BMC Bioinformatics

RESEARCH

Open Access

CytoPipeline and CytoPipelineGUI: a Bioconductor R package suite for building and visualizing automated pre-processing pipelines for flow cytometry data

Philippe Hauchamps[1], Babak Bayat[2], Simon Delandre[2], Mehdi Hamrouni[2], Marie Toussaint[2], Stephane Temmerman[2], Dan Lin[2] and Laurent Gatto[1*]

*Correspondence:
laurent.gatto@uclouvain.be

[1] Computational Biology and Bioinformatics, de duve Institute, UCLouvain, Brussels, Belgium
[2] GSK, Rixensart, Belgium

Abstract

Background: With the increase of the dimensionality in flow cytometry data over the past years, there is a growing need to replace or complement traditional manual analysis (i.e. iterative 2D gating) with automated data analysis pipelines. A crucial part of these pipelines consists of pre-processing and applying quality control filtering to the raw data, in order to use high quality events in the downstream analyses. This part can in turn be split into a number of elementary steps: signal compensation or unmixing, scale transformation, debris, doublets and dead cells removal, batch effect correction, etc. However, assembling and assessing the pre-processing part can be challenging for a number of reasons. First, each of the involved elementary steps can be implemented using various methods and R packages. Second, the order of the steps can have an impact on the downstream analysis results. Finally, each method typically comes with its specific, non standardized diagnostic and visualizations, making objective comparison difficult for the end user.

Results: Here, we present *CytoPipeline* and *CytoPipelineGUI*, two *R* packages to build, compare and assess pre-processing pipelines for flow cytometry data. To exemplify these new tools, we present the steps involved in designing a pre-processing pipeline on a real life dataset and demonstrate different visual assessment use cases. We also set up a benchmarking comparing two pre-processing pipelines differing by their quality control methods, and show how the package visualization utilities can provide crucial user insight into the obtained benchmark metrics.

Conclusion: *CytoPipeline* and *CytoPipelineGUI* are two Bioconductor *R* packages that help building, visualizing and assessing pre-processing pipelines for flow cytometry data. They increase productivity during pipeline development and testing, and complement benchmarking tools, by providing user intuitive insight into benchmarking results.

Keywords: Flow cytometry, Automated data analysis pipeline, Pre-processing, Quality control, Visualization

Background

With recent advances in flow cytometry technologies, it has become possible to measure up to 50 markers simultaneously for the same single cells [1]. As an immediate benefit, scientists now have access to richer flow cytometry experimental data. However, these advances also come at a cost, i.e. a more complex data analysis task. Indeed, traditional 'manual gating' data analysis procedures, which proceed by iterative hierarchical 2D representations of the data guided by biological knowledge, are unable to thoroughly extract the signal of interest from such high-dimensional data [2]. There is therefore a need to complement such manual, expert-based approaches with *computational flow cytometry*, i.e. a set of computational algorithms and methods for automated, reproducible and data-driven flow cytometry data analysis [2].

Those computational flow cytometry approaches translate into so-called data analysis pipelines. Examples of such automated pipelines have been published in the recent litterature (e.g. [3–7]). These consist of a series of data processing steps that are executed, one after the other, with the output of one step becoming the input of the next step. Schematically, for a typical flow cytometry data analysis, these numerous steps can usually be grouped into three big parts, coming after initial data sample acquisition:

- *data pre-processing and quality control*, which consists in both filtering undesirable and low quality events, and increasing the signal to noise ratio of the raw data, in order to feed the downstream steps with data of the highest possible quality;
- *population identification*, which aims at labelling the events with names of cell populations of interest;
- *downstream statistical analysis*, which can range from the simplest descriptive count/frequencies per population, to building complex prediction models for an outcome of interest, possibly for a high number of samples.

In what follows, we will mainly focus on the data pre-processing part, which can itself be split into several sub-tasks, or *steps* [8]: compensation, scale transformation, control (and possibly removal) of batch effects, control of signal stability in time (*QC in time*), filtering of undesirable events like debris, doublets and dead cells. All these steps are crucial to avoid that the subsequent analysis gets perturbed with erroneous signal (see e.g. [9] for compensation, [10] for scale transformation, [11] for signal stability in time, and [12] for other steps).

However, building good, automated pre-processing pipelines, suitable for the particular type of biological samples and biological question, can be challenging for a number of reasons. First, for each elementary step, there might exist a number of different computational methods, each of those having numerous parameters available to the user. For example, looking only into methods available on the Bioconductor project [13] for controlling the signal stability (*QC in time*), one finds at least four different methods available: *flowAI* [14], *flowClean* [15], *PeacoQC* [11], *flowCut* [16], and each of these methods comes with 7 to 11 different parameters. Second, the order of steps is not always set in stone, and applying different orders can lead to different outcomes, an effect coined steps interaction. These two facts lead to what we refer to as the *combinatorial problem of designing pipelines*, which means that, as the number of necessary elementary

steps increases, the number of possible pipeline designs grows exponentially. As a consequence, for the user, it becomes time consuming to build and assess even only a few of the possible step combinations, let alone testing a representative sample of them in a systematic manner.

On top of that combinatorial problem, the user is also faced with a lack of generic, standardized and user-friendly tools to evaluate and compare data pre-processing pipelines. On the one hand, each single pre-processing step method might come with its own approach for diagnostic and visualization (e.g. ad hoc plots, html or pdf reports), which allows the user neither to standardize the comparison process, nor to easily investigate the links and interactions between the different steps. On the other hand, there are a number of benchmarking studies comparing computational methods for flow cytometry data [17–20], but they tend to focus on only one part of the pipeline, which is usually the downstream analysis. Finally, there also exist some generic tools and frameworks to systematize the benchmarking process, including the interactions between different steps, such as for example *pipeComp* [21] and *CellBench* [22], and as well as some attempts to formally model the pipeline optimization problem in mathematical terms [23]. However, what is lacking for the end user is the ability to intuitively interpret the results of such benchmarkings. In other words, could one translate that a pipeline *A* outperforms a pipeline *B*, with respect to a specific performance metric, in terms of the obtained data characteristics, or number of filtered events. Therefore, there is still a need for flow cytometry practitioner-focused standardized tools for visual comparison of pre-processing pipelines.

Here, we present *CytoPipeline* and *CytoPipelineGUI*, two *R* packages aimed at facilitating the design and visual comparison of pre-processing pipelines for flow cytometry data. We describe the concepts underlying the software, provide some illustrative examples and demonstrate the use of the accompanying visualization utilities. We show that these new tools can help increasing the productivity during pipeline development and testing, and that they can complement benchmarking tools and studies, by providing the user with intuitive insight into benchmarking results. The *CytoPipeline* and *CytoPipelineGUI* packages are available on Bioconductor [13], as of version 3.17 and 3.18, respectively.

Methods

Implementation

In what follows, we assume that we have a dataset, provided as a set of files in Flow Cytometry Standard (*fcs*) format [24], on which we would like to apply a data pre-processing pipeline.

The *CytoPipeline* suite is composed of two main *R* packages, *CytoPipeline* and *CytoPipelineGUI*. While *CytoPipeline* is the main package, providing support for pipeline definition, running, monitoring and basic plotting functions, *CytoPipelineGUI* provides two interactive GUI applications enabling users to interactively explore and visualize the pipeline results. The *CytoPipeline* framework is based on two main concepts, namely *CytoPipeline* and *CytoProcessingStep*. A *CytoPipeline* object centralizes the pipeline definition, and specifies the run order of the different pipeline steps. These steps materialize as *CytoProcessingStep* objects, which store pipeline step names and the corresponding

R functions that will be called at execution time. These functions are either provided within the *CytoPipeline* package itself, exported from third party packages, or coded by the user themself. Together with the function name to be called, a *CytoProcessingStep* object also contains the list of parameters that are used as arguments to the function.

When creating a *CytoPipeline* object, the user can provide the description of the pipeline as a text file in *json* format [25]. Figure 1 shows the typical structure of such a *json* file. Note that, in practice, two different sets of processing steps, or pipelines, are described:

- A *scaleTransformProcessingSteps* pipeline, which describes the set of successive steps needed to generate the scale transformations that will be applied to the different channels of each of the *fcs* files that are included in the dataset. The *CytoPipeline* engine will run this pipeline first, and only once, prior to running the pre-processing on each *fcs* file.
- A *flowFramesPreProcessingSteps* pipeline, which describes the set of pre-processing steps that will be applied on each of the different *fcs* file independently.

Steps in both pipelines are described in the exact same way, i.e. by providing a user-chosen name for the step, the corresponding function that needs to be called by the engine upon running, and the set of arguments (i.e. the list of parameter names and corresponding values) that need to be provided to the function. Note that, on top of these explicitely defined arguments, the running engine will also take the output of each step as an implicit additional argument to the function called by the subsequent step.

The standard process for using *CytoPipeline* to build, run and inspect pre-processing pipelines is the following:

- define the pipeline by specifying the different steps using a descriptive text file, in *json* format;[1]
- run the pipeline, possibly for several data files in parallel, which involves writing and executing a short *R* script (see following sections);
- monitor the execution process thanks to a *CytoPipeline* provided workflow visualization utility;
- visualize and compare the results at different stages, using the *CytoPipelineGUI* interactive GUI applications.

In terms of technical infrastructure, the *CytoPipeline* package suite makes itself internal use of several technical *R* packages:

- *BiocParallel* [26] enabling parallel pre-processing of *fcs* sample files;
- *BiocFileCache* [27] enabling storage (i.e. *caching*) of all intermediary results for further inspection;
- *shiny* [28] for interactive visualizations.

[1] Note that *CytoPipeline* also provides methods to define a pipeline and its steps programmatically in *R*, without providing a text file as an input.

```
{
"scaleTransformProcessingSteps": [
    {
      "name": ["my_scale_transform_step1"],
      "FUN": ["scaleTransFormFunc1"],
      "ARGS": {
        "paramName1": ["value1"],
        "paramName2": ["value2"],
        "paramName3": ["value3"]
      }
    },
    {
      "name": ["my_scale_transform_step2"],
      "FUN": ["scaleTransFormFunc2"],
      "ARGS": {
        "paramName4": ["value4"]
    },

    (...) other scale transform processing steps

  ],
  "flowFramesPreProcessingSteps": [
    {
      "name": ["my_fcsfile_preprocessing_step1"],
      "FUN": ["preprocessingFunc1"],
      "ARGS": {
        "paramName5": ["value5"],
        "paramName6": ["value6"]
      }
    },
    {
      "name": ["my_fcsfile_preprocessing_step2"],
      "FUN": ["preprocessingFunc2"],
      "ARGS": []
    },

    (...) other flow frames preprocessing steps

  ]
}
```

Fig. 1 Structure of the user provided *json* file that describes a *CytoPipeline* object. The first pipeline (i.e. *"scaleTransformProcessingSteps"*) specifies how the preliminary calculation of the scale transformations is performed. Here only its first two steps are described. The first step, named *"my_scale_transform_step1"*, consists in calling the *"scaleTransFormFunc1"* function, with 3 parameters (*"paramName1"*, *"paramName2"* and *"paramName3"* provided as arguments, taking the specified *"value1"*, *"value2"* and *"value3"* value respectively. The second step, named *"my_scale_transform_step2"*, calls the *"scaleTransFormFunc2"*, with only one single parameter, i.e. *"paramName4"* taking *"value4"* as value. The second pipeline (i.e. *"flowFramesPreProcessingSteps"*) specifies the set of pre-processing steps performed on each data file independently. Here again, only the first two steps are described. The first one, named *"my_fcsfile_preprocessing_step1"* calls the *"preprocessingFunc1"* function with two parameters, while the second one, named *"my_fcsfile_preprocessing_step2"* calls the *"preprocessingFunc2"* function with no parameter (apart from the output of the previous step which is always used as an implicit additional argument)

Illustrative dataset

In order to demonstrate *CytoPipeline* functionalities, we make use of an illustrative dataset, the *HBV chronic mouse* dataset. This dataset was collected during a preclinical study aimed at assessing the effect of different therapeutic vaccine regimens on the immune response of Hepatitis B Virus transduced mice.

In this study, 56 male and female HLA.A2/DRB1 transgenic mice (transgenic for the human HLA-A2 and HLA-DRB1 molecules) were used. HLA.A2/DRB1 mice from groups 1, 2 and 4 were transduced at day 0 with adeno-associated virus serotype 2/8 (AAV2/8-HBV) vector carrying a replication-competent HBV DNA genome and randomized before immunization with 4 doses of vaccine candidate, based on level of HBs circulating antigen detected in the sera at day 21, age and gender proportions. Mice from group 3 were not transduced with AAV2/8-HBV viral vector and were immunized with four doses of vaccine candidate and finally, mice from group 5 were not transduced and received four doses of NaCl solution. Upon sacrifice, livers were collected, perfused with Phosphate Buffered Saline (PBS) to remove blood cells and after enzymatic treatment, lymphocytes were isolated, and stained with different monoclonal antibodies. The stained cells were acquired by flow cytometry using a BD Symphony A5 flow cytometer - the same instrument for all biological samples - and analyzed using the FlowJo v10.8 Software (BD Life Sciences).

Animal husbandry and experimental procedures were ethically reviewed and carried out in accordance with European Directive 2010/63/EU and the GlaxoSmithKline Biologicals' policy on the care, welfare and treatment of animals, in GSK animal facilities located in Rixensart, Belgium (AAALAC accredited). The ethical protocol of the GSK in vivo study was approved by the local GSK ethical committee.

This experiment resulted in the acquisition of 55 different *fcs* raw data file - one sample could not be acquired - with a flow cytometry panel of 12 different channels. The *HBV chronic mouse* dataset is available on Zenodo (DOI:10.5281/zenodo.8425840).

Applied pre-processing pipelines

Pipeline set-up

For the purpose of illustrating *CytoPipeline* functionalities, the 55 raw data files of the *HBV chronic mouse* dataset were used as input of two different pre-processing pipelines. Each pipeline was composed of the following steps:

- Reading of the raw *fcs* sample files, using the *flowCore* package [29].
- Margin events removal, which consists in identifying and removing the outliers using the *PeacoQC* package [11]. In short, manual boundaries per channel, corresponding to the instrument detection limits, are applied, and all events falling outside these boundaries are removed.
- Signal compensation, which consists in applying an existing compensation matrix. This matrix was generated by the flow cytometer at data acquisition time, and subsequently manually adjusted by the expert scientist.
- *QC in time*, which consists in eliminating parts of the signal that are not stable in time, using one of the corresponding QC algorithms (see below).
- Doublet removal, which consists in keeping the events that have a similar area vs. height ratio of the FSC channel signal pulse, and eliminating the doublets, which have a significantly higher ratio. This was performed using an ad hoc implementation in the *CytoPipeline* package.
- Debris removal, which consists in clustering the events in the (FSC-A, SSC-A) 2D representation, targetting a number of clusters provided by the user. After the clus-

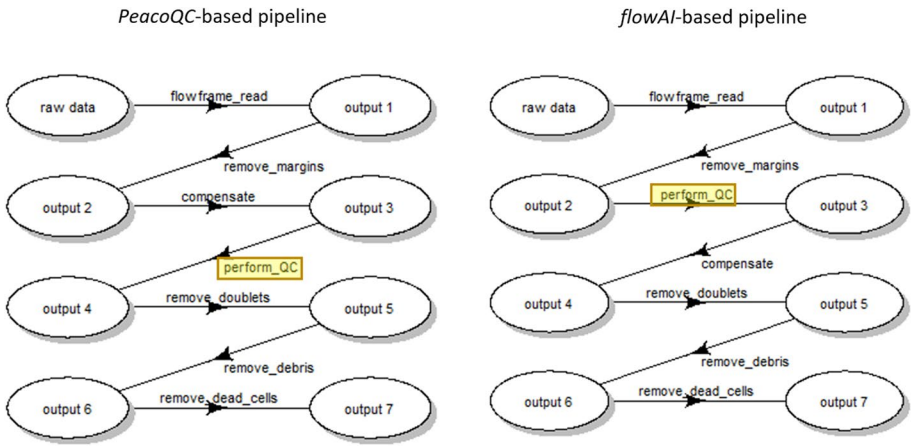

Fig. 2 Workflow of the subsequent steps applied in the pre-processing of each *fcs* file, for both pipelines. These plots have been generated using the *CytoPipeline* package

ters are obtained, the cluster of which the centroid lies nearest to the origin, i.e. with the smallest FSC-A (size) and smallest SSC-A (content, granularity), is considered as containing debris and removed. This was done using the *flowClust* package [30].

- Scale transformation, which consists in automatically estimating the parameters of a *logicle* transformation [31], using the *flowCore* package [29]. The obtained scale transformations were applied on all 55 sample files, and the parameters were estimated on an aggregation of a subset of 4 randomly chosen sample *fcs* files, after margin events removal and signal compensation.
- Dead cells removal, which consists in automatically setting a threshold between live cells and dead cells in the corresponding fluorescent 'Live & Dead' channel dimension, using the *flowDensity* package. The events having a 'Live & Dead' intensity above the found threshold are eliminated as dead cells.

However, the two pre-processing pipelines essentially differed by the method used for the *QC in time* step, as one used the *PeacoQC* package [11], while the other used the *flowAI* package [14]. In addition, the step order was also different, as the *PeacoQC* method is based on a peak detection algorithm which needs to run on compensated, scaled transformed data [11], while the *flowAI* method is advised to be applied on raw data [14]. Figure 2 outlines the different steps applied in the pre-processing of each *fcs* files, for both the *PeacoQC*-based pipeline, and the *flowAI*-based pipeline.

More detailed information on packages, versions and methods underlying each step (Additional file 1: Tables S1, S2 and S3), as well as the *json* configuration files defining respectively the *PeacoQC*-based and *flowAI*-based pipelines are available in the Additional file 1.

Running the pipelines and visualizing the results

In order to create the *CytoPipeline* objects representing the pipelines, run them and visualizing the results - including monitoring of the pipeline execution - a short *R* script needs to be written and executed. An example of such *R* script is provided in Additional file 1: Fig. S1. Note that, as a result of the centralization of the pipeline

definition, the code is very simple and concise, as for example, creating and running the pipeline boils down to essentially two *R* statements. Also, note that it is the same *R* code that triggers the execution of both pre-processing pipelines described in the previous section (except for the selection of the appropriate input *json* file and the choice of the experiment name under which to store the results). The distinctive part of the pipeline is located in the input *json* file, which describes the pipelines steps and their execution order.

Example benchmarking

Aiming at illustrating the use of *CytoPipeline* to provide insights into benchmarking results, we designed a benchmarking, which consisted in comparing the outcome of the two competing *PeacoQC*-based and *flowAI*-based pipelines described in the previous section, using the *HBV chronic mouse* dataset, to a ground truth. The latter was obtained by submitting the 55 raw data *fcs* files to an expert scientist, who manually pre-processed the files, gated the events using FlowJo. The obtained FlowJo workspace file was subsequently automatically processed using the *CytoML* package [32] version 2.12.0, and incorporated into a dedicated *CytoPipeline* ground truth pipeline for comparison with the two automated pipelines.

Regarding the benchmark evaluation metrics, for each single *fcs* file, the final output of each pipeline was compared to the ground truth, in terms of number of events, and the following metrics were calculated: sensitivity, specificity, precision and recall, which are defined as follows: let

- G (resp. B) be the set of events that are considered as **G**ood (resp. **B**ad) in the manual gating i.e. in the ground truth;
- F_G (resp F_B) be the set of events that are flagged as good (resp. flagged as bad) by the considered automated pipeline.

We can additionally define the following sets of events:

- $F_{G,\,\text{correct}} = F_G \cap G$, the set of events that are correctly flagged as good;
- $F_{B,\,\text{correct}} = F_B \cap B$, the set of events that are correctly flagged as bad.

The chosen evaluation metrics are then defined as:
$\text{sensitivity} = \frac{|F_{B,\,\text{correct}}|}{|B|}$; $\text{specificity} = \frac{|F_{G,\,\text{correct}}|}{|G|}$; $\text{precision} = \frac{|F_{B,\,\text{correct}}|}{|F_B|}$; $\text{recall} = \frac{|F_{G,\,\text{correct}}|}{|F_G|}$, where $|A|$ stands for the number of elements in the set A.

The benchmark was set up and performed using the *pipeComp* package [21], version 1.10.0. Indeed, *pipeComp* is a convenient tool to efficiently automate multiple alternative pipelines to be compared in the benchmark, as well as to automate the calculation of the evaluation metrics for each dataset used as benchmark input.

Results

Visual assessment and comparison of pipeline outputs

We used *CytoPipeline* to define both *PeacoQC* and *flowAI*-based pre-processing pipelines, as described in the Methods section, on the *HBV chronic mouse* dataset. We obtained results in the form of sets of data matrices (or *flowFrames*) after each step for each pre-processing pipeline. In the following paragraphs, we present some *CytoPipeline* visual assessment plots, according to 6 different use cases (Table 1). Use case #1 consists in visualizing a run and monitoring the status of the different steps. Use cases #2 to #5 consist in either looking at 'what happened' within a single pipeline for a single biological sample in isolation (use case #2), or comparing two different situations (flow frames) involving different pipelines (use cases #3 and #4), or involving different biological samples within the same pipeline (use case #5). Finally, use case #6 consists in assessing, and possibly modifying, the scale transformations obtained during a pipeline execution.

Table 1 Use cases of visual assessment and comparison of pipeline outputs. When the use case involves comparing two *flowFrames* obtained from different steps and/or different pipelines (i.e. use cases #2 to #4), or different samples (i.e. use case #5) the 3 columns 'sample', 'pipeline' and 'output' designate the initial *flowFrame* (referring to Fig. 2), while the 3 columns 'compared sample', 'compared pipeline' and 'compared output' designate the *flowFrame* that is compared to the initial *flowFrame*

Use case	Description	Sample	Pipeline	Output (cf. Fig. 2)	Compared sample	Compared pipeline	Compared output (cf. Fig. 2)
#1	Monitoring a run	*D91_G01*	*PeacoQC*	All	Not applicable	Not applicable	Not applicable
#2	Visualizing the effect of a single pipeline step	*D91_G01*	*PeacoQC*	Output 2 (before 'compensate')	*D91_G01*	*PeacoQC*	Output 3 (after 'compensate')
#3	Comparing the outcome of a pipeline step with different parameter values	*D91_G01*	*PeacoQC*	Output 6 (after 'remove_debris'), run with 3 clusters	*D91_G01*	*PeacoQC*	Output 6 (after 'remove_debris'), run with 2 clusters
#4	Comparing two different methods for one or several step(s)	*D91_A01*	*PeacoQC*	Output 7 (after 'remove_dead_cells')	*D91_A01*	*flowAI*	Output 7 (after 'remove_dead_cells')
#5	Comparing two different biological samples	*D91_A01*	*PeacoQC*	Output 7 (after 'remove_dead_cells')	*D93_B05*	*PeacoQC*	Output 7 (after 'remove_dead_cells')
#6	Visualization and update of generate scale transformations	*D91_G01*	*PeacoQC*	Not applicable	Not applicable	Not applicable	Not applicable

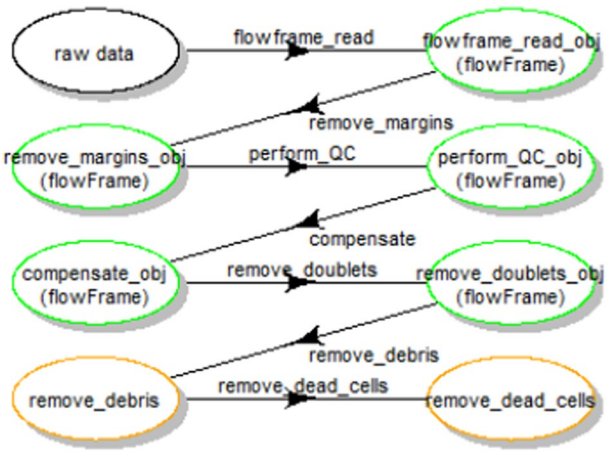

Fig. 3 Use case #1: summary workflow view of the run - green nodes correspond to steps that ran to completion for the selected sample file, orange nodes correspond to steps that have not generated an output yet

Use case #1: monitoring a run

As all the intermediate results produced during pipeline execution are saved (see Methods/Implementation section), it is possible to generate a summary workflow view, consecutive to a run. Figure 3 shows an example of such a display, obtained after running the *flowAI*-based pipeline described above, where there was a spelling error in one of the parameter names of the *"remove_debris"* step. On top of showing the sequence of steps, a colour code is used to highlight which of the steps have run to completion, and which of the steps need to be re-run. Here, for the selected sample, the pipeline ran correctly until the *"remove_doublets"* step (green nodes), but did not produce any output for the subsequent steps (orange nodes), which is due to the spelling error in the definition of the *"remove_debris"* step. Based on this summarized visual information, the user can now dig into the flagged problematic step, and/or track the particular characteristics of the sample which generated the error. More details on the colour code used in this plot can be found in Additional file 1: Fig. S2.

Use case #2: visualizing the effect of a single pipeline step

In Fig. 4, the user is visually assessing two consecutive states of the *flowFrame* of sample *D91_G01*, within the same run of the *PeacoQC*-based pipeline. To evaluate the effect of the compensation step, the *"before compensation"* (output 2, cf. Fig. 2) and the *"after compensation"* (output 3) states of the pipeline are visually compared. Note that this visualization can be done according to any pair of selected channels/markers (2D distribution representation), or according to a 1D marginal distribution representation for any selected channel/marker. Here, the (CD8, CD38) 2D view shows, on the left, that the fluorescence of the dye BB700 (CD38) spills into the CD8 channel. On the right, application of a pre-computed compensation matrix (see Methods section) has rectified the distribution of the two markers, revealing different ranges of CD38 (an activation marker) between the CD8+ and CD8- populations. A corresponding

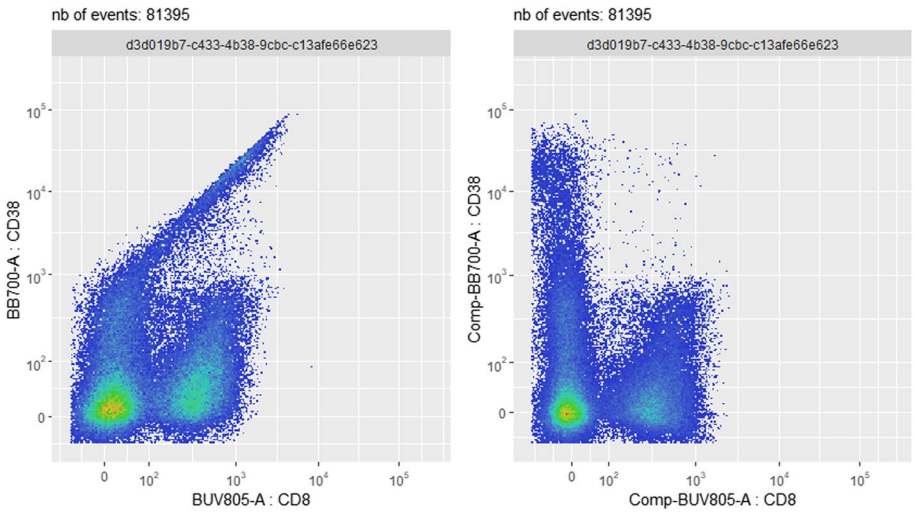

Fig. 4 Use case #2: effect of a single pipeline step - here, the compensation step of the *PeacoQC*-based pipeline for sample *D91_G01*. On the left, spillover of the BB700 (CD38) dye fluorescence into the CD8 channel creates a visual artefact, with events wrongly flagged as double positive CD8+ CD38+. On the right, compensation has rectified the bivariate distribution of the two markers

screenshot of the interactive GUI application, implemented in the *CytoPipelineGUI* package, can be found in Additional file 1: Fig. S3.

Use case #3: comparing the outcome of a pipeline step with different parameter values

This use case involves running the same pipeline, with the same steps but with amended values for one or several steps, in order to investigate which parameter combination performs better. An illustrating example is shown in Fig. 5, where the outcome of the debris removal step is compared when applying two different user input number of clusters (three on the left plot, vs. two on the middle plot). On the right plot, events coloured in red are the ones that are eliminated when applying the debris removal step when the number of clusters is two, but not eliminated when the number of clusters

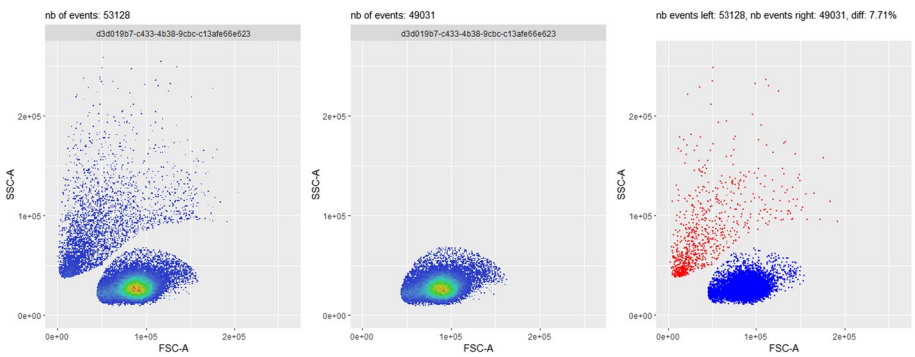

Fig. 5 Use case #3: comparison of two different parameter settings for the debris removal step, on sample *D91_G01*. The setting with two clusters (in the middle) better eliminates undesirable events than the setting with three clusters (on the left). On the right, an explicit comparison between the two *flowFrames* is performed. Red dots correspond to events that are present on the left hand side plot, but not present on the middle plot, while blue dots correspond to events that are present on both plots

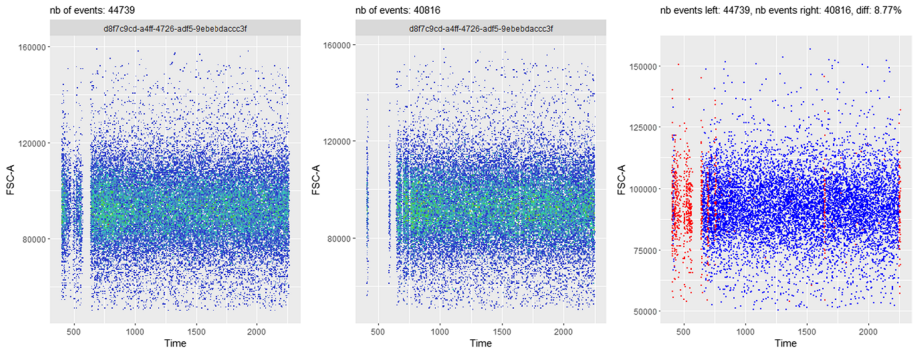

Fig. 6 Use case #4: comparison of the final state results, for sample *D91_A01*, between the *PeacoQC*-based pipeline and the *flowAI*-based pipeline - here using a (*FSC-A* vs. *Time*) plot. On the left, the end state of the *PeacoQC*-based pipeline is shown, while the end state of the *flowAI*-based pipeline is shown on the middle plot. On the right, an explicit comparison between the two *flowFrames* is performed. Red dots correspond to events that are present on the left hand side plot, but not present on the middle plot, while blue dots correspond to events that are present on both plots. This figure reveals that, for this particular sample, *flowAI* tends to remove time chunks more aggressively than *PeacoQC*

is three. Let us recall that the debris elimination step consists in clustering the events in a fixed number of clusters, followed by the elimination of the cluster nearest to the origin - see Methods section. Here, specifically, the user can conclude that the debris removal algorithm (based on *flowClust* package) does a better job selecting the target events when the appropriate number of target clusters is used, i.e. two clusters, as on the middle plot. This is because the cell population of interest, here a population of lymphocytes extracted from mice liver tissues, naturally groups into one single cluster in the (FSC-A, SSC-A) 2D representation. As a consequence, in this case, two is the optimal number of clusters (one cluster of debris, one cluster of lymphocytes). Additional file 1: Fig. S4 illustrates the removal of events during the debris removal step, for the 2 clusters and the 3 clusters cases.

Use case #4: comparing two different methods for one or several steps

This use case is a generalization of the preceeding one, where the user wants to compare the performance of two different methods for one or several steps of the pre-processing pipeline. For instance, Fig. 6 provides a comparison between the *PeacoQC*-based and the *flowAI*-based pipelines, applied on a particular biological sample of the *HBV chronic mouse* dataset. This comparison, obtained by plotting one specific channel (here the *FSC-A*) as a function of time, reveals that *flowAI* removes time chunks more aggressively than *PeacoQC*, for the current sample. Note that this comparison can also be done for any 2D combination of makers (not shown here).

Use case #5: comparing two different biological samples

It is also possible to compare two different biological samples of the same dataset, at any specific step of any pipeline. This allows e.g. to check that the methods used for the various pre-processing steps perform consistently across the whole dataset. One example is shown in Fig. 7, where two different samples are displayed in a 2D plot with the *FSC-A* and *Live &Dead* channels. In this case, the two samples show very similar bivariate

Fig. 7 Use case #5: comparison of the final state results between two different biological samples (on the left: sample *D91_A01* and on the right: sample *D93_B05*), within the same *PeacoQC*-based pipeline. For the particular channels chosen (*Live &Dead* vs. *FSC-A*), the two samples show very similar bivariate distributions. Based on this 2D representation, one could conclude that the pre-processing pipeline has correctly selected the target cell population in both cases

distributions. Based on this 2D representation, one could conclude that the pre-processing pipeline has correctly selected the target cell population in both cases. This would however need careful confirmation based on other 2D combinations of markers, e.g. *FSC-H* vs. *FSC-A* (for doublets elimination), and *FSC-A* vs. *SSC-A* (for debris elimination).

Use case #6: visualization and update of generated scale transformations

Besides the *flowFrame* comparison tool, *CytoPipelineGUI* also provides a second interactive GUI application, which is aimed at inspecting the scale transformations obtained from the corresponding *scaleTransformProcessingSteps* pipeline (see Methods/Implementation section). If the shape of the distribution after transformation needs adjustment (for example for better separation of negative and positive populations for a specific marker), the user can manually adapt the scale transformation parameters, interactively assess the impact of their modifications, and apply these modifications to the scale transformations for further use in the pre-processing pipelines (Fig. 8). These manual adjustments can be very useful, for example when the automatic transformation parameter adjustment algorithm has not worked satisfactorily. Figure 9 shows an example where the *logicle* transformation [31] applied on marker CD38 (left) shows spurious density oscillations in the negative domain. Manually adjusting the *positive decimals* parameter of the *logicle* transformation leads to a better looking density plot, where one can more easily distinguish CD38-, CD38+ and CD38++ populations.

Benchmarking results

As mentioned in the Methods section, we used *pipeComp* [21] to perform a benchmarking exercise, comparing two different pre-processing pipelines, i.e. the *PeacoQC*-based and the *flowAI*-based pipelines, on the 55 sample files of the *HBV chronic mouse* dataset,

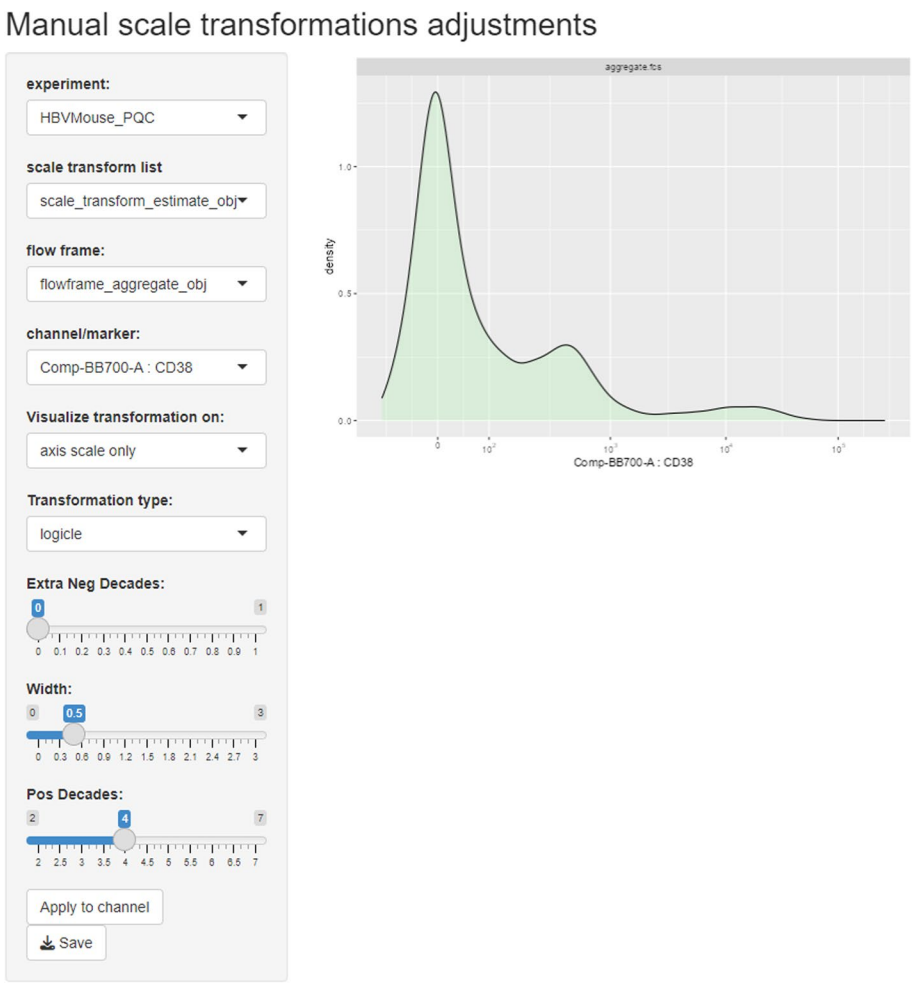

Fig. 8 Use case #6: screenshot of the *CytoPipelineGUI* interactive GUI application enabling the inspection, manual adjustment and save of pipeline generated scale transformations. Here the user is visualizing the transformation applied on marker CD38, for sample *D91_G01*

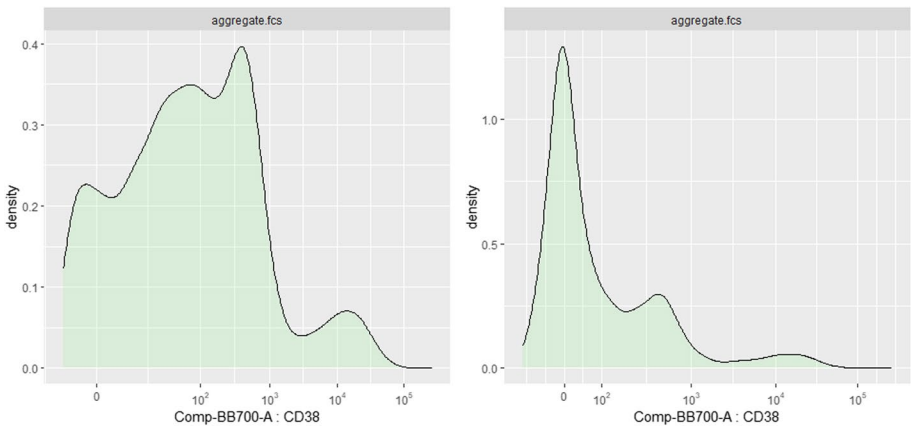

Fig. 9 Manual parameters adjustment of the *logicle* transformation applied on marker CD38, for sample *D91_G01*. On the left, the density plot shows spurious oscillations in the negative domain. On the right, manual adjustment on the *positive decimals* parameter of the *logicle* transformation leads to a better looking transformed density, where one can more easily identify CD38-, CD38+ and CD38++ populations

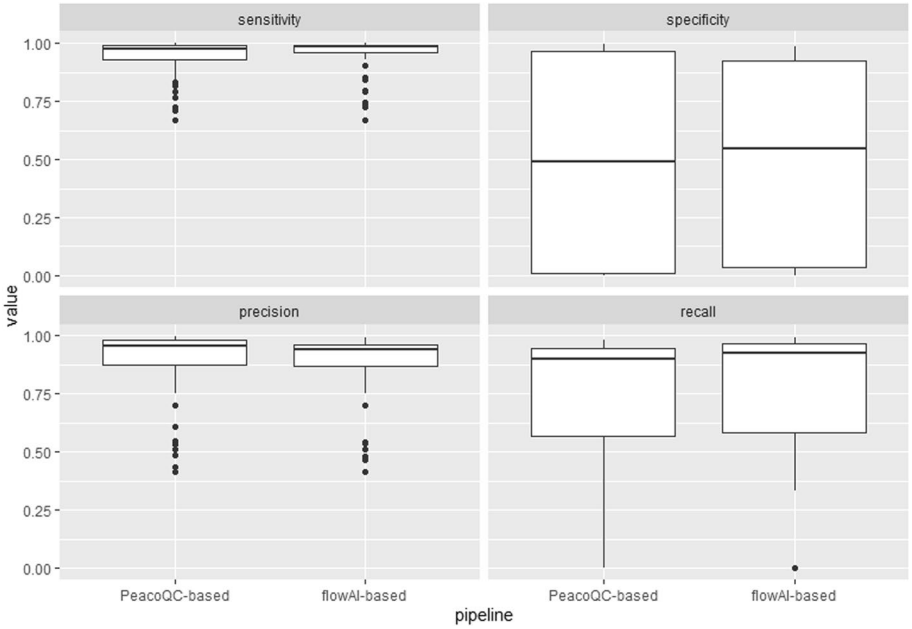

Fig. 10 Box plots of the distributions of calculated evaluation metrics per sample, for the two competing pipelines. Globally, both pipelines perform very similarly, for all four evaluation metrics, i.e. sensitivity, specificity, precision and recall

and calculating evaluation metrics in terms of how well the automated pipelines could match the manual pre-processing performed by an expert scientist ('ground truth'). A global assessment shows comparable results between the two competing pipelines, consistently across all metrics (Fig. 10). However, when directly contrasting sample by sample results (Fig. 11) one can identify that the pipeline performance is rather heterogeneous across the 55 biological samples.

In order to better understand the behaviour of the two competing automated pipelines on different samples, we selected three different samples, corresponding to different locations into the specificity plot of Fig. 11. We then used *CytoPipelineGUI* to inspect the results at different steps, for the two automated pipelines as well as for the 'ground truth':

- Sample *D91_C07* was an outlier for which the *PeacoQC*-based pipeline obtained an almost zero specificity, while *flowAI*-based pipeline specificity was around an acceptable level of above 0.8. However, as shown in Fig. 12, this was not due to the different *QC in time* algorithm (*PeacoQC* vs. *flowAI*), but to a lack of robustness of the dead cells removal algorithm, leading to an interaction phenomenon by which almost all events were removed in the dead cell removal step of the *PeacoQC*-based pipeline.
- Sample *D93_A05* resulted in a very low specificity for both pipelines. Investigation using *CytoPipelineGUI* revealed that this sample was in fact one of the low quality samples wherein the interesting cell population was a small minority of the events, while there was a great abundance of debris and dead cells (Additional file 1: Fig. S5). As a consequence, both pipelines were unable to automatically select the correct cell population, regardless of the *QC in time* method used.

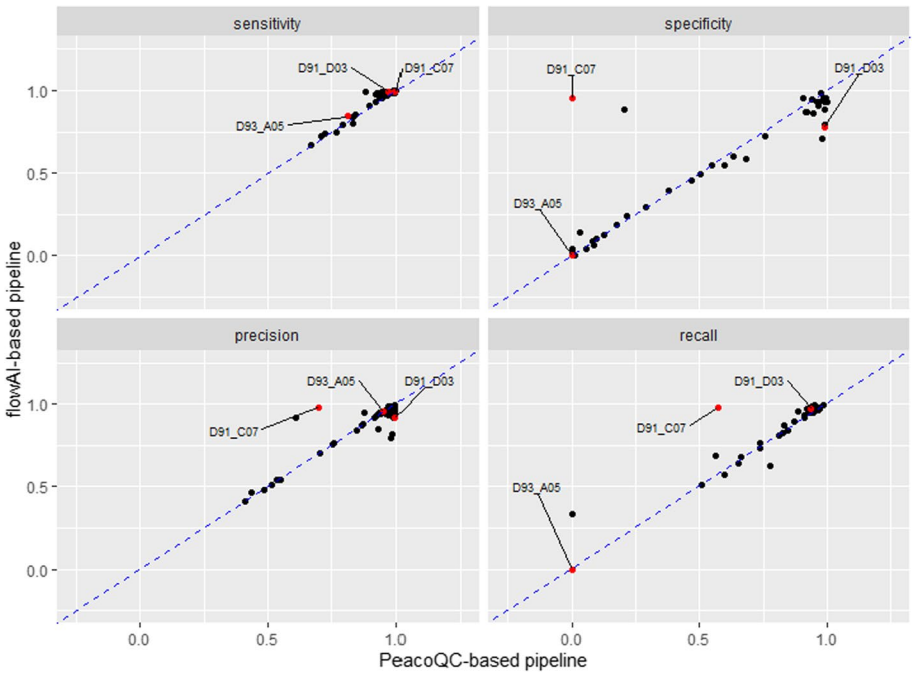

Fig. 11 Scatter plots comparing the two pre-processing pipelines, each dot representing one of the 55 samples. Three specific samples are highlighted in red, corresponding to very different comparative behaviour of the two competing pipelines. Sample *D91_C07* is a unique sample for which the *flowAI*-based pipeline has a high specificity, but the *PeacoQC*-based pipeline has very low specificity. Sample *D93_A05* is one of the samples leading to low specificity for both pipelines, while *sample D91_D03* is representative of the samples for which both pipelines provide good specificity

- Sample *D91_D03* was an example where both automatic pipelines performed adequately without major issues. Here, the difference in metrics is effectively related to the choice of *QC in time* method. Looking at a specific visualization where time is displayed on the x axis (Additional file 1: Fig. S6), and based on both qualitative plot inspection and number of events comparison with the manual gating ground truth, *CytoPipelineGUI* reveals that *flowAI* method is too agressive in this case, while *PeacoQC* is too liberal.

Note that the conclusions of these visual inspections are particularly precious to the scientist in charge of building the data analysis pipelines, who is now able to get precise and accurate insight into why one pipeline performs better than the other, for specific samples. In particular, they are much better equipped to distinguish between an intrinsic performance difference between some competing methods, and surprising artefacts like a side effect of low sample quality or an interaction between two different steps.

Discussion

CytoPipeline, a flexible framework for building and running pre-processing pipelines

In this work, we have demonstrated the use of the *CytoPipeline* suite by implementing pre-processing pipelines on the *HBV chronic mouse* dataset. The implementation of

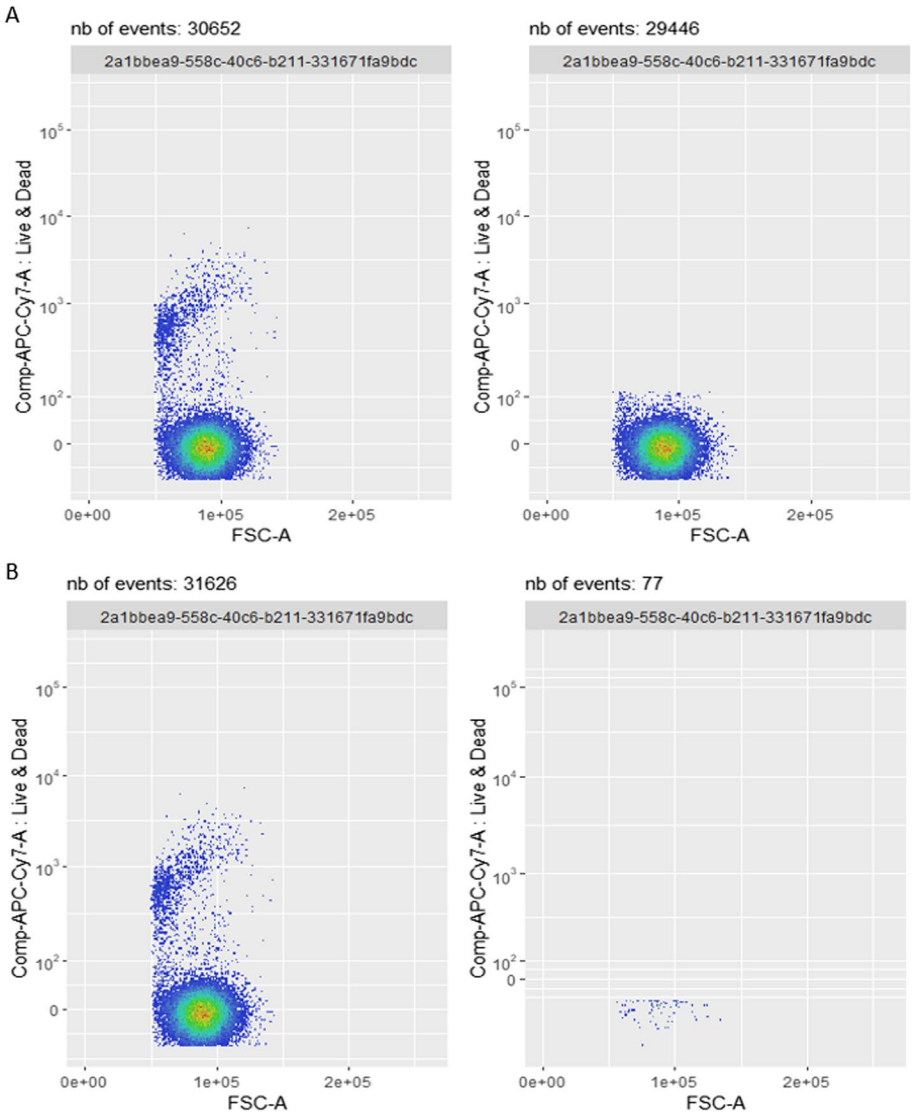

Fig. 12 Comparison between the dead cells removal step between the *flowAI* pipeline (**A**), and the *PeacoQC*-based pipeline (**B**), on sample *D91_C07*. While the input set of events look very similar (left plots of panels **A and B**), the dead cells removal step of the *PeacoQC*-based pipeline (right plot of panel B) wrongly removes most of the events. This reveals a lack of robustness of the algorithm, unrelated to the *QC in time* method used (*flowAI* vs. *PeacoQC*)

CytoPipeline, with a centralized specification of the pipeline definition in a *json* file, leads to a better design of the pipeline code. As a result, we believe that the user productivity, when coding and testing different pipelines, can be greatly improved.

In order to illustrate this, we implemented the two *PeacoQC*-based and *flowAI*-based competing pipelines, described in Methods, in two *R* scripts, without using *CytoPipeline* objects, and looked into the duplication effort as well as the future extensibility of the code. These pieces of code are provided in the *2023-CytoPipeline-code* GitHub repository (see Code Availability in Declarations section).

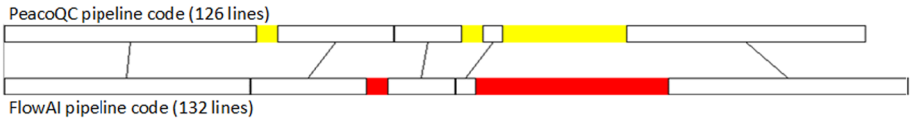

Fig. 13 Structure of the *R* script implementations of the *PeacoQC*-based and *flowAI*-based pipelines. The common parts are shown in white, *PeacoQC*-based pipeline specific parts in yellow, and *flowAI*-based pipeline specific parts in red. Between the two pipelines, 79% of the code is in common, and the pipeline specific parts are not fully gathered in one single location

Figure 13 provides a schematic comparison between these two pieces of code, as well as indicative number of code lines. Of course, these relates to one particular implementation, as there are countless ways to program the same pipelines. What is interesting to note, though, is that there is a high proportion of code duplication, but the differences are not only located in one single place, due to the subtle differences induced by the change of orders in the steps. This is likely to lead to a high code maintenance burden in the future, for instance when extending the program to many more pipeline instances, which can use different step methods, different method parameters etc. In constrast, let us recall that, when using *CytoPipeline*, the *R* code itself stays the same, as all differences are explicitly described in the input *json* file. This *json* file is easier to maintain and extend than the *R* scripts represented in Fig. 13.

CytoPipeline provides a standardized and user-friendly tool for visual investigations

We have presented a series of use cases of *CytoPipeline* visualizations. In all these use cases, we took advantage of the same set of visualization tools, in a standardized way, but translated into different contexts, whatever the underlying methods used for the pre-processing pipelines. Also during the investigation of the benchmarking results, visual comparisons could be made with a ground truth manual gating, again using the same tools. Besides, the interactive GUI applications, implemented in *CytoPipelineGUI*, provide user interactivity and facilitate the investigation process. As stated in the introduction, these visual assessments are extremely important for the scientists, as they provide a unique mean to:

- visually control for the quality of the data samples, and acquire insight on the corresponding sample variability;
- visually check the robustness of the methods used in a given pre-processing pipeline, including the adequacy of the chosen user input parameters;
- visually compare different pre-processing pipeline settings. This can range from comparing different possible choices of method for a particular step, to assessing which one of two or more competing pipelines, possibly mixing different step methods in different orders, is performing better for the considered dataset.

CytoPipeline allows user intuitive insight into benchmarking results

As part of this work, we have implemented a benchmarking comparing two competing pre-processing pipelines, with the main objective of showing the benefits of using *CytoPipeline* visualization tools, as a complement to the benchmarking itself.

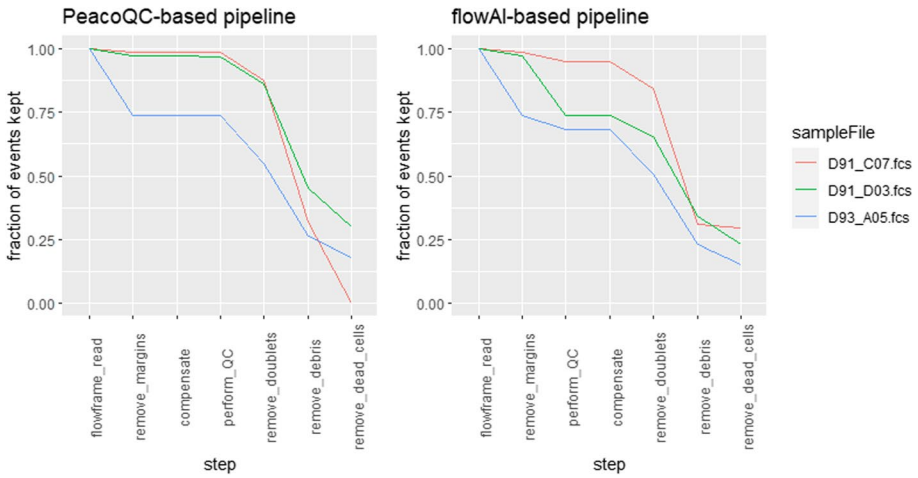

Fig. 14 Plots showing the proportion of retained events at each pre-processing step, for each sample. On the left, the *PeacoQC*-based pipeline shows, for sample *D91_C07*, a sharp drop in the last *remove_dead_cells* step. On the right, the *flowAI*-based pipeline does not show the same phenomenon

We showed that detailed comparison plots help the user investigating some specific benchmarking results, hence getting better intuition into the benchmarking outcome. We have indeed demonstrated that there can be numerous reasons why a pipeline instance performs better than another on specific samples, and it is key for the scientist to get a clear view of these reasons, and their possible links with sample characteristics. Therefore, we think that *CytoPipeline* is a powerful tool for interpreting the outcome of benchmarking studies.

Using the proportion of events kept at each step as a diagnostic tool

As was shown in various figures in the Results section (see e.g. Fig. 12), *CytoPipelineGUI* computes the number of events that are retained at each step (shown as subtitles in the individual density plots). Tracking these changes throughout the pre-processing steps of a pipeline for different samples is a useful quality control. This can be implemented using some of the *CytoPipeline* functions, and is shown on Fig. 14.

Limitations and possible extensions of the work

The *CytoPipeline* suite of *R* packages can be positioned as a tool to facilitate the design, testing and comparison of pre-processing pipelines for the end user. It is not meant to be:

- A novel pre-processing pipeline in itself, as it does not provide new methods for the various pre-processing steps (although it includes some functions calling some widely used methods), nor an innovative way to combine some of these.
- A tool facilitating benchmarking automation, like *pipeComp*. For example, unlike *pipeComp* [21], *CytoPipeline* does not provide any optimization solution to reduce the amount of CPU time and memory to run a potentially huge amount of (combi-

nations of) possible pipelines. However, as mentioned before, *CytoPipeline* is used to facilitate the interpretation of results produced with benchmarking tools.

Regarding scalability, one should distinguish CPU and memory from hard drive storage requirements. CPU- and memory-wise, *CytoPipeline* has no particular issues when dealing with large number of samples, as long as each single fcs file can fully reside in memory. Indeed, as described in the Methods section, the engine that executes pre-processing pipelines supports both sequential and parallel file processing, and benefits from all multi-tasking scheduling options provided by the *BiocParallel* [26] package. However, storage-wise, caching data at each step leads to large storage needs when processing many files. Typically, when analysing datasets including hundreds of fcs files, with several millions of events, compared across several pipelines and many processing steps, storage needs can require several terabytes. In those cases, users of *CytoPipeline* will typically need to call on high capacity storage facilities.

Another limitation of our work is the following: while *CytoPipelineGUI* is a powerful visualization tool for exploring specific pipeline steps for one or two samples, it does not provide an overall quality control of all samples at once. In that sense, it would be useful, especially for large datasets, to provide a global view of how samples differ at each pre-processing step. As mentioned above, one such diagnostic view can be obtained, by plotting the fraction of retained events at each pre-processing step (Fig. 14). Another promising approach focuses on the visualisation of all samples at once to identify specific outliers [33].

Finally, another possible extension would be to further develop *CytoPipeline*, as to not only include the building and assessment of pre-processing steps, but also include support for subsequent steps of the data analysis: batch correction, population identification, etc.

Conclusion

In this work, we have introduced a suite of *R* packages, *CytoPipeline* and *CytoPipelineGUI*, that helps building, visualizing and assessing pre-processing pipelines for flow cytometry data. We have demonstrated several use cases on a real life dataset, and highlighted several concrete benefits of these tools. For the new user, the packages come with ample documentation and tutorial videos, accessible through the package vignettes. We trust that using *CytoPipeline* will favour productivity in testing and assessing alternative data pre-processing pipelines, with the aim of designing good pre-processing and QC solutions for each particular context. The latter can be the specific type of biological sample, technology used (conventional flow cytometry, cytof, spectral flow cytometry), panel composition, experimental design etc., which in turn highly depend on the biological question at hand.

Supplementary Information

The online version contains supplementary material available at https://doi.org/10.1186/s12859-024-05691-z.

Additional file 1. Supplementary tables, figures and pipeline configuration files.

Author contributions
Conceptualization: PH, LG; Methodology: PH, LG, DL, SD; Software: PH, LG; Data collection: BB, MH; Writing - original draft: PH, BB; Writing - review & editing: LG, DL, ST, SD, MH, MT; Supervision: LG, DL, ST, MT.

Funding
This work was funded by GlaxoSmithKline Biologicals S.A., under a cooperative research and development agreement between GlaxoSmithKline Biologicals S.A. and de Duve Institute (UCLouvain).

Availability of data and materials
Raw flow cytometry data files, as well as the manual gating information considered as the ground truth for the benchmarking, are available on Zenodo (DOI:10.5281/zenodo.8425840).

Code availability
All code needed to reproduce the results presented in the current article is available on the following GitHub repository: https://github.com/UCLouvain-CBIO/2023-CytoPipeline-code, of which a release has been archived on Zenodo (DOI:10.5281/zenodo.8425840).

Declarations

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
P.H. is a student at the de Duve Institute (UCLouvain) and participates in a post graduate studentship program at GSK; B.B. is employee of the GSK group of companies, reports ownership of GSK shares and is listed as inventor on patent(s) owned by the GSK group of companies; S.D. is employee of the GSK group of companies and reports ownership of GSK shares; M.H. is employee of the GSK group of companies; M.T. is employee of the GSK group of companies, reports ownership of GSK shares and is listed as inventor on patent(s) owned by the GSK group of companies; S.T. is employee of the GSK group of companies, reports ownership of GSK shares and is listed as inventor on patent(s) owned by the GSK group of companies; D.L. is employee of the GSK group of companies and reports ownership of GSK shares; L.G. reports no competing interest.

Received: 10 November 2023 Accepted: 2 February 2024
Published online: 20 February 2024

References

1. McKinnon KM. Flow cytometry: an overview. Curr Protoc Immunol. 2018;120:511–5111.
2. Saeys Y, Van Gassen S, Lambrecht BN. Computational flow cytometry: helping to make sense of high-dimensional immunology data. Nat Rev Immunol. 2016;16(7):449–62.
3. Quintelier K, Couckuyt A, Emmaneel A, Aerts J, Saeys Y, Van Gassen S. Analyzing high-dimensional cytometry data using FlowSOM. Nat Protoc. 2021;16(8):3775–801.
4. Nowicka M, Krieg C, Crowell HL, Weber LM, Hartmann FJ, Guglietta S, et al. CyTOF workflow: differential discovery in high-throughput high-dimensional cytometry datasets. F1000Res. 2017;6:748.
5. Rybakowska P, Van Gassen S, Quintelier K, Saeys Y, Alarcón-Riquelme ME, Marañón C. Data processing workflow for large-scale immune monitoring studies by mass cytometry. Comput Struct Biotechnol J. 2021;19:3160–75.
6. Ashhurst TM, Marsh-Wakefield F, Putri GH, Spiteri AG, Shinko D, Read MN, et al. Integration, exploration, and analysis of high-dimensional single-cell cytometry data using Spectre. Cytometry A. 2022;101(3):237–53.
7. Rybakowska P, Van Gassen S, Martorell Marugán J, Quintelier K, Saeys Y, Alarcón-Riquelme ME, et al. Protocol for large scale whole blood immune monitoring by mass cytometry and Cyto Quality Pipeline. STAR Protoc. 2022;3(4):101697.
8. Liechti T, Weber LM, Ashhurst TM, Stanley N, Prlic M, Van Gassen S, et al. An updated guide for the perplexed: cytometry in the high-dimensional era. Nat Immunol. 2021;22(10):1190–7.
9. Mazza EMC, Brummelman J, Alvisi G, Roberto A, De Paoli F, Zanon V, et al. Background fluorescence and spreading error are major contributors of variability in high-dimensional flow cytometry data visualization by t-distributed stochastic neighboring embedding. Cytometry A. 2018;93(8):785–92.
10. Finak G, Perez JM, Weng A, Gottardo R. Optimizing transformations for automated, high throughput analysis of flow cytometry data. BMC Bioinform. 2010;11:546.
11. Emmaneel A, Quintelier K, Sichien D, Rybakowska P, Marañón C, Alarcón-Riquelme ME, et al. PeacoQC: peak-based selection of high quality cytometry data. Cytometry A. 2022;101(4):325–38.
12. den Braanker H, Bongenaar M, Lubberts E. How to prepare spectral flow cytometry datasets for high dimensional data analysis: a practical workflow. Front Immunol. 2021;12: 768113.
13. Huber W, Carey VJ, Gentleman R, Anders S, Carlson M, Carvalho BS, et al. Orchestrating high-throughput genomic analysis with Bioconductor. Nat Methods. 2015;12(2):115–21.
14. Monaco G, Chen H, Poidinger M, Chen J, de Magalhães JP, Larbi A. flowAI: automatic and interactive anomaly discerning tools for flow cytometry data. Bioinformatics. 2016;32(16):2473–80.

15. Fletez-Brant K, Špidlen J, Brinkman RR, Roederer M, Chattopadhyay PK. flowClean: automated identification and removal of fluorescence anomalies in flow cytometry data. Cytometry A. 2016;89(5):461–71.
16. Meskas J, Yokosawa D, Wang S, Segat GC, Brinkman RR. flowCut: an R package for automated removal of outlier events and flagging of files based on time versus fluorescence analysis. Cytometry A. 2023;103(1):71–81.
17. Liu X, Song W, Wong BY, Zhang T, Yu S, Lin GN, et al. A comparison framework and guideline of clustering methods for mass cytometry data. Genome Biol. 2019;20(1):297.
18. Weber LM, Robinson MD. Comparison of clustering methods for high-dimensional single-cell flow and mass cytometry data. Cytometry A. 2016;89(12):1084–96.
19. Cheung M, Campbell JJ, Thomas RJ, Braybrook J, Petzing J. Assessment of automated flow cytometry data analysis tools within cell and gene therapy manufacturing. Int J Mol Sci. 2022;23(6):3224.
20. Aghaeepour N, Chattopadhyay P, Chikina M, Dhaene T, Van Gassen S, Kursa M, et al. A benchmark for evaluation of algorithms for identification of cellular correlates of clinical outcomes. Cytometry A. 2016;89(1):16–21.
21. Germain PL, Sonrel A, Robinson MD. pipeComp, a general framework for the evaluation of computational pipelines, reveals performant single cell RNA-seq preprocessing tools. Genome Biol. 2020;21(1):227.
22. Su S, Tian L, Dong X, Hickey PF, Freytag S, Ritchie ME. Cell Bench: R/Bioconductor software for comparing single-cell RNA-seq analysis methods. Bioinformatics. 2020;36(7):2288–90.
23. Selega A, Campbell KR.: Multi-objective Bayesian optimization with heuristic objectives for biomedical and molecular data analysis workflows. Preprint at https://www.biorxiv.org/content/early/2022/06/12/2022.06.08.495370.
24. Spidlen J, Moore W, Parks D, Goldberg M, Bray C, Bierre P, et al. Data file standard for flow cytometry, version FCS 3.1. Cytometry A. 2010;77(1):97–100.
25. Pezoa F, Reutter JL, Suarez F, Ugarte M, Vrgoč D. Foundations of JSON schema. In: Proceedings of the 25th International Conference on World Wide Web. International World Wide Web Conferences Steering Committee; 2016. p. 263–273.
26. Morgan M, Wang J, Obenchain V, Lang M, Thompson R, Turaga N.: BiocParallel: Bioconductor facilities for parallel evaluation. R package version 1.34.0. Available from: https://bioconductor.org/packages/BiocParallel.
27. Shepherd L, Morgan M.: BiocFileCache: Manage Files Across Sessions. R package version 2.8.0. Available from: https://bioconductor.org/packages/BiocFileCache.
28. Chang W, Cheng J, Allaire J, Sievert C, Schloerke B, Xie Y, et al.: shiny: Web Application Framework for R. Available from: https://shiny.posit.co/.
29. Ellis B, Haaland P, Hahne F, Le Meur N, Gopalakrishnan N, Spidlen J, et al.: flowCore: Basic structures for flow cytometry data. R package version 2.12.0. Available from: https://bioconductor.org/packages/flowCore.
30. Lo K, Hahne F, Brinkman RR, Gottardo R. flowClust: a Bioconductor package for automated gating of flow cytometry data. BMC Bioinform. 2009;10:145.
31. Parks DR, Roederer M, Moore WA. A new "Logicle" display method avoids deceptive effects of logarithmic scaling for low signals and compensated data. Cytometry A. 2006;69(6):541–51.
32. Finak G, Jiang W, Gottardo R. CytoML for cross-platform cytometry data sharing. Cytometry A. 2018;93(12):1189–96.
33. Hauchamps P, Gatto L.: CytoMDS: Low Dimensions projection of cytometry samples. R package version 0.99.8. Available from: https://uclouvain-cbio.github.io/CytoMDS.

Publisher's Note