Received: 22 December 2023 | Revised: 17 June 2024 | Accepted: 23 August 2024

DOI: 10.1002/cyto.a.24896

ORIGINAL ARTICLE

ISAC CYTOMETRY Journal of Quantitative Cell Science PART A

SCIP: A scalable, reproducible, and open-source pipeline for morphological profiling image cytometry and microscopy data

Maxim Lippeveld[1,2] | Daniel Peralta[3] | Andrew Filby[4] | Yvan Saeys[1,2]

[1]Data Mining and Modelling for Biomedicine, VIB Center for Inflammation Research, Ghent, Belgium

[2]Department of Applied Mathematics, Computer Science and Statistics, Ghent University, Ghent, Belgium

[3]IDLab, Department of Information Technology, Ghent University – imec, Ghent, Belgium

[4]Biosciences Institute and Innovation Methodology and Application Research Theme, Newcastle University, Newcastle upon Tyne, UK

Correspondence
Daniel Peralta, IDLab, Department of Information Technology, Ghent University—imec, Ghent, Belgium.
Email: daniel.peralta@ugent.be

Funding information
Vlaamse regering; Fonds Wetenschappelijk Onderzoek

Abstract

Imaging flow cytometry (IFC) provides single-cell imaging data at a high acquisition rate. It is increasingly used in image-based profiling experiments consisting of hundreds of thousands of multi-channel images of cells. Currently available software solutions for processing microscopy data can provide good results in downstream analysis, but are limited in efficiency and scalability, and often ill-adapted to IFC data. In this work, we propose Scalable Cytometry Image Processing (SCIP), a Python software that efficiently processes images from IFC *and* standard microscopy datasets. We also propose a file format for efficiently storing IFC data. We showcase our contributions on two large-scale microscopy and one IFC datasets, all of which are publicly available. Our results show that SCIP can extract the same kind of information as other tools, in a much shorter time and in a more scalable manner.

KEYWORDS
data analysis, distributed computing, feature extraction, imaging flow cytometry, machine learning

1 | INTRODUCTION

Imaging flow cytometry (IFC) combines the high acquisition rate of conventional flow cytometry (CFC) with the single-cell imaging capabilities of microscopy [1]. It is used routinely in image-based profiling experiments consisting of hundreds of thousands of multi-channel images of cells [2, 3]. To extract all the information available in such rich data, efficient computational tools become necessary. Processing these images for downstream analysis involves denoising them, computing masks to delineate the cell shape, and extracting a broad set of features describing the cell. Although there currently exist several solutions for processing IFC data, these are still subject to some limitations, often because they are designed to handle microscopy data. For example, Cytek provides the IDEAS software to process data generated by the widely used ImageStream MK-II IFC platform. Even though IDEAS is a powerful software, it requires a paid license, is closed-source, runs only on Windows, and is tailored toward low-dimensional, manual gating analysis making it unsuitable for image-based profiling. CellProfiler [4] is an open-source package, which can be used to process IFC data. However, it is tailored toward processing standard microscopy datasets, which consist of fewer, large images with a fixed field of view (FOV) size, as opposed to the large volume of small images with a varying FOV size found in IFC datasets. To deal with this, images have to be resized and pooled into a fixed grid layout [5], which is rather inefficient and error-prone. Other open-source software for processing microscopy data, such as ImageJ [6, 7], QuPath [8], MCMICRO [9], or Orbit [10] have similar issues with IFC data.

To allow for timely and efficient processing of large-scale datasets, software must be scalable. This means including support for parallel processing on a single machine and in a distributed computing environment, such as a high performance computing cluster [11]. In CellProfiler, MCMICRO, QuPath and ImageJ, parallelization is achieved by concurrently processing images or image tiles. CellProfiler and MCMICRO also support scaling out horizontally by batching the

dataset and processing the batches in parallel on multiple compute nodes [4]. Distribution by batching, however, has limited flexibility. For instance, processing tasks cannot be dynamically assigned to compute nodes based on available or required resources, such as GPUs. There is also only limited support for exploiting data locality, and there are no mechanisms for fault tolerance. Orbit does provide these functionalities through a tile-based MapReduce framework backed by Apache Spark [12], but it is implemented in Java—making it more difficult to integrate with state-of-the-art Python image-processing tools—and as noted earlier, it is not compatible with IFC data. Even though distributed computing provides a very fine-grained parallelization, there is always a certain degree of overhead that should be taken into account. For instance, Amdahl's law [13] establishes the maximum speedup achievable given the fraction of processing that is not parallelizable. Moreover, additional overhead appears when parallelizing a task, due to the necessity of synchronizing different tasks or aggregate their results [12]. Therefore, a careful design is needed when addressing complex problems with parallel architectures.

Besides software-related issues, the standardized storage of IFC data is also an issue. Most device manufacturers provide their data in a proprietary file format only accessible through proprietary software. Some open-source file formats exist in the microscopy community, such as HDF5, OME-TIFF, N5 and `zarr`, which are focused on providing efficient access to microscopy datasets. The latter two are especially suitable for parallelization, as they allow for easy access to individual chunks of data from local storage or over a network. However, these formats are built for standard microscopy data and cannot deal with the varying FOV sizes in IFC.

Our contribution to solving these issues is twofold. First, we propose Scalable Cytometry Image Processing (SCIP), an open-source Python software that handles IFC *and* standard microscopy data in a scalable manner with the Dask framework for graph-based distributed computing, which enables fine-grained control over pipeline execution. By designing SCIP to handle both imaging modalities, only one software needs to be installed and maintained to process them, reducing the complexity of the processing pipeline. Second, we propose a storage format based on `zarr` optimized for IFC data, which allows to efficiently store variably-sized images.

We tested our proposal on three very different use cases (2 of them generated in-house and published, and 1 well-known benchmark), involving IFC and microscopy data, as well as different biological problems. SCIP showed similar performance to other tools in terms of classification accuracy and clustering analysis, while being implemented in a more scalable and flexible manner, enabling the analysis of larger datasets with more advanced processing pipelines.

2 | METHODS

2.1 | Image profiling pipeline

Our software implements a full profiling pipeline, whose steps are formalized in this section, discussing what function is executed and how it is concretely implemented.

First, SCIP accepts input data in the form of one or more directories of Tag Image File Format (TIFF) images, either with one file per channel, or with multiframe TIFFs where each frame represents one channel, Carl Zeiss Image (CZI) files, or our custom `zarr` files, as detailed in Section 2.1.1. The input to be processed is a matrix of pixels D arranged in four or five dimensions which encode signal intensity measured by an imaging system. All datasets will have at least four dimensions: X, Y, channel (C) and an index (I). In multi-focal datasets, a fifth Z dimension is also present, which indexes the different focal planes. We will refer to each position i on the index dimension of D as image $d_i \in \mathbb{R}^{c \times z \times x \times y}$.

Each image d_i also has a number of categorical or ordinal values associated with it, referred to as the meta data. This can include the plate and well in which the cell was located, some experiment identifier, or an object number.

2.1.1 | Storage

One of the bottlenecks of single-cell imaging data is the large amounts of images that are captured in each sample. Both imaging flow cytometers and microscopes can capture up to millions of individual cells in a typical experiment, but handling such a large amount of individual images can be very cumbersome. In a regular filesystem such as NTFS or ext-4, such large amounts of files lead to very long indexing times, complex master file tables and inode hierarchies, and reduced effective storage space due to fixed block sizes. This further causes a very significant overhead in I/O times, which becomes a critical bottleneck when processing the data.

A common solution to this problem is the use of multiframe TIFF files, which store multiple TIFF images into a single large file. However, this requires all single-cell images to have the same dimensions, which might not be desirable in IFC and segmented microscopy images. Small images must be enlarged with empty background pixels that cause a memory overhead, and large images might have to be cropped, removing potentially useful information.

To address these issues, we introduce a data format to efficiently work with IFC data. The format is based on `zarr`, which is optimized for storing chunked, N-dimensional arrays, such as images. We store the variably-sized images in the `zarr` format with the `VLenArray` codec provided in the `numcodec` package. This codec requires that images are stored as 1-dimensional arrays, so we also store each image's original shape for restoring it when loaded. By using our format, the large volumes of images of different sizes can be more easily and quickly accessed and shared, compared to proprietary file formats or a directory of individual image files, without resizing or wasting space. We provide code for converting IFC datasets to our format on Github.*

*https://github.com/saeyslab/ideas-to-zarr.

2.1.2 | Projection in multi-focal datasets

When the input is a multi-focal dataset, the images have to be projected onto one plane as shown in Equation 1.

$$\begin{aligned} \texttt{Project}: \mathbb{R}^{c\times z\times x\times y} &\rightarrow \mathbb{R}^{c\times x\times y} \\ d_i &\mapsto p(d_i), \end{aligned} \tag{1}$$

where p is a projection function over dimension z. SCIP provides implementations for *min*, *mean*, *max* and *median* projection, which respectively compute these statistics over the focal planes on each (c,x,y) location in d_i.

2.1.3 | Illumination correction

Next, the illumination correction step transforms images to mitigate the influence of non-homogeneous illumination. Different techniques exist; here, we focus on retrospective illumination correction, which combines data from all images acquired in a batch to produce an illumination correction function (ICF). The function is then applied to each image to make a corrected version (Equation 2) [14].

$$\begin{aligned} \texttt{Correct}: \mathbb{R}^{c\times x\times y} &\rightarrow \mathbb{R}^{c\times x\times y} \\ d_i &\mapsto \mathrm{ICF}(d_i) \end{aligned} \tag{2}$$

We implement a distributed version of the retrospective illumination correction proposed by Jones et al. [15] The ICF is computed by averaging all images per channel per experimental batch and smoothing the results. Say dataset D consists of m experimental batches D_j. Then, the ICF is computed for each batch D_j as follows

$$\overline{d}_j = \texttt{median\_smooth}\left(\frac{\sum_{d_i \in D_j} d_i}{|D_j|}\right), \tag{3}$$

where `median_smooth` is a median filter which smooths its input by replacing each pixel with the median value of the pixels in its neighborhood. The ICF is applied by dividing each image by its corresponding batch correction image $\overline{d}_j$.

2.1.4 | Segmentation

In the case that each $d_i \in D$ contains many cells, such as in high-throughput microscopy experiments, instance segmentation is required to identify the sub-regions in each channel of d_i that encode information about single cells. The segmentation operation is defined in Equation 4.

$$\begin{aligned} \texttt{Segment}: \mathbb{R}^{c\times x\times y} &\rightarrow \mathbb{R}^{n\times c\times x'\times y'} \\ d_i &\mapsto \{d_j | d_j \subset d_i, \forall_{j,m} d_j \cap d_m = \emptyset\} \end{aligned} \tag{4}$$

The Cellpose algorithm [16] is available for FOV segmentation and can be used with or without GPU acceleration. It is a state-of-the-art deep learning (DL)-based algorithm, applicable to a wide variety of imaging modalities without any further training or customization. Benchmarks [16–18] showed that Cellpose substantially outperformed previous state-of-the-art methods, such as Stardist and Mask R-CNN. It achieved this by a model design for cell segmentation based on the principles of watershed segmentation, as well as a training dataset containing a wide variety of images with manually segmented objects.

2.1.5 | Masking

In IFC-like data and in microscopy data after the segmentation step, each d_i is expected to contain only one cell. The masking operation (Equation 5) identifies the region of interest containing the cell's signal in each channel of the image.

$$\begin{aligned} \texttt{Mask}: \mathbb{R}^{c\times x\times y} &\rightarrow \{0,1\}^{c\times x\times y} \\ d_i &\mapsto m_i, \end{aligned} \tag{5}$$

where each (x,y) location is set to either 0 (background) or 1 (foreground) based on some decision function, typically a threshold based on the pixel value or the result of a segmentation algorithm. SCIP has implementations for threshold and spot masks, which mask the entire cell and the brightest spots in it, respectively.

2.1.6 | Profiling

Finally, the images are profiled. This operation (Equations 6 and 7, depending on whether masking is applied or not) maps the pixel data of d_i onto a feature vector that describes characteristics of the object captured in d_i.

$$\begin{aligned} \texttt{Profile}: \mathbb{R}^{c\times x\times y} &\rightarrow \mathbb{R}^{c\times k} \\ d_i &\mapsto f(d_i) \end{aligned} \tag{6}$$

$$\begin{aligned} \texttt{Profile}: \mathbb{R}^{c\times x\times y} \times \{0,1\}^{c\times x\times y} &\rightarrow \mathbb{R}^{c\times k} \\ (d_i, m_i) &\mapsto f(d_i \wedge m_i) \end{aligned} \tag{7}$$

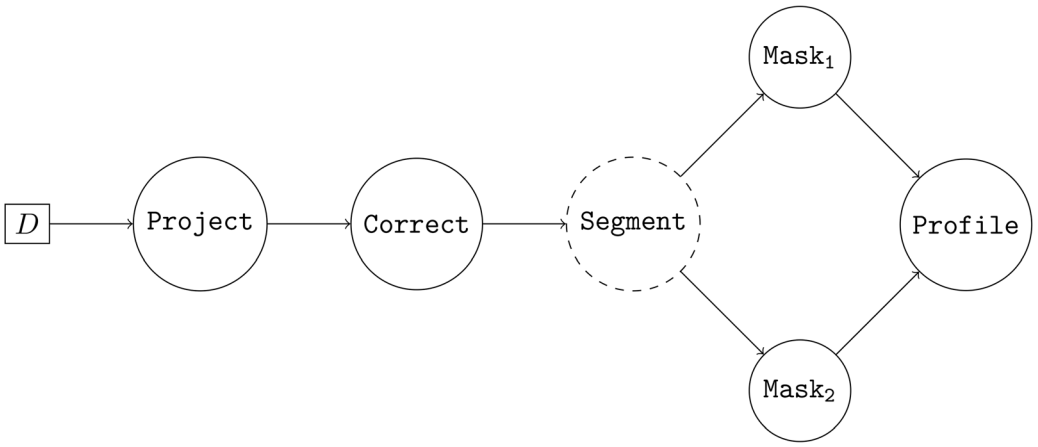
Three types of features are derived from the pixel data: intensity, shape, and texture. Table S1 gives an example of some of them, which are described in more detail in Section S1. All features are computed for every channel in the image using the channel-specific mask, as well as a union of all masks. The intensity and texture features are also computed on the edge pixels of the mask. To obtain the edge pixels, an eroded version of the mask is subtracted from the original one, leaving only the edge of the mask.

2.2 | Graph-based parallel execution enables fine-grained control over distributed pipeline execution

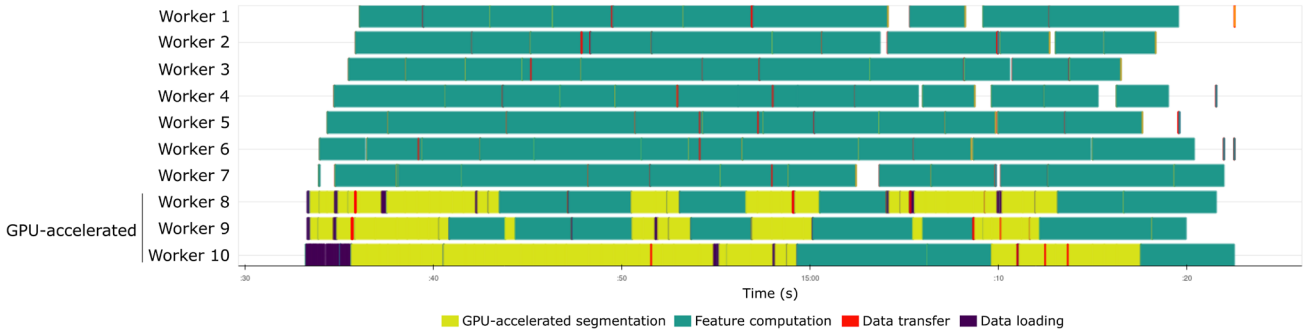
A naive parallel implementation of an image profiling pipeline would process all images d_i in dataset D using a number of isolated processes that each execute the full function f defined in Equation 8. This is referred to as the split-apply-combine approach.

$$f(d_i) = \texttt{Profile}(\texttt{Mask}(\texttt{Segment}(\texttt{Correct}(\texttt{Project}(d_i))))). \quad (8)$$

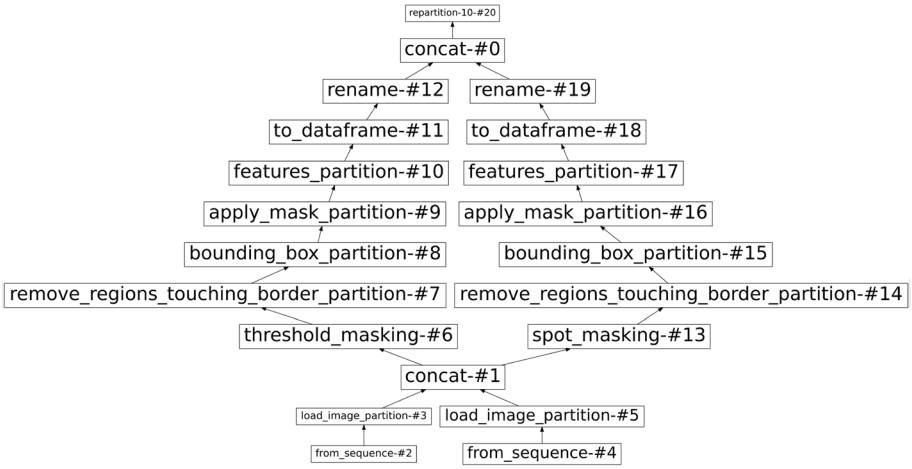
However, this approach is not flexible due to the lack of communication and cooperation between the processes. Each process has to execute the full pipeline, and it is therefore not possible, for example, to have processes that execute a specific step on a compute node with specialized hardware, such as a GPU. It is also not possible to reuse results for processes that are executing steps that depend on the same input, for example, when deriving profiles for multiple different masks of each image. Since the processes are unaware of each



(A) Basic task graph encoding the operations of Equation 8 to process dataset D. Nodes represent operations, edges represent dependencies between operations. Nodes with dashed borders have specific resource requirements.



(B) Dask task stream of a microscopy dataset being processed with SCIP showing an efficient use of heterogeneous computational resources by distributing steps of the pipeline across the cluster. In this pipeline, segmentation is done with CellPose on 3 GPU-accelerated nodes.



(C) Dask task graph visualizing the operations to be executed on the input images. This graph loads in images from two files, computes two masks from them (a threshold mask and a spot mask), and derives profiles from the masked images.

FIGURE 1 Overview of the proposed workflow on Dask. [Color figure can be viewed at wileyonlinelibrary.com]

other's state, there is no way to implement efficient fault tolerance or load balancing.

A way to overcome these limitations is by using graph-based parallel execution with task scheduling. In this approach, a directed acyclic graph (DAG), called the task graph, is constructed that encodes all tasks to be executed on the dataset as nodes and dependencies between tasks as edges. A task graph executing the same functions as Equation 8 is shown in Figure 1A. The task graph is analyzed by a task scheduler instructing the worker processes to execute individual tasks, leveraging parallelization and reusing results where possible. The scheduler can also keep track of what each worker is executing to resubmit tasks in the event of a worker's fault and rebalance the workload if necessary.

In a graph-based parallel system with task scheduling, many processors are connected to each other over a network to execute a program. To handle larger workloads, the system can be expanded by adding more processors; in theory, doubling the amount of available processors to the program should halve the execution time. However, in practice, the speedup is usually smaller because this expansion causes communication overhead (as more orchestration is required), and some parts of the computation are non-parallelizable, as described by Amdahl's law [13]. In Section 2.4, we present results on the scalability of SCIP.

A number of big data frameworks implement the graph-based parallel execution approach, including Apache Spark [12], Ray [19] and Dask. We chose to implement SCIP using the latter because it implements functionality and distributed data structures that are a natural fit to the pipeline introduced above.

2.3 | Dask-based implementation enables scalability to large datasets

Algorithm 1 shows how the pipeline is implemented using operations on distributed Dask Bag and DataFrame collections. We use the `map_partitions` operation, which applies a function to all elements in a bag, and the `foldby` operation, which performs an efficient grouped reduction on the elements in a bag. These operations are used to construct a task graph that applies the functions defined in Section 2.1 on the input images. The task graph is then sent to the

Algorithm 1 **Extract image profiles from image dataset *D* with SCIP**

SCIP(D, *segment*, *masks*)
Require: Distributed scheduler S
inputs :
- Directory containing image files d
- Segmentation flag *segment* $\in \{\text{TRUE}, \text{FALSE}\}$,
- Illumination correction flag *correct* $\in \{\text{TRUE}, \text{FALSE}\}$,
- Metadata key indicating group membership for illumination correction *group*,
- list of masking methods *masks*

outputs: Image profiles $P = \{f_i \in \mathbb{R}^k\}$ stored on disk

```
# Create directed acyclic graph (DAG) of tasks
B ← bag_from_directory(d);
D ← B.map_partitions(Load);
if z > 1 then
    D ← D.map_partitions(Project);
if correct then
    d̄_group ← D.foldby(group, ICF);
    D ← D.map_partitions(Correct, d̄_group);
if segment then
    with dask.annotate(GPU)
        D ← D.map_partitions(Segment);
C ← ∅;
foreach mask ∈ masks do
    D_m ← D.map_partitions(Mask);
    D_m ← D_m.map_partitions(Profile);
    D_m ← D_m.to_dataframe();
    C ← C ∧ D_m;
D ← dask.dataframe.concat(C);
D ← Export(D);
# Execute delayed operations on D on S
dask.compute(D);
```

Dask distributed scheduler that decides what operation each worker needs to execute in what order to compute the desired result as efficiently as possible.

First, paths pointing to input files are stored in a Dask Bag *B* of dictionaries holding the meta data about each image d_i. In the loading operation, pixel data is added to each dictionary in the Bag. By using a Bag of dictionaries, SCIP can handle images with varying *x* and *y* dimensions and keep the meta data linked to the pixel data.

Illumination correction is available in SCIP through a distributed implementation of the method introduced in Section 2.1.3. First, a `foldby` operation is used to compute a distributed sum of all images in a group and divide it by the number of images in it, obtaining one average image per group. Second, each averaged image is smoothed using a median filter. To reduce the memory footprint of the filter, the image can be downscaled.

Next, the instances are segmented using watershed or Cellpose. All channels are segmented separately; objects detected in different channels are assigned to the overlapping parent object detected in the user-defined parent channel. Typically, the parent channel will contain a cytoplasm stain that captures the entire cell. Finally, the segmentation step restructures *B* from a *per-file* to a *per-object* Bag, which contains one element per cell, such that the identified cells are each in their own dictionary.

Dask allows to annotate tasks with required resources, such as a GPU, which is specifically suited to GPU-accelerated segmentation methods. These annotations will cause the scheduler to only assign the task to a worker with the requested resource. Figure 1B shows the stream of tasks executed by the workers, three of which have a GPU resource. It illustrates how heterogeneous computational resources can be used efficiently by distributing steps of the pipeline across the cluster: while GPU-accelerated nodes are performing segmentation, other nodes are loading data from disk or computing feature profiles.

Multiple masks can be computed to produce specific views on the images, such as a threshold and a spot mask. Figure 1C shows how this multi-mask approach translates to the Dask task graph. Dask's smart task scheduler will cache as many of the loaded images in distributed memory as possible to avoid re-loading them from disk for each mask.

After segmentation and masking, the profiling operation is applied to *B*. This operation converts the Bag into a DataFrame containing a profile for each detected cell.

Finally, the profiles are exported to AnnData h5ad-files or Parquet-files. The former is provided for interoperability with the `ScanPy` [20] library, the latter is a format that can be opened in many different programming languages. It is also possible to export segmentation results for quality control or visualization. The segmentation labels are stored in a `numpy` file that can be processed further with Python scripting.

2.4 | SCIP scales well for increasing dataset sizes and can handle a large amount of workers

In this section, we probe the performance and scalability of SCIP in three experiments, corresponding to three parameters that determine the runtime and maximum memory usage of our approach: the dataset size. *d*, the number of workers *n* and the partition size *p*. We probe the scalability of SCIP by alternatively fixing *n* or *d* and varying the other two, using the values shown in Figure 2D.

The measurements were collected on a subset of the IFC white blood cell (WBC) dataset from Section 3.1. The dataset contains images with slightly varying dimensions of around 60×90 pixels. Measurements were made for the computation of the full feature set for 2 of the 12 channels per image. Scripts to reproduce these results can be found on Github.[†]

First, we find that SCIP scales well with the number of workers. Figure 2A shows that doubling the number of workers doubles the number of images processed per second. When the number of workers exceeds 8, the measured runtimes diverge from the ideal speedup for this particular dataset due to the incurred communication overhead between the workers. The partition size does not have a strong influence in this experiment.

Secondly, we find that SCIP also scales well with respect to dataset size for a fixed number of workers. As expected, Figure 2B shows that for lower dataset sizes, the distribution overhead is large, leading to a low number of images processed per second. For dataset sizes over 10,000, the overhead becomes negligible, and the processing speed reaches a plateau of 300–350 images per second. This experiment does show an influence of partition size: logically, for smaller datasets, a smaller partition size allows for more parallelization and shorter runtimes. For larger dataset sizes, a larger partition size is preferable, as this reduces communication overhead. As a rule of thumb, for datasets with less than 100,000 objects, the partition size should be between 200 and 400. When going up to 1,000,000 objects, the partition size should be at least 1000.

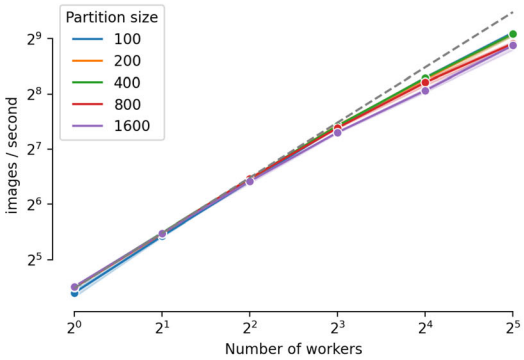
Finally, we find that SCIP exhibits good scaling behavior in terms of maximum memory usage. Figure 2C shows that memory usage scales linearly for small datasets and sublinearly when dataset size exceeds 10,000. This is likely due to Dask's smart task scheduling, which takes into account memory availability before scheduling new tasks, and which traverses the task graph depth-first allowing for quicker reduction of the images in memory to profiles, which take up less memory. From these experiments, we can conclude there is a trade-off between memory usage and speed when choosing partition size: maximum memory usage tends to be lower for smaller partition sizes, but processing speed is higher for larger partition sizes.

3 | USE CASES

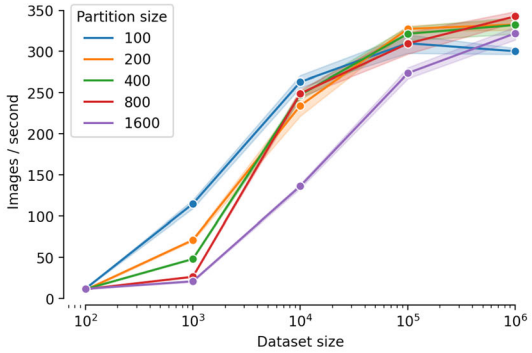
We demonstrate the use of SCIP on three datasets. All necessary code to reproduce our results is provided as a Snakemake [21] workflow on Github.[‡] Datasets for use cases 1 and 3 were generated at Newcastle University. All datasets are publicly available.

Two different hardware platforms were used to run these use cases. Use case 1 (Section 3.1) used the HPC infrastructure at Ghent University, specifically the *doduo* cluster, which contains 128 nodes,

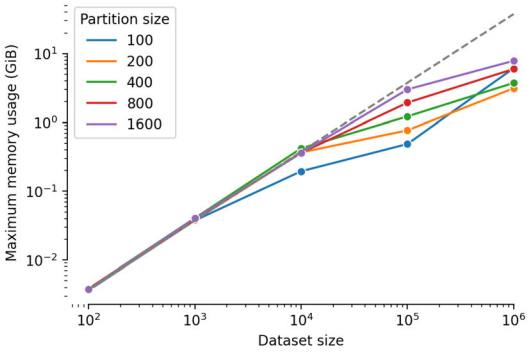
[†]https://github.com/ScalableCytometryImageProcessing/benchmark.
[‡]https://github.com/ScalableCytometryImageProcessing/scip-use-case-workflows.



(A) Number of images processed per second in function of number of workers. The dataset is limited at 100,000 images. The speedup diverges from the ideal case due to communication overhead as the number of workers increases.



(B) Number of images processed per second in function of dataset size. For small dataset sizes, the distribution overhead is large as reflected in the low number of images processed per second. Distribution overhead becomes small for larger dataset sizes.



(C) Maximum memory usage in GiB in function of dataset size. For small dataset sizes, maximum memory usage scales linearly, and as size increases maximum memory usage scales better then linear.

	Number of workers	Dataset size	Readout
(a)	$\{2^i \mid i \in \{1,2,3,4,5\}\}$	80,000	Runtime
(b)	16	$\{10^i \mid i \in \{2,3,4,5,6\}\}$	Runtime
(c)			Max. memory usage

(D) Values tested of number of workers and dataset size in experiments to probe the scalability of SCIP. All experiments were replicated for $p \in \{100 \cdot 2^i \mid i \in \{0,1,2,3,4\}\}$

FIGURE 2 Results of scaling experiments. The measurements were collected on a subset of the imaging flow cytometry white blood cell dataset from Section 3.1. The dataset contains images with slightly varying dimensions of around 60×90 pixels. [Color figure can be viewed at wileyonlinelibrary.com]

each of which hosts two 48-core AMD EPYC 7552 processors (Rome @ 2.2 GHz), 250 GiB RAM, and 180GB SSD local storage, interconnected by an HDR-100 InfiniBand newtork. All nodes use RHEL 8 as operating system. Use cases 2 and 3 (Sections 3.2 and 3.3) were run on a GPU cluster hosted by the Saeys lab research group, which hosts a 32-core Intel Xeon Silver 4110 processor, 5 NVIDIA GeForce RTX 3090 GPUs (of which one was used for each experiment), 345GiB RAM, and 128GB SSD local storage.

3.1 | WBC classification

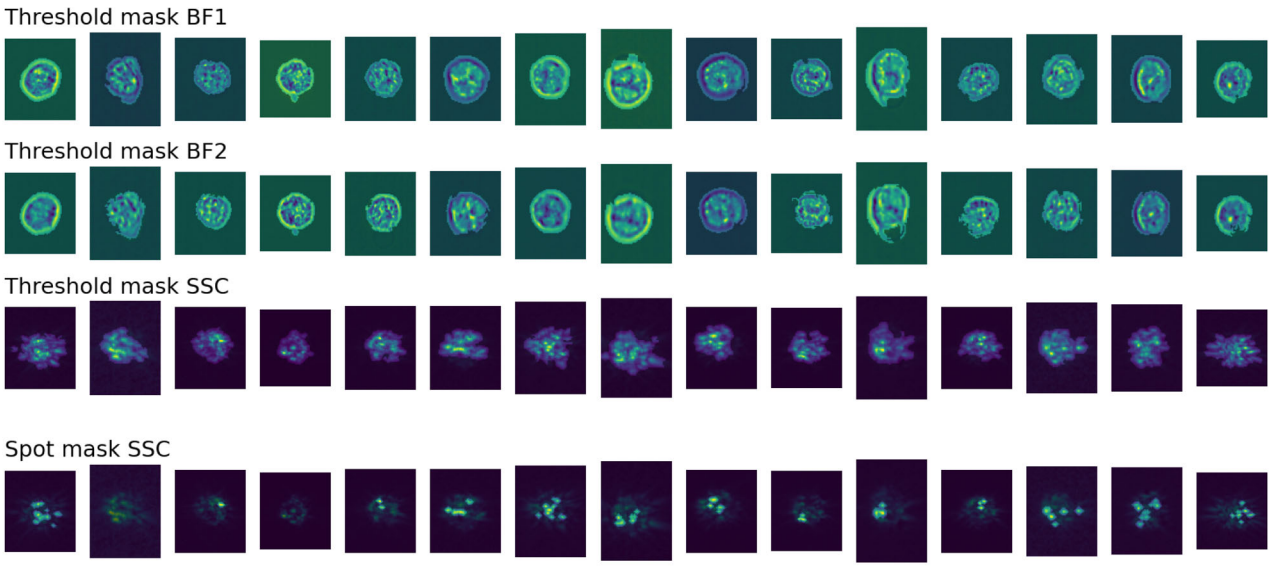
The dataset consists of 3 human whole blood samples, each stained with 9 fluorescent markers. Each acquired image has 12 channels: 9 fluorescent, 2 brightfield, and 1 darkfield channel. A ground truth label was assigned to all cells with manual gating on the fluorescent stains; the objective of this study is to perform cell type classification using only the 3 stain-free channels. We refer to [22] for details on the data acquisition and gating procedure. All data is accessible under study S-BIAD452 on the Bioimage Archive.[§]

To profile the dataset with SCIP, we exported 16-bit TIFF images from the IDEAS software and stored them in the `zarr` format [23]. Feature profiles were extracted for 247,993 12-channel images masked with both a spot and threshold mask (see Figure 3A). Profiling took 2.48 h using 16 workers; 4338 features were computed in total per image. After filtering out doublets and debris based on the derived profiles, 233,262 cells remained for classification.

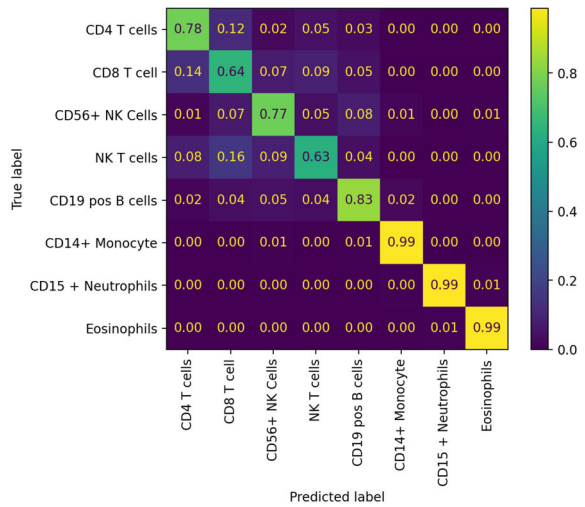
Classification results were obtained using an eXtreme Gradient Boosting (XGB) model. We used random successive halving hyper-parameter optimization [24] validated with nested fivefold stratified cross-validation to find optimal hyper-parameter values for the XGB model. Optimal configurations were selected based on balanced accuracy.

To counter class imbalance, the majority class (neutrophils) was first randomly under-sampled to the same level as the second most abundant class (monocytes), after which all minority classes were randomly over-sampled to the same level of the monocytes.

[§]https://www.ebi.ac.uk/biostudies/studies/S-BIAD452.



(A)



(B)

Phase	test	train
balanced accuracy	0.769 (0.004)	0.841 (0.012)
f1 macro	0.681 (0.005)	0.741 (0.012)
precision macro	0.636 (0.004)	0.690 (0.011)
recall macro	0.769 (0.004)	0.841 (0.012)

(C)

FIGURE 3 Overview and results of cross-validated stain-free leukocyte classification with an eXtreme Gradient Boosting classifier. (A) Example images of threshold and spot masks computed by SCIP for brightfield 1 (BF1), brightfield 2 (BF2), and darkfield (SSC) channels. Note that the images are captured by the device with different sizes. (B) Confusion matrix. (C) Classification metrics. [Color figure can be viewed at wileyonlinelibrary.com]

As we can see in the results (Figure 3), the classifier succeeded in distinguishing between the eight cell types using only the brightfield and darkfield channels, indicating that the relevant information encoded in those images was successfully and efficiently captured by SCIP. We obtained a balanced accuracy of 76.9%, which is similar to the 77.8% obtained in [22], where the IDEAS software was used for profiling.

3.2 | Predicting mechanism-of-action for small molecule treatments

The publicly available BBBC021 dataset [25] consists of high-throughput microscopy images of MCF-7 cells. Each image corresponds to a set of cells treated with one of 103 (38 compounds, 1–7 concentrations per compound). Each compound-concentration combination was assigned to 12 mechanism of actions (MOAs), such as protein synthesis. This use case involves predicting each MOA based on morphological information extracted from the microscopy images.

Several classification results on this prediction task have been published. Feature-based approaches follow similar steps [26]: (i) profile the cells using CellProfiler, (ii) summarize the profiles per treatment, and (iii) use a nearest-neighbor classifier to predict the MOA for a treatment. Other works present variants of this workflow, such as performing feature clustering and dimensionality reduction [27], or performing illumination correction prior to profiling [14]. The latter was implemented in SCIP (see Section 2.1.3).

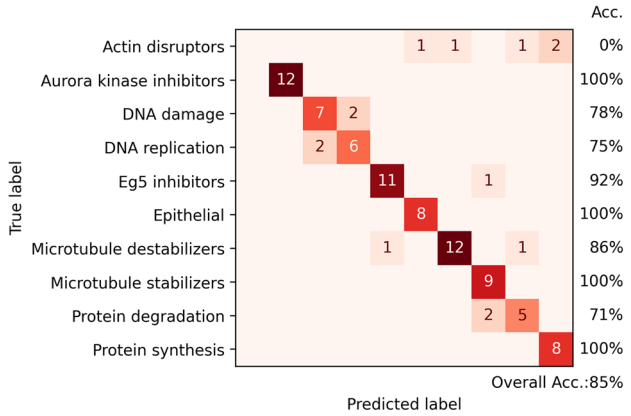


FIGURE 4 Confusion matrix for the mechanism-of-action prediction task on the BBBC021 benchmark dataset obtained with SCIP using illumination correction and factor analysis. [Color figure can be viewed at wileyonlinelibrary.com]

Summarizing cell profiles per treatment is either done using factor analysis (FA) or the means method, both according to [26]. In the former, an FA model is trained on a subset of the DMSO-treated cells, after which the model is used to transform the treated cells. This new representation is averaged to obtain the per-sample profile. Ljosa et al. achieved optimal classification performance using 50 factors, which we also used to showcase the performance of SCIP. The median of the three replicate samples is then taken to produce a treatment profile for each compound-concentration combination.

To obtain classification results, we profiled the dataset using SCIP with illumination correction [14], and GPU-accelerated Cellpose segmentation. It took 17.6 h to extract profiles for 2.18 million cells using 11 workers, of which 4 were GPU-accelerated. Cells (470,127) remained when keeping only cells from the subset of images annotated with an MOA. We achieved an accuracy of 85%, which was also obtained by [14]. Figure 4 shows the obtained confusion matrix.

Runtimes of CellProfiler and SCIP were compared on a subset of data. CellProfiler ran for 25 min and 47 s when the workload was processed in parallel over eight batches. SCIP processed the data in 25 min and 10 s with 8 workers, of which four were GPU-accelerated for segmentation.

3.3 | Unsupervised profiling of human white blood cells

Blood from a healthy donor was stained and imaged on a Zeiss Cell-discoverer 7 platform. The images contain seven channels: 4 fluorescence channels, a brightfield channel, an oblique channel, and a phase-gradient contrast channel. All data are accessible under study S-BIAD452 on the Bioimage Archive.[¶]

For acquisition, all cells were fixed and stained with CD45, Siglec 8, and CD15 fluorophore-conjugated antibody markers; cell nuclei were stained with DAPI. Images of cells were acquired in 5 wells, each well was imaged at 25 non-contiguous locations in 3 focal planes, each 2 µm apart in the *Z*-axis. Manual inspection flagged 12 out of 125 well images with uneven illumination that could not be corrected (see Figure 5B).

We clustered cells into four distinct phenotypes using Leiden clustering: granulocytes, eosinophils, lymphocytes, and monocytes. We also identified one unclassified cluster, which had a low response for all markers in the panel. We hypothesize this cluster is made up of cell debris and platelets. Figure 6D shows some example images of objects from the unclassified cluster. Finally, we embedded the features with Uniform Manifold Approximation and Projection (UMAP) to visualize the cell populations. Figure 5A shows an overview of the annotation.

Objects whose centers were closer than 30 pixels to the border of the image were discarded. Figure 5B shows an example of a segmented image. Multiplets were gated out using aspect ratio and eccentricity, as shown in Figure 5C. Objects where no DAPI signal was found were also discarded. Objects (34,838) remained after filtering.

Next, we aligned distributions of the fluorescence intensities across the image positions and replicated them to account for technical variation. From replicate 1, 32 outliers were removed based on CD45 marker intensity and another 9 outliers based on Siglec 8 marker intensity. After this, an arcsinh-transform and Z-score normalization were performed per image position per replicate to align the fluorescence signals and all other intensity-derived features such as signal upper quartile or mean. All other features were Z-score normalized. Figure 6A shows the aligned intensity distributions.

Using the mutual information (MI) score, 66 features were identified and removed that had a strong relation to the replicate number after normalization. MI was computed using 30-nearest neighbors with the estimation introduced by [28].

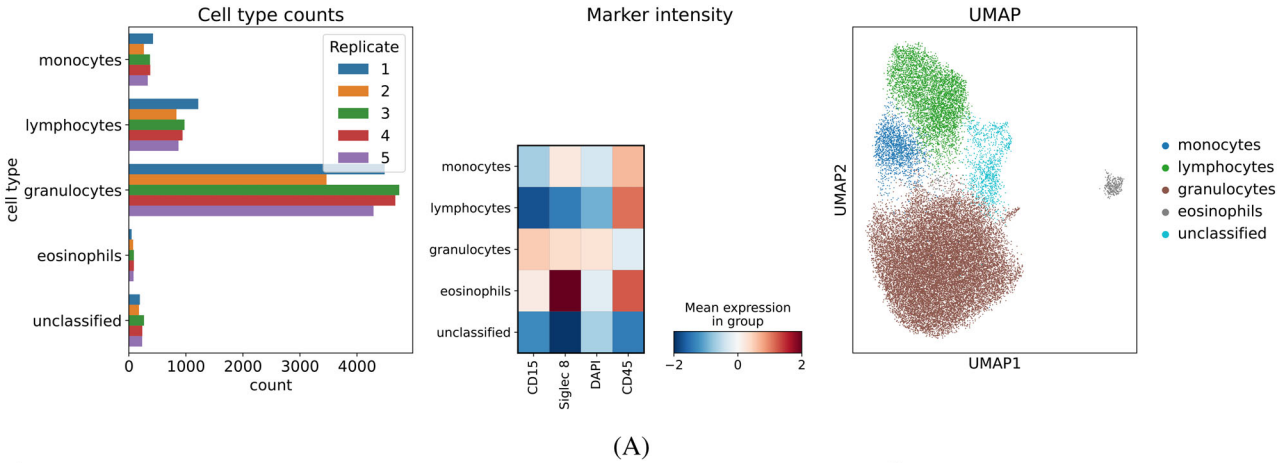
To do so, we first filtered out multiplets and debris using the eccentricity, major axis length, and minor axis length features. Features were then Z-score normalized and PCA-transformed. We constructed a 30-nearest neighbor graph based on the first 50 PCA components using UMAP distance for computing connectivity. We ran Leiden clustering with a resolution of 0.75 on this graph to discover 10 clusters. Scatter plots (see Figure 6B,C) of fluorescent marker intensity were used to annotate clusters with cell types.

This analysis shows the power of SCIP to extract meaningful features that can be used in an unsupervised analysis, especially when used in combination with a feature selection procedure that can identify the features that are relevant to the dataset being processed.

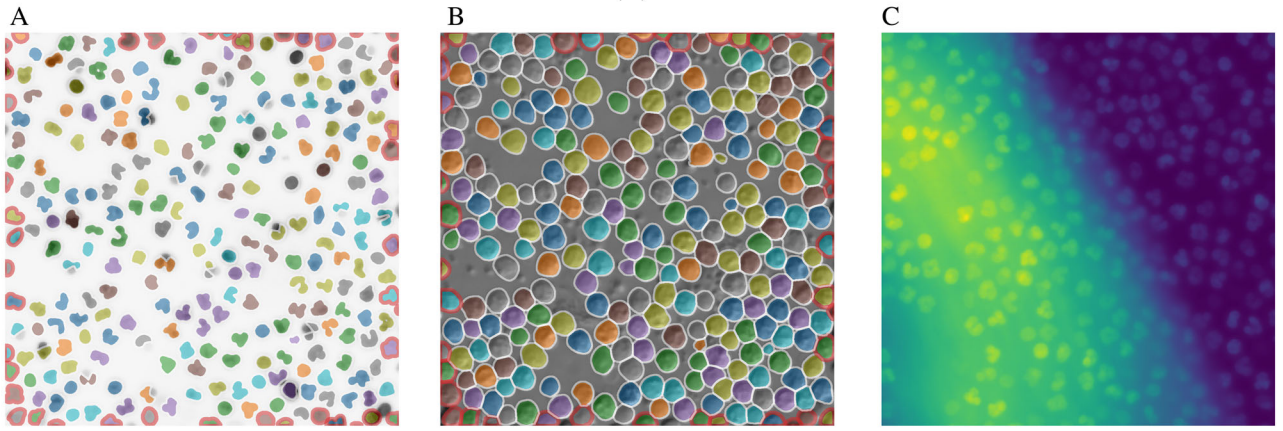
4 | DISCUSSION

In this work, we introduced SCIP, an open-source software tool for processing large-scale bioimaging datasets implemented using the Dask framework. We discussed SCIP's implementation and design details, highlighting the flexible data and task parallelism, which allow

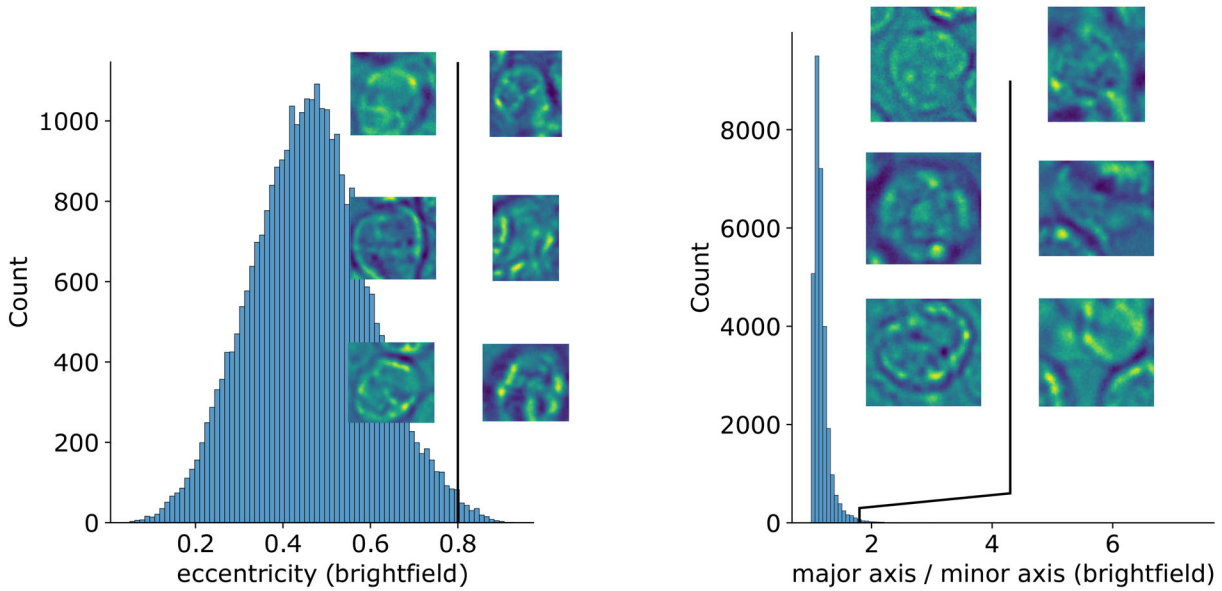
[¶]https://www.ebi.ac.uk/biostudies/studies/S-BIAD505.



(A)



(B)



(C)

FIGURE 5 Analysis of CD7 microscopy dataset. (A) Left Cell type counts per replicate show an equal distribution over the replicates, as expected. Middle Mean expression of marker per cell type. Right UMAP embedding shows all populations separated in two dimensions. Note the unclassified cluster, which has low expression for all markers and is likely made up of debris and platelets. (B) A, B Example segmentation of DAPI and oblique channel, respectively, with CellPose. Cells having a red border are discarded due to their proximity to the image border. C Example of image with uneven illumination, which is discarded from the dataset. (C) Multiplets were filtered out using eccentricity and aspect ratio (major axis length over minor axis length). For both figures, events with values to the right of the black line were discarded Some example images are shown as well. [Color figure can be viewed at wileyonlinelibrary.com]

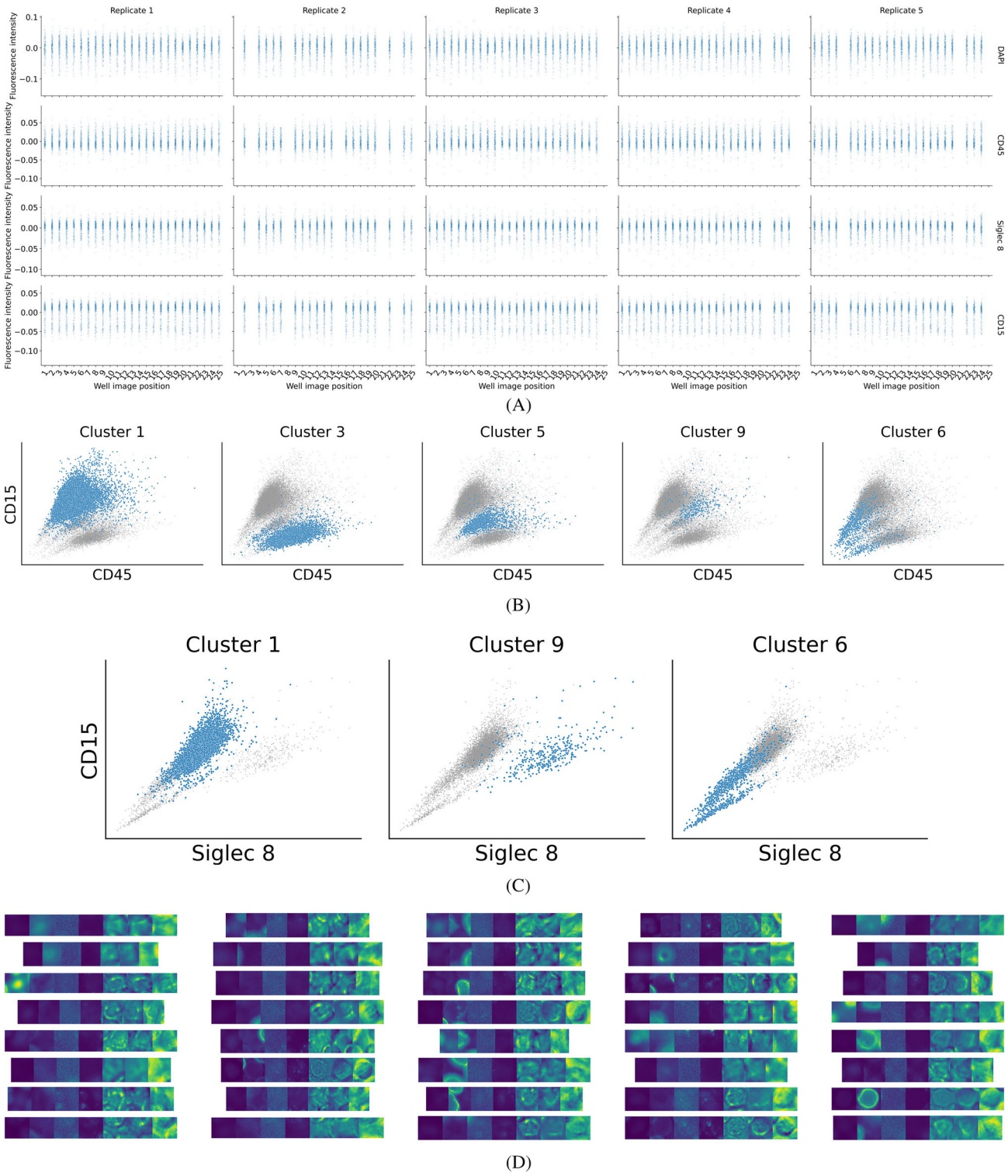


FIGURE 6 Distributions and clusters in CD7 microscopy dataset. (A) Strip plots of aligned intensity distributions after normalization and filtering of outlier events. (B) Scatterplots of CD45 versus CD15 intensity per cluster. (C) Scatterplots of Siglec 8 versus CD15 intensity per cluster. (D) Example images of the unclassified cluster. Channel names from left to right are DAPI, EGFP, RPe, APC, Bright, Oblique, PGC. [Color figure can be viewed at wileyonlinelibrary.com]

it to process both IFC and microscopy datasets using complex processing steps such as DL-based segmentation, while providing functionality such as load balancing, and fault tolerance.

We demonstrated in three use cases how SCIP is used to analyze various datasets. First, we reproduced earlier results obtained on the stain-free classification of eight WBC types. The models were able to

correctly predict cell types based on features computed by SCIP. Next, we performed unsupervised clustering to profile a healthy blood sample and were able to distinguish four major WBC types. Finally, we reproduced the results of MOA prediction using morphological features. Using our software, we were able to predict the 12 MOA classes accurately. In all cases, SCIP led to results comparable to those of previously published tools, confirming its utility for a variety of applications.

The three use cases made use of data collected with different devices, including one imaging flow cytometry dataset and two microscopy datasets, which demonstrates the versatility of SCIP to process images coming from different types of domains, manufacturers, scales, throughputs, and resolutions. We have shown how SCIP is not tied to a specific device; after converting manufacturer-specific formats to our custom-formatted `zarr` files, any kind of single-cell image data can be processed by SCIP.

With the use cases we have shown that SCIP provides a scalable, reproducible and versatile solution for working with large bioimaging datasets. We believe that the architecture of our software can enable researchers to address research questions previously limited by the capabilities of existing image-processing tools.

While a direct comparison of all available tools is beyond the scope of this work, we can contextualize SCIP within the broader ecosystem of image-processing tools. There are two widely used software packages that officially support IFC data: the closed-source IDEAS and the open-source CellProfiler. In use cases 1 and 2, we found that classification performance obtained with their features is similar to that of SCIP. In terms of scalability, IDEAS is limited as it can only be run on a single Windows workstation. CellProfiler does provide batch-based distribution, but this is less flexible compared to SCIP's graph-based distribution. This is demonstrated, for instance, in use case 2 and 3 where the data was processed with a mix of CPU-only workers and GPU-accelerated workers for segmentation. This efficient use of heterogeneous resources is not possible with CellProfiler, since the full pipeline is always ran on one machine.

For microscopy data, SCIP is a promising alternative to existing tools like CellProfiler, Orbit, MCMICRO, ImageJ, or QuPath. While QuPath and ImageJ are widely used, they are primarily GUI-based and designed for single workstations, limiting their scalability. Orbit and MCMICRO, on the other hand, are powerful tools primarily used for large whole-slide imaging datasets. Their functionality goes beyond the microscopy scope of SCIP. Therefore, our software is especially interesting for users who prefer the Python ecosystem, want to use Cellpose for segmentation, and like the functionality of the Dask framework.

4.1 | Future work

Currently, SCIP relies mostly on traditional image processing and manually engineered feature extraction, although segmentation is already done with the DL Cellpose model. While these steps are powerful, incorporating more DL-enabled methods represents a key area for expansion. Python-based deep learning frameworks (such as TensorFlow or PyTorch) can be integrated easily within our software's architecture—as shown already with segmentation—offering the potential to extract features with pretrained DL models, and to perform image denoising and illumination correction.

Currently, SCIP is used through a command-line interface, and while it can be used on a desktop computer, its full scalability potential relies on a certain degree of technical expertise in areas such as distributed computing. This focus on a programmatic interface can present a barrier for researchers who are less familiar with these environments. Developing a graphical user interface, perhaps in the form of a web application, would enhance usability, making SCIP's powerful capabilities accessible to a broader range of researchers.

Besides mitigating these limitations, we want to improve SCIP further by supporting more input formats, such as the open-source Open Microscopy Environment (OME)-TIFF[**] and OME-Next Generation File Format (NGFF) [29]. Due to the widespread adoption of these formats in the microscopy community, this would allow users to more easily convert their data to a format compatible with SCIP. Finally, we also want to explore the use of our software for spatial omics data, as the processing of these large and complex datasets can certainly benefit from SCIP's scalability.

AUTHOR CONTRIBUTIONS

Maxim Lippeveld: Investigation; methodology; validation; visualization; software; writing – original draft. **Daniel Peralta:** Conceptualization; methodology; supervision; writing – original draft; writing – review and editing; investigation. **Andrew Filby:** Writing – review and editing; conceptualization; methodology; investigation; funding acquisition; supervision; resources; data curation; validation. **Yvan Saeys:** Writing – review and editing; conceptualization; methodology; investigation; funding acquisition; supervision.

ACKNOWLEDGMENTS

The compute resources and services used in this work were provided by the VSC (Flemish Supercomputer Center), funded by the Research Foundation—Flanders (FWO) and the Flemish Government. M. Lippeveld is a Predoctoral Fellow of the Fund for Scientific Research FWO Flanders (1SB9421N). This research received funding from the Flemish Government under the "Onderzoeksprogramma Artificiële Intelligentie Vlaanderen." The authors want to thank Sander Thierens for his work in the initial stages of developing SCIP.

PEER REVIEW

The peer review history for this article is available at https://www.webofscience.com/api/gateway/wos/peer-review/10.1002/cyto.a.24896.

DATA AVAILABILITY STATEMENT

SCIP is available to install from Github (https://github.com/ScalableCytometryImageProcessing/scip) or PyPi (https://pypi.org/project/scip/). The documentation (https://scalable-cytometry-image-processing.readthedocs.io) contains usage information online.

**https://docs.openmicroscopy.org/ome-model/5.6.3/ome-tiff/specification.html.

ORCID

Maxim Lippeveld https://orcid.org/0000-0002-9527-7162

Andrew Filby https://orcid.org/0000-0001-9078-4360

Yvan Saeys https://orcid.org/0000-0002-0415-1506

REFERENCES

1. Rees P, Summers HD, Filby A, Carpenter AE, Doan M. Imaging flow cytometry. Nat Rev Methods Primers. 2022;2(1):1–13.
2. Mochalova EN, Kotov IA, Lifanov DA, Chakraborti S, Nikitin MP. Imaging flow cytometry data analysis using convolutional neural network for quantitative investigation of phagocytosis. Biotechnol Bioeng. 2022;119(2):626–35.
3. Luo S, Nguyen KT, Nguyen BTT, Feng S, Shi Y, Elsayed A, et al. Deep learning-enabled imaging flow cytometry for high-speed cryptosporidium and giardia detection. Cytometry A. 2021;99(11):1123–33.
4. Carpenter AE, Jones TR, Lamprecht MR, Clarke C, Kang I, Friman O, et al. CellProfiler: image analysis software for identifying and quantifying cell phenotypes. Genome Biol. 2006;7(10):R100.
5. Blasi T, Hennig H, Summers HD, Theis FJ, Cerveira J, Patterson JO, et al. Label-free cell cycle analysis for high-throughput imaging flow cytometry. Nat Commun. 2016;7:10256.
6. Rueden CT, Schindelin J, Hiner MC, DeZonia BE, Walter AE, Arena ET, et al. ImageJ2: ImageJ for the next generation of scientific image data. BMC Bioinformatics. 2017;18(1):529.
7. Schindelin J, Rueden CT, Hiner MC, Eliceiri KW. The ImageJ ecosystem: an open platform for biomedical image analysis. Mol Reprod Dev. 2015;82(7–8):518–29.
8. Bankhead P, Loughrey MB, Fernández JA, Dombrowski Y, McArt DG, Dunne PD, et al. QuPath: open source software for digital pathology image analysis. Sci Rep. 2017;7(1):16878.
9. Schapiro D, Sokolov A, Yapp C, Chen YA, Muhlich JL, Hess J, et al. MCMICRO: a scalable, modular image-processing pipeline for multiplexed tissue imaging. Nat Methods. 2022;19(3):311–5.
10. Stritt M, Stalder AK, Vezzali E. Orbit image analysis: an open-source whole slide image analysis tool. PLoS Comput Biol. 2020;16(2):e1007313.
11. Peralta D, Saeys Y. Distributed, numerically stable distance and covariance computation with MPI for extremely large datasets. 2019 IEEE International Congress on Big Data (Big-DataCongress). 2019, pp. 77–84.
12. Zaharia M, Xin RS, Wendell P, Das T, Armbrust M, Dave A, et al. Apache spark: a unified engine for big data processing. Commun ACM. 2016;59(11):56–65.
13. Amdahl GM. Validity of the single processor approach to achieving large scale computing capabilities. *Proceedings of the April 18–20, 1967, spring joint computer conference*. AFIPS'67 (Spring). New York, NY, USA: Association for Computing Machinery, April 1967: 483–485.
14. Singh S, Bray MA, Jones T, Carpenter A. Pipeline for illumination correction of images for high-throughput microscopy. J Microsc. 2014;256(3):231–6.
15. Jones TR, Carpenter AE, Sabatini DM, Golland P. Methods for high-content, high-throughput image-based cell screening. Proceedings of the Workshop on Microscopic Image Analysis with Applications in Biology. 2006, vol. 5, pp. 65–72.
16. Stringer C, Wang T, Michaelos M, Pachitariu M. Cellpose: a generalist algorithm for cellular segmentation. Nat Methods. 2021;18(1):100–6.
17. Waisman A, Norris A, Costa M, Kopinke D. Automatic and unbiased segmentation and quantification of myofibers in skeletal muscle. Sci Rep. 2021;11:11.
18. Han S, Phasouk K, Zhu J, Fong Y. Optimizing deep learning-based segmentation of densely packed cells using cell surface markers. Res Square. 2023;24:124.
19. Moritz P, Nishihara R, Wang S, Tumanov A, Liaw R, Liang E, et al. Ray: a distributed framework for emerging AI applications. arXiv. 2018; arXiv: 561-577.
20. Wolf FA, Angerer P, Theis FJ. SCANPY: large-scale single-cell gene expression data analysis. Genome Biol. 2018;19(1):15.
21. Mölder F, Jablonski KP, Letcher B, Hall MB, Tomkins-Tinch CH, Sochat V, et al. Sustainable data analysis with Snakemake. F1000Res. 2021;10:33.
22. Lippeveld M, Knill C, Ladlow E, Fuller A, Michaelis LJ, Saeys Y, et al. Classification of human white blood cells using machine learning for stain-free imaging flow cytometry. Cytometry A. 2020;97(3):308–19.
23. Miles A, Kirkham J, Durant M, Bourbeau J, Onalan T, Hamman J, et al. Zarr-developers/Zarr-python: V2.4.0. Zenodo. 2020.
24. Jamieson K, Talwalkar A. Non-stochastic best arm identification and hyperparameter optimization. *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*. PMLR, May 2016, pp. 240–248.
25. Caie PD, Walls RE, Ingleston-orme A, Daya S, Houslay T, Eagle R, et al. High-content phenotypic profiling of drug response signatures across distinct cancer cells. Mol Cancer Ther. 2010;9(6):1913–26.
26. Ljosa V, Caie PD, Ter Horst R, Sokolnicki KL, Jenkins EL, Daya S, et al. Comparison of methods for image-based profiling of cellular morphological responses to small-molecule treatment. SLAS Discov. 2013;18(10):1321–9.
27. Peralta D, Saeys Y. Robust unsupervised dimensionality reduction based on feature clustering for single-cell imaging data. Appl Soft Comput. 2020;93:106421.
28. Ross BC. Mutual information between discrete and continuous data sets. PLoS One. 2014;9(2):e87357.
29. Moore J. Ome/NGFF: next-generation file format (NGFF) specifications for storing bioimaging data in the cloud. Zenodo. 2020.

SUPPORTING INFORMATION

Additional supporting information can be found online in the Supporting Information section at the end of this article.

How to cite this article: Lippeveld M, Peralta D, Filby A, Saeys Y. SCIP: A scalable, reproducible, and open-source pipeline for morphological profiling image cytometry and microscopy data. Cytometry. 2024. https://doi.org/10.1002/cyto.a.24896