from the *policy model*, which is used to select actions, although they take the same form.

We used the OPENAI GYM package [95] to develop the environment that serves as a wrapper around the virtual accelerator complex model described above in Sec. V A to interact with the RL agent. The observation state space is simply the aforementioned five variables in the surrogate model section above, shown to causally relate to the measure of regulation error, **B:IMINER**. The action state space only contains one free parameter of control: adjustments to **B:VIMIN**. The seven discrete control options relative to the previous **B:VIMIN** are 0 (no change), ± 0.0001 , ± 0.005 , and ± 0.001 . The choice of these values was based on the actual distribution of the changes in **B:VIMIN** observed in the data.

At the start of each episode, 150 time steps from the data are used to initialize the system state, as is required for the environment surrogate model. The 150th step defines the system state used by the agent. For each step thereafter, the agent provides a new action by specifying a change to **B:VIMIN**, and the system state is updated. The new system state is then used by the surrogate model to predict the resulting value of **B:IMINER**. After the prediction the system state is incremented to the next time step. The current state, reward, and status for each step is passed to the agent to be used for training the DQN policy model. RL algorithms learn from the reward provided by the environment, which in this study is given by Eqn. 4.

During training, event samples are placed into a buffer before calculating the loss. This *memory buffer* is sampled randomly in a process called experience replay [80] in order to remove instabilities found to arise from training on time-ordered samples. Once the memory buffer has sufficient experiences (32 experiences for this study) the active policy model begins training and continuously updating. We use the ϵ -greedy [96] method to control the agent’s trade-off between exploration (random choice of action) and exploitation (deterministic action dictated by the current policy), in which the optimal action according to the current policy is chosen with probability $1 - \epsilon$, while a random action is selected with probability ϵ . At the beginning of the training session we set $\epsilon = 1$ with a decay factor of 0.9995, applied multiplicatively whenever an exploration action is selected, until a minimum value of $\epsilon = 0.0025$ is reached. For this study, we use a multilayer perceptron (MLP) as the pol-

icy model (and target model) architecture, and rectified linear unit (ReLU) activation functions [97], as summarized in Table III. The active policy model is continuously updated during training by using randomly selected experiences from the memory buffer. At each training step the weights of the target model θ_{target} are incrementally updated to reflect the weights of the active policy model θ_{policy} ,

$$\theta_{\text{target}} \mapsto \theta_{\text{target}}(1 - \tau) + \tau\theta_{\text{policy}}, \quad (8)$$

where we set $\tau = 0.5$ [83].

The result of the DQN MLP training, in terms of a rolling average of the total reward over 10 episodes versus the number of episodes, is shown in Fig. 7 (top). Here we show the results of consecutive batch initialization within a restricted region of 4,000 consecutive samples. These samples fell within the first 250,000 samples that the surrogate model was trained on. This was done to ensure that the environment would still reliably respond to the action. Note that in the training, there is additional randomness introduced due to the epsilon-greedy approach.

Additionally, in Fig. 7 (bottom), we display the results of testing (no rolling average taken) versus the number of episodes. Likewise, the testing start points were generated via consecutive batch initialization within an orthogonal region of 1,000 samples that still fell within the region the surrogate model was trained on.

For both the training and testing reward plots, the current controller reward, the black dashed line, was determined using the data by summing the values of $R_t = -|\mathbf{B:IMINER}(t)|$ throughout each 50 step episode. The DQN MLP controller reward, the solid red line, shows improvement over the current system by approximately a factor of 2 in both the training and testing sets. The downward reward spikes in both the training and testing correspond to occurrences of a relatively rare but regular operation: the 6s resonant extraction from the Main Injector at 120 GeV, sending beam through a long beam transport line, which requires certain large power supplies to operate at high current for the duration, strongly influencing the electrical environment of the GMPS. The RL policy is expected to learn to treat this appropriately once trained on more data. As is evident here, only experiencing these events four times is not enough for the RL policy to perfectly accommodate this circumstance. Nevertheless the RL policy outperforms the current PID implementation in this context. It would be interesting to see the corresponding improvement to injection losses and Booster efficiency for a controller regulating GMPS with this RL policy, because these improvements are not thought to be linearly related to the size of the regulation error.

Additionally, we quantified the step by step difference between the RL agent’s actions and the current PID’s historical actions (defined as the change in **B:VIMIN**) over the same data during the test set. The distribution of differences in action is approximately normal and has mean

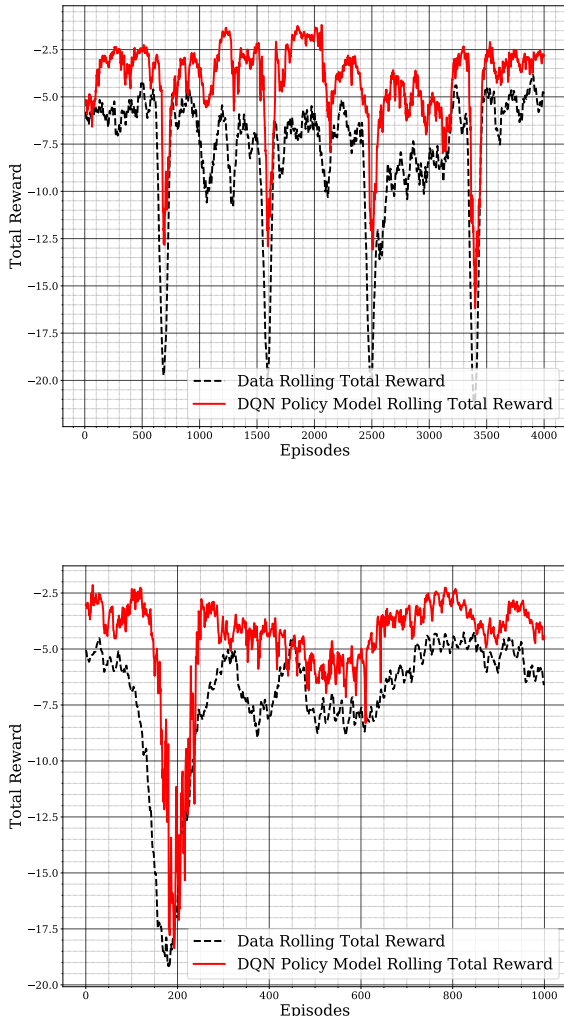


FIG. 7. Top: Total rolling reward per episode versus number of training episodes for the DQN MLP algorithm (solid red line) at top. During training, the 10-episode rolling window determines the first entry at the 10th episode on the plot. Bottom: The corresponding testing results (without a rolling average) are shown below.

$0.0006 \pm 5.769 \times 10^{-5}$ and standard deviation 0.0129, giving us confidence that the learned RL approach is reasonable. We expect that future development work focused on finetuning the discretization of the action space will decrease this standard deviation.

VI. IMPLEMENTATION IN FAST ELECTRONICS

Fast GMPS control electronics are required to collect information from the Booster environment, decide whether to apply a corrective action, and distribute the

corresponding control signal, all with the low latency requirement set by the Booster’s 15 Hz cycle. An FPGA is a natural choice to implement the corresponding circuit, accommodating latencies far below those achievable with a CPU or GPU while allowing reconfigurability impossible in a custom application-specific integrated circuit (ASIC) solution. The DQN MLP GMPS regulation model proposed in Sec V B requires an efficient, but adjustable, implementation of NN algorithms, strongly suggesting an FPGA-based implementation. As a preliminary step, we take the offline-trained DQN MLP with weights fixed and deploy it in an FPGA.

The following subsections review the computational steps required for a single NN inference (§VIA); describe the basic elements of an FPGA and how a deep NN calculation can be efficiently mapped to a corresponding circuit (§VIB); present an implementation of the DQN MLP described above in Sec V and the impact of various design choices (§VIC); and lastly discuss possible extensions of the implementation to accommodate more complex algorithms which are of interest (§VID).

A. Elements of NN Inference

The structure of an MLP is a series of alternating linear and nonlinear transformations (layers), with the i th layer mapping a set of inputs x_i (features) to a discrete list of outputs y_i . In the present application, the features may include any measurements of the GMPS environment, such as digitized traces from the reference magnet system, line voltage frequency, and equipment gallery temperature. For the DQN MLP, the outputs y_i are scores associated to a discrete set of possible actions, with the highest-scoring action being the one taken by the controller. An MLP layer f yielding m outputs may be written in terms of its action on a set of inputs $\{x_i\}_{i=1,\dots,n}$ as

$$f : x_i \rightarrow \sigma \left(\sum_i w_{ij} x_i + b_j \right), \quad (9)$$

where w_{ij} (the $n \times m$ weight matrix) and b_j (the m -dimensional bias vector) are configurable parameters of the linear translation and σ is an m -to- m nonlinear activation function. For each layer, the activation function is prescribed as a part of the model architecture while optimal values for the weights and biases are found through a training procedure. The DQN MLP utilizes the linear (identity) $\text{ReLU}(x_i) = \max(x_i, 0)$ activation functions. The complete, k -layer NN is specified by an ordered composition of layers $y = f^{(1)} f^{(2)} \dots f^{(k)}(x)$. While the input and output dimensions are fixed by the set of features and actions, the dimensionality of intermediate layers is arbitrary. Table III describes the architecture of the DQN MLP, in addition to the number of configurable parameters and total multiply-and-accumulate (MAC) operations required.

TABLE III. Implemented DQN MLP model architecture. The first NN layer receives five input values.

Layer	Outputs	Activation	Parameters	MACs
1	128	ReLU	768	640
2	128	ReLU	16 512	16 384
3	128	ReLU	16 512	16 384
4	7	Linear	903	896
Total	34 695	34 304

B. NN Inference on FPGAs

An FPGA consists of an array of logic gates that may be programmed to emulate any circuit (up to the physical resource constraints of the specific hardware device). This allows FPGA designs to profit from the many advantages of custom ASICs including massive parallelization and low power consumption while maintaining reconfigurability. However, a significant advantage of the FPGA architecture lies in the fact that it is not simply a homogeneous fabric of low-level gates (e.g. NAND gates). Rather, modern FPGAs are heterogeneous structures including more complex logical blocks, each specialized for a dedicated task, repeated many times. In this way, FPGA designs can simultaneously exploit both the flexibility of a programmable architecture and the performance of a dedicated printed circuit.

An efficient implementation of the NN model in firmware requires a design that exploits the FPGA’s specialized computational units to perform each step of the NN calculation. Digital signal processor (DSP) slices are flexible circuits for addition, multiplication, wide-array bitwise logical operations, and more. DSPs may be further chained to accommodate more complex operations. In the Intel Arria 10 FPGA, to be deployed in the GMPS control system, DSP blocks may be configured to multiply and accumulate fixed-point numbers up to 27 bits, providing a solution for the linear component of Equation 9. The affine map from m to n dimensions requires mn scalar multiplications and sums that, in a fully parallelized design, may be accomplished with mn cascading DSPs. To evaluate an arbitrarily complex activation function in FPGAs, it is more efficient to store a pre-computed table of values than to re-calculate the function many times per inference. This may be accomplished using block RAM (BRAM), embedded memory that is configurable for read/write access. BRAMs are available in segments of 20 kb in the Arria 10 to store, for example, a bank of 1024 function values at 20-bit precision. Registers are groups of flip flops used to record temporary numerical values or internal states, and to facilitate signal routing across the major computational blocks of the design. Finally, ALMs are lightweight, configurable modules of combinational logic elements, used throughout designs for basic operations such as simple arithmetic

and logical operations.

C. Implementation of the GMPS Regulator Model

The GMPS regulator model described in Sec V must be converted to firmware in a manner that takes full advantage of the FPGA’s architectural features described in Sec VIB. This is accomplished through the translation of the KERAS description of the NN function into high-level synthesis (HLS) code using the HLS4ML [58] toolkit, whose functionality has been recently extended to Intel FPGAs [98]. The HLS design is converted to firmware using INTEL QUARTUS [99]. The use of HLS4ML brings the significant advantage of enabling a fast development cycle from model prototyping to implementation in firmware. Thus, the present work has focused not only on achieving an optimal design for the benchmark ML algorithm proposed in Sec VB but also on more generic design-space exploration. Establishing scalable strategies such as FPGA implementation is critical for scaling up to more complex ML models that will inevitably become necessary as larger data sets allow for increasingly nuanced treatments of the control problem.

The conversion of the KERAS model to firmware requires a number of design choices. Chief among these are the numerical precision to which the calculation is carried out and the degree of parallelism incorporated into the design. Fixed-point values with a specified number of total and integer bits are used to represent model inputs, weights, and all intermediate results of the calculation. To determine the range of values to encode in the fixed-point representation, the number of integer bits is set to be at least as large as the maximum weight value. The number of total bits, which sets the number of bits used to encode the fractional component of the weight (once the integer part is specified), is set to minimize the impact of quantization.

Figure 8 displays a histogram of the weight values for the trained DQN MLP model. The total number of bits retained for each weight is selected by comparing the floating-point model inference with that of the fixed-point model for a range of bit widths, scanning over a representative sample of input test data. Figure 9 shows that using 14 bits to encode the fractional component of all operands is sufficient to replicate the decision taken by the floating-point model for over 99.5% of the test data. This performance comparison of fixed and floating-point models shows that while nine fractional bits are sufficiently precise to represent the weights, additional precision in the representation of the intermediate sums in the NN calculation is necessary to achieve full performance. Following this, comparisons were performed using a 20-bit representation for all internal fixed-point parameters (5 integer bits, 14 fractional bits, and a sign bit) where not explicitly varied.

The degree of design parallelization can be motivated by the number of operations required for each NN infer-

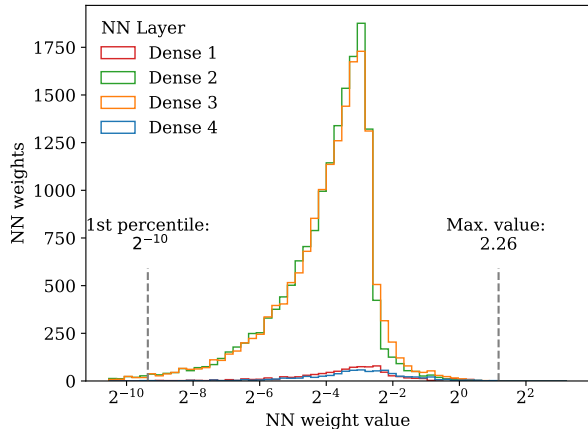


FIG. 8. Separate histograms display the magnitude of the floating-point weights obtained for each many-to-many (“dense”) layer after training the DQN MLP, in bins of logarithmically-varying width. Over 99% of weights are found to have absolute value greater than $1/2^{10}$, with a maximum value of 2.26.

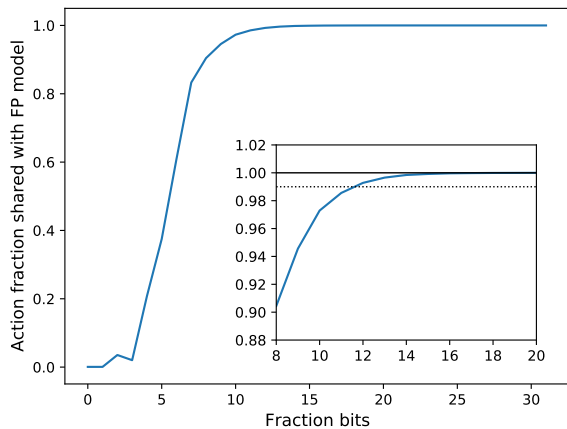


FIG. 9. The fraction of decisions that the quantized NN implementation shares with the floating-point calculation across a set of representative input Booster data is shown as a function of the fixed-point precision. Here the number of bits to encode the integer part is fixed to five plus a sign bit, while the number of bits encoding the fractional part is varied. The inset shows the same measurements, highlighting the region where the shared action fraction is over 90%. At very low precision, statistical fluctuations are observed that depend on the specific model weights and rounding conventions.

ence in comparison with the total available resources on the target FPGA. One constraint comes from the MAC operations that are efficiently computed in a single clock cycle using dedicated DSP slices for each operation. As discussed in Sec VIA, the MLP agent requires 34 304 MACs per inference, compared to the 1518 DSP slices available in the Arria 10 FPGA. The approach taken to

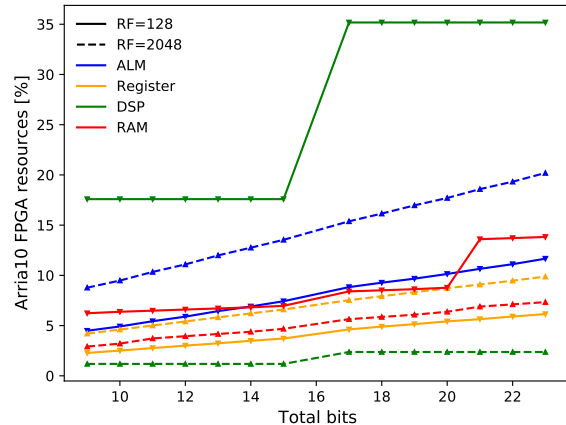


FIG. 10. FPGA resources required for the implementation of the DQN MLP are shown as a function of the fixed-point precision utilized for internal NN operations. All resources are normalized to the total available in a benchmark Arria 10 device (see Table IV). Results are shown for implementations with reuse factors of 128 (solid lines) and 2048 (dashed lines).

address this is to assign a reuse factor to each NN layer, specifying the number of operations each physical MAC unit may contribute to the set of necessary computations required by that layer. Larger reuse factors will result in a design utilizing fewer FPGA resources at the expense of longer inference latency. For simplicity a single, model-level reuse factor is considered in the following, where the per-layer reuse factor is given by the greatest common divisor of the reuse factor and the product of input and output multiplicity.

Figure 10 demonstrates how the resources required to implement the NN algorithm are affected by the precision to which internal calculations are carried out, and the degree of parallelization specified by the selected reuse factor. In general, the required low-level resources such as ALMs and registers scale linearly with precision to accommodate the widths of increasing data paths. Conversely, a single DSP slice can accommodate a range of operand bit-widths, up to the limit of the design specification at which point a second DSP must be used per calculation. In the case of reuse factor of 128, the largest burden on FPGA resources comes from the required DSPs (either 4 or 8% of the Arria 10 total, depending on the necessary precision), while ALMs are the limiting factor (3–8%) in the case of reuse factor of 1568, where the design parallelization is at most a factor of four in each NN layer. While inference latency depends on the degree of parallelism directly through the reuse factor, it is essentially invariant under changes to the operand precision.

Table IV compares several implementations for constant precision (20 total bits) and various reuse factors. In general the algorithm latency increases as a function of increasing reuse factor while the numbers of DSPs and

TABLE IV. The required FPGA resources and corresponding latency for the NN algorithm are shown for three possible implementations corresponding to various reuse factors. In addition to design parameters, the maximum available resources are shown for an Intel Arria 10 benchmark FPGA. Memory logic array blocks (MLABs) are configured from ten ALMs and hence no device maximum is shown.

reuse factor	DSP	BRAM	MLAB	ALM	Register	Latency
128	534	238	672	43.3 k	92.6 k	3.9 μ s
256	274	231	642	48.9 k	112.3 k	7.3 μ s
512	144	195	7467	152.6 k	252.0 k	13.6 μ s
1024	68	171	4088	111.7 k	202.4 k	24.6 μ s
2048	36	173	1960	75.6 k	149.1 k	39.3 μ s
Available	1518	2713	...	427 k	1.7 M	...

BRAMs required are inversely proportional to the reuse factor. Variations in the required registers and ALMs are generally not significant by comparison. These results demonstrate a range of feasible firmware implementations of the algorithm that fit comfortably within the available resources of the GMPS control board and 1/15 sec (66.7 ms) latency budget. The ability to tune resource usage provides significant flexibility to accommodate future scenarios where the NN algorithm may significantly grow in complexity and, further, must coexist on a single FPGA with additional control logic that may present inflexible resource constraints of its own.

D. Extensions to More Complex Algorithms

Up to this point, the discussion of the hardware implementation has centered around the three-hidden-layer MLP architecture found to be performant for the GMPS control problem in the context of RL studies described in Sec V. However, the conclusions of the studies described above may be extended to more complex NN algorithms providing improved GMPS performance in tandem with the experience gained through future data-taking campaigns.

The simplest extension to the single MLP solution, well-motivated in the context of RL studies, is to run inference with an ensemble of multiple copies of the network in parallel on the FPGA, to improve robustness of performance. Each NN may be programmed with a unique set of weights, allowing for disagreement among the models, where additional voter logic determines the final action to be taken by the control system. This is straightforward to achieve for models with similar complexity to the one studied in Sec VIC. Achieving designs that consume $\leq 6\%$ of all available resources suggests that an ensemble of $\mathcal{O}(10)$ models is feasible.

Alternatively, instead of an ensemble of relatively simple models, more complex networks can be pursued. The MLP architecture studied can be extended to additional

layers and larger numbers of nodes per layer maintaining an acceptable footprint through corresponding adjustment of the reuse factor. The theoretical scaling behavior was shown in the calculations of Sec VIA and observed in the implementation using Quartus HLS. As an illustrative example, one could consider a refinement of the baseline architecture where the number of nodes per layer is uniformly increased by a scaling factor s . In this case, the number of required multipliers may be kept constant by simultaneously increasing the reuse factor by a factor of s^2 , at the expense of a small corresponding increase in algorithm latency. This strategy would allow more powerful solutions with similar footprint to take advantage of the full latency budget of ≈ 66 ms. More sophisticated architectures such as convolutional and recurrent NNs may also be considered, taking advantage of their representations as compositions of multiple dense sub-layers. A detailed study of such possibilities is left to future work.

VII. SUMMARY AND OUTLOOK

In this paper, we have described a method for controlling the gradient magnet power supply (GMPS), an important subsystem of the Fermilab Booster accelerator, using machine learning models and demonstrated the feasibility of embedding such a model on a field-programmable gate array (FPGA) for a high-uptime, low-latency implementation. We first developed a surrogate LSTM model, based on a recurrent neural network, to reproduce the behaviors of the real GMPS system in the context of the accelerator complex, establishing a safe environment for training reinforcement learning algorithms. Within this environment, we trained a deep Q -network, based on a multilayer perceptron, to choose an optimal action (adjustment of one control knob) to maximize the long-term reward, taken from the negative absolute value of the regulation error (difference between the set and observed values of the minimum GMPS current). We found this surrogate-trained network achieved a factor of 2 improvement over the existing controller in terms of the achieved rewards. Finally, we implemented this network on an Intel Arria 10 FPGA and found it reproduces the CPU-based model, consumes less than 6% of the total FPGA resources, and executes with a latency as low as 2.8 μ s, which bodes well for future extensions.

Real-time and operations-hardened solutions will be critical for deploying this technology in an accelerator control context, but we believe a large number of other application spaces will be able to benefit from reinforcement learning on embedded systems. Surrogate models appear promising for supplying the large training data volumes required by reinforcement learning agents. This is particularly important for accelerator facilities where large-scale simulations of the entire complex are absent. Although many open questions remain, this proof-of-principle provides confidence to test our proposed con-

cept on “live” hardware. The next steps of this work, including mechanisms for online training and model updates for systems operating with a running accelerator, will be the subject of a future report.

In general, the future for machine learning algorithms in accelerator control is bright. The proliferation of shared tools and open datasets like the one developed for and used in this paper will doubtlessly enable rapid progress. We note that Ref. [33] also adopted the OPENAI GYM [95] as a programming interface for training reinforcement learning agents for use in an accelerator complex. Adoption of common tools will make it easy for researchers in this space to share code and especially to share access to datasets. This should allow multi-institution collaboration to prosper and greatly enhance the pace of progress in the field of artificial intelligence for accelerator applications.

ACKNOWLEDGMENTS

We would like to thank Jonathan Edelen and Jan Strube for useful discussions and a careful reading of the manuscript.

This document was prepared using the resources of

the Fermi National Accelerator Laboratory (Fermilab), a U.S. Department of Energy (DOE), Office of Science, HEP User Facility. Fermilab is managed by Fermi Research Alliance, LLC (FRA), acting under Contract No. DE-AC02-07CH11359. This research was sponsored by the Fermilab Laboratory Directed Research and Development Program under Project ID FNAL-LDRD-2019-027: “Accelerator Control with Artificial Intelligence” and partially funded by the National Science Foundation (NSF) Mathematical Sciences Graduate Internship Program (MSGI).

Pacific Northwest National Laboratory (PNNL) is a multi-program national laboratory operated by Battelle for DOE under Contract No. DE-AC05-76RL01830. This work at PNNL was supported by the DOE Office of High Energy Physics.

J. M. D. is supported by DOE Office of Science, Office of High Energy Physics Early Career Research program under Award No. DE-SC0021187. R. K. was supported by the NSF MSGI and the NSF Graduate Research Fellowship Program under Grant No. DGE-1644869.

We acknowledge the Fast Machine Learning collective as an open community of multi-domain experts and collaborators. This community was important for the development of the FPGA implementation for this project.

-
- [1] J. Amundson, A. Macridin, and P. Spentzouris, High performance computing modeling advances accelerator science for high energy physics, *IEEE Comput. Sci. Eng.* **16**, 32 (2014).
 - [2] Y. LeCun, Y. Bengio, and G. Hinton, Deep learning, *Nature* **521**, 436 (2015).
 - [3] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning* (MIT Press, 2016) <http://www.deeplearningbook.org>.
 - [4] G. Carleo, I. Cirac, K. Cranmer, L. Daudet, M. Schuld, N. Tishby, L. Vogt-Maranto, and L. Zdeborová, Machine learning and the physical sciences, *Rev. Mod. Phys.* **91**, 045002 (2019), [arXiv:1903.10563](https://arxiv.org/abs/1903.10563).
 - [5] DeepMind, Safety-first AI for autonomous data centre cooling and industrial control, <https://deepmind.com/blog/safety-first-ai-autonomous-data-centre-cooling-and-industrial-control/> (2018).
 - [6] A. Davies, The WIRED Guide to Self-Driving Cars, <https://www.wired.com/story/guide-self-driving-cars/> (2018).
 - [7] Jennifer Langston, How AI is building better gas stations and transforming Shell’s global energy business, <https://blogs.microsoft.com/ai/shell-iot-ai-safety-intelligent-tools/> (2018).
 - [8] Will Knight, This Factory Robot Learns a New Job Overnight, <https://www.technologyreview.com/s/601045/this-factory-robot-learns-a-new-job-overnight/> (2018).
 - [9] E. Hubbard *et al.*, *Booster Synchrotron*, Fermilab Technical Memo (1973).
 - [10] S. H. Kim and F. Boukouvala, Machine learning-based surrogate modeling for data-driven optimization: a comparison of subset selection for regression techniques, *Optim. Lett.* **14**, 989 (2020).
 - [11] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. (MIT Press, Cambridge, MA, USA, 2018).
 - [12] V. François-Lavet, P. Henderson, R. Islam, M. G. Belle-mare, and J. Pineau, An introduction to deep reinforcement learning, *Found. Trends Mach. Learn.* **11**, 219 (2018).
 - [13] C. Emma, A. Edelen, M. J. Hogan, B. O’Shea, G. White, and V. Yakimenko, Machine learning-based longitudinal phase space prediction of particle accelerators, *Phys. Rev. Accel. Beams* **21**, 112802 (2018).
 - [14] C. Tennant, A. Carpenter, T. Powers, A. Shabalina Solopova, L. Vidyaratne, and K. Iftekharuddin, Superconducting radio-frequency cavity fault classification using machine learning at jefferson laboratory, *Phys. Rev. Accel. Beams* **23**, 114601 (2020).
 - [15] A. L. Edelen, S. G. Biedron, B. E. Chase, D. Edstrom, S. V. Milton, and P. Stabile, Neural networks for modeling and control of particle accelerators, *IEEE Trans. Nucl. Sci.* **63**, 878 (2016).
 - [16] A. Edelen, S. Biedron, D. Bowring, B. Chase, J. Edelen, S. Milton, and J. Steimel, Neural network model of the PXIE RFQ cooling system and resonant frequency response, in *7th International Particle Accelerator Conference (IPAC)* (JACoW, Geneva, Switzerland, 2016) p. 4131, [arXiv:1612.07237](https://arxiv.org/abs/1612.07237).
 - [17] A. L. Edelen, S. G. Biedron, S. V. Milton, D. Bowring, B. E. Chase, J. P. Edelen, D. Nicklaus, and J. Steimel, Resonant frequency control for the PIP-II injector test RFQ: Control framework and initial results, in *2nd*

- North American Particle Accelerator Conference (NAPAC)* (JACoW, Geneva, Switzerland, 2017) p. 109, arXiv:1612.05659.
- [18] A. Edelen, S. Biedron, J. Edelen, and S. Milton, First steps toward incorporating image based diagnostics into particle accelerator control systems using convolutional neural networks, in *2nd North American Particle Accelerator Conference (NAPAC)* (JACoW, Geneva, Switzerland, 2017) p. 390, arXiv:1612.05662.
- [19] A. Edelen, J. Edelen, S. Biedron, S. Milton, and P. van der Slot, Using neural network control policies for rapid switching between beam parameters in a free electron laser, in *Deep Learning for Physical Sciences Workshop at the 31st Conference on Neural Information Processing Systems* (<https://dl4physicsciences.github.io>, Long Beach, CA, USA, 2017).
- [20] A. Edelen, S. Biedron, J. Edelen, S. Milton, and P. van der Slot, Using a neural network control policy for rapid switching between beam parameters in an FEL, in *International Free Electron Laser Conference (FEL'17)* (JACoW, Geneva, Switzerland, 2018) p. 488.
- [21] A. Scheinker and S. Gessner, Adaptive method for electron bunch profile prediction, *Phys. Rev. ST Accel. Beams* **18**, 102801 (2015).
- [22] A. Scheinker, A. Edelen, D. Bohler, C. Emma, and A. Lutman, Demonstration of model-independent control of the longitudinal phase space of electron beams in the linac-coherent light source with femtosecond resolution, *Phys. Rev. Lett.* **121**, 044801 (2018).
- [23] S. Hirlander and N. Bruchon, Model-free and bayesian ensembling model-based deep reinforcement learning for particle accelerator control demonstrated on the fermi fel (2020), arXiv:2012.09737.
- [24] J. Ögren, C. Gohil, and D. Schulte, Surrogate modeling of the CLIC final-focus system using artificial neural networks, *J. Instrum.* **2021** (05), P05012, arXiv:2009.06454.
- [25] A. Edelen, N. Neveu, C. Emma, D. Ratner, and C. Mayes, Machine learning models for optimization and control of X-ray free electron lasers, in *Machine Learning and the Physical Sciences Workshop at the 33rd Conference on Neural Information Processing Systems* (<https://ml4physicsciences.github.io>, Vancouver, Canada, 2019).
- [26] A. Edelen, N. Neveu, M. Frey, Y. Huber, C. Mayes, and A. Adelman, Machine learning for orders of magnitude speedup in multiobjective optimization of particle accelerator systems, *Phys. Rev. Accel. Beams* **23**, 044601 (2020).
- [27] J. Kirschner, M. Mutný, N. Hiller, R. Ischebeck, and A. Krause, Adaptive and safe Bayesian optimization in high dimensions via one-dimensional subspaces, in *Proceedings of the 36th International Conference on Machine Learning*, Vol. 97, edited by K. Chaudhuri and R. Salakhutdinov (PMLR, Long Beach, CA, USA, 2019) p. 3429, arXiv:1902.03229.
- [28] J. Duris, D. Kennedy, A. Hanuka, J. Shtalenkova, A. Edelen, P. Baxevanis, A. Egger, T. Cope, M. McIntire, S. Ermon, *et al.*, Bayesian optimization of a free-electron laser, *Phys. Rev. Lett.* **124**, 124801 (2020), arXiv:1909.05963.
- [29] A. Hanuka, X. Huang, J. Shtalenkova, D. Kennedy, A. Edelen, Z. Zhang, V. R. Lalchand, D. Ratner, and J. Duris, Physics model-informed Gaussian process for online optimization of particle accelerators, *Phys. Rev. Accel. Beams* **24**, 072802 (2021), arXiv:2009.03566.
- [30] A. Scheinker, C. Emma, A. L. Edelen, and S. Gessner, *Advanced Control Methods for Particle Accelerators (ACM4PA) 2019 Workshop Report*, Tech. Rep. (2020) arXiv:2001.05461.
- [31] P. Arpaia *et al.*, Machine learning for beam dynamics studies at the CERN Large Hadron Collider, *Nucl. Instrum. Methods Phys. Res. A* **985**, 164652 (2021), arXiv:2009.08109.
- [32] A. Edelen *et al.*, Opportunities in Machine Learning for Particle Accelerators, arXiv:1811.03172 (2018).
- [33] S. Hirlander, V. Kain, and M. Schenk, Automatic performance optimisation and first steps towards reinforcement learning at the CERN Low Energy Ion Ring, in *2nd ICFA Mini-Workshop on Machine Learning for Charged Particle Accelerators* (PSI, Villigen, Switzerland, 2019).
- [34] 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).
- [35] T. Boltz, E. Bründermann, M. Caselle, A. Kopmann, W. Mexner, A.-S. Müller, and W. Wang, Accelerating machine learning for machine physics (an AMALEA-project at KIT), in *Proceedings, International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'19)* (JACoW, Geneva, Switzerland, 2020) p. 781.
- [36] 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).
- [37] 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).
- [38] A. G. Barto, R. S. Sutton, and C. W. Anderson, Neuron-like adaptive elements that can solve difficult learning control problems, *IEEE Trans. Sys. Man Cybern.* **SMC-13**, 834 (1983).
- [39] C. R. Banbury, V. J. Reddi, M. Lam, W. Fu, A. Fazel, J. Holleman, X. Huang, R. Hurtado, D. Kanter, A. Lokhmotov, D. Patterson, D. Pau, J. sun Seo, J. Sieracki, U. Thakker, M. Verhelst, and P. Yadav, Benchmarking TinyML systems: Challenges and direction, arXiv:2003.04821 (2020).
- [40] CMS Collaboration, *The Phase-2 upgrade of the CMS Level-1 trigger*, CMS Technical Design Report CERN-LHCC-2020-004. CMS-TDR-021 (2020).
- [41] S. I. Venieris, A. Kouris, and C.-S. Bouganis, Toolflows for mapping convolutional neural networks on FPGAs: A survey and future directions, *ACM Comput. Surv.* **51**, 1 (2018), arXiv:1803.05900.
- [42] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, A survey of FPGA-based neural network inference accelerators, *ACM Trans. Reconfigurable Technol. Syst.* **12**, 1 (2019), arXiv:1712.08934.
- [43] A. Shawahna, S. M. Sait, and A. El-Maleh, FPGA-based accelerators of deep learning networks for learning and classification: A review, *IEEE Access* **7**, 7823 (2019), arXiv:1901.00121.
- [44] K. Abdelouahab, M. Pelcat, J. Serot, and F. Berry, Accelerating cnn inference on FPGAs: A survey, arXiv:1806.01683 (2018).
- [45] S. I. Venieris and C. S. Bouganis, FPGAConvNet:

- A toolflow for mapping diverse convolutional neural networks on embedded FPGAs, in *NIPS 2017 Workshop on Machine Learning on the Phone and other Consumer Devices* (<https://sites.google.com/view/nips-2017-on-device-ml>, Long Beach, CA, USA, 2017) arXiv:1711.08740.
- [46] S. I. Venieris and C. S. Bouganis, FPGACONVNET: Automated mapping of convolutional neural networks on FPGAs, in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (ACM, New York, NY, USA, 2017) p. 291.
- [47] S. I. Venieris and C. S. Bouganis, Latency-driven design for FPGA-based convolutional neural networks, in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)* (IEEE, New York, NY, USA, 2017) p. 1.
- [48] S. I. Venieris and C. S. Bouganis, FPGACONVNET: A framework for mapping convolutional neural networks on FPGAs, in *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)* (IEEE, New York, NY, USA, 2016) p. 40.
- [49] Y. Guan, H. Liang, N. Xu, W. Wang, S. Shi, X. Chen, G. Sun, W. Zhang, and J. Cong, FP-DNN: An automated framework for mapping deep neural networks onto FPGAs with RTL-HLS hybrid templates, in *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)* (IEEE, New York, NY, USA, 2017) p. 152.
- [50] H. Sharma, J. Park, D. Mahajan, E. Amaro, J. K. Kim, C. Shao, A. Mishra, and H. Esmailzadeh, From high-level deep neural models to FPGAs, in *49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)* (IEEE, New York, NY, USA, 2016) p. 1.
- [51] R. DiCecco, G. Lacey, J. Vasiljevic, P. Chow, G. Taylor, and S. Areibi, Caffeinated FPGAs: FPGA framework for convolutional neural networks, in *2016 International Conference on Field-Programmable Technology (FPT)* (IEEE, New York, NY, USA, 2016) p. 265, arXiv:1609.09671.
- [52] Xilinx, Xilinx/VITIS-AI, <https://github.com/Xilinx/Vitis-AI> (2021).
- [53] M. Blott, T. Preusser, N. Fraser, G. Gambardella, K. O'Brien, and Y. Umuroglu, FINN-R: An end-to-end deep-learning framework for fast exploration of quantized neural networks, *ACM Trans. Reconfigurable Technol. Syst.* **11**, 1 (2018), arXiv:1809.04570.
- [54] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, FINN: A framework for fast, scalable binarized neural network inference, in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (ACM, New York, NY, USA, 2017) p. 65, arXiv:1612.07119.
- [55] Y. Umuroglu *et al.*, Xilinx/FINN, <https://github.com/Xilinx/finn> (2021).
- [56] P. Whatmough, C. Zhou, P. Hansen, S. Venkataramanah, J.-s. Seo, and M. Mattina, FIXYNN: Energy-efficient real-time mobile computer vision hardware acceleration via transfer learning, in *Proceedings of Machine Learning and Systems*, Vol. 1, edited by A. Talwalkar, V. Smith, and M. Zaharia (mlsys.org, Palo Alto, CA, USA, 2019) p. 107, arXiv:1902.11128.
- [57] P. N. Whatmough *et al.*, ARM-software/DEEPPFREEZE, <https://github.com/ARM-software/DeepFreeze> (2019).
- [58] J. Duarte *et al.*, Fast inference of deep neural networks in FPGAs for particle physics, *J. Instrum.* **2018** (07), P07027, arXiv:1804.06913.
- [59] V. Loncar, S. Summers, N. Tran, B. Kreis, J. Ngadiuba, J. Duarte, D. Hoang, E. Kreinar, D. Golubovic, Y. Iiyama, Z. Wu, P. Cretaro, P. Zejdl, and D. Rankin, *fastmachinelearning/HLS4ML: Aster* (2020), <https://doi.org/10.5281/zenodo.4161550>.
- [60] J. Ngadiuba, V. Loncar, M. Pierini, S. Summers, G. Di Guglielmo, J. Duarte, P. Harris, D. Rankin, S. Jindariani, M. Liu, *et al.*, Compressing deep neural networks on FPGAs to binary and ternary precision with HLS4ML, *Mach. Learn.: Sci. Technol.* **2**, 015001 (2020), arXiv:2003.06308.
- [61] S. Summers, G. D. Guglielmo, J. Duarte, P. Harris, D. Hoang, S. Jindariani, E. Kreinar, V. Loncar, J. Ngadiuba, M. Pierini, D. Rankin, N. Tran, and Z. Wu, Fast inference of boosted decision trees in FPGAs for particle physics, *J. Instrum.* **2020** (05), P05026, arXiv:2002.02534.
- [62] C. N. Coelho, A. Kuusela, S. Li, H. Zhuang, J. Ngadiuba, T. K. Aarrestad, V. Loncar, M. Pierini, A. A. Pol, and S. Summers, Automatic heterogeneous quantization of deep neural networks for low-latency inference on the edge for particle detectors, *Nat. Mach. Intell.* **3**, 675 (2021), arXiv:2006.10159.
- [63] T. Aarrestad, V. Loncar, N. Ghilmetti, M. Pierini, S. Summers, J. Ngadiuba, C. Petersson, H. Linander, Y. Iiyama, G. D. Guglielmo, J. Duarte, P. Harris, D. Rankin, S. Jindariani, K. Pedro, N. Tran, M. Liu, E. Kreinar, Z. Wu, and D. Hoang, Fast convolutional neural networks on FPGAs with HLS4ML, *Mach. Learn.: Sci. Tech.* **2**, 045015 (2021), arXiv:2101.05108 [cs.LG].
- [64] Y. Iiyama *et al.*, Distance-weighted graph neural networks on FPGAs for real-time particle reconstruction in high energy physics, *Front. Big Data* **3**, 44 (2021), arXiv:2008.03601.
- [65] A. Heintz, V. Razavimaleki, J. Duarte, G. DeZoort, I. Ojalvo, S. Thais, M. Atkinson, M. Neubauer, L. Gray, S. Jindariani, N. Tran, P. Harris, D. Rankin, T. Aarrestad, V. Loncar, M. Pierini, S. Summers, J. Ngadiuba, M. Liu, E. Kreinar, and Z. Wu, Accelerated charged particle tracking with graph neural networks on FPGAs, in *3rd Machine Learning and the Physical Sciences Workshop at the 34th Conference on Neural Information Processing Systems* (ml4physicalsciences.github.io, Vancouver, Canada, 2020) arXiv:2012.01563.
- [66] F. Fahim, B. Hawks, C. Herwig, J. Hirschauer, S. Jindariani, N. Tran, L. Carloni, G. D. Guglielmo, P. Harris, J. Krupa, D. Rankin, M. B. Valentin, J. Hester, Y. Luo, J. Mamish, S. Memik, T. Aarrestad, H. Javed, V. Loncar, M. Pierini, A. A. Pol, S. Summers, J. Duarte, S. Hauck, S.-C. Hsu, J. Ngadiuba, M. Liu, D. Hoang, E. Kreinar, and Z. Wu, HLS4ML: An open-source codesign workflow to empower scientific low-power machine learning devices, in *1st tinyML Research Symposium* (OpenReview, Burlingame, CA, USA, 2021) arXiv:2103.05579.
- [67] K. Majumder and U. Bondhugula, A flexible FPGA accelerator for convolutional neural networks, arXiv:1912.07284 (2019).
- [68] A. Rahman, J. Lee, and K. Choi, Efficient FPGA acceleration of convolutional neural networks using logical-3D compute array, in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)* (IEEE, New York, NY, USA, 2016) p. 1393.

- [69] G. B. Hacene, V. Gripon, M. Arzel, N. Farrugia, and Y. Bengio, Quantized guided pruning for efficient hardware implementations of convolutional neural networks, in *2020 18th IEEE International New Circuits and Systems Conference (NEWCAS)* (IEEE, New York, NY, USA, 2020) p. 206, [arXiv:1812.11337](https://arxiv.org/abs/1812.11337).
- [70] S.-E. Chang, Y. Li, M. Sun, R. Shi, H. K. H. So, X. Qian, Y. Wang, and X. Lin, Mix and match: A novel FPGA-centric deep neural network quantization framework, in *27th IEEE International Symposium on High-Performance Computer Architecture (HPCA)* (IEEE, New York, NY, USA, 2021) [arXiv:2012.04240](https://arxiv.org/abs/2012.04240).
- [71] J. Crawford, B. Pellico, J. Morgan, C. Gattuso, J. Reyna, B. Drendel, T. Sullivan, C. Broy, A. Meyhoefer, D. Newhart, B. Worthel, *et al.*, *Booster Rookie Book*, Tech. Rep. (Fermilab, Batavia, IL, USA, 2009) https://operations.fnal.gov/rookie_books/Booster_V4.1.pdf.
- [72] J. Ryk, *Gradient Magnet Power Supply for the Fermilab 8-GeV Proton Synchrotron*, Tech. Rep. FERMILAB-PUB-74-085 (1974).
- [73] N. Minorsky., Directional stability of automatically steered bodies, *J. Am. Soc. Nav. Engineers* **34**, 280 (1922).
- [74] J. G. Ziegler and N. B. Nichols, Optimum settings for automatic controllers, *J. Dyn. Syst. Meas. Control* **64**, 759 (1942).
- [75] K. Cahill *et al.*, The Fermilab accelerator control system, *ICFA Beam Dyn. Newslett.* **47**, 106 (2008).
- [76] C. Briegel, G. Johnson, and L. Winterowd, The Fermilab ACNET upgrade, *Nucl. Instrum. Methods Phys. Res. A* **293**, 235 (1990).
- [77] D. Kafkes and J. St. John, BOOSTR: A dataset for accelerator control systems, *Data* **6**, 42 (2021), [arXiv:2101.08359](https://arxiv.org/abs/2101.08359).
- [78] C. W. J. Granger, Investigating causal relations by econometric models and cross-spectral methods, *Econometrica* **37**, 424 (1969).
- [79] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, and I. Stoica, Apache Spark: A unified engine for big data processing, *Commun. ACM* **59**, 56 (2016).
- [80] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, Human-level control through deep reinforcement learning, *Nature* **518**, 529 (2015).
- [81] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, Playing Atari with deep reinforcement learning, in *NIPS Deep Learning Workshop 2013* (<https://sites.google.com/site/deeplearningworkshopnips2013>, Lake Tahoe, CA, USA, 2013) [arXiv:1312.5602](https://arxiv.org/abs/1312.5602).
- [82] R. Bellman, On the theory of dynamic programming, *Proc. Natl. Acad. Sci. U.S.A* **38**, 716 (1952).
- [83] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, Continuous control with deep reinforcement learning, in *4th International Conference on Learning Representations (ICLR 2016)* (OpenReview, Puerto Rico,, 2016) [arXiv:1509.02971](https://arxiv.org/abs/1509.02971).
- [84] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, Proximal policy optimization algorithms (2017), [arXiv:1707.06347](https://arxiv.org/abs/1707.06347).
- [85] S. Fujimoto, H. van Hoof, and D. Meger, Addressing function approximation error in actor-critic methods, in *35th International Conference on Machine Learning*, Proceedings of Machine Learning Research, Vol. 80, edited by J. Dy and A. Krause (PMLR, Stockholm, Sweden, 2018) p. 1587, [arXiv:1802.09477](https://arxiv.org/abs/1802.09477) [cs.AI].
- [86] A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel, Overcoming exploration in reinforcement learning with demonstrations, in *2018 IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, New York, NY, USA, 2018) p. 6292, [arXiv:1709.10089](https://arxiv.org/abs/1709.10089).
- [87] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, Hindsight experience replay, in *Advances in Neural Information Processing Systems 30*, edited by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Curran Associates, Inc., Red Hook, NY, USA, 2017) p. 5048, [arXiv:1707.01495](https://arxiv.org/abs/1707.01495).
- [88] S. Hochreiter and J. Schmidhuber, Long short-term memory, *Neural Comput.* **9**, 1735 (1997).
- [89] R. Pascanu, T. Mikolov, and Y. Bengio, On the difficulty of training recurrent neural networks (PMLR, Atlanta, Georgia, USA, 2013) p. 1310, [arXiv:1211.5063](https://arxiv.org/abs/1211.5063).
- [90] Y. Bengio, P. Simard, and P. Frasconi, Learning long-term dependencies with gradient descent is difficult, *IEEE Trans. Neural Netw.* **5**, 157 (1994).
- [91] F. Chollet *et al.*, Keras, <https://keras.io> (2015).
- [92] D. P. Kingma and J. Ba, Adam: A method for stochastic optimization, in *3rd International Conference on Learning Representations, ICLR 2015* (ICLR, San Diego, CA, USA, 2015) [arXiv:1412.6980](https://arxiv.org/abs/1412.6980).
- [93] H. V. Hasselt, Double Q-learning, in *Advances in Neural Information Processing Systems 23*, edited by J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta (Curran Associates, Inc., Red Hook, NY, USA, 2010) p. 2613.
- [94] H. van Hasselt, A. Guez, and D. Silver, Deep reinforcement learning with double Q-learning, in *30th AAAI Conference on Artificial Intelligence (AAAI, Palo Alto, CA, USA, 2016)* [arXiv:1509.06461](https://arxiv.org/abs/1509.06461).
- [95] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, OpenAI gym, [arXiv:1606.01540](https://arxiv.org/abs/1606.01540) (2016).
- [96] C. J. C. H. Watkins, *Learning from Delayed Rewards*, Ph.D. thesis, King's College, Cambridge, UK (1989).
- [97] A. F. Agarap, Deep learning using rectified linear units (ReLU), [arXiv:1803.08375](https://arxiv.org/abs/1803.08375) (2018).
- [98] H. Javed, A Quartus backend for HLS4ML: Deploying low-latency neural networks on Intel FPGAs, in *Fast Machine Learning for Science Workshop* (Zenodo, Geneva, Switzerland, 2020).
- [99] Intel Corporation, *Intel Quartus Prime Pro Edition User Guide*, Tech. Rep. UG-20129 (Intel, Santa Clara, CA, USA, 2021) <https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ug/ug-qpp-getting-started.pdf>.