The scale and complexity of biological data have increased substantially in recent years. Advances in single-cell technologies and spatially resolved assays such as matrix-assisted laser desorption/ionization (MALDI) imaging have enabled the collection of datasets with tens of thousands of cells or pixels and thousands of measured features per sample. For example, MALDI datasets may contain over 98,000 pixels and nearly 3,000 spectral peaks, resulting in matrices with millions of values. As the dimensionality of these datasets grows, efficient methods for reducing dimensionality are essential to group samples, such as pixels, cells, or bins, and to perform comparative biological analyses.

Principal component analysis (PCA) is a widely used technique for dimensionality reduction in biological research. In the R programming environment, which remains a primary tool for many researchers, PCA is commonly performed using functions such as prcomp (which computes full singular value decomposition, or SVD) and irlba (which uses a truncated SVD algorithm). However, prcomp becomes computationally impractical for large datasets due to its memory and time requirements, and irlba, while more efficient, still exhibits performance limitations when estimating a large number of components.

To address these challenges, we developed **FastPCA**, an R package that wraps high-performance Python-based SVD functionality using PyTorch. FastPCA enables researchers to perform scalable and accelerated dimensionality reduction directly within R. The package supports both randomized and exact SVD methods and can utilize either CPU or GPU resources. Benchmarks using a MALDI dataset with 98,647 pixels and 2,925 features show that FastPCA achieves significantly faster execution times compared to irlba and the PCAone package. For example, FastPCA’s randomized GPU implementation completed the decomposition in under one second (elapsed time: 0.939 s), while irlba required over 145 seconds under similar conditions. These results are based on direct timing measurements using R’s system.time() function.

FastPCA also produces eigenvalues and principal components that are consistent with those from other established methods. When comparing the top 100 components, FastPCA’s randomized and exact methods closely match the outputs from irlba and PCAone’s 'alg1' implementation. Deviations between methods begin to appear beyond the 50th component, particularly with PCAone’s 'alg2', which diverges more noticeably from the others.

By integrating optimized Python matrix operations into R workflows, FastPCA provides a practical and reproducible solution for researchers working with large-scale biological data. Installation is available via GitHub (ACSoupir/FastPCA), and setup can be completed using either a preconfigured conda environment or the built-in setup\_py\_env() function. This approach allows users to retain their existing R-based analysis pipelines while benefiting from the computational efficiency of modern Python libraries.