GENERIC PYTHON ALGORITHM PLUGIN FOR THE EPICS AREA DETECTOR FRAMEWORK

M. Marn*, J. Varlec†, T. Zagar, Cosylab, Ljubljana, Slovenia

Abstract

Area Detector is widely used in accelerator and observatory control systems to build data processing pipelines for images and waveforms, but adding new processing stages normally requires specialized C++ development. This slows prototyping and limits access to modern analysis tools that are predominantly available in Python. We present a new Area Detector plugin that executes user-defined Python algorithms directly on acquired images and their metadata. The plugin integrates advanced scientific libraries and machine learning toolkits, supports GPU acceleration, and enables parallel execution through multi-stage pipelines or Python multiprocessing. Algorithms can be updated without recompiling the IOC, allowing rapid iteration during operations. An automatically generated GUI exposes configurable parameters to operators. The approach is being evaluated on ESA's NEOSTED telescope system, where Python-based routines are used for telescope autofocusing.

INTRODUCTION

Area Detector (AD) [1] is the Experimental Physics and Industrial Control System (EPICS) framework for building image and waveform processing pipelines. At particle accelerators it underpins accelerator diagnostics and beamline instrumentation; at observatories it serves the camera control system. A pipeline is composed as a graph of plugins, each implemented as a C++ class derived from a common base class. Adding a new processing stage therefore requires both C++ and EPICS expertise. This barrier slows prototyping and puts the modern Python scientific computing and machine-learning ecosystem out of reach.

We present `NDPluginPython`, a generic AD plugin that executes user-defined Python algorithms directly on acquired images and their attached metadata. The plugin is being developed in the Telescope On-Site Image Processing (TOSIP) project for the European Space Agency (ESA), with the NEOSTED (Near-Earth Object Survey Telescope Deployment) autofocus use case [2] as its first concrete deployment. While accelerator imaging diagnostics are not the project's first target, the design is fully generic and applies wherever AD is already used.

AREA DETECTOR BACKGROUND

Images travel through an AD pipeline as `NDArray` objects [1]. The `NDArray` is a self-describing container holding pixel data, dimensions, data type, identifier, timestamp, and an extensible list of named attributes (`NDAttribute`). Attributes carry arbitrary named metadata associated with the array. Pipeline elements come in two kinds: drivers, which produce `NDArrays` from a source such as a sensor or a file, and plugins, which consume them through an asynchronous input queue and may produce further `NDArrays`. Both expose parameter records and base operator screens, and connect to one another through named ports that may be reconfigured at run time.

PYTHON PLUGIN ARCHITECTURE

Process Isolation

In current versions of Python's global interpreter lock prevents threads in a single interpreter from running pure Python in parallel [3], which conflicts with the throughput goals of an AD pipeline. Each `NDPluginPython` instance therefore runs the user algorithm in a dedicated child process. The Input/Output Controller (IOC) binary itself is not linked to Python; the child is a separate executable that embeds an interpreter (Fig. 1).

This separation provides two levels of parallelism without changes to the user algorithm. Instantiating the plugin multiple times runs independent interpreters across cores, while the algorithm itself is free to use the standard Python multiprocessing libraries. Graphics processing unit (GPU) acceleration follows naturally: because the child process binds to a user-chosen Python environment, libraries such as CuPy [4], PyTorch [5], or TensorFlow [6] work as they would in a stand-alone script, with no GPU support code required in the IOC. The same isolation contains memory growth and crashes in user code, leaving the IOC unaffected.

Inter-Process Communication

Image data is produced in the parent and consumed in the child. Copying it is costly — the sensors targeted by TOSIP produce frames on the order of 32 MB — so image data and metadata are copied only once, into shared memory, and the algorithm sees the image as a read-only NumPy [7] view backed by that memory. A message-based protocol synchronises parent and child, carrying short events such as image-ready and processing-complete; payloads are written to and read from named locations in the shared segment. The segment is grown on demand to accommodate variable image sizes. A future version will eliminate even this single copy by allocating `NDArrays` directly in the shared memory segment.

Process Lifetime

The plugin spawns and supervises its child. Shutdown follows a graceful-then-forced sequence. Should the parent die unexpectedly, the orphaned child is terminated automatically,

* matic.marn@cosylab.com
† jure.varlec@cosylab.com

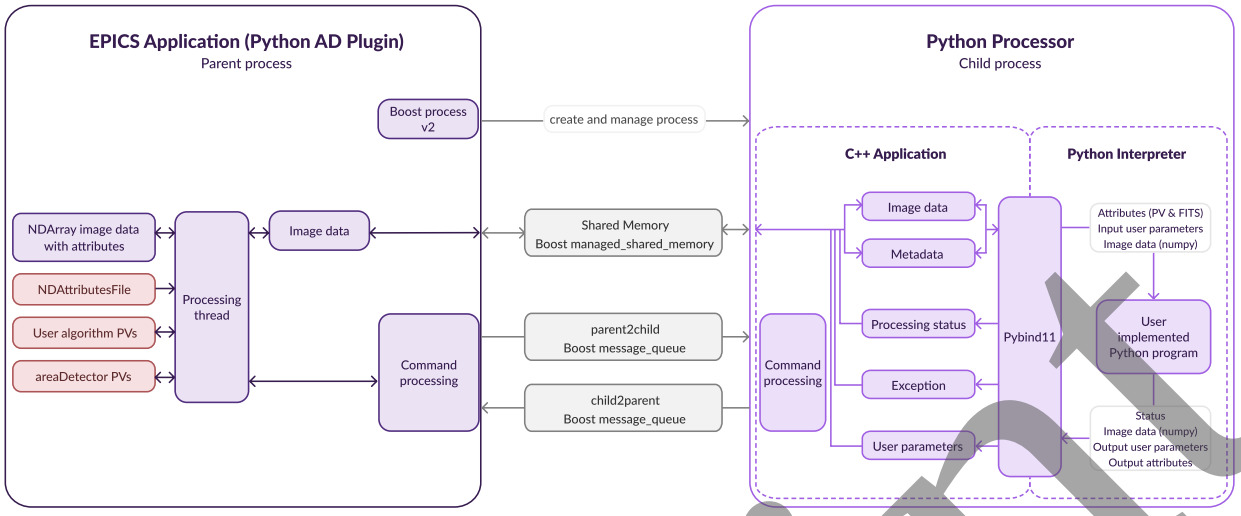

Figure 1: Architecture of `NDPluginPython`.

preventing leaked Python interpreters from accumulating across IOC restarts.

ALGORITHM INTERFACE

Python Algorithm Class

The user supplies a Python class that inherits from a TOSIP-provided base class and implements a single method, `process(image, params, attributes, metadata)`. Because the algorithm is a class rather than a plain function, it can be stateful — accumulating statistics, caching trained models, or reusing GPU buffers across frames. The class also declares the user parameters that the algorithm consumes as inputs and the parameters it exposes as outputs, together with their types.

At call time, `image` is a NumPy view of the shared-memory buffer; `attributes` carries the `NDAttribute` values attached to the array; `params` carries the current operator-set parameter values; and `metadata` describes pixel layout. The return value carries a status code, an optional message, the processed image, updated parameters, and any modified attributes.

Auto-Generated Records and GUI

On start-up, the plugin inspects the algorithm class, creates EPICS records matching each declared parameter, and generates a Phoebus [8] operator screen for the plugin instance. Algorithm developers therefore obtain a working interface for any algorithm without hand-written display files, and algorithm authors can iterate on the parameter set with the graphical user interface (GUI) following automatically.

Deploying and Updating Python Algorithms

User code lives in a Python virtual environment (based on pip [9], conda [10] or anything else) configured independently from the IOC. Algorithm modules can be replaced without rebuilding the IOC; an IOC restart is required only when the declared parameter list changes, since the EPICS record set must follow. The IOC may continue to embed its own Python interpreter for unrelated purposes, for example a PyDevice instance running with the system Python, without affecting the plugin's choice of environment.

APPLICATION: NEOSTED AUTOFOCUS

The first deployment target is NEOSTED, an instance of ESA's Flyeye asteroid-surveillance telescope series [2]. The telescope is a reflector type whose primary mirror focuses light onto a secondary that splits the field across 16 cameras, yielding a very wide field of view. Existing static autofocus procedures are time-consuming and currently performed only at the start of an observing session, even though ambient temperature and atmospheric seeing can shift the optimum during the night.

TOSIP equips NEOSTED with dynamic focus monitoring. The existing Python focus-estimation algorithm, which depends on common scientific libraries, is ported to TOSIP without modification. In the existing NEOSTED setup the cameras upload their Flexible Image Transport System (FITS) [11] files directly to the ESA File Transfer Protocol (FTP) archive. To avoid any change on the camera side, a new FTP gateway is inserted in front of the archive: cameras upload to it instead, and it forwards each file unchanged to the ESA archive while also making the file available to a companion AD driver, `ADFits`, which reads the FITS data and exposes its header keywords as `NDAttributes` for the Python algorithm to consume (Fig. 2). The same driver supports a batch mode that replays archived images without a live camera or EPICS network, which is useful for offline algorithm development.

The operator GUI displays per-camera and field-wide figure of merit alongside ambient temperature and atmospheric seeing, with configurable alarm thresholds, so that operators

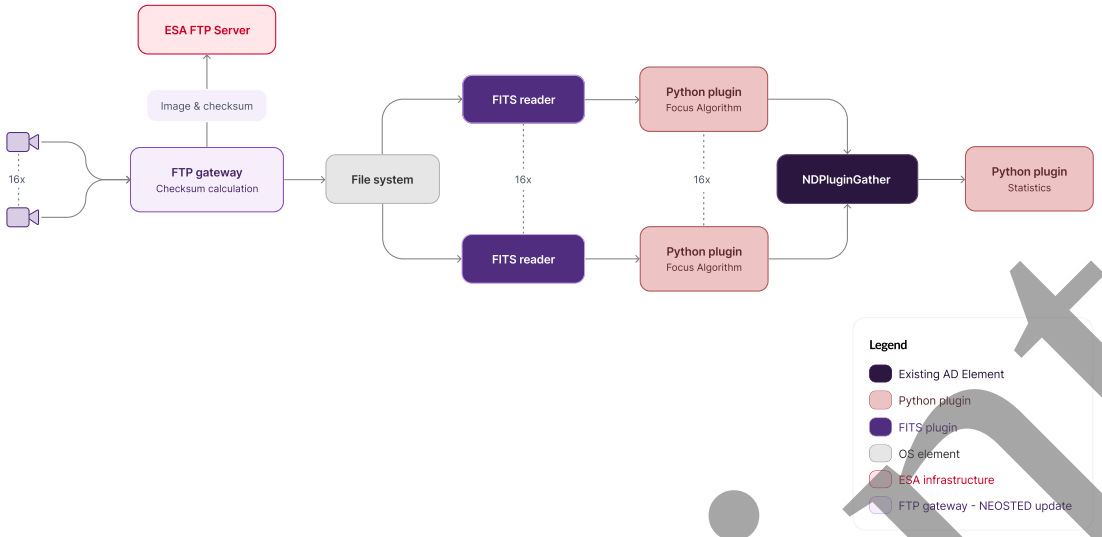

Figure 2: TOSIP pipeline at NEOSTED.

can decide when a static autofocus rerun is justified rather than running it on a fixed schedule.

The same algorithm is planned for the next-generation telescope in the series, where cameras are integrated through Area Detector. This forward compatibility motivated the choice of a generic AD plugin over a one-off Python service: the same component will serve both deployments unchanged. The architecture also supports upgrading to more sophisticated algorithms, such as real-time cloud detection.

CONCLUSION

`NDPluginPython` lifts the C++ barrier for adding processing stages to Area Detector pipelines, opening AD to the broader Python scientific computing and machine-learning ecosystem. Process isolation buys ecosystem freedom — including straightforward GPU acceleration — while keeping the IOC robust against errors in user code. Auto-generated parameter records and engineering screens, together with the ability to swap algorithms without rebuilding the IOC, fit the realities of operations. Validation is under way on NEOSTED, and the same design applies directly to other AD pipelines, including accelerator imaging diagnostics.

REFERENCES

[1] M. Rivers, AreaDetector: EPICS software for area detectors, `https://areadetector.github.io/areaDetector/`.

[2] L. Cibin *et al.*, "FlyEye ground-based telescope: unveiling new frontiers in astronomical science", in *Proc. SPIE*, vol. 13096, 2024, p. 13096A7. doi:10.1117/12.3020200

[3] S. Gross, PEP 703 — Making the Global Interpreter Lock optional in CPython, Python Enhancement Proposal, 2023. `https://peps.python.org/pep-0703/`

[4] R. Okuta, Y. Unno, D. Nishino, S. Hido, and C. Loomis, "CuPy: A NumPy-compatible library for NVIDIA GPU calculations", in *Proc. Workshop on Machine Learning Systems (LearningSys) at NeurIPS*, Long Beach, CA, USA, Dec. 2017. `http://learningsys.org/nips17/assets/papers/paper_16.pdf`

[5] A. Paszke *et al.*, "PyTorch: An imperative style, high-performance deep learning library", in *Adv. Neural Inf. Process. Syst. 32 (NeurIPS)*, Vancouver, Canada, Dec. 2019, pp. 8024–8035.

[6] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning", in *Proc. 12th USENIX Symp. Operating Systems Design and Implementation (OSDI)*, Savannah, GA, USA, Nov. 2016, pp. 265–283.

[7] C. R. Harris *et al.*, "Array programming with NumPy", *Nature*, vol. 585, pp. 357–362, 2020. doi:10.1038/s41586-020-2649-2

[8] Phoebus/Control System Studio `https://control-system-studio.readthedocs.io/`

[9] Python Packaging Authority, pip — The Python package installer, `https://pip.pypa.io/`.

[10] Anaconda Inc., conda documentation, `https://docs.conda.io/`.

[11] W. D. Pence, L. Chiappetti, C. G. Page, R. A. Shaw, and E. Stobie, "Definition of the Flexible Image Transport System (FITS), version 3.0", *Astron. Astrophys.*, vol. 524, p. A42, 2010. doi:10.1051/0004-6361/201015362