https://doi.org/10.1038/s41592-022-01655-4

PyImageJ: A library for integrating ImageJ and Python

New advancements in biological image processing, such as object segmentation, tracking[1] and machine-learning frameworks, have enabled researchers to extract more information and ask additional questions of their image data. Increasingly, these innovations are written in the Python programming language, making use of its extensive software library (for example, NumPy[2] and SciPy[3]) and its accessibility to researchers at various programming proficiency levels. As the Python software library has grown over the years to address new image-processing needs, so too has ImageJ – a Java-based open-source software package and platform widely used for scientific image analysis. ImageJ allows researchers to perform a variety of image-processing and analysis tasks such as edge detection, tiled image stitching, object and cell lineage tracking; morphological operations such as skeletonization; and various data projections. All of these operations can be combined to construct complex workflows in the form of macros and scripts. Additionally, the functionality of ImageJ has been extended through the use of plugins – new features written in Java and accessible directly from ImageJ, capable of bringing cutting edge technologies to the ImageJ platform. ImageJ supports an active community of software developers who produce and maintain these three extension types, which in recent years include deep-learning capabilities[4].

Unfortunately, Java-based and Python-based programs do not work together or share data seamlessly. Without the ability to easily exchange data between Python and ImageJ, features must be re-implemented in each respective environment or targeted wrappers built; this is not scalable. Communities across both languages require a bridge enabling seamless feature integration without duplicated effort.

To address this need, we present here PyImageJ, a Python-based package built on ImageJ2 (ref. [5]) that provides fundamental interoperability between Python and ImageJ-based software including the original ImageJ, ImageJ2 and the Fiji distribution of ImageJ[6]. With PyImageJ, we aim to support both software developers wanting to combine ImageJ and its plugin library with Python-based routines, and bench scientists wanting to do the same within their analysis workflows. PyImageJ is cross-platform, running on Linux, macOS and Windows operating systems, and can be installed from PyPI and conda-forge. PyImageJ enables two-way communication between ImageJ and Python by initializing Java as a subprocess of Python, such that any Java-based functionality can be used from Python programs, and new Python-based routines can be written to augment Java programs. The other paradigm, initializing Python as a subprocess of Java, is also useful in some scenarios and is currently under development. The architecture of ImageJ2 consists of libraries built on two key layers: SciJava (https://scijava.org/), which offers foundational infrastructure and is not image specific; and ImgLib2 (ref. [7]), which provides the core image data model (Fig. 1). Accordingly, PyImageJ is built on two foundational Python packages – scyjava (https://github.com/scijava/scyjava) and imglyb (https://github.com/imglib/imglyb) – which act as Python-based integration layers for the Java-based SciJava and ImgLib2 packages, respectively.

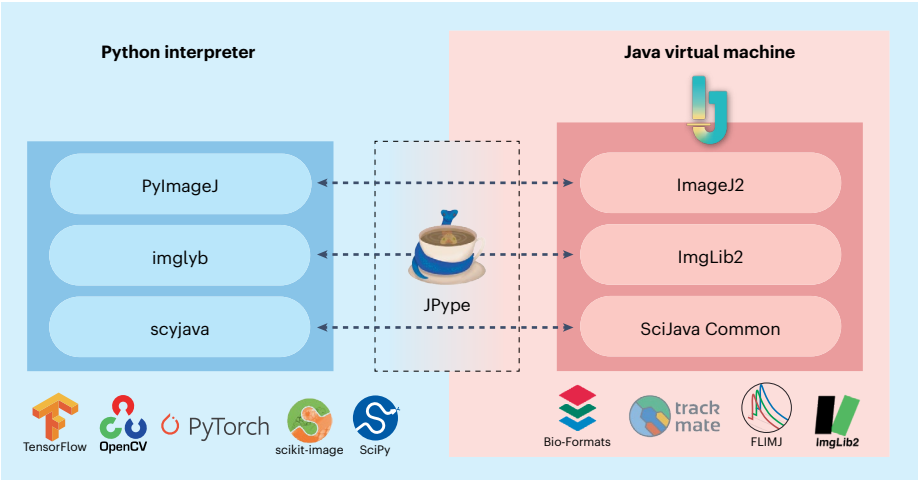

Fig. 1 | The software architecture of PyImageJ. Blue shows the Python environment and example Python applications. Red shows the ImageJ2 software stack with example plugins running in a special Python-integrated Java virtual machine (JVM). In the Python environment, PyImageJ uses JPype (from the scyjava layer) to create the Python-integrated JVM that will run the ImageJ2 software stack. In the Java environment, this encapsulated JVM incorporates all the user-requested Java libraries, including ImageJ, ImageJ2 and additional plugins – for example, from Fiji and/or other ImageJ update sites. The top Python layer, PyImageJ, provides access to the ImageJ2 gateway and Python convenience functions. The Python-side imglyb interfaces with the Java-side ImgLib2. Finally, the Python scyjava layer provides the foundational components such as JVM configuration and type conversions.

The first layer, scyjava, utilizes the jgo project (https://github.com/scijava/jgo) to retrieve the ImageJ2 Java libraries, and JPype (https://jpype.readthedocs.io/en/latest/) to create a special Python-integrated Java environment that includes the libraries. This design enables scyjava to transparently download and cache Java libraries packaged from remote online repositories, start the Java environment as a subprocess with those libraries included, wrap Java classes as dynamically generated Python classes with all the same functions, and convert common data structures such as lists, sets and dictionaries or maps between Java and Python. Notably, the scyjava package is potentially useful for any in-process integration of Java-based libraries

into Python programs and can be used independently of PyImageJ.

Exchanging image data between Python and ImageJ is accomplished through the imglyb layer, which provides zero-copy access to NumPy arrays and metadata-rich xarray data through shared memory. Using shared memory to store image data not only reduces memory use and processing time, but also enables the user to see the results of ImageJ processing immediately on Python-based images. ImageJ images that have been converted into an appropriate Python type (that is, NumPy or xarray) can be accessed by Python-based image-processing tools such as napari[8], a fast and interactive multi-dimensional image viewer, or CellProfiler[9], a workflow tool for reproducibly scaling analyses to large batches of data. Improved performance has already been shown with the RunImageJScript CellProfiler plugin, which, for example, enables a user to apply models from Trainable Weka Segmentation[10] (an ImageJ plugin) as one step in their feature classification workflow. PyImageJ offers the user interactive access to the full ImageJ2 Application Programming Interface (API), including all of ImageJ2's functionality and plugins, as well as the original ImageJ API, accessible via the backwards compatibility legacy layer of ImageJ2. PyImageJ also supports a headless mode without graphical user interface (GUI) elements, enabling workflows on systems with no computer monitor (for example, a remote server) — all ImageJ2 commands are available in headless mode, although functions of the original ImageJ are limited by its underlying dependence on GUI elements.

In summary, PyImageJ gives users access to the best of both Python and ImageJ, by fundamentally integrating the two software ecosystems. PyImageJ supports robust data interoperability between both Python and ImageJ, enabling users to create workflows that incorporate both Python and ImageJ elements.

Data availability

All data used for PyImageJ usecases are available at https://github.com/imagej/pyimagej/tree/master/doc/sample-data.

Code availability

The source code, documentation, tutorials and use cases for PyImageJ, which is made available under the open-source Apache software license, can be found online at https://github.com/imagej/pyimagej.

Curtis T. Rueden[1], Mark C. Hiner[1], Edward L. Evans III[1,2], Michael A. Pinkert[1,2,3], Alice M. Lucas[4], Anne E. Carpenter[4], Beth A. Cimini[4] and Kevin W. Eliceiri[1,2,3,5] ✉

[1]Center for Quantitative Cell Imaging, University of Wisconsin-Madison, Madison, WI, USA. [2]Morgridge Institute for Research, Madison, WI, USA. [3]Department of Medical Physics, University of Wisconsin-Madison, Madison, WI, USA. [4]Imaging Platform, Broad Institute of Harvard and MIT, Cambridge, MA, USA. [5]Department of Biomedical Engineering, University of Wisconsin-Madison, Madison, WI, USA.

✉e-mail: eliceiri@wisc.edu

Published online: 17 October 2022

References

1. Tinevez, J.-Y. et al. *Methods* **115**, 80–90 (2017).
2. Harris, C. R. et al. *Nature* **585**, 357–362 (2020).
3. Virtanen, P. et al. *Nat. Methods* **17**, 261–272 (2020).
4. Gómez-de-Mariscal, E. et al. *Nat. Methods* **18**, 1192–1195 (2021).
5. Rueden, C. T. et al. *BMC Bioinformatics* **18**, 529 (2017).
6. Schindelin, J. et al. *Nat. Methods* **9**, 676–682 (2012).
7. Pietzsch, T., Preibisch, S., Tomančák, P. & Saalfeld, S. *Bioinformatics* **28**, 3009–3011 (2012).
8. Sofroniew, N. et al. *napari/napari: 0.4.15rc1* (Zenodo, 2022); https://doi.org/10.5281/zenodo.6325333
9. Stirling, D. R. et al. *BMC Bioinformatics* **22**, 433 (2021).
10. Arganda-Carreras, I. et al. *Bioinformatics* **33**, 2424–2426 (2017).

Acknowledgements

This package was only made possible through the work of jgo and SciJava plugin frameworks. We are grateful for the contributions of P. Hanslovsky, the architect of imglyb and co-author of jgo. The authors also thank several individuals for various contributions and suggestions including E. T. A Dobson, J. Eglinger, S. Griffin, R. Haase, Y. Liu, H. Mary, and L. Yang. We also thank the PyImageJ user community for their great input and feedback. This work has been supported by the National Institutes of Health (P41GM135019 to A.E.C., B.A.C. and K.W.E.; T32CA009206 to M.A.P.; T32GM008349 to M.A.P.); Chan Zuckerberg Initiative funding to B.A.C., C.T.R. and K.W.E.; National Science Foundation (1429045 to K.W.E.); and additional internal funding from the Laboratory for Optical and Computational Instrumentation and the Morgridge Institute for Research.

Author contributions

Code concept and design was done by C.T.R., M.H. and K.W.E. PyImageJ coding development and implementation was done by C.T.R., M.C.H. and E.L.E.; case work by C.T.R., M.C.H., E.L.E., M.A.P., A.M.L., B.A.C. and K.W.E.; manuscript organizing and writing by C.T.R., M.C.H., E.L.E.,M.A.P., A.M.L., A.E.C., B.A.C. and K.W.E.; and funding and project administration by A.E.C., B.A.C. and K.W.E.

Competing interests

The authors declare no competing interests.

Additional information

Peer review information *Nature Methods* thanks Guillaume Jacquemet, Juan Nunez-Iglesias and Daniel Sage for their contribution to the peer review of this work.