3094
doi:10.1017/S1431927622011539
Microsc. Microanal. 28 (Suppl 1), 2022

Infrastructure for Analysis of Large Microscopy and Microanalysis Data Sets

Jingrui Wei[1], Carter Francis[1], Dane Morgan[1], KJ Schmidt[2], Aristana Scourtas[2], Ian Foster[2], Ben Blaiszik[2] and Paul M. Voyles[1*]

[1.] Department of Materials Science and Engineering, University of Wisconsin-Madison, Madison WI, United States.
[2.] University of Chicago, Globus / Argonne National Laboratory – Data Science and Learning Division
* Corresponding author: paul.voyles@wisc.edu

Microscopy and microanalysis research increasingly depends on sophisticated computational analysis to derive meaningful signals from large, complex, multidimensional, and often noisy data sets. Analysis methods span a range from physically-motivated approaches with no adjustable parameters, through unsupervised machine learning (ML) methods which require careful hyperparameter optimization, to supervised ML methods which require labeled training data.

We have employed two different infrastructure approaches to disseminating advanced analysis in microscopy. One approach is to support the open source analysis platform hyperspy [1] and the associated 4D STEM analysis package pxyem [2] to improve their ability to handle large datasets. With the advent of large format and ultrafast direct electron detectors for 4D STEM, datasets that are 100s of GB to TBs in size are becoming commonplace. These datasets cannot be held entirely in memory on most computers, so a "lazy" approach, in which subsets of the data in chunks are loaded into memory, operated on, then unloaded, is required. Hyperspy implemented lazy data processing with the widely-used dask library. We have updated the hyperspy implementation to operate efficiently on 4D STEM data by improving how functions are applied to chunks of data, dividing data into more efficient chunks, and implementing support for zarr, a package for fast reading and writing of data using efficient compression and multiprocessing. The resulting system implements the powerful analytics and development tools provided by dask, scales seamless from desktop computing to high-performance computing environments, and supports shared memory multiprocessing on data chunks. It supports all of the analysis capabilities of hyperspy and pyxem, including ML analyses like component analysis and physics-based analyses like orientation imaging for crystals or fluctuation electron microscopy for glasses.

Our other infrastructure approach leverages the Foundry software [3], as summarized in Figure 1. Foundry is software built on services to simplify dissemination and reuse of ML models and their associated training and test data. Foundry unifies two existing services: the Materials Data Facility, which offers large scale storage and permanent accessibility to data, and DLHub, which publishes and executes ML models. Foundry associates the data with the model under a single digital object identifier (DOI) and provides a unified Python interface for working with both data and models.

Foundry stores and serves ML models from containers, which removes the need to configure a computing environment to run a model from just the software distributed through (for example) Github. Models can be executed using Foundry-provided computing, on cloud computing resources including Google Colab or NSF Jetstream, or using local computing resources with a local install of the Foundry execution client. Foundry provides interfaces and containers for models built using the common ML libraries Scikit-Learn, Tensorflow, Keras, and PyTorch. It can also host models that execute arbitrary

Python code, which accommodates custom pre- and post-processing of data and libraries without specialized support. At this point, publishing models requires involvement of Foundry staff or collaborators, but direct user upload will be implemented in the future. Models are reviewed for appropriateness for the Foundry before being published.

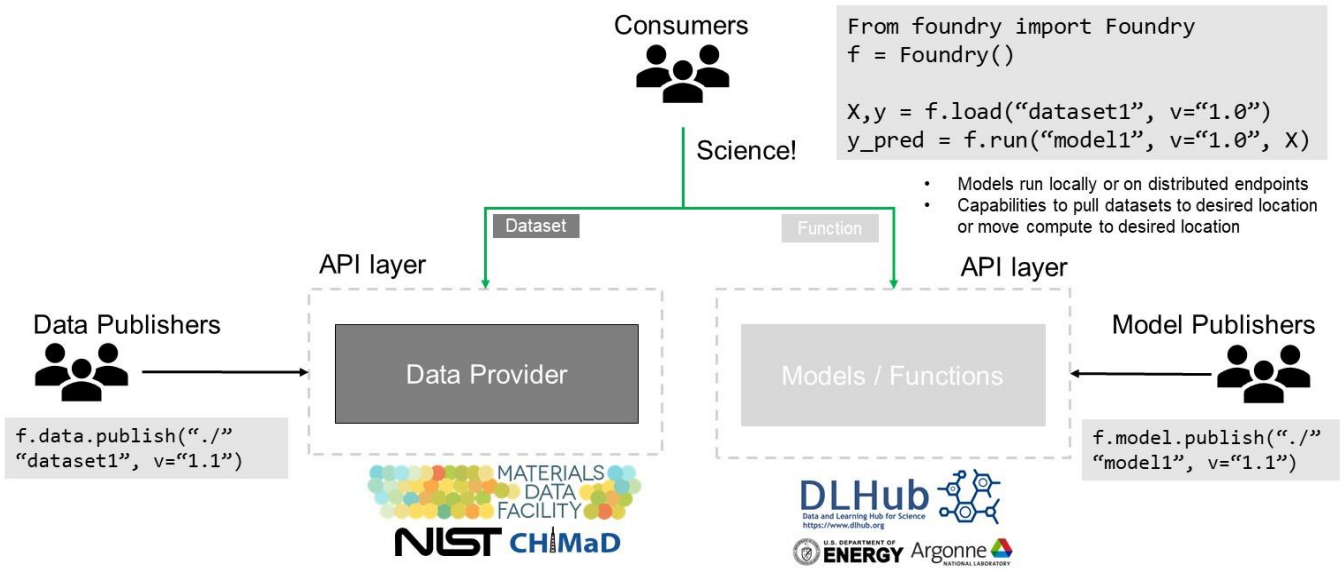

Figure 1: Foundry combines published data and models through a single Python interface to enable simplified access to training datasets and models for both execution of existing models or publication of updated or completely new models.

Foundry datasets must conform to defined shapes and contain specified metadata which are designed to ease access and reuse for training ML models, so they are not domain-science specific. Shapes and metadata are currently available for tabular data, JSON-based, and hierarchical data (*e.g.* HDF5). HDF5 can be used for data sets consisting of images or spectra. In addition to basic labels, the Foundry metadata captures splits of the data, for example into subsets for training and validation.

We imagine three general use cases for Foundry. The first use case is reuse of containerized models. The user installs Foundry in their Python environment, then calls existing published models with a single line of code with nothing else to install. Such models might identify all the atoms in the high-resolution image, denoise a noisy spectrum image, or locate all the diffraction disks in a 4D STEM dataset. Additional models might take those outputs and use them to quantify defect densities from atom positions or determine crystal orientations from disk positions. Foundry-hosted models from different researchers and non-Foundry software can be flexibly combined to create complex, powerful analysis workflows.

The second use case is dissemination of ML models. Disseminating models and data through Foundry follows FAIR principles [4], making them findable, accessible, interoperable, and reusable. FAIR is the emerging standard for open science, and FAIR ML models are more likely to find use and impact outside the group that developed them.

The third use case is retraining and evolving ML models. By permanently linking data and models, Foundry makes it possible for users to train a different type of model on the same training data or to test

https://doi.org/10.1017/S1431927622011539 Published online by Cambridge University Press

a new model on the same validation data. This type of head-to-head performance comparison offers opportunities for rapid advancement in the state-of-the-art of ML models for microscopy. The ultimate use of Foundry is to enable users to augment training data with new data or by combining data sets, then use the larger data to retrain existing models or train new models. Foundry offers tools for data set and model versioning and citing to ensure proper credit for all researcher's work [5].

References:

[1] hyperspy, DOI: 10.5281/zenodo.5608741
[2] pxyem, DOI: 10.5281/zenodo.2649351
[3] Foundry: Data, Models, Science, https://ai-materials-and-chemistry.gitbook.io/foundry/
[4] MD Wilkinson et al., Scientific Data **3**, 160018 (2016). DOI: 10.1038/sdata.2016.18
[5] Development of pyxem at UW-Madison is supported by the Wisconsin MRSEC (DMR-1720415). Development of Foundry is support by the National Science Foundation (OAC-1931298).