Next steps: AI/ML Pipeline

To make these image datasets and metadata available for ingestion for an AL/ML pipeline, two things need to be taken into consideration: preparation and storage the images and metadata for ingestion and designing the system to carry out the process. Here are the steps I would take into consideration:

Preparing and storing the image datasets

For the images and metadata, we would need to apply feature engineering to ensure that the data is preprocessed and formatted in a way that the AI/ML software can understand and use effectively. To support efficient ingestion of data, standardizing the file formats of the image datasets so they are all the same file type is crucial. The best file format would be Zarr because it is good for organizing large data and is versatile – it integrates well with many Python libraries including Zarr, CloudVolume, Dask and Xarray and code languages like JavaScript and C++. This is ideal for any data analysis tasks in the future. Since we are working with large images, storing these datasets in Zarr format is also ideal because they can be compressed and stored in chunks. Another benefit of Zarr format is that they are cloud storage friendly and suited for parallel I/O tasks.

Preparing and storing the metadata table

The metadata table is important for this pipeline because it contains the information needed to create the blocks. For the metadata table, making sure that the numerical values are normalized and that we are extracting all the relevant metadata the AI/ML pipeline would need. Important metadata for a software that requests a pixel block at various locations, would be parameters such as the image shape, size, resolution dtype and file location. It will be important to store the table in a way that is easily accessible for the software. The metadata table should be saved in a JSON format for easy access by the pipeline and easy readability for users. JSON is also version-friendly and can be easily updated using GitHub. Just like Zarr, JSON is easily adapted into coding languages like JavaScript and Rush.

API to handle block sizes

The next thing to take into consideration is the pipeline itself. Having an API that oversees the block sections would be a good thing to implement. This API could take in dataset_id, image shape, and the block region (eg: 128x128x128) and use that information to know where to retrieve the pixel block. For this, you could also use a Pytorch Dataset or write a function to create the block function. Lastly, if the goal is to feed this data into an ML model as input, it would be important to normalize, smooth, resize and get rid of artifacts of the raw files is always a good first step in working with image data. For outputs, any predictions made can be saved as .zarr files for further data analysis or stored using CloudVolume to Neuroglancer for web visualization. If the output is an embedding or feature vector, these can be stored in a database.

Additional optional feature: caching service

Since we are dealing with large datasets, having some sort of caching service would be helpful for any repeated tasks for faster retrieval. For example, if the same block is used consistently, store it in a cache so for faster retrieval. A cache service would also be a great place to store any model outputs.

In summary, the steps needed to make the datasets and metadata available for input into an AI/ML pipeline are to first standardize the file format of all the image datasets, so they are the same and to preprocess the raw files if needed and storing the datasets either locally or on the cloud. The next step is to prepare the metadata table by normalizing the numerical metadata across all datasets and making sure the table contains metadata that would help the software find the right locations to create the blocks in. Just like the image datasets, it is important to store the metadata table: a JSON file would work well for version control and fast access. For the system itself, create a block access function or API that ingests the metadata info to create pixel block chunks in various locations on the image datasets. A good way to do this would be to either write a function or use a Pytorch's Dataset class. Including a caching service to store frequently used blocks would help improve processing time. A pipeline set up in this way would allow for successful integration with model training pipelines and could be easily adaptable to new datasets.