ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TU1BCO01

A WORKFLOW FOR TRAINING AND DEPLOYING MACHINE LEARNING MODELS IN EPICS

M. Leputa*, K. R. L. Baker, M. Romanovschi
ISIS Neutron and Muon Source, Didcot, United Kingdom

Abstract

The transition to EPICS as the control system for the ISIS Neutron and Muon Source accelerators is an opportunity to more easily integrate machine learning into operations. However, developing high-quality machine learning (ML) models is insufficient. Integration into critical operations requires good development practices to ensure stability and reliability during deployment and to allow robust and easy maintenance. For these reasons we implemented a workflow for training and deploying models that utilise off-the-shelf, industry-standard tools such as MLflow. Our experience of how adoption of these tools can make developer's lives easier during the training phase of a project is discussed. We describe how these tools may be used in an automated deployment pipeline to allow the ML model to interact with our EPICS ecosystem through Python-based IOCs within a containerised environment. This reduces the developer effort required to produce GUIs to interact with the models within the ISIS Main Control Room as tools familiar to operators, such as Phoebus, may be used.

INTRODUCTION

The ISIS Neutron and Muon Source accelerators [1] control system is presently undergoing a transition from a Vsystem (Vista Control system [2]) to an EPICS (Experimental Physics and Industrial Control system [3]) based control system [4].

Alongside this migration, an expansion of data monitoring and collection systems has been implemented, incorporating many new software stacks; including but not limited to EPICS Achiever Appliance [5], and Influx-DB [6] as well as the related metrics collection and analysis suites (Telegraf, InfluxDB, Grafana, etc.) [7].

Moreover, a significant milestone has been achieved in the form of digitisation of the Analog Waveform System (AWS) [8, 9], which has provided digital waveforms of key accelerator systems; to date only stored as hourly images of screen-captures from oscilloscopes.

All of these enhancements along with an increase in the quality and volume of data, as well as developments in machine learning frameworks (such as TensorFlow [10] and PyTorch [11]) and commonly available powerful hardware accelerators, have enabled the ISIS controls group to leverage these advances and begin development of advanced control, optimisation and monitoring systems.

In this paper, we will discuss the tooling, workflow and two sample deployments of machine learning projects using EPICS. We will demonstrate how the deployments conform to a user interface that the operators in the ISIS main control room (MCR) are already familiar with and how the deployments require minimal setup from the operators side. The advantages of this workflow are discussed as well as plans for further development.

MACHINE LEARNING OPERATIONS

Motivation

The ISIS controls group is a relatively small team with many overlapping responsibilities throughout the accelerator; as such it is an implicit requirement that any software and infrastructure developed by the team is easy to maintain, develop and hand over to other members. To that end, the team follows a set of best practices and principles in their software and infrastructure development; the core of which include, but are not limited to:

- **Modular Architecture** - The team designs software in a modular fashion, ensuring minimal coupling between different objects both at the code level and service level (when said code is deployed).
- **Version Control Systems (VCS)** - As is standard in most software development teams the team uses a version control system (in our case git [12] and GitLab [13]) to manage the code-base. This allows for tracking changes, better collaboration, and facilitates continuous knowledge transfer within the team.
- **Continuous Integration and Continuous Deployment (CI/CD)** - CI/CD pipelines built on top of the controls group VCS allows for easy automation of repetitive tasks, including but not limited to testing, deployment, and evaluation of the code-base.

These practices have been singled out due to their significant impact on productivity relative to the time invested in their implementation. Consequently, they were chosen as a baseline requirement for the target machine learning workflow that the team would adopt.

Machine learning projects also face additional challenges, these are best exemplified in Burkov's chapter on why machine learning projects fail [14]. With the most relevant and actionable within the current scope of the project being:

- **Siloed Organizations and Lack of Collaboration** - Lack of standardised workflows and pipelines leads to every project being a "one-off" collection of scripts and code that only the original developer can realistically deploy and maintain.

* mateusz.leputa@stfc.ac.uk

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TU1BCO01

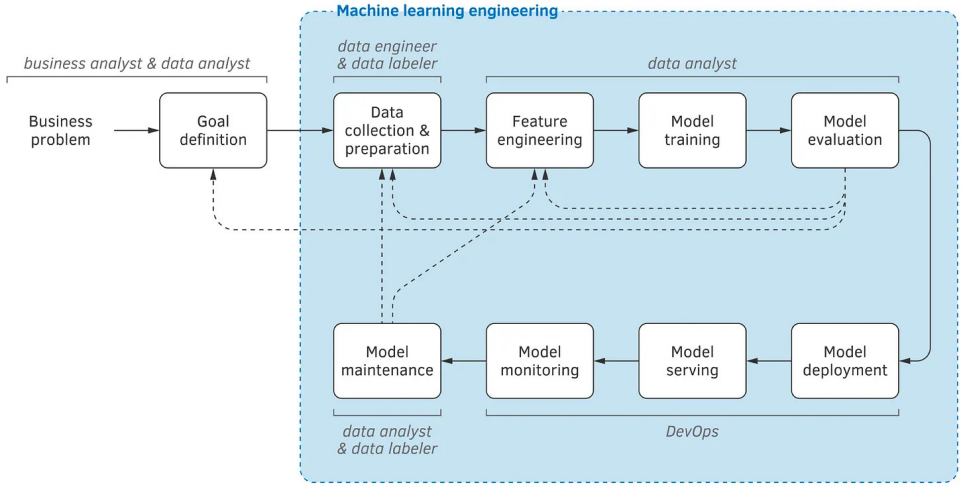


Figure 1: Machine Learning Project Life cycle, Burkov (2020)

- **Missing Data Infrastructure** - Lack of easy access to data, especially through legacy systems and APIs; lack of standardised ways of obtaining and pre-processing data, will generally lead to flaky and opaque workflows that end up being unmaintainable in the long term.

We, therefore, seek to address the machine learning development challenges by adopting the already well-established software development best practices in addition to using domain-specific tooling and practices with minimal development and maintenance overhead. We also aim to minimise the time machine learning developers spend on non-machine learning tasks as well as the time required for handovers and setup of new projects. For the purposes of this paper we will adopt machine learning lifecycle terminology as described in Burkov [15] and shown in Fig. 1. We will also generally refer to the first five steps of *Goal Definition* though *Model Evaluation* as *development*.

System Architecture and Hardware

Figure 2 represents the current system architecture of the ISIS controls group machine learning development stack. Excluding the NFS file share and client applications, each unit represented in the diagram is a dockerised micro-service. These services are all located on clustered docker swarm servers with NFS mounted to the developer's clients as well as the remote development servers.

The development servers run an instance of JupyterHub [16] which manages and launches JupyterLab [17] instances to each system user. Each user has their own private work area as well as a shared area for collaboration. The JupyterHub host is a GPU-enabled machine hosting two Nvidia A100 [18] GPUs.

In the current setup, the use of GPUs is not restricted per instance but should the need to do so arise one can restrict the resources using a number of methods ranging from restrictions within the client code; such as the TensorFlow's or PyTorch's ability to limit GPU usage to sub-dividing the GPU using the Multi-Instance GPUs [19] or similar techniques. The CPU and host memory usage is, however, restricted through JupyterHub's configuration.

When users are developing models they use a model archiving and storage service, in our case MLflow [20], to archive the training artifacts (such as model files, sample outputs and model definitions) as well as model performance metrics recorded during training. MLflow provides a web front-end that can be used to compare model metrics, take notes, and mark models as ready for deployment. It also provides an API to do all of the above in a programmatic fashion. Note that there is an increasing number of alternative model archiving and life-cycle management tools with varying feature sets and commercialisation models to choose from. MLflow was chosen due to developer familiarity, the relatively low resource reqired to self-host, a permissive license, and being fully free and open-source.

The next stage in the workflow is automated using a docker-hosted model-deployment service. The service is developed in-house and uses the MLflow API. It checks if a model has been marked as ready for production and automatically deploys it to a TF-Serve [21] instance. Torch models still require manual deployments but TorchServe automated deployment [22] is under development. TorchServe is a PyTorch equivalent of TF-Serve. Both frameworks serve ML models as HTTP/S endpoints for client applications to access. The client applications are generally lightweight containerised services that ingest data from users or other systems (primarily EPICS) and pass the outputs to the front-end client where the end users of our systems are.

Development

The development environment set up on remote, GPU-enabled servers has so far proven to be of great advantage to ML developers. The JupyterLabs simplify the process of setting up a developer's workspace and dependencies as well as providing access the otherwise difficult-to-attain level of computing resources. Before the deployment of JupyterHub

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TU1BCO01

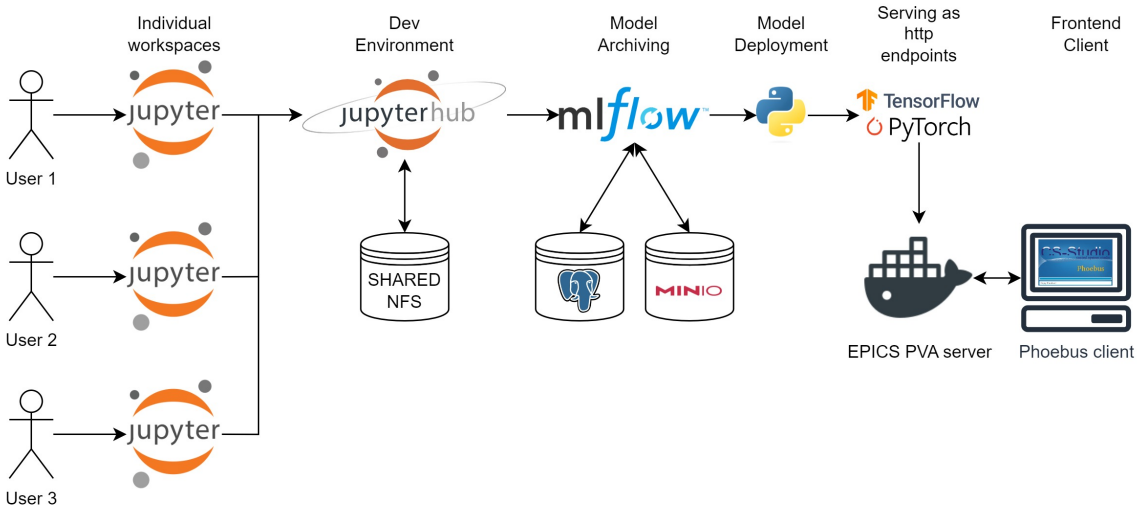


Figure 2: System architecture of the ML development setup for the ISIS controls group.

and MLflow, the developers would work entirely locally on their individual machines (mostly laptops) which made resource-intensive methods such as hyperparameter searches impossible. Whilst sharing the code base between the developers was well facilitated by the usage of git and GitLab, storage and sharing of data and AI models was not, along with the training and evaluation results. Moreover, each new model would require developers' time spent writing evaluation boilerplate code and model comparison. Much of this was addressed by the use of a model archiving framework, in our case MLflow, which through a web UI and an easy API allows one to track many experiments and make direct comparisons of chosen metrics. The models, results, and other artifacts can now be easily queried from the servers. This allows the developers to focus on the model quality. Greater computing resources also allow exploring a larger space of possible models in a shorter time frame, increasing overall team throughput. This setup also records the experiments' state, making a return to shelved or impeded projects easier for developers.

Deployment

When we started the development of the machine learning projects within the ISIS controls group, the deployment process was complex and demanded a significant amount of manual effort. The software deployment was done using USB drives, resulting in a relatively long turnaround time for new features, bug fixes, and model updates.

Recognising that many ML projects shared substantial amounts of code and had similar requirements, a clear path to a standardised workflow was established. The new deployment process takes advantage of the MLflow API and the readily available serving frameworks from TensorFlow and PyTorch. When a developer reaches an agreed level of the model's performance, the model in question is marked as production-ready using the MLflow web interface. A microservice referred to as a *model manager* periodically checks the MLFlow database for models with the production tag. Once such a model is added, it is downloaded and added to the deployments folder. The serving frameworks then serve the models as HTTP/S endpoints. The next step of the deployment is creating a client application, in the example in Fig. 2 a containerised EPICS pvAccess [23] server that captures the user input from a Phoebus [24] screen, performs the necessary formatting to make the request, makes a HTTP/S request and returns the data to the EPICS control system as EPICS process variables, which are then displayed on the operators' screen. The size of the deployment on the client side is now only reduced to the size of a Phoebus client; which in case of most of the operators was already installed on their machines. This installation is only in the order of 10s to 100s of MBytes where as a full client installation would have previously involved including the machine learning frameworks and other dependencies, which would frequently sum up to the order of GBytes.

EXAMPLES

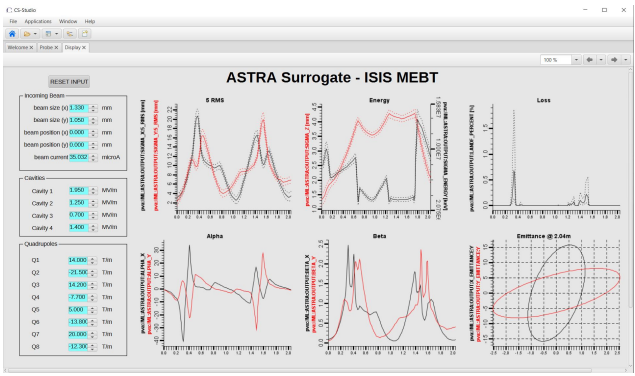
This section delves into two distinct machine learning deployments within ISIS:

- ASTRA Medium Energy Beam Transport Surrogate model (MEBT)
- Low Energy Beam Transport Optimisation model (LEBT)

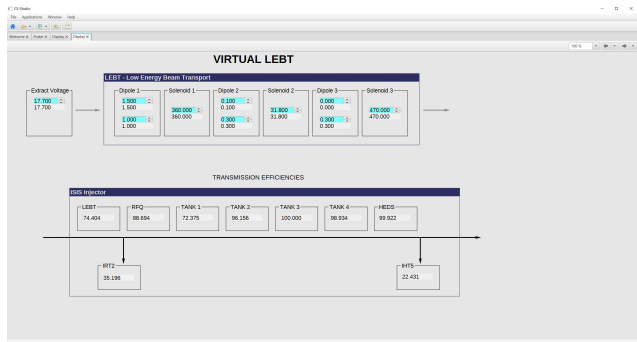
Each is tailored to enhance the performance and control of different beam transport systems.

ASTRA MEBT Surrogate Model The ASTRA MEBT Surrogate Model [25] is a PyTorch-based neural network. It is a surrogate model for the ASTRA particle tracking code [26] that simulates beam descriptors for the ISIS Medium Energy Beam Transport. It uses a combination of incoming beam and machine settings to predict the beam

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TU1BCO01



(a) ASTRA Phoebus Virtual diagnostic screen



(b) LEBT Phoebus Virtual diagnostic screen

Figure 3: Phoebus-based front-end examples for model deployments at ISIS Controls Group

envelope, energy spread, alpha and beta parameters, and beam loss along the MEBT. The model uses a relatively straightforward recurrent neural network architecture of 5 layers of 128 nodes each. The model inputs comprise the machine settings and distance along the MEBT at which we are predicting the beam parameters at discrete steps along the length of the instrument. The model is deployed as a containerised service which in its current state also hosts the production model, data processing, and a pvAccess server. Upon change to the model or any of the code-base; the container is rebuilt, tested, and redeployed using a CI/CD pipeline. The front end used to control the model can be seen in Fig. 3a. A deployment that does not host the model inside the service container, but instead inside the dedicated model serving frameworks is under active development.

LEBT Optimisation Model The LEBT machine learning project is a surrogate model for the ISIS Low Energy Beam Transport used as a tool for tuning and testing optimisation offline as ISIS moves towards wider automation across the control systems. The model inputs comprise 10 parameters defining changes to parameters in the LEBT, predicting the efficiencies after subsequent components to the linear accelerator. Similar to the ASTRA MEBT model, the model is a PyTorch neural network consisting of 3 layers of 64 nodes. This model is deployed using the TorchServe service as a HTTPS endpoint which is called by a service that implements a pvAserver that takes the inputs from the operator screen shown in Fig. 3b, sends them to the neural network and displays the output on the operator screen. The model also includes all the pre- and post-processing, saving the developers the need to manage three different models.

In both cases, the client utilises the p4p [27] library to create EPICS PVs (process variables) for each of the inputs and outputs, as well as some additional PVs calculated from the outputs if desired by the users. Phoebus is used to create control screens for each of the clients, although this is purely for convenience, other tools can be used to create GUIs if the application requires a user front end.

FUTURE DEVELOPMENTS

Further Automation While most of the existing development operations were adopted and adjusted to create better machine learning workflows, a portion of the work remains to be automated. For instance the HTTP/S to EPICS pvAccess services. These, are written manually but share substantial overlap between deployments; with the only major differences consisting of pre- and post-processing steps. These steps can also be deployed from the model archive as stand-alone scripts or integrated into the model itself.

At the time of writing only TensorFlow models are automatically deployed from MLflow using the model manager script. A PyTorch alternative is in development.

Profiling The ASTRA and LEBT models described earlier are relatively simple models with low inference times, especially when run through serving frameworks on GPU-enabled servers. The latency for the said queries ranges between 16 ms to 40 ms when querying the deployed applications, most of which is attributed to network latency. As the models that are deployed grow bigger this might not be the case anymore. Adoption or development of standardised profiling tools is therefore necessary as larger models and more applications in parallel will require more efficient use of available resources.

CONCLUSIONS

This paper presented a comprehensive overview of the current workflow for developing and deploying machine learning systems while also discussing possible future developments to further automate and streamline machine learning systems development and production. We illustrated these concepts through the examination of two deployments: ASTRA MEBT surrogate and LEBT optimisation surrogate.

Through this examination, we have demonstrated the advantages of the web-hosted development environment as well as the model archiving system MLflow. The models served using the MLflow API. Additionally, a road-map towards achieving a more complete system deployment automation was discussed.

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TU1BCO01

REFERENCES

[1] J. W. G. Thomason, "The ISIS spallation neutron and muon Source—The first thirty-three years", *Nucl. Instrum. Methods Phys. Res. Sect. A*, vol. 917, pp. 61–67, 2019. doi:10.1016/j.nima.2018.11.129

[2] Vista control system, https://www.vista-control.com

[3] EPICS control system, https://epics-controls.org

[4] I. D. Finch *et al.*, "Progress of the EPICS Transition at the ISIS Accelerators", presented at ICALEPCS'23, Cape Town, South Africa, 2023, paper TUPDP108, this conference.

[5] EPICS archiver applience, https://slacmshankar.github.io/epicsarchiver_docs/index.html

[6] InfluxDB, https://www.influxdata.com

[7] I. Finch, G. Howells, and A. Saoulis, "Controls Data Archiving at the ISIS Neutron and Muon Source for In-Depth Analysis and ML Applications", in *Proc. ICALEPCS'21*, Shanghai, China, 2022, paper WEPV049, pp. 780–783. doi:10.18429/JACoW-ICALEPCS2021-WEPV049

[8] W. Frank, B. Aljamal, and R. Washington, "Digitisation of the Analogue Waveform System at ISIS", in *Proc. ICALEPCS'21*, Shanghai, China, 2022, paper MOPV020, pp. 169–172. doi:10.18429/JACoW-ICALEPCS2021-MOPV020

[9] K. Koh and R. A. Washington, "Full scale system test of prototype digitised waveform system at ISIS", presented at ICALEPCS'23, Cape Town, South Africa, 2023, paper TH-PDP075, this conference.

[10] TensorFlow, https://www.tensorflow.org/

[11] PyTorch, https://pytorch.org/

[12] Git, https://git-scm.com/

[13] GitLab, https://about.gitlab.com/

[14] A. Burkov, "Chapter 1, section 2.5 - why machine learning projects fail", in *Machine Learning Engineering*. True Positive Inc., 2020.

[15] A. Burkov, "Chapter 1, section 1.7 - machine learning project lifecycle", in *Machine Learning Engineering*. True Positive Inc., 2020.

[16] JupyterHub, https://jupyter.org/hub

[17] JupyterLab, https://jupyter.org/

[18] Nvidia A100, https://www.nvidia.com/en-us/data-center/a100/

[19] Nvidia MIG, https://www.nvidia.com/en-gb/technologies/multi-instance-gpu/

[20] MLflow, https://mlflow.org/

[21] TensorFlow serving, https://www.tensorflow.org/tfx/guide/serving

[22] TorchServe, https://pytorch.org/serve/

[23] pvAccess and pvData, https://epics-controls.org/resources-and-support/documents/pvaccess/

[24] CS-Studio/Phoebus Documentation, https://control-system-studio.readthedocs.io/en/latest/

[25] K. R. L. Baker *et al.*, "Development of a Virtual Diagnostic for Estimating Key Beam Descriptors", in *Proc. IPAC'22*, Bangkok, Thailand, 2022, pp. 969–972. doi:10.18429/JACoW-IPAC2022-TUPOST048

[26] ASTRA particle tracking code, https://www.desy.de/~mpyflo/

[27] p4p, https://mdavidsaver.github.io/pvxs/overview.html