Benchmark of pre-processing pipelines of single cell RNA sequencing data

Universitat Oberta de Catalunya

Dídac Jiménez Sánchez
MU Bioinformàtica i Bioestadística
Omics data analysis area

Final project Tutor
Alfonso Saera Vila

Date of submittal:
20/06/2023

FITXA DEL TREBALL FINAL

Títol del treball:	*Benchmark of pre-processing pipelines of single cell RNA sequencing data*
Nom de l'autor:	*Dídac Jiménez Sánchez*
Nom del consultor/a:	*Alfonso Saera Vila*
Nom del PRA:	*Alfonso Saera Vila*
Data de lliurament (mm/aaaa):	*06/2023*
Titulació o programa:	*Màster universitari en Bioinformática i bioestadística*
Àrea del Treball Final:	*Anàlisi de dades òmiques*
Idioma del treball:	*Anglès*
Paraules clau	*scRNA-seq, bioinformatics, preprocessing*

Resum del Treball

El *single-cell RNA sequencing* (scRNA-Seq) és una eina molt potent per estudiar el transcriptoma. Hi ha diverses eines disponibles per preprocessar les dades que se n'obtenen. Tanmateix, no existeix cap estàndard. L'objectiu d'aquest treball és comparar quatre eines de preprocessament de scRNA-Seq, extreure conclusions sobre els avenços dels darrers anys, i determinar si hi ha una eina superior. Les eines comparades han sigut (i) UMI Tools, una de les primeres eines publicades, que serveix de punt de referència, (ii) Salmon Alevin, una eina amb moltes característiques interessants, (iii) Kallisto Bustools, una eina centrada en l'eficiència computacional i (iv) STARSolo, una implementació recent sobre un *aligner* popular. He dissenyat 4 pipelines en bash per implementar cada eina i he implementat un anàlisi *downstream* per a avaluar la significació biològica dels resultats. He comparat la velocitat i l'eficiència computacional, les *count matrix* produïdes i els resultats biològics. Per a les comparacions, he utilitzat els conjunts de dades proporcionats per Tabula Muris com a *ground truth*. He trobat que Kallisto Bustools és significativament més ràpid i eficient, mentre que UMI Tools és l'eina més lenta. Les *count matrix* produïdes per UMI Tools i Kallisto Bustools han estat coherents amb la *ground truth*, mentre que les produïdes per Salmon Alevin i STARSolo presentaven inconsistències. Els resultats biològics han sigut coherents, tot i que Salmon Alevin ha presentat problemes. Concloc que el preprocessat de dades de scRNA-Seq ha progressat en els darrers anys, però més en eficiència computacional. Kallisto Bustools es l'eina més ràpida i consistent entre les avaluades.

Abstract

Single cell RNA sequencing (scRNA-Seq) is a very powerful tool to study the transcriptome. Several tools are available to pre-process the data it produces.

However, no standard exists. The objective of this work is to compare four scRNA-Seq pre-processing tools and extract conclusions regarding the advancements over recent years, and to determine the superior tool. The tools compared were (i) UMI Tools, one of the first published tools, which served as reference point, (ii) Salmon Alevin, a tool with many interesting features, (iii) Kallisto Bustools, a tool focused on computational efficiency and (iv) STARSolo, a recent implementation to a popular aligner. I designed 4 pipelines in bash to implement each tool, and I implemented a downstream analysis to evaluate the biological significance of the results. I compared the computational speed and efficiency, the count matrices produced and the biological results. For the comparisons, I used the datasets provided by Tabula Muris as ground truth. I found that Kallisto Bustools was significantly more efficient and faster, while UMI Tools was the slowest tool. The count matrices produced were consistent with the ground truth for UMI Tools and Kallisto Bustools while Salmon Alevin and STARSolo presented inconsistencies. The biological results were coherent, although Salmon Alevin showed problematics. I concluded that scRNA-Seq has progressed in recent years but more so in computational efficiency. Kallisto Bustools was the fastest and most consistent tool among those evaluated.

Contents

1. Introduction

1.1. Context and justification of the thesis

The central dogma of molecular biology states that genes encoded in DNA are transcribed into messenger ribonucleic acid (RNA) which in turn is translated into proteins. However, not every gene is transcribed in the same frequency and not all transcripts are translated into proteins. This system comprised by all RNA molecules makes up the transcriptome which is the first instance of the phenotype of a cell. The expression profile is different in each cell type and changes with the cell cycle. This, in turn, will affect the proteins that are synthesized which will be tailored for the functions and needs of the cell. Therefore, studying the transcriptome at a cellular level is key to understand the role of each cell type and, in consequence, it is necessary to develop techniques to obtain transcriptomic data as precisely as possible.

In the past two decades, significant progress has been made in transcriptomic technologies. RNA microarrays, capable of detecting thousands of transcripts, were developed as early as 1995 [1]. However, these arrays are limited because they only detect a finite number of transcripts that must be present as probes. Recently, Next Generation Sequencing (NGS) techniques have become more cost-effective and are now widely recognized as the primary method for genomic expression analysis. RNA sequencing (RNA-Seq), which involves sequencing reverse transcribed RNA samples, has replaced microarrays as the preferred method for gene expression analysis [2], [3].

The samples of traditional RNA-Seq, however, are obtained from bulk tissue, without discriminating its cell population. During the last decade, significant technological advancements have been made in the with the goal of obtaining RNA sequencing data at a cell resolution thus developing the technology known as single-cell RNA Sequencing (scRNA-Seq). Single-cell RNA sequencing now enables the analysis of transcriptomes of millions of cells in a single study at the single-cell level. This ability to classify, characterize, and distinguish cells at the transcriptome level has enabled the identification of rare cell populations that are functionally important [4].

The raw output of these techniques are sequencing files (FASTQ) that contain millions of sequences (reads) of transcripts. Each read contains a nucleotide sequence that encodes a barcode and a Unique Molecular Identifier (UMI). The barcode associates the read with a cell, while the UMI associates the read with an RNA molecule that existed before amplification. These files cannot be analysed directly. It is necessary to pre-process the raw data to produce a

counts matrix, which contains the number of transcripts for each gene in each cell. With a counts matrix, we can then study the transcriptomic profile of our cell sample.

The pre-processing step can be computationally intensive, depending on the size of the sample. In consequence, a variety of specialized tools to carry and optimize this task have been developed. Despite that, there is still no procedure considered as a standard. Consequently, it is interesting to compare the different available tools to find the one that best suits our needs, computational resources, and specifications of the experiment.

1.2. Objectives

Main objectives:

1. Compare the performance of four single-cell RNA Sequencing data pre-processing pipelines, each built through different available software tools.
2. Extract conclusions about the improvement in scRNA-Seq data pre-processing over the last years, and determine which tool is the superior option or in which circumstances each tool performs better.

Specific objectives:

1. Design, in bash, scRNA-Seq data pre-processing pipelines with the tools: UMI-Tools/STAR, STARsolo, Kallisto|Bustools and SalmonAlevin.
2. Design an R Markdown notebook to perform the downstream analysis: quality control, dimensional reduction, clusterization and marker gene obtention.
3. Run the pipelines and notebook with a dataset well characterized and compare the performance and results obtained by each tool.

1.3. Impact in sustainability, ethics, and diversity

The impact of this work on environmental sustainability and ecological footprint is negligible or non-existent. As for energy consumption, it is not in the scope of this work to compare consumption. The computations were carried in a personal computer with a commercial CPU. Therefore, it is not a relevant subject because the energy used for computation is negligible. The significance of computational efficiency remains crucial in the field of omics, as it enables operations to be executed on personal computers and optimizes the utilization of high-performance computer clusters. However, pre-processing is not meant to be computed perpetually nor in extreme volumes of data, and, therefore, its

energy consumption is not an issue. Regarding material consumption, waste, and pollution, since this work is focused on comparing computational processes, there is no direct generation of waste or pollution. As for legislation and regulations, there are none that directly affect this topic. The data used comes from mouses (*Mus musculus*), but it is publicly available and no experiments, on animals or otherwise, were carried explicitly for this work. This work does not directly impact any of the Sustained Development Goals (SDG) related to environmental sustainability.

Just as well, the impact of this work on ethical-social aspects is negligible. The objectives focus on the evaluation of technical methodologies, rather than addressing broader societal or ethical concerns. This work, has the potential to contribute to advancements in applications of single-cell RNA sequencing, which could have implications for areas such as healthcare, precision medicine, and biological research. These areas align with SDG 3 (Good Health and Well-being). However, this work is unlikely to be key or to have any direct impact. Therefore, it is not reasonable to attach responsibility in addressing these issues. In a similar line, the gender and race dimensions of society have no impact in the research of this work. Data is from mouse, and the data is not the focus of study and thus gender is not relevant.

1.4. Approach and methodology

Regarding the choice of tools of pre-processing software, I have chosen UMI-Tools because it was one of the first publicly available tools and should serve as a contrast against the other more vanguard tools [5]. Second, STARsolo is an integration with the STAR aligner for single cell analysis [6]. These two tools require mapping the reads to the genome which is a computationally intense process. On the other hand, Kallisto Bustools uses pseudomapping [7]. Instead of aligning reads to a genome, a pseudomapping algorithm identifies which transcripts are compatible with each read to quantify the transcript, resulting in a faster process. Last, Salmon Alevin, uses a lightweight algorithm that the authors claim is faster than traditional aligners [8].

I have written four pre-processing pipelines in bash, meant to be executed in terminal. The inputs for the pipelines are: The raw reads in fastq files, a genome index for the alignment and, depending on the tool, other files like a transcript to gene list, the genome sequence, or an annotation file. The main output is the counts matrix, which is a matrix describing the number of transcripts found for each gene and cell. Obtaining this file from the raw reads is what is known as pre-processing and is the focus of this work. Along with it, the pipelines provide: The list of genes and of cell barcodes as separate files, a file containing the logs printed in terminal, and a file that tracks execution times and resource usage.

To evaluate the pre-processing, it is necessary to analyse the counts matrix, in a process known as downstream analysis. For this I have used the Seurat library to write an RMarkdown file in R [9]. This type of file enables code execution in separate chunks, which is convinient for some steps. It also allows writing in Markdown between these chunks to document the analysis. Additionally, it allows "knitting" the document together with the output of each chunk into an HTML or PDF file. The downstream analysis I have implemented consists in quality control, dimensional reduction, clusterization and differential expression analysis to find marker genes. A workflow chart of the methodology is provided in Figure 1.

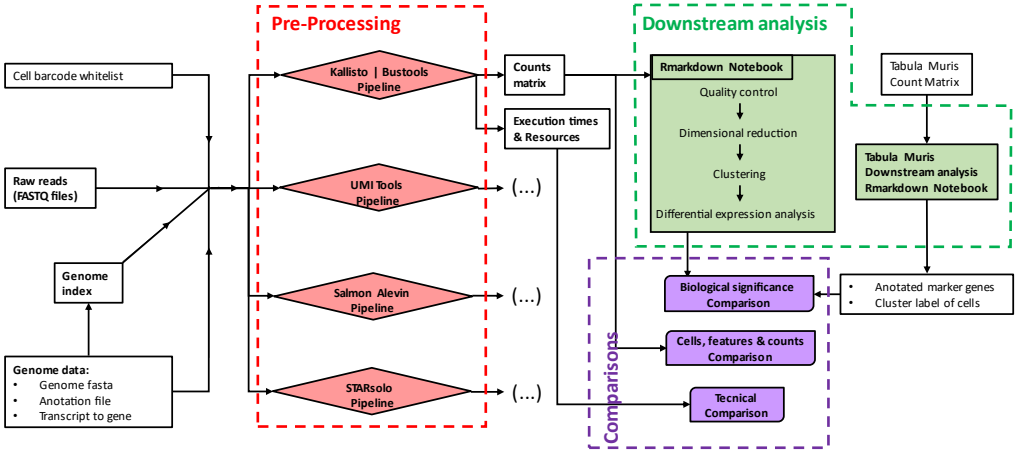

Figure 1: Workflow of the approach taken. In white are files needed or produced, in red are the pre-processing pipelines, in green the downstream notebooks and in purple the comparisons. Note that the right-side is repeated for each pre-processing tool.

I have compared the pre-processing tools on three different accounts. First, in terms of computational efficiency by comparing the execution times and the resources used. Second, by evaluating the number of genes and cells obtained. And lastly, by comparing the biological significance of the obtained results. For the first comparison, execution times and resources used can be compared directly. For the second and third comparisons, the results need to be compared with a ground truth. For this I have chosen a dataset provided by the Tabula Muris project, a compendium of single cell transcriptome data from the model organism Mus musculus [10]. Tabula Muris provides both the raw reads and already pre-processed counts matrices from datasets originated from different tissues. For the second comparison, the Tabula Muris matrix provides the expected number of cells and genes for each dataset. For the comparison of the biological significance of the results, I have written a separate RMarkdown notebook. It performs the same downstream analysis on the counts matrix provided by Tabula Muris and saves key results to files. Hence, I can compare the results obtained from the ground truth to the results obtained with my count matrices. I have taken two different approaches for these comparisons. First is

to compare if the marker genes of a given cluster match, together, with the marker genes of a cluster from the ground truth. Consequently, we can infer that we have identified a set of cells with a function matching that of another set in the ground truth. The second comparison is checking the proportion of cells from a cluster that are found together in the same cluster of the ground truth. From these comparison, we can infer that the features that classify the dataset are kept through the pre-processing.

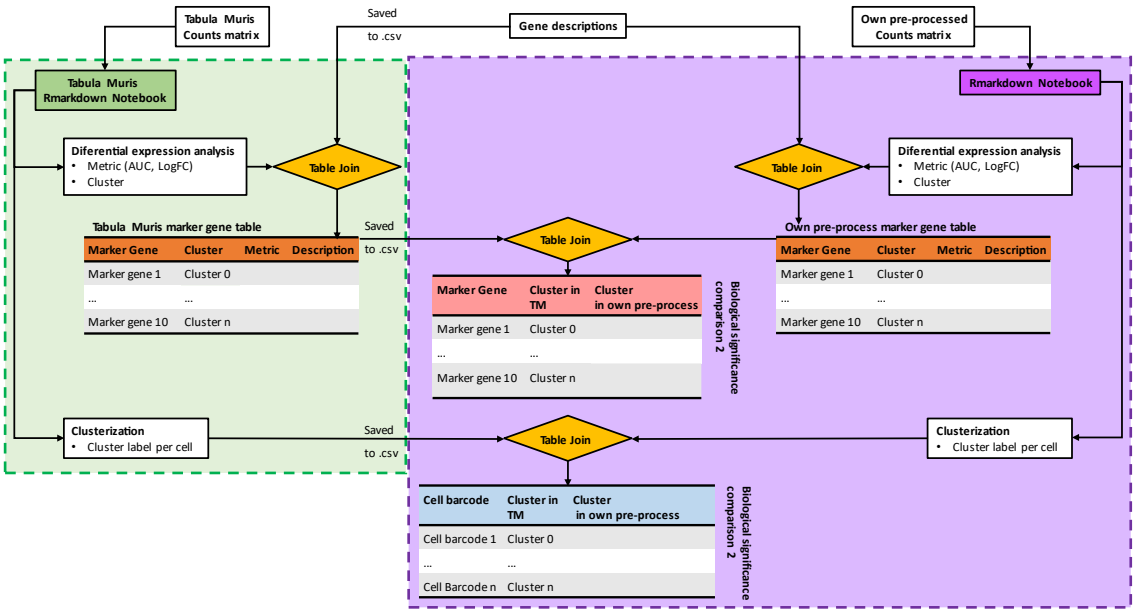

Figure 2: Workflow chart for the biological comparisons. Computations in purple are done in the main RMarkdown and computations in green happen in the RMarkdown designed for Tabula Muris matrices.

Finally, to keep version control and for code availability I used git and I published all the code in my GitHub.

1.5. Planification of the project

The tasks of this project were: To write the pre-processing pipelines, to write the downstream analysis, to find a suitable dataset that served as ground truth, implement the comparisons of the different aspects of the pipeline, and execute the pipelines and analysis.

In a first phase, I learnt how to use UMI Tools and Kallisto Bustools, I built the genome indices for Kallisto and STAR and I wrote pipelines. Then, I designed the downstream analysis before I proceeded with the Salmon Alevin and STARSolo pipelines. Next, I decided to use the Tabula Muris dataset. I designed the biological comparison section of the notebook, and I created the second notebook to analyse the ground truth. At this point I invested time into polishing the pipeline: I added error codes, getopts parameters, help messages, code commentary, and other improvements to ease-of-use and reproducibility.

As part of these improvements, I started using Git for version control and I uploaded the project to GitHub. Next, before the final execution, I fine-tuned the parameters of both pre-processing and downstream analysis. I changed the metric used to identify marker genes and I changed the identification of barcodes for certain pipelines. Finally, I executed the code on its last version with all datasets to produce the results.

1.6. Summary of obtained products

The products obtained are the code I have written, and the data obtained, along with the GitHub page and this memory. The code is divided between the pre-processing bash pipelines and the RMarkdown notebooks for the downstream analysis. The pipelines are four scripts written in bash and a module file that stores utility functions. The purpose of these scripts is to pre-process single-cell RNA sequencing raw data into a cell by gene counts matrix. The downstream notebooks are RMarkdown files, with its code chunks written in R. Their purpose is to perform the downstream analysis on the counts matrices. The data obtained is comprised by the count matrices, the records of the pre-processing performance and the reports produced by the notebooks.

1.7. Brief description of the chapters

State of the art: I describe the current methods to perform single cell RNA sequencing and I describe the relevant methods to process and analyse the data.

Materials and methods: In materials I describe the hardware, software, and data that I used in this work. In methods I describe the process to obtain the results and I describe, step by step, the workflow of each pipeline and of the downstream notebooks.

Results: Here I present the comparisons among the pre-processing tools. I compare the computational speed and efficiency, the raw and filtered count matrices obtained and the biological significance of the analysed results against the ground truth.

Conclusions: Summary and discussion of the obtained results, evaluation of the success in the objectives and possible future lines of work.

2. State of the art

2.1. Cell isolation and RNA sequencing

Every single-cell RNA-seq protocol is based in four fundamental stages: (i) individual cell isolation, (ii) mRNA retro transcription (RT), (iii) cDNA amplification, and (iv) preparation of next-generation sequencing libraries. Methods for obtaining DNA libraries from the transcriptome of single cells, were pioneered by Brady and colleagues in 1990 and Eberwine and colleagues in 1992, as methods of DNA amplification had recently become available [11], [12]. The former, isolated a cell on which they performed retro-transcription and amplification, while the latter microinjected the necessary enzymes into an isolated neuron. Initially, these libraries were then analyzed through microarray chips [13]. Later, as early as 2009, the methods were adapted to be used with next generation sequencing methods [14].

These methods, however, rely on isolating cells through high dilutions or micromanipulation with specialized pipettes. As such, they have a very low throughput and are very time-consuming. A common solution is Fluorescence Activated Cell Sorting (FACS), which allows to record phenotype data of each cell while also separating them into wells onto which the rest of stages can be performed [15], [16]. This method requires a higher volume of cells but offers higher sensitivity, and the phenotype data it gathers can be useful for several purposes [17].

Another predominant technology is nanodroplet encapsulation, developed and provided commercially by 10x Genomics Inc. The key step of this technology is the encapsulation of isolated cells in gel beads. The gel beads contain oligonucleotides and primers while the necessary enzymes are added along the cells. After encapsulation cell lysis and retro transcription happen and library preparation can start. The final library structure depends on the brand of 10x Genomics protocol and the purpose of the experiment, although the constituents are common. Libraries will always include: (i) adapters for paired end sequencing, (ii) two sequencing primers, one per each read, (iii) a cell barcode, unique to the cell, (iv) a unique molecular identifier (UMI), unique to each transcript, (v) the transcript sequence (vi) a poly-T sequence, which hybridizes with the poly-A sequence added by the reverse transcriptase and (vii) a sample barcode, to allow running more than one sample. The specific library structure of the 10x protocol used to obtain the data of this work is described in Figure 3. Once the library is ready, it is amplified and sequenced [18], [19].

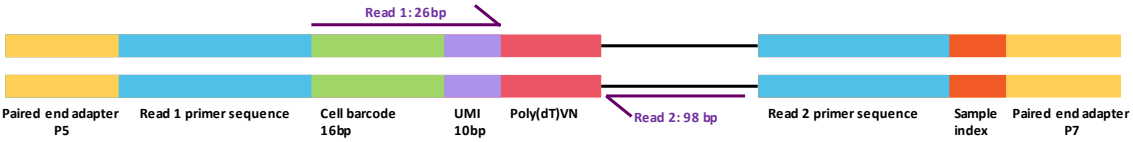

Figure 3: Specific library structure used in the data of this work.

The product of this protocol is two fastq files: One storing the UMI and barcode, read 1, and another storing the read itself, read 2. From this point, the pre-processing can start with the objective of obtaining the counts matrix from these files.

2.2. Pre-processing

All existing single cell RNA sequencing pre-processing methods, generally, follow a common structure. The first step is identifying the cell barcodes of the sample, usually saving them to a whitelist file. Second is the deduplication of all reads with the same UMI. Since all molecules with a shared UMI will have originated in amplification from a single transcript, only one transcript should be counted for that gene. The next process is assigning each read to the gene it was transcribed from. The most common method for this step is aligning the reads to a genome through mapping algorithms, as with bulk RNA sequencing. The last step is to quantify the gene counts per cell in a matrix.

In 2017 Smith T. and colleagues published UMI-tools, an open-source software tool written in python that performed deduplication of scRNA-Seq data, although it does not perform mapping and instead relies in other software [5]. In their work, the authors demonstrated that miscallings in the sequencing of UMIs are common, which leads to false UMIs being counted as genuine. The UMI-tools deduplication function can correct these errors by building networks of UMIs of the same genetic loci. Similarly, UMI-tools can correct miscallings in cell barcodes to ensure that no artificial cells are counted. UMI-tools has been used extensively and the publication has over a thousand citations [5], [20]. Many other tools developed ideas presented by UMI-tools and used it for benchmarking [21], [22].

Along with droplet-based technologies, 10X Genomics also developed Cell Ranger, a tool to preprocess scRNA-Seq data [18]. Cell Ranger identifies UMIs and barcodes by correcting both to a hamming distance of one and uses the STAR aligner. Its advantages are that the pipeline can be run through a single integrated function, its ease of use and that it is well maintained by 10X Genomics. However, it can only process sequences obtained through 10X Genomics protocols and it is not as fast as other available tools.

In 2019, Srivastava A. and colleagues published Alevin an open-source software tool written in C++ that performed all pre-processing steps [8]. It works within the Salmon framework, which is a tool developed for bulk RNA sequencing. It integrates whitelisting, mapping, deduplication and counting [23]. Salmon utilizes a lightweight mapping algorithm that the authors claim is much faster than other aligners. The authors of alevin implemented a novel deduplication algorithm that uses transcript level information and that considers multimapping reads to identify the correct UMIs. Transcript level information such as isoforms is kept by building equivalence classes using the method described by Turro E. and colleagues in 2011 [24]. Using these classes, reads with similar UMIs that map to the same gene may not be collapsed if the transcripts are distinct. Also, on deduplication, the authors identified that other tools discarded reads that map to more than one gene (multimapped) and proved that this approach lost information. In Alevin's algorithm these reads are kept, and their counts are distributed through an expectation maximization (EM) algorithm [25].

In 2021, Melsted P and collegues presented the Kallisto Bustools pipeline, which is focuses on computational speed and efficiency [7]. This pipeline uses Kallisto to assign reads to genes, but instead of aligning the reads traditionally, Kallisto uses pseudo-alignment [26]. Pseudoalignment is a technology developed with the objective of reducing the computational resources that traditional alignment needs. Instead of aligning all bases of a sequence to a reference, it breaks the read into "k-mers" (sub-sequences of length k) and finds which transcripts it is compatible with. Then, the algorithm builds a De Brujin graph of the compatible transcripts and keeps those that are valid for all k-mers as an equivalence class. The key to the efficiency of this algorithm is that it does not care about read orientation nor exact per base alignment. Furthermore, it can skip the evaluation of most k-mers since the authors show that two are often enough to determine the original transcript. Thus, Kallisto outputs a "Barcode, UMI, Set" (bus) file that contains the UMI, barcode and equivalence class of each read. The authors developed both the file type and Bustools, a software meant to manipulate bus files [27]. Through Bustools, then, the counts of multimapped reads are distributed with EM. Furthering the focus on computational speed, the authors of alevin deem the miscallings in UMIs to be negligible, only correcting those that are a Hamming distance of 1 away. Kallisto Bustools has shown to be a very efficient tool and has already been used in multiple studies [28], [29].

STARsolo is an integration to the STAR aligner developed to quantify single cell data [6]. The design is meant to be a "drop-in" replacement for CellRanger. It claims to be the fastest tool among those that have a higher accuracy and presents a tight pipeline that does not use intermediary files.

3. Materials and methods

3.3. Materials

3.3.1. Hardware

The hardware that I used in this work is my personal computer with the following specifications:

- CPU: AMD Ryzen 5 2600X Six-Core Processor
- RAM: 16 GB DDR4 + 40GB of swap RAM
- Disk: WD Blue SN570 500GB
- Storage: ST1000DM010-2EP102 - 1TB

3.3.2. Software

The operating system used was Ubuntu 22.04.1 LTS with release 5.1.16(1) of bash. The software tools used for pre-processing were:

- UMI tools: umi_tools 1.1.4, STAR 2.7.10a, samtools 1.16.1, featureCounts 2.0.3
- Salmon Alevin: Salmon 1.9.0
- Kallisto Bustools: kb_python 0.27.3, bustools 0.39.3
- STARsolo: STAR 2.7.10a

Other tools used in designing the pipelines include time and getopts from GNU.

For the downstream analysis, I used R 4.2.2 on RStudio 2022.07.02. The libraries used are:

- dplyr v1.0.10
- dropletUtils v1.16.0
- ggplot2 v3.4.0
- knitr v1.40
- Matrix v1.5-1
- Seurat v4.3.0
- tidyverse v1.3.2

Also, to obtain certain gene lists I used biomaRt v2.52.0.

3.4. Methods

3.4.1. Data obtention

All data utilized was originated from the Tabula Muris Project. Tabula Muris provides the already pre-processed data in a repository, available in their website [30]. The count matrices generated by Tabula Muris can be found in this repository, along with the corresponding R objects that can be loaded into R to obtain the downstream results obtained by Tabula Muris. The raw data, however, is only available through the Sequence Read Archive (SRA), under Gene Expression Omnibus (GEO) series accession GSE109774. The data in this study is categorized into various sample formats, and for this particular work, the selected formats were GEO accessions GSM3040906 (derived from lung tissue) and GSM3040917 (derived from trachea tissue). Data is also available in various formats, and the one downloaded was the original 10X Genomics bam file. Once downloaded, bedtools provides a tool, “bamtofastq”, that allowed me to obtain the original FASTQ files. These files are divided into read 1 (UMI and barcode), read 2 (real read) and indexes 1 and 2.

Other external data needed was the reference genome of *Mus musculus* (assembly GRCm39 or mm39), gene annotation files (GRCm39 from ensemble and M32 from Gencode) and lists obtained through BioMart (ensemble id of mouse mitochondrial genes, transcript to gene id associations and gene description). The whitelist, which is a list of the barcodes used in the experimental part, is identical for all experiments done with the same version of the chemistry. In our case it is the whitelist used for V2 chemistry. It is available for download in the github of Cell Ranger.

3.4.2. Genome indexes

Mouse (*Mus* musculus) genome indexes were built for each aligner, STAR, Kallisto and Salmon. Indexes are generated with built-in functions of the aligners and are one-time processes since the same index can be used for every execution of an aligner. The methods used to generate the indexes were those provided in the documentation of the aligners. STAR and kallisto used the genome fasta sequence and the annotation file from Ensembl (GRCm39). Kallisto also used the transcript to gene dictionary. To build the Salmon index I used the approach of using the entire genome as a decoy sequence, also following the documentation. It used both the transcriptome the genome from Gencode (M32).

3.4.3. Pre-processing

All pre-processing pipelines are written in bash and are meant to be executed as:

```
$ bash <path-to-script> [options]
```

There are parts of the code that are common to all pipelines. As a first step, all pipelines load these functions, described latter, that perform common processes. Then, using getopts, the script stores the required parameters in variables. These parameters are the paths to various files, including the raw data directory, aligner-specific index files, the barcode whitelist and the transcript-to-gene mapping file. Also, the path where the user wants to store the output files must be indicated. If any necessary option is not provided, the script then returns a help message and indicates to the user what is missing. Likewise, the help message is shown if a file or directory at the indicated paths is missing.

After saving the paths, all pipelines run a function called merge. This function first asks for the number of files. If its four, then they correspond to read 1, read 2, index 1 and index 2. The function then saves the path to reads 1 and 2 in two variables. If more than 4 files are found in the directory, it means that reads are separated into lanes. If that is the case, the files of reads 1 (R1) and 2 (R2) are concatenated by category. The resulting files are saved in the output directory and their path is saved to variables. These files will be the ones used in the rest of the functions.

After this step, the pipelines differ from each other. The Kallisto Bustools requires a transcript to gene file, a Kallisto index file and the cell isolation technology name, as well as the raw reads. After that, the files are directed to “kb count” a wrapper function provided by the authors. It performs sequentially “kallisto bus”, “bustools correct”, “bustools sort” and “bustools count”. This function produces de counts matrix directly.

The STARsolo pipeline takes the reads, the STAR index directory and the whitelist. After merging, the pipeline unzips the FASTQ files. Then the inputs are passed to STAR, with the parameter “`--soloType`" set to “`CB_UMI_Simple`”. The number of threads I used for this function is 6. It also produces the counts matrix directly.

The Salmon Alevin script requires the transcript to gene file, the salmon index file and the whitelist. After that, the files are directed to “salmon alevin” which performs the pre-processing and saves a counts matrix.

Last, the UMI tools pipeline takes the STAR index directory, the transcript to gene file and the gene annotation file. First, "`umi_tools whitelist`" takes the R1 file and the structure of the UMI and barcode and it finds the valid barcodes of the dataset, saving them to a new whitelist file. The rest of the pipeline can use a whitelist provided externally, but the UMI correction methodology of UMI tools requires a specific format. I have chosen to use this function to obtain the valid barcodes to test this feature. After that, the reads and the whitelist are passed to "`umi_tools extract`" to append the barcode to the read names, which is necessary for subsequent steps. Then, the extracted reads along with the STAR index are used to align the reads to the genome with STAR. I used 6 cores to run this function and I discarded all multimapping reads with the flag "`--outFilterMultimapNmax 1`". After this step we need to assign each read to a gene because we aligned the reads to the genome and because fragmentation happens after PCR amplification. This means that two reads with different mapping locations may still be duplicates. To do this I used `featureCounts`, from the subread package, which returns a bam file with the gene assignment in the XS tag. After using `featureCounts`, however, the reads are not sorted which is needed for quantification. To sort the bam file this I used `samtools sort` followed by `samtools index`, which take the bam file as input and return it sorted. This bam file can then be passed to `umi_tools count`, which deduplicates and quantifies the transcripts by gene and cell (using `--per-gene` and `--per-cell`).

All the functions used that are related to the pre-processing, were benchmarked and timed using `time -v -a -o` and the output was saved to a text file for latter comparison. Additionally, I redirected the output printed to terminal to a text file to help with debugging and development.

I executed the pipelines with both the dataset of trachea cells and of lung cells.

3.4.4. Downstream analysis

Downstream analysis was written in R into an RMarkdown notebook with the purpose of performing the downstream analysis and comparing the biological significance of each counts matrix. I designed two notebooks: One to perform the analysis on the count matrices provided by Tabula Muris and save its results to files, and another to perform the analysis on the count matrices that I obtained. The notebooks take as inputs the count matrices, a list of the ensemble IDs of mitochondrial genes and, in some cases, a list of genes and barcodes.

The notebook starts by loading all libraries, saving the path to the count matrix to be analysed and saving the tool used to pre-process it as a pre-determined string. The second part is loading the counts matrix to R as a sparse matrix. As the output formats vary slightly, I used conditionals to be able to use the correct

functions to load the data. Next, mitochondrial genes needed a tag on their name so that I could use the proportion of mitochondrial transcripts as a quality control measure. To do so, I loaded the mitochondrial gene list and I iterated over the present genes to add “MT-“ before the corresponding ensembl ID in the sparse matrix. After this, the sparse matrix is used to create a Seurat object. While creating the object, I filtered: Genes that appear in less than 5 cells and cells that have less than 5 genes. This is a small filter that removes data points that would be remover later and speeds up computations.

The first part of the analysis itself is the quality control. I have applied quality control by setting thresholds on certain per-cell metrics: Number of unique genes, total count of molecules and percentage of mitochondrial cells. The percentage of mitochondrial transcripts was set to less than 5% in all cases. To choose the other thresholds, I plotted the metrics as violin plots as well as the number of molecules against the number of genes in a dispersion graph. The thresholds were set, then, such that no obvious outliers remained after the plot and such that the dispersion graph was approximately linear.

The second part of the analysis was normalizing the data, selecting the most variable genes, and scaling the data. The purpose of normalization is so that the expression values of cells can be compared among themselves. Each value (expression A of gene i and cell j is divided by the total expression in its cell, multiplied by a scale factor and then log transformed (Equation 1). After normalizing, we select the most variable genes with the purpose of focusing the analysis on the genes that differentiate the cells. The metric used for this purpose is the standardized log dispersion, as described in Equation 2. Following Tabula Muris approach, I selected as highly variable those genes that had: $d_i > 0.1 \text{ AND } m_i > 0.5$. To end the second part of the analysis, I scaled the data so that the mean across cells would be 0 and the variance across cells 1. The formula applied is that of Equation 3. While scaling the data the percentage of mitochondrial counts is regressed out, to minimize its impact.

Equation 1: Scaling of expression value A from gene i and cell j into normalized expression value N.

$$N_{ij} = \log\left(1 + 10^4 \frac{A_{ij}}{\sum_j A_{ij}}\right)$$

Equation 2: Standarized log dispersion of gene i with mean m and variance v.

$$d_i = \log\left(\frac{v_i}{m_i}\right)$$

Equation 3: Scaled value X of gene i in cell j with normalized value N,and cell mean and standard deviation m and s

$$X_{ij} = (N_{ij} - m_i)/s_i$$

The next section carries out the dimensional reduction and clustering of cells. I performed dimensional reduction through principal component analysis. Cells have thousands of genes expressed so I needed to reduce the dimensions to proceed with the rest of the analysis while keeping as much variance as possible. After computing the components, I plotted the cumulative variance and, in each analysis, I selected the number of components before the elbow. With the components, then, I clustered the cells through shared nearest neighbours. Crucially, the cluster label of each cell is saved in the metadata of the Seurat object. I also computed the t-distributed stochastic neighbour embedding dimensional reduction from the principal components to mimic the graphics obtained by Tabula Muris.

The last part of the downstream analysis is the differential expression analysis to identify the marker genes of each cluster. The metric I used is the area under the curve (AUC). To calculate it, a Seurat function, `FindAllMarkers`, constructs a classifier for every pairing of gene and cluster. This classifier uses the gene as the sole predictor and differentiates between the cluster and the remaining cells as the two distinct classes. To evaluate each classifier, the function computes the area under the receiver operating characteristic curve (ROC). This curve built from the true positive rate against the false positive rate. Essentially, it measures how well a gene discriminates between the cells of a cluster and the rest of the cells. An AUC value of 1 means that expression values for this gene alone can perfectly classify the two groupings. This means that each of the cells in the evaluated cluster exhibit a higher expression level of the evaluated gene than in the cells of the rest of clusters. An AUC value of 0 also signifies perfect classification. However, I only evaluated genes that are more expressed than average in the cluster, as I want to find marker genes and not build a classifier. The function kept all marker genes with an AUC of over 0.7. However, I kept the top 10 markers of each cluster.

The analysis was identical in both notebooks with a few exceptions. The notebook designed to analyse the data from Tabula Muris takes as input an robj file, which contains the counts matrix in a seuratObject. These objects are in Seurat's version 3 of the class while the version I used is the number 4. As such the objects had to be updated. Also, the objects contain data from two different samples of the same tissue, therefore, only the data of the sample I used in the pre-processing was used. Another difference in the notebooks is that the main notebook carries out the comparison while the one designed for Tabula muris data saves key data for the biological comparison. The data saved, to a text file, was the cluster identity of each cell, and a table that detailed the top 10 marker genes of each cluster. The latter included: (i) The name of the marker gene, (ii) the cluster it was a marker gene for, (iii) the AUC metric, (iv) the log fold change between its expression and the mean and (v) the description of the gene

provided by Mouse Genome Informatics (MGI). Hence, I produce two of each of these files, one for each dataset.

3.4.5. Comparisons

To directly compare the results of each pre-processing I compared, among the pipelines: (i) The number of barcodes identified as correct, (ii) the total number of counts, (iii) the mean and median number of counts per barcode, (iv) the mean and median number of genes per barcode. And again, the same values, after applying the quality control filters.

I based the biological evaluation of the pipeline in comparisons between the results of the downstream analysis and the same analysis applied to the Tabula Muris-provided counts matrix. To compare the clusters directly, I computed the number of cells from a given cluster that coincide in identity in both analyses.

In another biological comparison, I first produced a table of the top marker genes as described in the las section of the downstream analysis and I loaded the corresponding file. Then, I inner-joined the tables by description, so that genes that were found to be marker genes in both analyses were joined. The information provided by this resulting table is how many, out of 10 marker genes per cluster, are common in both analyses.

4. Results

4.1. Computational speed and efficiency

I benchmarked the time each pipeline needed to run with GNU's time function. I measured the real time elapsed and the time the processes spent in the CPU, both in user and kernel mode. The time spent in CPU is in every case higher than the time elapsed because of parallelization, where a process can be undertaken by more than one CPU core at the same time. This way the time needed to execute the full pipeline in a single core is the user time plus the kernel time. To measure the grade of parallelization, we can calculate the percentage of time spent in CPU of the time elapsed to execute with Equation 4.

Equation 4: Formula of the percentage of CPU used by a pipeline. Used to measure parallelization.

$$\%\ \text{of CPU used} = \frac{User\ time + Kernel\ time}{Elapsed\ time} * 100$$

Since the time spent in CPU is higher than the elapsed time, this value indicates if a pipeline uses the available resources efficiently. This is heavily affected by the number of cores used, which can be set as an argument for all pipelines except UMI Tools. However, using a higher number of cores requires loading more data into RAM, which was the limiting factor in the STARSolo and Salmon Alevin pipelines. The number of cores, the maximum memory load used and the rest of the data is presented in Table 1, for the trachea dataset and Table 4 for the lung dataset.

Table 1: Table 2: Technical benchmark results of the preprocessing of the lung cell data set.

	Umi Tools	STARSolo	Kallisto Bustools	Salmon Alevin
Elapsed (hh:mm:ss)	02:08:50	00:28:21	14:32.0	00:39:39
User time (hh:mm:ss)	03:33:06	02:44:56	01:26:28	01:17:12
Kernel time (hh:mm:ss)	00:02:22	00:01:38	00:00:32	00:00:41
% of CPU used	% 167,1	% 587,6	% 598.6	% 196,4
Maximum memory used (Gb)	14,70	11,55	4,29	14,26
Cores	1 (6 for STAR)	6	8	4

As expected, UMI tools was the slower pipeline, taking over 2 hours to preprocess the trachea dataset, more than twice as long as next slower pipeline. There are several reasons for this. First, being written in python is a handicap when compared to other, more efficient, languages. Second it is fully modular and relies in intermediary files: `umi_tools extract` produces intermediary FASTQ files equal in length to the inputs, and `STAR`, `featureCounts`,

`samtools sort` and `umi_tools count` each produce an intermediary bam file. Lastly, the UMI Tools pipeline has no parallelization except for the read mapping which is done with STAR and for the read assignment. A detailed per-function summary of the pipeline is presented in Table 3.

Table 3: Per-function breakdown of the UMI Tools pipeline

	Trachea dataset		Lung Dataset	
Function	**Elapsed**	**CPU time**	**Elapsed**	**CPU time**
`umi_tools whitelist`	00:25:14	00:25:14	00:15:19	00:15:19
`umi_tools extract`	00:53:52	00:53:52	01:59:10	01:59:07
`STAR`	00:24:32	01:46:39	00:44:29	04:22:23
`featureCounts`	00:00:31	00:05:03	0:01:32	00:15:43
`samtools sort`	00:10:31	00:10:30	00:30:55	00:30:50
`samtools index`	00:00:54	00:00:54	00:02:22	00:02:22
`umi_tools count`	00:13:16	00:13:16	0:25:15	00:25:15

The elapsed time and the CPU time are almost identical in every function, which means that it had no parallelization. The exception is the STAR aligner that used 434% and 589% of the CPU respectively. This allowed the pipeline to finish 1h22min and 3h37min faster. The read assignment with featureCounts also had a high parallelization of 977% and 1020%, which is possible because it is a lightweight tool that could use as much memory as it needed. However, it only accelerated the process by 4 min and 14 min respectively.

Table 4: Technical benchmark results of the preprocessing of the lung cell data set.

	Umi Tools	STARSolo	Kallisto Bustools	Salmon Alevin
Elapsed (hh:mm:ss)	3:59:02	0:55:56	00:10:30	01:20:24
User time (hh:mm:ss)	07:48:13	05:25:46	00:39:34	02:36:12
Kernel time (hh:mm:ss)	00:02:47	00:02:50	00:00:45	00:00:39
% of CPU used	% 197,0	% 587,5	% 383,9	% 195,1
Maximum memory used (Gb)	11,49	11,62	4,20	14,51
Cores	1 (6 for STAR)	6	8	4

Also expected was that Kallisto Bustools was the fastest pipeline and the one that used the least memory. It finished the process in 14 min and 10 min for the trachea and lung datasets respectively and used only 4 Gb. I was able to run this pipeline with any number of cores and, as such, it had the highest parallelization. The next fastest tool, STARSolo took twice as much time for the trachea dataset and 5.5 times more for the lung dataset. STARSolo and Salmon Alevin were close to each other, with the former being 10 to 20 minutes faster and the latter using less memory. However, STARSolo required over twice the

amount of CPU time, as it relies on traditional genome aligning. Despite this, it still outperformed Salmon Alevin with a 587% of CPU usage.

4.2. Count Matrices

To evaluate the result produced by the pipelines I compared the number of transcripts, cells, and genes of each count matrix, including those provided by Tabula Muris. I also show the number of barcodes that passed the quality control filters that also passed the quality control filters in the Tabula Muris analysis. The data is presented in Table 5 and Table 6. The quality control filters applied were: (i) More than 2000 molecules per cell and less than 12000 molecules per cell (ii) more than 500 genes per cell and less than 3700 genes per cell (iii) less than 5% of counts of mitochondrial origin.

Table 5: Number of counts, cells and genes in the count matrices obtained from the trachea dataset., before and after the application of filters. Also shown, the number of cells after filtering that coincide by barcode with the Tabula Muris analysis.

		TabulaMuris	UMITools	KallistoBustools	Salmon alevin	STARSolo
Unfiltered	Counts	22,334,002	23,855,028	28,364,583	1,897,657	8,020,154
	Cells	4,643	11,975	86,415	5,398	75,097
	Genes	23,341	21,534	23,309	9,521	6,728
Filtered	Counts	17,965,131	18,063,628	18,227,971	1,703,939	2,212,169
	Cells	4,306	3,633	3,694	3,849	764
	Genes	23,341	21,534	23,309	9,521	6,728
Coinciding			3,139	3,191	3,217	216

Table 6: Number of counts, cells and genes in the count matrices obtained from the lung dataset., before and after the application of filters. Also shown, the number of cells after filtering that coincide by barcode with the Tabula Muris analysis.

		TabulaMuris	UMITools	KallistoBustools	Salmon alevin	STARSolo
Unfiltered	Counts	3,873,187	5,847,402	7,737,860	472,031	7,448,203
	Cells	621	5,982	111,480	868	113,172
	Genes	23,341	18,157	19,657	7,134	19,023
Filtered	Counts	2,072,285	3,108,475	3,160,955	418,749	2,978,967
	Cells	461	558	493	602	494
	Genes	23,341	18,157	19,657	7,134	19,023
Coinciding			411	420	442	411

UMI Tools produced the most similar results to Tabula Muris on all three accounts before filtering. After filtering all matrices were similar in magnitude of these values, with a few exceptions. As can be seen in both tables, the count matrices of Salmon alevin appear to be filtered beforehand, according to the number of cells. Furthermore, the per-cell values of counts and genes followed

a different distribution than in the other matrices as can be seen in Figure 4. Thus, different filters were used for Salmon Alevin: 100 to 2000 counts per cell, 100 to 750 genes per cell and less that 5% of mitochondrial counts per cell. With these filters, the count matrices of Salmon Alevin were very similar to the other filtered matrices, albeit with significantly less unique genes.

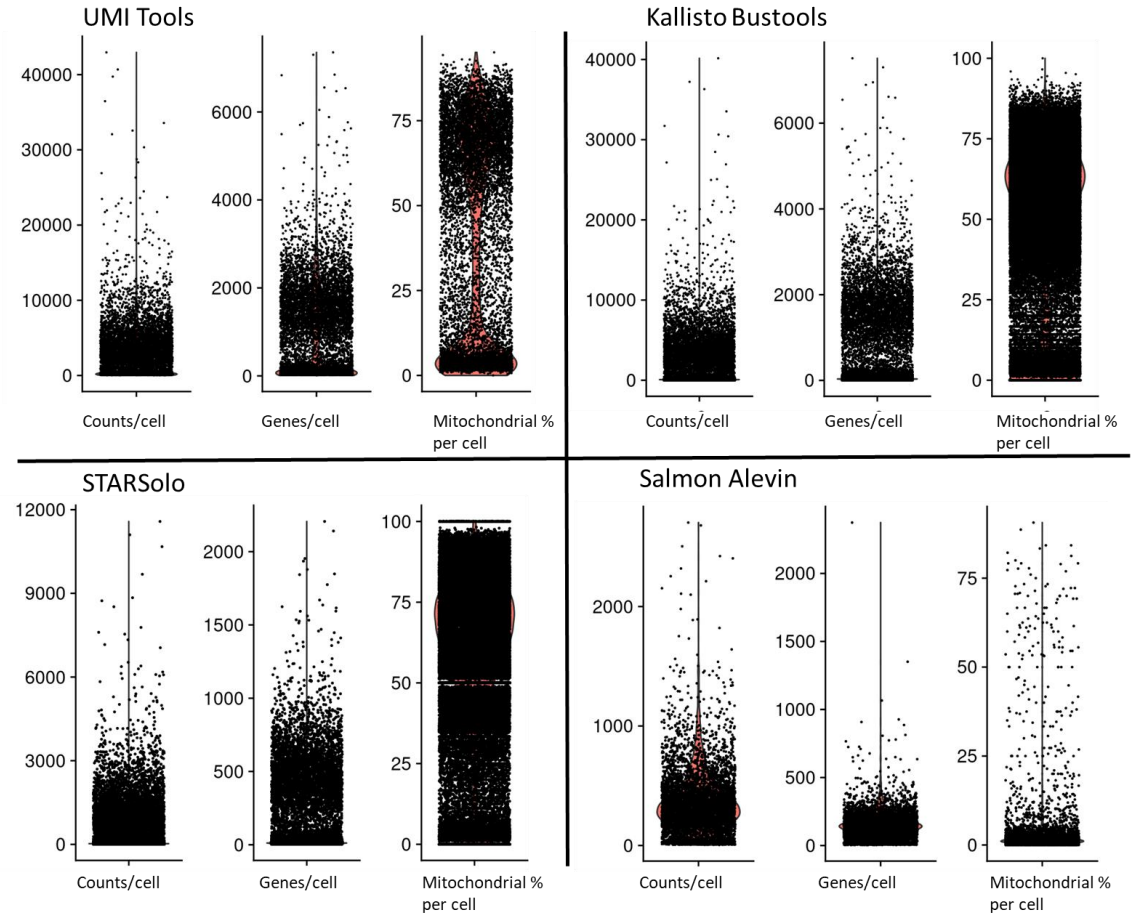

Figure 4: Distribution of the counts per cell, the genes per cell and the percentage of mitochondrial genes per cell in the matrices obtained from the trachea data set before filtering.

In figure 4 it can also be observed the relevance of filtering barcodes with a high proportion of mitochondrial genes. Kallisto Bustools and STARSolo overestimated the number of cells by as much as 22 times. After applying filtering, Kallisto Bustools yielded the anticipated values, whereas STARSolo exhibited a considerably reduced cell count in the trachea dataset, due to the application of the mitochondrial percentage filter.

Following the application of filters, the barcodes that remained had a percentage of coincidence with the ground truth ranging from 73% to 96%. The analysis with the most coinciding barcodes was salmon Alevin in the lung dataset.

4.3. Biological significance

To compare the cell clusters obtained with those from the ground truth dataset I generated a joint table that matched the cluster identities of each cell barcode across the two datasets. In the clustering of the Tabula Muris trachea dataset, 7 clusters were identified. The results of these analyses can be found in Figure 5 and Figure 6.

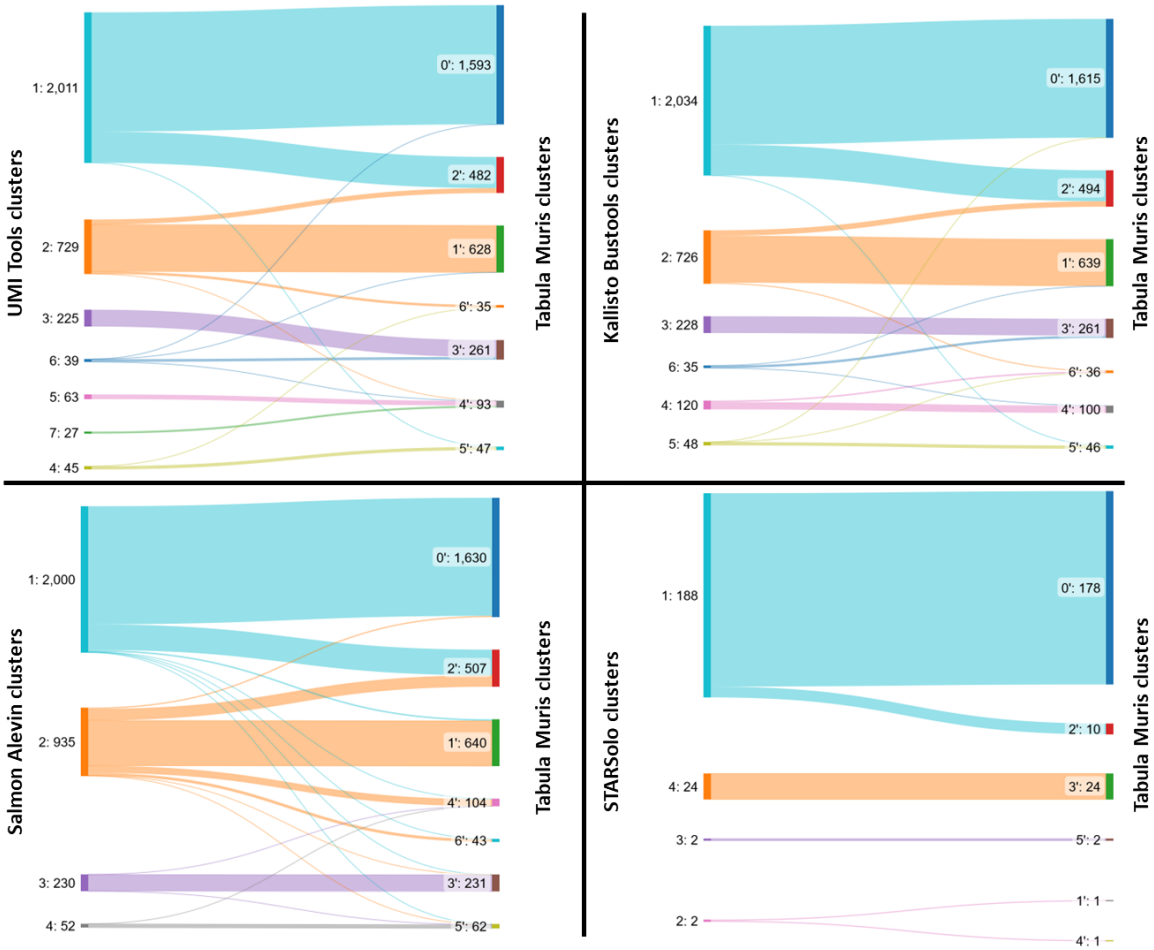

Figure 5: Cell flow diagram showing where the trachea cells from a given cluster are found in the ground truth clustering.

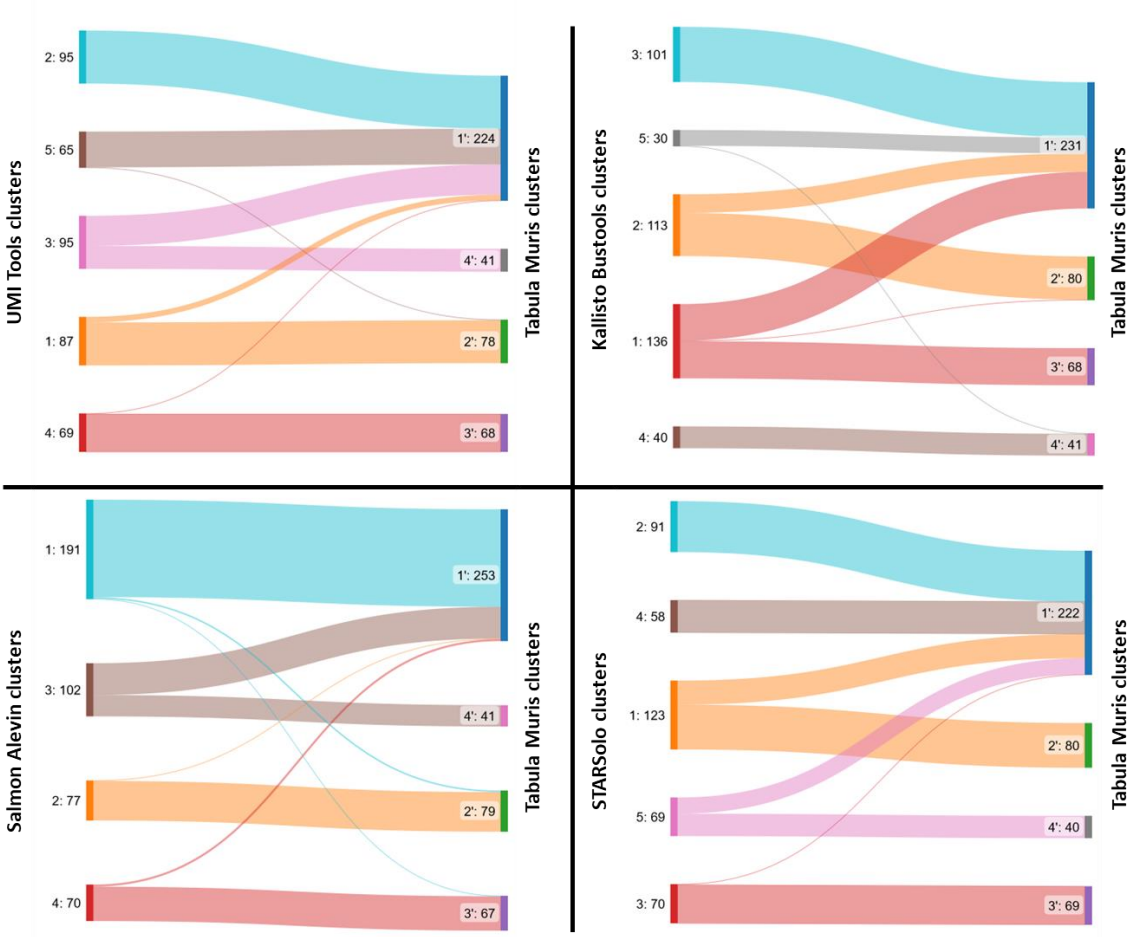

Figure 6: Cell flow diagram showing where the lung cells from a given cluster are found in the ground truth clustering.

To evaluate if the clusters of each analysis match with the ground truth I counted the number of cells that are in a correct cluster equivalence. For example, from the UMI Tools counts matrix analysis, the cells of the cluster number 1 are divided between clusters 0 and 2 of the ground truth. Therefore, that is set as an equivalence, since it is arguable that they belong to a distinct type of cell. However, there are 66 cells of the cluster number 2 of the analysis that are also found in the cluster number 2 of the ground truth. Since the ground truth cluster number 2 better matches cluster 1 and UMI tools cluster 2 better matches ground truth cluster 1, these 66 cells are wrongly assigned**Error! Reference source not found.**. The proportion of correctly assigned cells for each analysis is shown in Table 7 Table 7.

Table 7: Percentage of cells with a consistent assignation, per tool and dataset

	Trachea	Lung
UMI Tools	97,61%	87,10%
Kallisto Bustools	97,01%	77,61%
Salmon Alevin	90,02%	88,64%
STARSolo	100%	82,24%

The results were better for the trachea dataset. Most of the error in all analyses of the trachea dataset originates from a number of cells from the second cluster of ground truth being found in the second cluster of the analysis. In the analyses

of the lung dataset, the cells of the biggest Tabula Muris cluster are found distributed on various clusters, which produces error. The clustering of the STARSolo matrix produced results that perfectly matched the ground truth, though this analysis had a lower number of cells. Overall, the best results in this aspect are those obtained from the UMI Tools matrix.

In another approach to compare the biological significance of the results I compared the marker genes obtained in each analysis with those obtained from the ground truth matrix. The top ten marker genes by AUC of each cluster were considered. Then, similarly to the previous comparison, I traced how many markers for a cluster were also found in the ground truth and if they were found together. These results are summarized in Figure 7 and Figure 8.

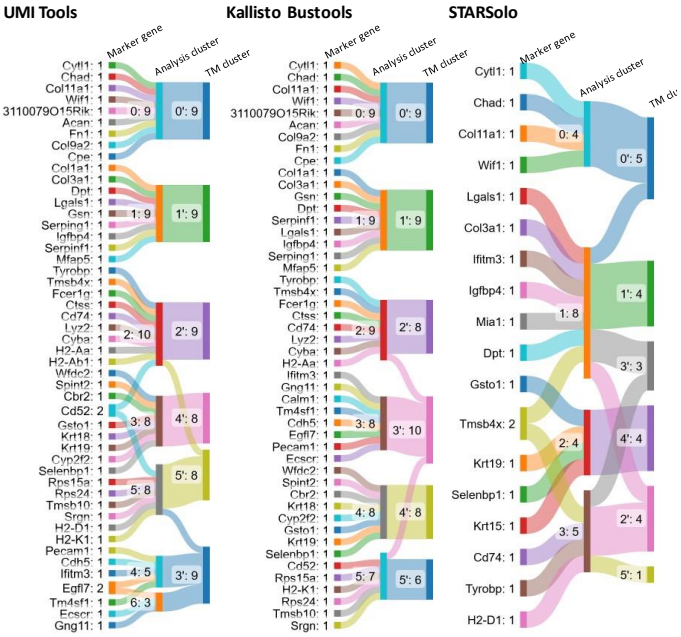

Figure 7: Flow chart of marker genes found in each analysis of the trachea dataset that were also found to mark a cluster of the Ground truth.

In the trachea dataset, for UMI Tools 53 out of 70 markers matched, for Kallisto Bustools 50 out of 60 markers matched and for STARSolo 19 out of 40 markers matched. No markers matched for Salmon Alevin. I found a consensuated cluster identified by Cytl1, Chad, Coll11a and Wif1 among others. Nine out of ten top markers found in the UMI Tools and Kallisto Bustools matrices coincide with those found in the Tabula Muris Matrix. This cluster is the largest in all cases and these are genes related to cartilaginous tissue. Therefore, the cells belonging to this tissue are likely the structural part of the endoderm of the trachea. The second largest cluster for all cases, is marked by collagen and collagen related genes as well as cell matrix related genes. These results are consistent with the cell identity correspondence of the previous comparison.

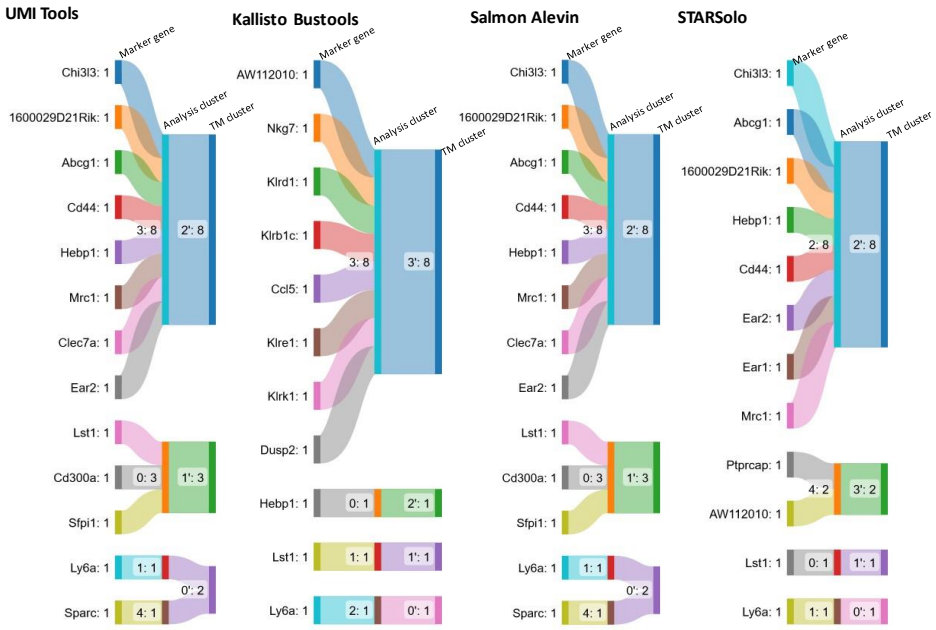

Figure 8: Flow chart of marker genes found in each analysis of the lung dataset that were also found to mark a cluster of the Ground truth.

In the lung dataset, less markers coincided with the ground truth. For UMI Tools, 13 out of 50 markers matched, for Kallisto Bustools 11 out of 50 markers matched, for Salmon Alevin 13 out of 40 markers matched and for STARSolo 12 out of 50 markers matched. A cluster with 8 markers that coincide with those of a ground truth cluster were found in the UMI Tools, Salmon Alevin and STARSolo matrices. The coinciding markers are Chi3l3, Abcg1, Cd44, Hebp1, Mrc1 and Ear2. The gene Lst1 was found to mark the second largest cluster in all analyses. In the Salmon Alevin analysis, however, it is a false coincidence since the cells of the cluster are not the same as the cells of the ground truth cluster. Overall, this comparison in the lung dataset is less useful because all pipelines found a low proportion of coinciding markers with the ground truth, although it also is consistent with the results of the cell identity comparison.

5. Conclusions and future works

As expected, UMI Tools was the slowest and least efficient pre-processing tool, while also being the hardest to implement. Kallisto Bustools was the fastest and most memory efficient tool which is also expected due to the implementation of pseudo-alignment and the compromises made in its design. STARSolo was faster than Salmon Alevin, which was unexpected, since Salmon does not fully align the reads to the genome unlike STAR. The key to the speed of STAR is its parallelization, because even though it spent more time being processed in the CPU than Salmon, it needed less real time to complete the process.

In the count matrices produced, UMI Tools and Kallisto Bustools produced matrices that were consistent with the ground truth in number of molecules, genes, and cells. Salmon Alevin needed to use less strict quality control filters due to having a different distribution in the metrics that were considered. STARSolo, unexpectedly, produced a very low-quality count matrix for the lung dataset due to high mitochondrial gene count proportions. This resulted in a low number of cells remaining after filtering, which affected latter results.

In the biological significance comparison, the clusterization of cells was found to be consistent, with over 75% of cells being clustered together both in the analysis of my count matrices and the Tabula Muris matrices. The marker genes were consistent with the ground truth, although less so in the lung cell dataset, likely due to being smaller in size. However, no marker genes of the Salmon alevin trachea matrix matched any marker gene in the Tabula Muris matrix. This is possibly due to the different distribution and lesser filters applied.

Thus, the first objective of comparing and benchmarking the pre-processing tools was successful. As for the second objective, it is clear that pre-processing has improved substantially in the las half decade in terms of computational efficiency. All tools performed significantly faster than UMI Tools, were easier to implement and used equal or less memory. However, in terms of matrix quality and biological results, the matrix produced by UMI Tools produced good results when compared to the ground truth in all accounts. Kallisto Bustools also produced good results in all accounts while STARSolo and Salmon Alevin showed inconsistencies. Since Kallisto Bustools was the fastest software, according to this work, it should be the superior option for general purpose single cell RNA sequencing preprocessing.

To further the objectives of this work, other tools could be easily evaluated within this workflow as well as other datasets. With more data volume and comparing more software, more certain conclusions could be extracted. Another

possible implementation would be a system to assign a cell type to clusters, which would improve the biological significance comparison.

6. Glossary

- AUC — Area Under the Curve
- BAM — Binary Alignment and Map
- BUS — Barcode, UMI, Set
- CPU — Central Processing Unit
- DNA — DeoxyriboNucleic Acid
- EM — Expectation Maximization
- FACS — Fluorescence-Activated Cell Sorting
- GEO — Gene Expression Omnibus
- mRNA — messenger RiboNucleic Acid
- NGS — Next Generation Sequencing
- PCR — Polymerase Chain Reaction
- RAM — Random Access memory
- RNA — RiboNucleic Acid
- ROC — Receiver Operating Characteristic (curve)
- RT — Retrotranscription
- scRNA-Seq — single-cell RNA Sequencing
- SDG — Sustainable Development Goals
- SRA — Sequence Read Archive
- UMI — Unique Molecular Identifier

7. Bibliography

[1] M. Schena, D. Shalon, R. W. Davis, and P. O. Brown, “Quantitative Monitoring of Gene Expression Patterns with a Complementary DNA Microarray,” *Science (1979)*, vol. 270, no. 5235, pp. 467–470, Oct. 1995, doi: 10.1126/SCIENCE.270.5235.467.

[2] Z. Wang, M. Gerstein, and M. Snyder, “RNA-Seq: A revolutionary tool for transcriptomics,” *Nat Rev Genet*, vol. 10, no. 1, pp. 57–63, Jan. 2009, doi: 10.1038/NRG2484.

[3] R. Stark, M. Grzelak, and J. Hadfield, “RNA sequencing: the teenage years,” *Nat Rev Genet*, vol. 20, no. 11, pp. 631–656, Nov. 2019, doi: 10.1038/S41576-019-0150-2.

[4] B. Hwang, J. H. Lee, and D. Bang, “Single-cell RNA sequencing technologies and bioinformatics pipelines,” *Experimental and Molecular Medicine*, vol. 50, no. 8. Nature Publishing Group, Aug. 01, 2018. doi: 10.1038/s12276-018-0071-8.

[5] T. Smith, A. Heger, and I. Sudbery, “UMI-tools: modeling sequencing errors in Unique Molecular Identifiers to improve quantification accuracy,” *Genome Res*, vol. 27, no. 3, pp. 491–499, Mar. 2017, doi: 10.1101/GR.209601.116.

[6] B. Kaminow, D. Yunusov, and A. Dobin, “STARsolo: accurate, fast and versatile mapping/quantification of single-cell and single-nucleus RNA-seq data,” *bioRxiv*, p. 2021.05.05.442755, May 2021, doi: 10.1101/2021.05.05.442755.

[7] P. Melsted *et al.*, “Modular, efficient and constant-memory single-cell RNA-seq preprocessing,” *Nature Biotechnology 2021 39:7*, vol. 39, no. 7, pp. 813–818, Apr. 2021, doi: 10.1038/s41587-021-00870-2.

[8] A. Srivastava, L. Malik, T. Smith, I. Sudbery, and R. Patro, “Alevin efficiently estimates accurate gene abundances from dscRNA-seq data,” *Genome Biol*, vol. 20, no. 1, pp. 1–16, Mar. 2019, doi: 10.1186/S13059-019-1670-Y/FIGURES/8.

[9] Y. Hao *et al.*, “Integrated analysis of multimodal single-cell data,” *Cell*, vol. 184, no. 13, pp. 3573-3587.e29, Jun. 2021, doi: 10.1016/J.CELL.2021.04.048/ATTACHMENT/1E5EB5C1-59EE-4B2B-8BFA-14B48A54FF8F/MMC3.XLSX.

[10] N. Schaum *et al.*, “Single-cell transcriptomics of 20 mouse organs creates a Tabula Muris,” *Nature 2018 562:7727*, vol. 562, no. 7727, pp. 367–372, Oct. 2018, doi: 10.1038/s41586-018-0590-4.

[11] J. Eberwine *et al.*, “Analysis of gene expression in single live neurons,” *Proc Natl Acad Sci U S A*, vol. 89, no. 7, pp. 3010–3014, 1992, doi: 10.1073/pnas.89.7.3010.

[12] G. Brady, M. Barbara, and N. N. Iscove, “Representative in vitro cDNA amplification from individual hemopoietic cells and colonies,” *Methods Mol Cell Biol*, vol. 2, no. 1, pp. 17–25, 1990, Accessed: May 14, 2023. [Online]. Available: http://wwwlabs.uhnresearch.ca/iscove/MMCB90.pdf

[13] C. A. Klein *et al.*, “Combined transcriptome and genome analysis of single micrometastatic cells,” *Nature Biotechnology 2002 20:4*, vol. 20, no. 4, pp. 387–392, 2002, doi: 10.1038/nbt0402-387.

[14] F. Tang *et al.*, “mRNA-Seq whole-transcriptome analysis of a single cell,” *Nature Methods 2009 6:5*, vol. 6, no. 5, pp. 377–382, Apr. 2009, doi: 10.1038/nmeth.1315.
[15] M. H. Julius, T. Masuda, and L. A. Herzenberg, “Demonstration That Antigen-Binding Cells Are Precursors of Antibody-Producing Cells After Purification with a Fluorescence-Activated Cell Sorter,” *Proc Natl Acad Sci U S A*, vol. 69, no. 7, p. 1934, 1972, doi: 10.1073/PNAS.69.7.1934.
[16] N. M. Clark, A. P. Fisher, and R. Sozzani, “Identifying Differentially Expressed Genes Using Fluorescence-Activated Cell Sorting (FACS) and RNA Sequencing from Low Input Samples,” *Methods Mol Biol*, vol. 1819, pp. 139–151, 2018, doi: 10.1007/978-1-4939-8618-7_6.
[17] N. Attaf *et al.*, “FB5P-seq: FACS-Based 5-Prime End Single-Cell RNA-seq for Integrative Analysis of Transcriptome and Antigen Receptor Repertoire in B and T Cells,” *Front Immunol*, vol. 11, p. 216, Mar. 2020, doi: 10.3389/FIMMU.2020.00216/BIBTEX.
[18] G. X. Y. Zheng *et al.*, “Massively parallel digital transcriptional profiling of single cells,” *Nature Communications 2017 8:1*, vol. 8, no. 1, pp. 1–12, Jan. 2017, doi: 10.1038/ncomms14049.
[19] “10x Genomics Website.” https://www.10xgenomics.com/ (accessed May 18, 2023).
[20] P. Kindgren, M. Ivanov, and S. Marquardt, “Native elongation transcript sequencing reveals temperature dependent dynamics of nascent RNAPII transcription in Arabidopsis,” *Nucleic Acids Res*, vol. 48, no. 5, pp. 2332–2347, Mar. 2020, doi: 10.1093/NAR/GKZ1189.
[21] S. Parekh, C. Ziegenhain, B. Vieth, W. Enard, and I. Hellmann, “zUMIs - A fast and flexible pipeline to process RNA sequencing data with UMIs,” *Gigascience*, vol. 7, no. 6, pp. 1–9, Jun. 2018, doi: 10.1093/GIGASCIENCE/GIY059.
[22] S. Chen, Y. Zhou, Y. Chen, and J. Gu, “fastp: an ultra-fast all-in-one FASTQ preprocessor,” *Bioinformatics*, vol. 34, no. 17, pp. i884–i890, Sep. 2018, doi: 10.1093/BIOINFORMATICS/BTY560.
[23] R. Patro, G. Duggal, M. I. Love, R. A. Irizarry, and C. Kingsford, “Salmon provides fast and bias-aware quantification of transcript expression,” *Nat Methods*, vol. 14, no. 4, pp. 417–419, Mar. 2017, doi: 10.1038/nmeth.4197.
[24] E. Turro, S. Y. Su, Â. Gonçalves, L. J. M. Coin, S. Richardson, and A. Lewin, “Haplotype and isoform specific expression estimation using multi-mapping RNA-seq reads,” *Genome Biol*, vol. 12, no. 2, pp. 1–15, Feb. 2011, doi: 10.1186/GB-2011-12-2-R13/TABLES/3.
[25] A. P. Dempster, N. M. Laird, and D. B. Rubin, “Maximum Likelihood from Incomplete Data Via the EM Algorithm,” *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 39, no. 1, pp. 1–22, Sep. 1977, doi: 10.1111/J.2517-6161.1977.TB01600.X.
[26] N. L. Bray, H. Pimentel, P. Melsted, and L. Pachter, “Near-optimal probabilistic RNA-seq quantification,” *Nat Biotechnol*, vol. 34, no. 5, pp. 525–527, May 2016, doi: 10.1038/NBT.3519.
[27] P. Melsted, V. Ntranos, and L. Pachter, “The barcode, UMI, set format and BUStools,” *Bioinformatics*, vol. 35, no. 21, pp. 4472–4473, Nov. 2019, doi: 10.1093/BIOINFORMATICS/BTZ279.

[28] A. S. Booeshaghi *et al.*, “Reliable and accurate diagnostics from highly multiplexed sequencing assays,” *Scientific Reports 2020 10:1*, vol. 10, no. 1, pp. 1–7, Dec. 2020, doi: 10.1038/s41598-020-78942-7.
[29] T. Brink Buus *et al.*, “Improving oligo-conjugated antibody signal in multimodal single-cell analysis,” *bioRxiv*, p. 2020.06.15.153080, Mar. 2021, doi: 10.1101/2020.06.15.153080.
[30] “Tabula Muris.” https://tabula-muris.ds.czbiohub.org/ (accessed May 23, 2023).