END-TO-END DIFFERENTIABLE DIGITAL TWIN FOR THE IOTA/FAST FACILITY*

N. Kuklev†, M. Wallbank, N. Banerjee, J. Jarvis, A. Romanov
Fermi National National Laboratory, Batavia, IL, USA

Abstract

As the design complexity of modern accelerators grows, there is more interest in using controllable-fidelity simulations that have fast execution time or can yield additional insights about accelerator state. One notable example of additional information are gradients of physical observables with respect to design parameters produced by differentiable simulations. The IOTA/FAST facility has recently begun a program to implement and experimentally validate an end-to-end digital twin to serve as a virtual accelerator test stand, allowing for rapid prototyping of new software and experiments with minimal beam time costs. In this contribution we will discuss our plans and progress. Specifically, we will cover the selection and benchmarking of both physics and ML codes for linac and ring simulation, the development of generic interfaces between surrogate and physics-based sections, and presenting the control interface as either a deterministic event loop or a fully asynchronous EPICS soft input/output controller. We will also discuss challenges in model calibration and uncertainty quantification, as well as future plans to implement larger proton accelerators like PIPII and Booster.

INTRODUCTION

Particle accelerator projects face increasing performance demands, resulting in tighter tolerances on lattice accuracy and stability. Achieving these tolerances requires both more advanced diagnostics and more commissioning time, which contributes to rising costs and lower beam availability.

A common approach to reduce commissioning risks is to implement a realistic version of the accelerator lattice as a simulation and perform full simulated commissioning. A recent successful example of this approach is the Advanced Photon Source Upgrade Project (APS-U), which developed a commissioning plan that included advanced ELEGANT simulations, detailed task scheduling, and many validation studies, including within a dedicated EPICS control network [1, 2].

However, so far most commissioning simulations have focused on the specific needs and quirks of their respective machines, avoiding complexity through implementing only physics models and control system behavior relevant to a particular project. This approach is not scalable, especially for a research facility like IOTA/FAST, since our machines are in constant flux and undergoing upgrades as the various experiments and elements are installed. Moreover, Fermilab main complex is currently undergoing several major upgrades, PIP-II (Proton Improvement Plan II) and ACORN (Accelerator Controls Operations Research Network). These will result in a mix of ACNET and EPICS controls systems, with brand new devices having to interface with those over 40 years old.

Given above complexity, it is clear that significant simulated efforts will be needed to ensure smooth commissioning and operation. We require a tool capable of testing classic methods (such as optics measurements via LOCO [3,4]), but also amenable to training and validating a new generation of machine learning (ML) tools. Notable examples of the latter include Bayesian optimization (BO) [5–7], reinforcement learning (RL) [8], and phase space reconstruction [9]. ML methods are especially promising to address issues that have previously relied on operator experience and incomplete, qualitative simulations due to both errors/uncertainties in the models and unsuitability of classical optimization methods. This paper presents our implementation of a 'digital twin' toolkit as a potential unified solution. We will first describe the key requirements. Then, architectural details will be presented, with emphasis on differentiability. Finally, we will show use cases and our first upcoming commissioning project, the IOTA proton injector.

DIGITAL TWINS

Throughout this paper, we will use the term '(surrogate) model' to describe the specific (approximate) implementations of accelerator simulations, for example an ImpactX simulation or a neural network that can approximate the tracking of a beam with space charge. We will use the term 'digital twin' to describe a complete framework that exposes and coordinates both standard and surrogate models to form a complete virtual environment. One can think of digital twins as backend engines for a virtual accelerator test stand, whereby various programs and users can interact with the models in a configurable environment with appropriate mix of control systems, simulation fidelity, and computational requirements.

Our plan for digital twin development has three main goals aligned both with internal Fermilab project priorities and more global challenges described in the recent DOE GARD accelerator and beam physics roadmap [10, 11]:

- To provide a high fidelity virtual test stand for testing tools and procedures, including commissioning and operational applications as well as research codes

* This manuscript has been authored by FermiForward Discovery Group, LLC under Contract No. 89243024CSC000002 with the U.S. Department of Energy, Office of Science, Office of High Energy Physics.
† nkuklev@fnal.gov

- To enable training and validation environments for ML-based surrogates and computational methods (differentiable modelling), including integration with both standard physics codes and ML frameworks
- To generate a near real-time model of the accelerator that is adaptively calibrated against incoming data and provides accurate prediction and uncertainty quantification of key lattice and beam parameters

ARCHITECTURE

While interlinked, above goals place unique requirement on the digital twin architecture. For instance, to ensure compatibility of several simulation codes, data structures are necessary that can keep track of and convert phase space coordinates. To enable model calibration, a bidirectional data flow interface is necessary along with capability to 'assimilate' experimental readings. At its core however, most of these challenges boil down to either optimization (of real or model parameters) and/or doing simulations quickly. Thus, inspired by Greek mythology, we chose to name our library Apollo (Accelerator Pipeline for Optimization and Low-Latency Operation). Figure 1 shows the general 3 layer architecture of Apollo.

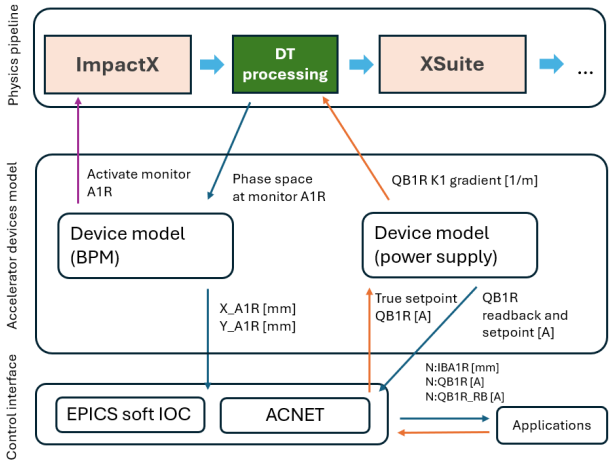

Figure 1: Apollo modular architecture.

Physics pipeline

The first layer of Apollo serves to create or wrap various standard and ML-based simulation code into a pipeline that can be portably executed as a single unit. After a large review of active simulation codes, we chose to support two natively - ImpactX [12] and Xsuite [13]. Key determining factors were availability of Python bindings, active development community, and enough physics models to cover most of the elements we expect to use. Of special importance is support for intense space charge and other collective effects. Other Fermilab groups also had good experiences with these codes. Native support indicates that Apollo can 'synthesize' both simulation commands and lattices on the fly from a common shared format, transparently handling implementation details like dipole edges and coordinate transformations. For other codes, or custom models, a black-box interface specification can be used. For differentiable models, we support Jax and PyTorch-based arbitrary modules and provide our own tracking code, as discussed in the next section.

Device models

Second layer of Apollo is responsible for determining how to convert parameters and outputs between the physics pipeline and the real world. For example, calculating the magnet gradient based on the virtual power supply current, or beam position from the phase space. Device interface is kept generic, resembling that of discrete event simulators and allowing for complex time-dependent implementation logic. For example, our standard power supply model includes hysteresis, ADC noise, and overshoot behavior. Active devices set determines which simulation outputs are requested, so as to minimize unnecessary computations.

Control interface

Final layer of Apollo is responsible for presenting the devices to clients through a control system layer. It runs asynchronously to the simulation loop, ensuring prompt network communication with multiple clients. We use external libraries for EPICS implementation, with CA interface through 'caproto' and PV access through 'pvapy'. Both present devices as configurable sets of records, with appropriate fields and support for monitoring and scanning. No Python server implementation exists for ACNET, so we supply a custom shim that transparently reroutes client requests.

System design

Throughout Apollo we imposed additional requirements to ensure the implementation remains flexible, portable, and yet robust to misconfigurations or bugs. Namely, each layer must be independently serializable and executable (i.e. in cluster environment). Parameters and return values should be validated against specifications at every step. This is accomplished by using 'Pydantic' library and a set of declarative inputs and output for each model and device. Pydantic enables serialization to and from json, yaml, and other formats. For example, a model might declare phase space output through OpenPMD standard [14] or as a custom one with:

```
"outputs": {
    "phase_space": {
        "type": "coordinates",
        "dtype": "float64",
        "tsystem": "momenta",
        "lsystem": "zeta_ptau",
        ...
    }
    ...
}
```

We avoid complexity of nested data containers by using a single shared namespace, and a single state object that carries both current values and history. This ensures synchronized modification, but does limit scalability, especially in Python. We mitigate this by offloading execution to distributed frameworks like Ray and Dask, keeping coordinator thread free.

DIFFERENTIABLE SIMULATIONS

Modern ML frameworks have made it feasible to implement complex physics models with automatic differentiation. Such differentiable models can be run just like regular simulations, but in addition to output values yield the gradients of outputs with respect to a designated set of parameters (from lattice or beam). They are useful in accelerator design, optimization, model calibration, and machine learning. Apollo implements a special pipeline type if all component models can do gradient propagation, with support for PyTorch and Jax. The former is more dynamic and easier to work with, while the latter is significantly faster. Both ecosystems also have packages for NN training/serving, RL, and Bayesian inference (Pyro [15], NumPyro [16]), and will be useful for both calibration and surrogate modelling [17]. Uniquely for PyTorch, we support uncertainty-aware modules like Gaussian processes.

Because Apollo prioritizes fast execution, we developed a new native Jax-based accelerator tracking code called JACC (Jax for ACCelerators). It implements standard 6x6 first order matrices as well as select second order matrices and drift-kick integrators through bends, quads, multipoles, and RF elements. There is also a single-bunch space charge implementation, and implicit vectorization capability consistent with Jax 'vmap' style. The whole beamline is compiled as a single unit to either CPU or GPU, with pruning and element fusion where possible.

USE CASES

IOTA proton injector

IOTA proton injector (IPI) project aims to add a 2.5MeV proton injector alongside existing 150MeV electron injection line [18, 19]. Scientifically, this will add capability to study space charge dominated beams and their control methods using a stable storage ring with lattice accuracy comparable to light sources. Hardware is currently being installed, with commissioning set to begin in the fall of 2025. Diagram of the IPI is shown in Figure 2.

IPI will be the first machine to use Apollo to simulate commissioning and infer currently unclear beamline parameters, such as the beam phase space coming from the RFQ and element misalignments. Full MEBT section lattice is already implemented in both standard and differentiable formats, and work is ongoing to construct an open-source RFQ tracking model.

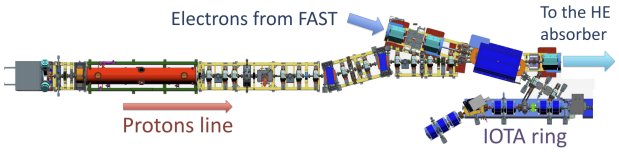

Figure 2: IOTA proton injector layout and integration with existing electron beamlines. Proton line consists of the duoplasmotron source (left end, gray), LEBT (yellow, short), RFQ (red), and MEBT (yellow, long) section with dogleg, debuncher, and various diagnostics.

FNAL main complex in the PIP-II era

Supporting all main complex machines is the long term goal for Apollo. This is made difficult due to the complex physics (SRF cavities, ramping machines, impedances, feedback systems) and a large number of codes used by various departments. For PIP-II, we have started adding advanced cavity and RFQ models to better match the reference TraceWin implementation of the PIP-II linac. Because TraceWin is commercial software, we cannot use it as one of the models directly. For Booster, for key beam power limiting factors like beam loss locations we are training data-driven surrogates, since simulations do not yet have quantitative agreement. Main Injector already uses Xsuite simulation extensively, and we expect no major roadblocks.

CONCLUSION

Robust virtual simulation and testing environments are important for ensuring efficient design, commissioning, and operation of the new generation of particle accelerators. They enable development of many other capabilities, and thanks to differentiable codes are a powerful inference and model calibration tool in their own right. In this paper we have described our development of a differentiable digital twin framework 'Apollo'. It is our hope that using a modern software stack, robust system design practices, and a modular architecture will cover the majority of use cases at Fermilab and beyond, creating a first generic digital twin framework for accelerators. We are planning on using Apollo for IOTA IPI commissioning this fall, and through this early experimental testing discover and address any issues well ahead of PIP-II commissioning. Future work will focus on improving user experience, integration with live data readout, as well as extending the differentiable capabilities with advanced impedance, cavity, and feedback models.

REFERENCES

[1] V. Sajaev, "Commissioning simulations for the argonne advanced photon source upgrade lattice", *Phys. Rev. Accel. Beams*, vol. 22, no. 4, p. 040 102, 2019. doi:10.1103/PhysRevAccelBeams.22.040102

[2] V. Sajaev, "Improvements to the commissioning simulations of the APS Upgrade storage ring", in *Proc. IPAC'23*, Venice, Italy, pp. 3112–3115, 2023. doi:10.18429/JACoW-IPAC2023-WEPL006

[3] D. Vilsmeier, R. Singh, and M. Bai, "Inverse modeling of circular lattices via orbit response measurements in the presence of degeneracy", *Phys. Rev. Accel. Beams*, vol. 26, no. 3, p. 032 803, 2023. doi:10.1103/PhysRevAccelBeams.26.032803

[4] A. Romanov *et al.*, *Correction of magnetic optics and beam trajectory using loco based algorithm with expanded experimental data sets*, 2017. doi:10.48550/arXiv.1703.09757

[5] N. Kuklev, M. Borland, G. I. Fystro, H. Shang, and Y. Sun, "Online Accelerator Tuning with Adaptive Bayesian Optimization", in *Proc. NAPAC'22*, Albuquerque, NM, USA, pp. 842–845, 2022. doi:10.18429/JACoW-NAPAC2022-THXD4

[6] N. Kuklev, M. Borland, G. Fystro, H. Shang, and Y. Sun, "Robust adaptive bayesian optimization", in *Proc. IPAC'23*, Venice, Italy, pp. 4428–4431, 2023. doi:10.18429/JACoW-IPAC2023-THPL007

[7] R. Roussel *et al.*, "Bayesian optimization algorithms for accelerator physics", *Phys. Rev. Accel. Beams*, vol. 27, no. 8, p. 084 801, 2024. doi:10.1103/PhysRevAccelBeams.27.084801

[8] V. Francois-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, "An Introduction to Deep Reinforcement Learning", *FNT in Machine Learning*, vol. 11, no. 3-4, pp. 219–354, 2018. doi:10.1561/2200000071

[9] R. Roussel *et al.*, "Efficient six-dimensional phase space reconstructions from experimental measurements using generative machine learning", *Phys. Rev. Accel. Beams*, vol. 27, no. 9, p. 094 601, 2024. doi:10.1103/PhysRevAccelBeams.27.094601

[10] *Accelerator beam physics research roadmap*. https://science.osti.gov/hep/-/media/hep/pdf/2022/ABP_Roadmap_2023_final.pdf

[11] S. Nagaitsev *et al.*, *Accelerator and beam physics research goals and opportunities*, 2021. https://arxiv.org/abs/2101.04107

[12] A. Huebl *et al.*, "Next Generation Computational Tools for the Modeling and Design of Particle Accelerators at Exascale", in *Proc. NAPAC'22*, Albuquerque, NM, USA, pp. 302–306, 2022. doi:10.18429/JACoW-NAPAC2022-TUYE2

[13] G. Iadarola *et al.*, "Xsuite: An Integrated Beam Physics Simulation Framework", *JACoW*, vol. HB2023, p. TUA2I1, 2024. doi:10.18429/JACoW-HB2023-TUA2I1

[14] *C++ and python api for scientific i/o with openpmd*. https://github.com/openPMD/openPMD-api

[15] E. Bingham *et al.*, "Pyro: Deep universal probabilistic programming", *J. Mach. Learn. Res.*, vol. 20, p. 28:1–28:6, 2019. http://jmlr.org/papers/v20/18-403.html

[16] J. Bradbury *et al.*, *JAX: Composable transformations of Python+NumPy programs*, version 0.4.8, 2023. http://github.com/google/jax

[17] N. Kuklev, "Differentiable beam optics optimization and measurement", in *Proc. IPAC'23*, Venice, Italy, pp. 3108–3111, 2023. doi:10.18429/JACoW-IPAC2023-WEPL004

[18] D. Edstrom *et al.*, "IOTA Proton Injector Beamline Installation", in *Proc. IPAC'23*, Venice, Italy, pp. 1737–1739, 2023. doi:10.18429/JACoW-IPAC2023-TUPA183

[19] J. Wieland and A. Romanov, "Injection simulations of space charge dominated proton beams in IOTA".