# Software, tools, and methods development for single-cell transcriptomics

Thesis by
Delaney Kalcey Sullivan

In Partial Fulfillment of the Requirements
for the degree of
Doctor of Philosophy


CALIFORNIA INSTITUTE OF TECHNOLOGY
Pasadena, California

2025
(Defended May 1st, 2025)

Delaney Kalcey Sullivan
ORCID: 0000-0002-8359-6705

# ACKNOWLEDGEMENTS

I would like to first thank my mom, Yihua Lin, for her unwavering support of my career and ambitions. Her love and encouragement have been the bedrock of my journey. I also wish to acknowledge my father, Billy C. Sullivan, who passed away from cancer during my childhood. His passing deeply shaped my path, inspiring me to pursue medicine and sparking my curiosity about the mechanisms of disease.

I am profoundly grateful to Lior Pachter and Mitch Guttman, who advised me throughout my Ph.D. training. Their mentorship has both challenged and inspired me. I would additionally like to express my heartfelt gratitude to my other thesis committee members, Barbara Wold (committee chair) and Harold Pimentel, for their insight and support.

Next, I would like to thank my "BE/Bi 103 a" course partners, Lynn Fang and Nina Le, who were the first classmates I befriended at Caltech. Their camaraderie helped me feel at home in this chapter of my academic journey.

A special thanks to Olivia Ettlin (olive!) for being my co-developer on the SWIFT-seq method in the Guttman lab; collaborating with her has been both productive and a lot of fun. I also want to thank our superstar mentor, Mario Blanco, whose expertise and enthusiasm were instrumental in our success. A sincere thank you to other members of the Guttman lab as well for their friendship, laughs, and constant source of support, including Jimmy Guo, Ben Yeh, Amy Chow, Carl Urbinati, Asuka Sato (another SWIFT-seq developer), Jolo Ferrer, Noah Epstein, Praneeth Goli, Drew Honson, Linlin Chen, Allen Chen, Ryan Hong, Drew Perez, Isabel Goronzy, Paulomi Bhattacharya, Prashant Bhat, Mackenzie Strehle, and Anthony Vasquez.

I am deeply grateful to Laura Luebbert, Kristján Eldjárn Hjörleifsson, Bekah Loving, and Mayuko Boffelli for our work together developing k-mer-based sequence analysis methods in the Pachter lab. Bonding over the struggles of debugging code has been quite the experience. I would also like to extend my appreciation to other members of the Pachter lab for fostering a positive, fun, and collaborative environment during my time at Caltech,

including Lambda Moses, Maria Carilli, Vera Beilinson, Tara Chari, Taleen Dilanyan, Meichen Fang, Cat Felce, Kayla Jackson, Anne Kil, Joe Rich, Nikki Swarna, Sina Booeshaghi, Ángel Gálvez- Merchán, Conrad Oakes, Gennady Gorin, Charlene Kim, and Joseph Min.

I am also grateful to Ali Mortazavi (and members of his lab), Diane Trout, and Brian Williams for important feedback on my work. Additionally, I would like to thank to Verona Yue for trying out and testing my code. Furthermore, I would like to thank Páll Melsted and Guillaume Holley for their help and advice on data structures.

I thank my former mentors of the Felsher Lab where I had worked during my bachelor's and master's degree years at Stanford University: Dean Felsher, Daniel Liefwalker, Renu Dhanasekaran, Anja Deutzmann, Arvin Gouw, and Srivi Swaminathan. It is thanks to them that I decided to pursue science further. I am also grateful to Jonathan Chen for his mentorship during my time at Stanford.

I would like to thank Danni Lu, whom I had the honor of serving as the best man for his wedding.

I would like to thank my UCLA-Caltech MSTP friends, with a particular shout out to Eric Lin for always being there to brighten my day.

Lastly, to the countless people who have brought joy, encouragement, and inspiration into my life during the highs and lows of my Ph.D.: while not everyone is listed, please know how much your presence has meant to me. Thank you all for making this journey unforgettable.

# ABSTRACT

Advances in transcriptomics have transformed the study of gene expression, enabling a shift from low-throughput bulk RNA measurements to high-resolution, large-scale single-cell RNA-sequencing (scRNA-seq). This work refines existing methodologies and introduces new strategies for achieving precise, versatile, and scalable transcriptomic analyses across a broad spectrum of assays and biological contexts.

On the computational front, this dissertation introduces new methods for adaptable preprocessing of sequencing reads, enabling the handling of very complex read structures. It refines existing strategies for efficiently querying large-scale transcriptomic datasets and enhances approaches for quantifying nascent and mature RNA species. A general framework is introduced for discovering and organizing biologically informative sequences directly from raw sequencing data, facilitating the detection of sample-specific or condition-specific variation. On the experimental front, a novel single-cell RNA sequencing method is presented that is cost-effective, open source, and scalable, supporting large-scale studies with substantial cell numbers and high per-cell resolution.

These developments collectively expand the toolkit for transcriptomics, enabling more efficient and comprehensive exploration of RNA biology.

# PUBLISHED CONTENT AND CONTRIBUTIONS

Luebbert L, **Sullivan DK**, Carilli M, Hjörleifsson KE , Winnett AV, Chari T, Pachter L. Detection of viral sequences at single-cell resolution identifies novel viruses associated with host gene expression changes. ***Nature Biotechnology***. 2025.
doi: 10.1038/s41587-025-02614-y

D.K.S. implemented the viral sequence detection algorithm (with L.L.).

**Sullivan DK***, Hjörleifsson KE*, Swarna NP, Oakes C, Holley G, Melsted P[#], Pachter L[#]. Accurate quantification of nascent and mature RNAs from single-cell and single-nucleus RNA-seq. ***Nucleic Acids Research***. 2025;53(1):gkae1137.
doi: 10.1093/nar/gkae1137

*D.K.S. and K.E.H. contributed equally to this work.
[#]L.P. and P.M. are co-corresponding authors on this work.
D.K.S. conceived this study (with K.E.H., P.M. and L.P.), implemented the methods in software (with K.E.H.), benchmarked the methods (with K.E.H. and N.P.S.), produced analyses and figures (with K.E.H.), and wrote the paper (with K.E.H. and L.P.).

**Sullivan DK**, Min KHJ, Hjörleifsson KE, Luebbert L, Holley G, Moses L, Gustafsson J, Bray NL, Pimentel H, Booeshaghi AS[#], Melsted P[#], Pachter L[#]. ***Nature Protocols***. 2024.
doi: 10.1038/s41596-024-01057-0

[#]L.P., P.M., and A.S.B. are co-corresponding authors on this work.
D.K.S. conceived the writing of this paper (with L.P., P.M., and A.S.B.) and wrote the initial draft of the paper. D.K.S. led the development of the latest versions (at the time of publication) of kallisto (version 0.50.1), bustools (version 0.43.2), and kb-python (version 0.28.2). D.K.S. created all figures and tables for this paper.

**Sullivan DK**, Pachter L. Flexible parsing, interpretation, and editing of technical sequences with *splitcode*. ***Bioinformatics***. 2024;40(6):btae331.
doi: 10.1093/bioinformatics/btae331

D.K.S. conceived of the work, developed the methods and software, and drafted the manuscript.

Bhat P, Chow A, Emert B, Ettlin O, Quinodoz SA, Strehle M, Takei Y, Burr A, Goronzy IN, Chen AW, Huang W, Ferrer JLM, Soehalim E, Goh ST, Chari T, **Sullivan DK**, Blanco MR, Guttman M. Genome organization around nuclear speckles drives mRNA splicing efficiency. ***Nature***. 2024;629:1165-1173.
doi: 10.1038/s41586-024-07429-6

D.K.S. performed RNA-seq analysis of single-cell SPLiT-seq data from mouse C2C12 myoblasts (with P.B.).

# TABLE OF CONTENTS

*Chapter 1*

# INTRODUCTION

## Overview of transcriptomics

The field of transcriptomics, which encompasses the study of RNA molecules produced by the genome, is fundamental to understanding how genetic information is expressed and regulated within cells. At its core, transcriptomics is built upon the central dogma of molecular biology: DNA is transcribed into RNA, which is subsequently translated into proteins (Crick, 1958). This process begins with transcription, where RNA polymerase in the nucleus synthesizes RNA from a DNA template. Transcription produces a diverse collection of RNA molecules, which may then be modified by chemical processing (such as splicing), further expanding their diversity. The end result is a wide variety of RNA molecules, including messenger RNAs (mRNAs), which are transported to the cytoplasm for protein synthesis, and non-coding RNAs, which serve important regulatory and structural roles.

Gene expression can be assessed by measuring RNA abundance. Early methods, such as reverse transcription quantitative polymerase chain reaction (RT-qPCR) (Wang et al., 1989), enabled precise measurement of specific RNA molecules, providing a robust way to quantify gene expression. However, RT-qPCR is limited in throughput, as only a few genes can be quantified at a time.

The advent of gene expression microarrays enabled simultaneous measurement of expression levels across thousands of genes (Schena et al., 1995). These microarrays are based on hybridization between chemically labeled RNA or cDNA molecules and complementary DNA probes fixed on a solid surface. Thus, a comprehensive probe set could provide an extensive transcriptome profile. However, since microarrays are constrained by their dependence on predefined probes, they cannot be used to detect and quantify novel transcripts or any other sequences that are not pre-designed on the array.

RNA sequencing (RNA-seq) revolutionized transcriptomics by enabling comprehensive, high-resolution transcriptome profiling without reliance on predefined probes (Mortazavi et al., 2008; Wold and Myers, 2008). RNA-seq involves a series of experimental steps, referred to as a library preparation, which convert an RNA isolate into a concentrated collection of double-stranded DNA molecules that can be sequenced via high-throughput sequencing. With the data generated from RNA sequencing, bioinformatics can empower many downstream analyses including transcript abundance quantification (Mortazavi et al., 2008), differential expression analysis (Oshlack et al., 2010), gene structure determination and novel transcript discovery (Guttman et al., 2010; Trapnell et al., 2010), alternative splicing detection (Griffith et al., 2010), and expression quantitative trait loci (eQTL) analysis (Pickrell et al., 2010) (Figure 1.1).

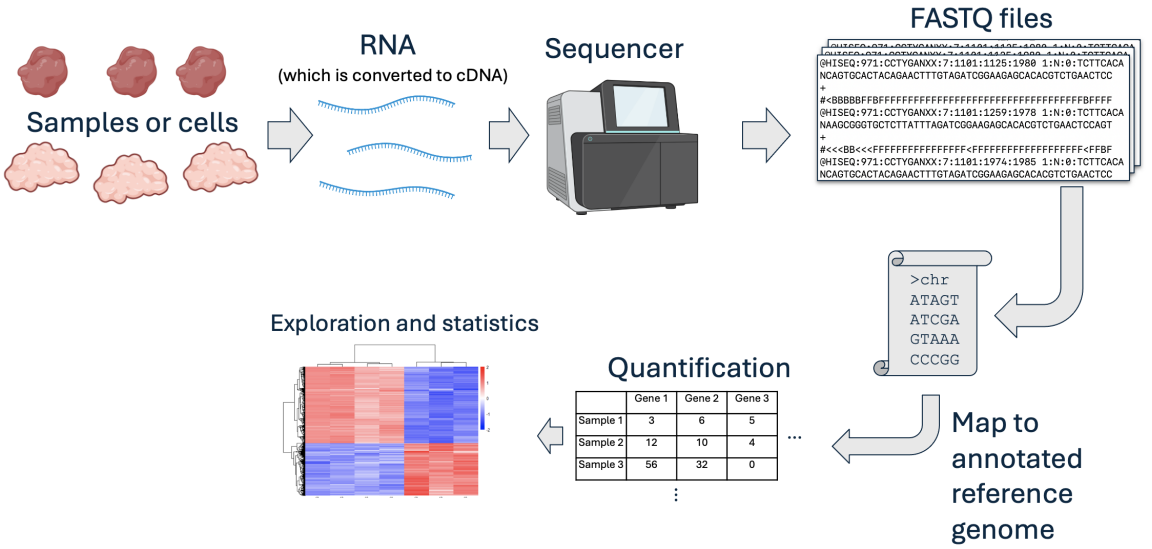



***Figure 1.1****: Overview of a simple RNA-seq experiment.*
*RNA is prepared from biological samples of interest (very commonly, control samples and treatment samples) then, after being processed through a series of biochemical steps, sequenced via high-throughput sequencing. The output produced are sequencing reads, which are stored in a FASTQ file. The FASTQ file is a text file that stores each read as four lines: 1) the sequence name, 2) the sequence itself, 3) a spacer, and 4) the sequence quality scores. The provenance of each read is determined by mapping (i.e. aligning) them to an annotated reference genome. After counting how many reads map to each gene, one can perform various analyses such as determining which genes are statistically significantly differentially expressed between the treatment samples and the control samples. Parts of this illustration were created with BioRender (biorender.com).*

Traditionally, RNA-seq was conducted on bulk tissue samples (i.e. bulk RNA-seq), but single-cell RNA-seq has been gaining increasing popularity. Single-cell RNA-seq can unlock multiple types of analyses that cannot be done with bulk RNA-seq. For complex tissue, single-cell technology can enable subpopulation and cell type identification from the tissue whereas bulk RNA-seq would simply produce an ensemble gene expression quantification, even though some cell types in the mixture may vary widely in expression profiles. The 'compartmentalization' powered by single-cell RNA-seq can thus preserve information about the composition of the mixture (Trapnell, 2015). Even among cells of a given cell type, single-cell RNA-seq affords assessment of inter-cell heterogeneity, which effectively means than rather than having a single biological signal, we can capture thousands of distinct biological measurements for a cell type. Additionally, single cell clustering can reveal transition paths between cell states, enabling the reconstruction of cellular trajectories through processes such as differentiation (Trapnell et al., 2014). Finally, and more recently, parameters describing the stochasticity of processes such as transcription, splicing, and degradation can be inferred by biophysical models thanks to the ability of single-cell RNA-seq to capture multimodal measurements (e.g. both nascent RNA and mature RNA) (La Manno et al., 2018) across a large number of cells (Gorin et al., 2022b, 2023). Of course, single-cell transcriptomics comes with its own challenges, including data sparsity and higher levels of noise, in comparison to bulk RNA-seq. Nonetheless, the single cell resolution can unlock many biological insights that would be obscured by bulk analysis.

With genomic and transcriptomic technology development on the laboratory side comes the need to develop methods on the computational front to process and analyze the resulting data. The next section focuses on principles that underly effective bioinformatics software development.

**Bioinformatics software development**

The growing complexity of bioinformatics workflows, driven by rapid advances in biological and computational methods, hampers the integration of existing and new tools into cohesive pipelines. Pipelines can involve a large number of components and software tools that interact with each other. The selection of software to use at a particular point in the pipeline is often complicated by the fact that different tools, even those performing the same or similar tasks, ingest different input formats and produce different output types. Restricting input and output formats can limit software to a specific use case or create dependencies between tools. Additionally, software programs often combine multiple processing steps in the back end, thereby reducing the ability to customize or modify individual steps within the pipeline. As a result, venturing outside the realm of the specific analysis the tool was designed for, e.g. to adapt the workflow to a new data type, is challenging. While there is no one-size-fits-all solution to address this problem fully, here, we describe specific design principles that can enhance the adaptability and utility of bioinformatics tools. We focus on three such design principles: *flexibility*, *modularity*, and *cross-compatibility* (Figure 1.2), which are partly based on the UNIX philosophy (McIlroy et al., 1978). These design principles can significantly enhance pipeline development for researchers using bioinformatics tools, giving them greater control over their data processing steps and the types of analysis they can perform. Following these principles along with documented best practices can result in more robust and versatile workflows while simultaneously ensuring proper usage.

*Flexibility*: Here, flexibility refers to the ability of a software tool to process diverse input types and to produce output that can be used in many different downstream applications or analyses. An inflexible tool would be one that accepts only one input structure, limiting its applicability across different datasets and its adaptability to datasets produced by future technological developments. Furthermore, an inflexible tool may produce output that is usable for only one specific type of downstream analysis. To illustrate the notion of flexibility, consider the processing of RNA-sequencing data. There exist many different

RNA-seq technologies (Aldridge and Teichmann, 2020), such as bulk RNA-seq, droplet single-cell RNA-seq, plate-based single-cell RNA-seq, and split-and-pool barcoding-based single-cell RNA-seq, each with a distinct raw sequencing read structure. A flexible tool would be able to accommodate multiple or, ideally, all technologies. However, it may be infeasible for an RNA-seq read mapping tool to be compatible with every single past and future sequencing technology. File formats outputted by different sequencers (e.g. Illumina vs. Nanopore) may vary, some assays have complex arrangements of molecular tags in the reads that require specific parsing, etc. Therefore, an upstream tool may be developed to reformat the sequencing reads into a compatible format, thereby augmenting the mapping tool's flexibility. For example, while the popular Cell Ranger software (Zheng et al., 2017) can only process single-cell RNA-seq data from 10X Genomics technologies, tools that reformat read structure for use with Cell Ranger have been developed (Battenberg et al., 2022). An approach that uses an external tool to perform this type of preprocessing would be a modular solution toward enhancing flexibility. Additionally, a flexible tool should produce output compatible with multiple types of downstream analyses. For example, in RNA-seq, a flexible tool would be able to perform isoform-level, gene-level, allelic, and joint nascent and mature RNA quantification. Since quantification is distinct from sequencing read mapping, these different quantifications need not be performed by a single tool. A preferable and more modular solution would be to produce output with which a downstream tool (or even a separate component of the read mapping software) can perform the different quantifications. The goal is to retain flexibility while avoiding having one component of a tool do too many things, which we will discuss further in the following section.

*Modularity*: Modularity here refers to the separation of processing steps into distinct, customizable components. A non-modular tool that performs an entire analysis within one package can limit customization and hinder the ability to utilize intermediate results for other analyses. A modular processing solution, however, separates individual steps and produces intermediate outputs that users can inspect and modify. Since there might be overhead from writing to disk, in many cases, the intermediate output of one step can be

directed to standard output, which can then be directly piped into the next step if the intermediate output file is not needed for a particular use case. Consider sequencing data processing. Sequencing data requires multiple steps of processing, from trimming to alignment to quantification or variant calling followed by further downstream analysis. By separating these steps and providing intermediate outputs, modular tools enable users to customize their workflows and optimize each stage according to their specific needs. Of course, the number of customizations and options can be large for a tool that is modular and flexible, so a best practices guide, based on known benchmarking results, should be provided to guide users towards proper usage of the tool and the most accurate results. Oftentimes, a wrapper for the most common use cases would be desirable, so a user does not need to manually execute the individual steps that they don't wish to customize.

*Cross-compatibility*: refers to a tool being able to use the output of another tool as input and/or to produce output that is compatible with a wide range of downstream tools. For example, a differential gene expression program should be able to take the results of different aligners and, likewise, different aligners should produce results that are compatible with the differential gene expression program. Of important note, cross-compatible solutions should use appropriate standard data file formats for each step (List et al., 2017), such as BAM/SAM files (Li et al., 2009), FASTA/FASTQ files (Pearson and Lipman, 1988), Matrix Market files (Boisvert et al., 1996), JSON files (Pezoa et al., 2016), YAML files (Ben-Kiki et al., 2004), etc. If a new file format is introduced, developers should include a detailed description of the new file format along with a tool to easily process it. Some file formats are less amenable to cross-compatibility. For example, we discourage from storing results in HDF (Hierarchical Data Format) files because HDF is not straightforward to work with or convert to another file format and other tools do not have built-in support. It may be inconvenient for a user to install half a dozen software tools to complete an analysis, however, workflow managers, such as Snakemake (Mölder et al., 2021) or Nextflow, or containers such as Docker (Merkel, 2014) or conda (https://conda.io), greatly facilitate tool installation, compatibility, and management.

Altogether, ensuring cross compatibility through standard input and output file formats promotes interoperability and simplifies the integration of diverse tools within a pipeline.

There are many factors that ensure the long-term success of bioinformatics software, including prompt user support, regular software maintenance, ease of installation and use, comprehensive documentation, and use of package managers such as conda. While much attention has been given to the principles of clear documentation (Karimzadeh and Hoffman, 2018) and modular pipeline design (Hoon et al., 2003; Roy et al., 2018), these discussions primarily focus on software and pipeline usability, with less emphasis on the underlying software design. Although there has been considerable discussion on making tools easy-to-use and easy-to-understand for both biologists and bioinformaticians (Ahmed et al., 2014; Bolchini et al., 2009; Kumar and Dudley, 2007; Pavelin et al., 2012), there has been less focus on software design that enhances its utility for data processing and analysis, beyond just usability. Flexibility, modularity, and cross-compatibility in software design are essential for creating adaptable tools that can handle diverse input formats, integrate seamlessly with other software, and be customized for various analyses. Incorporating these three principles into bioinformatics software can advance the capabilities of data processing and analysis pipelines.

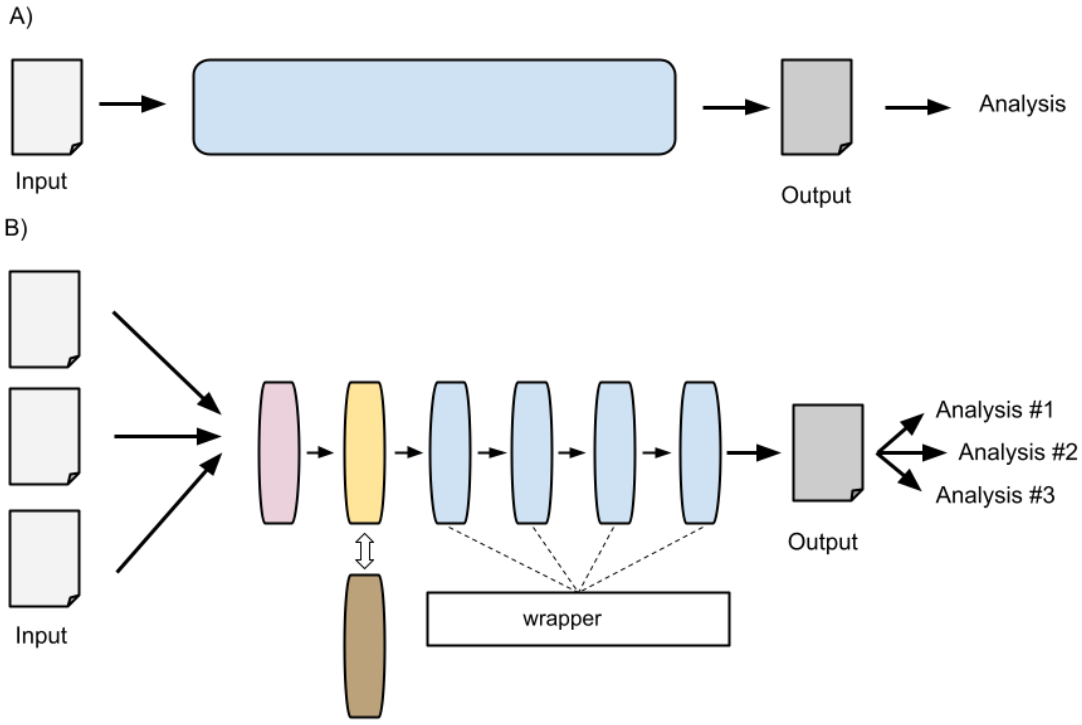



***Figure 1.2****: Principles of flexibility, modularity, and cross-compatibility.*
***A)*** *Schematic of a pipeline that does not contain flexible, modular, or cross-compatible tools. Lack of flexibility: The processing software only accepts one input and produces output that can only be used in one specific analysis. Lack of modularity: The software bundles all processing steps into a single component. Lack of cross-compatibility: The software can not integrate with other tools and no tools can be substituted into any step of the processing.* ***B)*** *Schematic of a pipeline that contains tools that are flexible, modular and cross-compatible. Flexibility: Tools in the pipeline accept multiple input types and produce output that can be used in multiple downstream analyses. Modularity: The processing is divided into multiple steps. The software in blue is broken down into multiple components and a "wrapper" around its components can be executed to facilitate running the components together. Cross-compatibility: The four different tools (shown in pink, yellow, brown, and blue) are able to interface with one another such that each tool can accept another tool's input and/or produce output compatible with another tool. The tool shown in brown can be substituted in for the tool shown in yellow.*

*Chapter 2*

# SEQUENCING READ PREPROCESSING

## Flexible processing of technical sequences

The reads that result from next-generation sequencing libraries can contain many types of synthetic constructs, or technical sequences, including adapters, primers, indices, barcodes, and unique molecular identifiers (UMIs) (Booeshaghi et al., 2024; Johnson et al., 2023; Kebschull and Zador, 2018; Kivioja et al., 2012; Martin, 2011; Melsted et al., 2019). These oligonucleotide sequences are defined by the technicalities of sequencing-based assays and experiments, with each sequence being either a completely unknown sequence, a known sequence, or an unknown sequence that is a member of a set of known sequences.

There are many read preprocessing tools for editing and extracting information from such sequences, including the widely used tools cutadapt (Martin, 2011), fastp (Chen et al., 2018), and Trimmomatic (Bolger et al., 2014) for adapter and quality trimming, UMI-tools (Smith et al., 2017) and zUMIs (Parekh et al., 2018) for UMI processing, BBTools (https://sourceforge.net/projects/bbtools/) (Bushnell et al., 2017) and reaper (Davis et al., 2013) for general filtering operations, Picard (https://github.com/broadinstitute/picard) and fgbio (https://github.com/fulcrumgenomics/fgbio) for many read manipulation operations, INTERSTELLAR for read structure interpretation (Kijima et al., 2023), among many other tools (Battenberg et al., 2022; Cheng et al., 2024; Kong, 2011; Liu, 2019; Roehr et al., 2017). Many of these tools define a “read structure” to describe the layout of a read, e.g. fgbio uses a sequence of <number><operator> operators where the number of the length of a segment and the operator describes how the segment should be processed. However, no one tool can adequately address all technical sequence preprocessing tasks. Some methods, such as adapter trimming methods, can only remove identified technical sequences from reads but lack the ability to store information about technical sequences that are relevant to the provenance of the read. Other methods can extract and store

technical sequences from reads but are limited to only extracting sequences at defined positions of defined lengths within reads, and may present limited options for handling variable position and variable length segments. Still other methods are designed for only a specific type of assay, such as single-cell RNA-seq. Technologies such as (long-read) SPLiT-seq (Rebboah et al., 2021; Rosenberg et al., 2018), SPRITE (Quinodoz et al., 2022, 2018), and Smart-seq3 (Hagemann-Jensen et al., 2020), contain complex, multifaceted technical sequences that currently are processed by custom scripts or specific use-case modifications to existing tools.

The tool that we developed, splitcode (Sullivan and Pachter, 2024), introduces versatile new features for general preprocessing needs. splitcode is a flexible solution with a low memory and computational footprint that can reliably, efficiently, and error-tolerantly preprocess technical sequences based on a user-supplied structure of how those sequences are organized within reads. For example, splitcode can simultaneously trim technical sequences, parse combinatorial barcodes that are variable in length and inconsistent in location within a read, and extract UMIs that are defined in location with respect to other technical sequences rather than at a set position within a read. These features make splitcode a suitable tool for processing variable length staggers at the start of reads; such staggers are often introduced to enhance nucleotide diversity during the early cycles of sequencing, preventing monotemplate issues that would arise from sequencing identical nucleotides during those cycles. The technical sequences that splitcode may be useful for identifying include not only barcodes or UMIs but also ligation linkers, integrase attachment sites, and Tn5 transposase mosaic ends. Moreover, splitcode can seamlessly interface with other command-line tools, including other read sequencing read preprocessors as well as read mappers, by streaming the pre-processed reads into those tools. Thus, splitcode can eliminate the need to write an entirely new file to disk at every step of preprocessing, a practice that currently results in inefficient use of time and disk space. Furthermore, splitcode can stream reads into itself and also can directly accommodate multiple steps (i.e. the output from one set of user-defined instructions can

be directly fed into another), enabling multiple preprocessing steps to be performed in sequence for more complicated assays.

*Software*:

The splitcode software is written in C++11 and is freely available and open source under the BSD-2 clause license. The source code for the splitcode program is available at https://github.com/pachterlab/splitcode. Documentation for the software is available at https://splitcode.readthedocs.io/.

*Framework and Usage*:

We refer to the synthetic constructs, or technical sequences that can be identified in reads as tags. Tags are described in the splitcode config file with several parameters including a tag ID, the sequence itself, the error-tolerance for identifying that tag, and options such as where the tag might be found within sequencing reads and conditions under which the tag should be searched for. A collection of tags forms a barcode, which can be used to demultiplex reads according to the tags identified within a read. Within the config file, a user can also specify extraction options to delineate how certain subsequences within reads should be extracted. Subsequences can be extracted by using tags as anchor points or can be extracted at user-defined positions within reads. This feature is particularly useful for unique molecular identifier (UMI) sequences which are generally unknown sequences that exist at defined locations within reads. Additionally, in the config file, a user can specify read editing options including trimming and whether identified tags should be replaced with a particular sequence. Thus, identified technical sequences can be modified or trimmed *in situ*. Taken together, this array of options makes it possible for splitcode to parse data from a large variety of sequencing assays, including those with many levels of multiplexing (Figure 2.1).

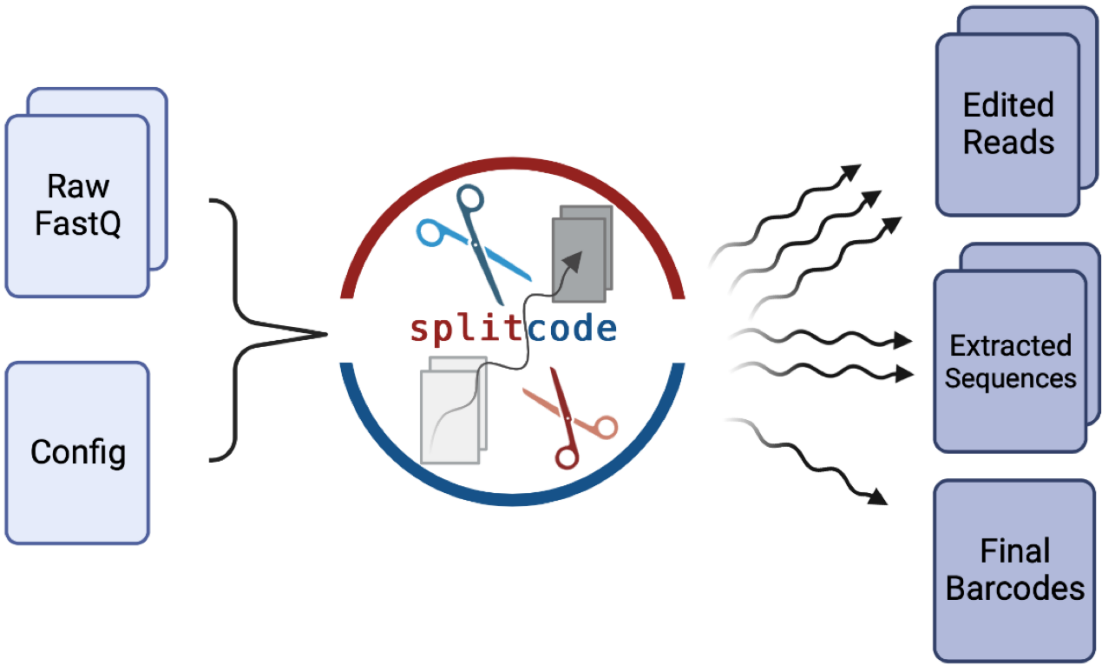



***Figure 2.1**: Overview of the splitcode workflow.*
*The splitcode program takes in a set of FASTQ files and a user-specified config file, which serves as a recipe describing how the reads should be parsed. The user executes splitcode on the command-line, specifying command-line options on how the output should be formatted. The output consists of one or more of the following: the original FASTQ files (possibly edited), the extracted sequences (e.g. UMI sequences which are unknown and need to be extracted by using location information or anchor points), and the final barcodes which are unique for each combination of identified tags. The output may take the form of FASTQ files, gzip-compressed FASTQ files, BAM files, or interleaved sequences directed to standard output, depending on what the user specifies. This figure was created with BioRender (biorender.com).*

Following construction of the config file (Figure 2.2), users can supply the config file to the splitcode program on the command-line. Users can further specify the output options for how the final barcode, the (possibly edited) reads, the extracted subsequences should be outputted. The program presents many options for outputting reads, allowing seamless integration with many downstream tools. Importantly, the output can be interleaved and directed to standard output, which can then be directly piped into tools (including splitcode itself if another round of read processing is needed) that support such input. This feature makes it possible to send processed reads directly to a read mapper, therefore eschewing the inefficiencies of creating large intermediate files on disk.

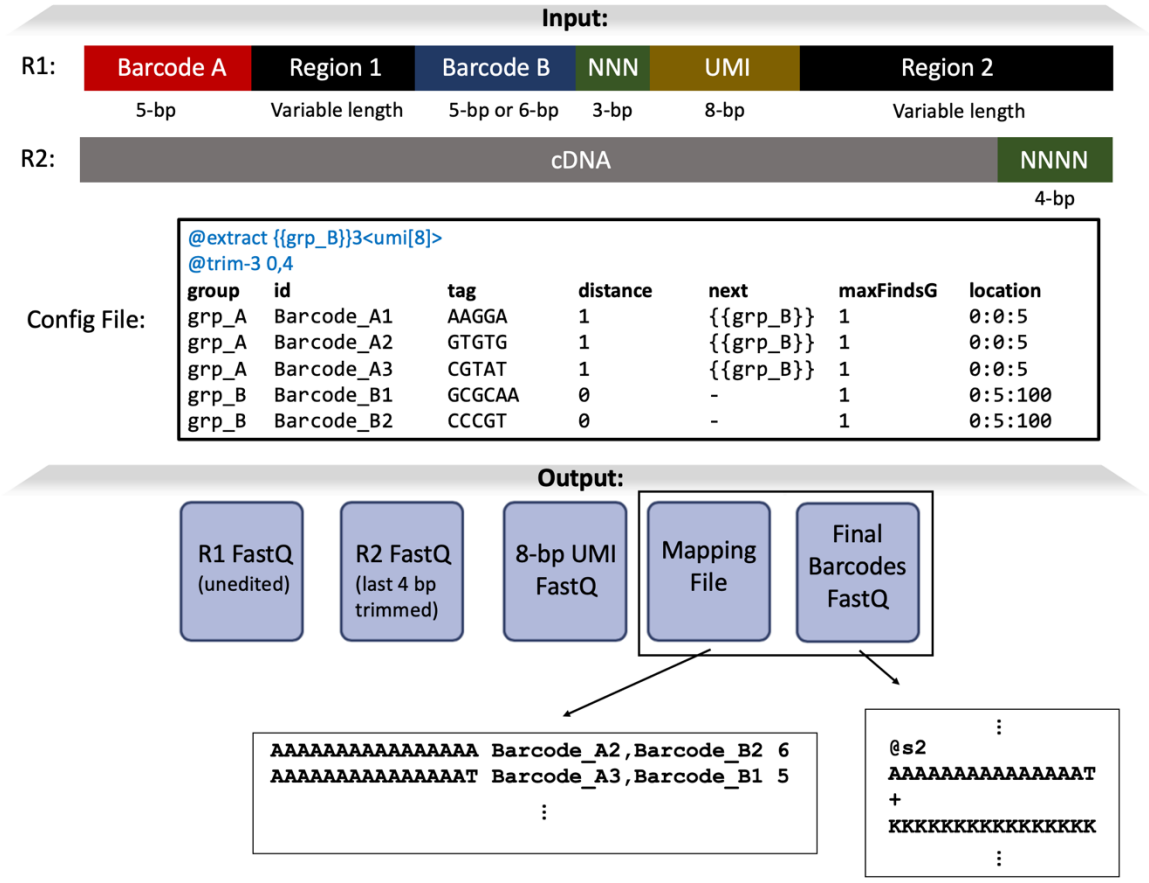



***Figure 2.2**: Example of splitcode usage.*
*The structure of the reads from this hypothetical sequencing technology contains multiple regions that need to be parsed, including some of variable length. In the config file, each region that needs to be parsed is organized into groups and each "group" contains multiple tags. The tags in the grp_A group have the value 1 in the "distance" column, meaning a hamming distance 1 error tolerance. The values in the "next" column indicate that after a grp_A tag (i.e. Barcode_A1, Barcode_A2, or Barcode_A3) is found, we should next search only for tags in the grp_B group. The "maxFindsG" values of 1 mean that the maximum number of times a specific group can be found is 1 (e.g. after finding a tag in grp_A, stop searching for tags in grp_A). The "location" for grp_A tags have the value 0:0:5, meaning that the tag is found in file #0 (i.e. the R1 file) within positions 0–5 of the read; for grp_B tags, splitcode searches file #0 within positions 5–100. In the header of the config file, the @extract option contains an expression indicating that we should extract an 8-bp sequence, which we name umi, 3 bases following identification of a grp_B tag. The supplied @trim-3 option means that only 3'-end trimming of 0 bases and 4 bases of the R1 file and the R2 file, respectively, should be performed. Thus, here, the output R1 file will contain the original R1 sequences (i.e. the entirety of Barcode A, Region 1, Barcode B, NNN, UMI, and Region 2) while the output R2 file will contain just the cDNA. The output "Final Barcodes" FASTQ file will contain a sequence uniquely identifying a combination*

*of tags and the mapping file allows us to map the final barcode sequence back to the tag combination (the numbers in the right-most column of the mapping file represent how many reads that tag combination was found in). Finally, it is important to note that this is simply one of many ways to parse this read structure with splitcode and users can configure the options how they see fit. Further, users can also customize the output options. For example, users can choose to output reads that contain both grp_A and grp_B tags into one set of files and direct all other reads into a separate set of files, and users can choose whether to output the 8-bp UMI sequence into an independent file or to put it in the FASTQ header of the outputted reads. Users also have the option to output reads as a BAM file with the 8-bp UMI sequence encoded in a SAM tag.*

The splitcode program has many options, some of which can be supplied in the config file and others of which (namely the output options) must be supplied on the command line. In the config file, a user can specify "sequence identification" options for finding tags in reads as well as editing reads in situ based on identified tags as well as "read modification and extraction" options for general read trimming and extracting UMI-like sequences. The latter option group is supplied in the header of the config file while the "sequence identification" options are supplied as tab-separated values in a tabular format in the file, an example of which is shown in Figure 2. Finally, splitcode is efficient software: On 150-bp paired-end reads in gzip FASTQ format, splitcode can reach throughputs exceeding 10 million reads per minute with memory usage on the order of a few hundred megabytes on a standard laptop, although these performance results vary depending on the task at hand.

*Tag sequence identification:*

Each sequence in the splitcode config file along with all sequences within the sequence's allowable hamming distance and/or indel error tolerance is indexed in a hash map. Each sequence is associated with the tag(s) from which it originated. Note that sequences can also be supplied in an external text file, which is useful when working with a long list of "cell identification barcodes" as is common in single-cell RNA-seq. Reads in FASTQ files are scanned from start to end to identify tags based on hash map lookups. Additionally, users can specify locations and conditions within which a specific tag may appear and only tags satisfying such conditions are identified. Further, by restricting tag identification to

only specific regions of reads, the number of hash map queries is reduced, therefore improving runtime.

*Final barcode sequences:*

Each combination of tags is assigned a numerical ID, which begins at 0 and is incremented for every newly encountered combination. Each numerical ID, a 32-bit unsigned integer, can be converted to a unique 16-bp final barcode sequence (i.e. a pseudobarcode) by mapping each nucleotide to a 2-bit binary representation as follows: A = 00, C = 01, G = 10, T = 11. It follows that the numerical ID can be represented in nucleotide-space based on the integer's binary representation. For example, the numerical ID 0 is AAAAAAAAAAAAAAAA, the numerical ID 1 is AAAAAAAAAAAAAAAT, and the numerical ID 30 is AAAAAAAAAAAAACTG. This interconversion between numerical IDs and nucleotide sequences facilitates simplifying complex barcodes.

In certain applications, an optimized barcode encoding scheme may be preferable to a sequential numerical assignment, particularly when efficiency or consistency across datasets is required. Such a scheme ensures that each unique combination of tags is deterministically mapped to a fixed final barcode sequence, regardless of dataset composition or order of processing. This can be achieved through a mixed radix encoding algorithm, which leverages the size of each tag group (for example, the number of barcodes in a given round of a split-and-pool combinatorial barcoding strategy) to optimize the representation of combinations. Given each tag group $i$, with $n_i$ possible elements (tags), the total number of bits required to encode all tag groups is:

$$Total\ bits = \left\lceil log_2 \prod_{i=1}^{k} n_i \right\rceil$$

where $k$ is the number of groups. For instance, encoding three tag groups each containing 96 elements requires $\lceil log_2(96)^3 \rceil = 20$ bits, corresponding to a barcode length of 10 nucleotides. The mixed-radix approach calculates a unique numerical ID for each

combination of tags by treating the combination as a mixed-radix number. Each tag value $v_i$ is multiplied by the product of the domain sizes of all preceding groups as follows:

$$Barcode\ ID = \sum_{i=1}^{k} \left( v_i \prod_{j=1}^{i-1} n_j \right)$$

The ID can then be converted into nucleotide space, by mapping each nucleotide to a 2-bit binary representation as before. Decoding the ID into its constituent tag compositions can also be done.

*Discussion*:

The preprocessing of FASTQ files is an important first step in bioinformatics pipelines. This step is frequently inefficient, involving multiple steps with the creation of large intermediate files or writing and running of custom unoptimized scripts which can be challenging with large-scale sequencing data. Splitcode alleviates some of these inefficiencies via a modular and flexible design to effectively and efficiently handle intricate, hierarchical read structures produced by technologies with many layers of multiplexing. While many of splitcode's features overlap with those of existing bioinformatics software, splitcode is not intended to fully recapitulate all the features of existing tools or to replace or outperform any one tool. Rather, splitcode is intended to serve as one additional, flexible and versatile tool in a bioinformatics arsenal, and has been designed to be interoperable with other tools. Indeed, splitcode was designed with the principles of flexibility, modularity, and cross-compatibility (as described in the first chapter of this dissertation) in mind. Splitcode operates not as an alignment algorithm, but on a principle of dictionary lookups. In this approach, technical sequences along with their permissible mismatches are cataloged in a hash table. This makes splitcode apt for scenarios requiring identification, interpretation, and modification of short sequences within reads, and it effectively manages extensive lists of lookup sequences. Algorithms like cutadapt, which use dynamic programming score matrix to optimize alignment, are more suitable for cases, such as general adapter trimming, that require finding the best

possible alignment between two sequences or for finding long technical sequences. This is because in such cases, storing the allowable mismatches in a hash table is computationally infeasible; given the alphabet {A, T, C, G, N} and a sequence of length L, the number of sequences with M substitution mismatches or fewer follows the formula:

$$Number\ of\ sequences = \sum_{k=0}^{M} \binom{L}{k} 4^k$$

In any case, we anticipate that splitcode will be used in tandem with other preprocessing tools to provide an effective solution for many bioinformatics needs. Furthermore, we expect that splitcode will continue to expand in functionality based on user feedback, user needs, and possibly the introduction of more complicated read structures that may arise from the development of novel sequence census assays. Lastly, splitcode was utilized to process portions of the data presented in this dissertation and will be revisited in later chapters.

*Chapter 3*

# ENHANCING SINGLE-CELL AND SINGLE-NUCLEUS QUANTIFICATION

## Quantifying nascent and mature RNAs

The utility of single-cell RNA sequencing (RNA-seq) measurements for defining cell types has represented a marked improvement over bulk RNA-seq, and has driven rapid development and adoption of single-cell RNA-seq assays (Zeng, 2022). One application of single-cell RNA-seq that is not possible with bulk RNA-seq is the study of cell transitions and transcription dynamics, even via snapshot single-cell RNA-seq experiments (Gorin et al., 2023, 2022a; La Manno et al., 2018). Such applications of single-cell RNA-seq are based on the quantification of both unprocessed and processed messenger RNAs (mRNAs) (Figure 3.1), lending import to the computational problem of accurately and separately quantifying these two modalities (Soneson et al., 2021). The importance of quantifying unprocessed mRNAs in addition to processed mRNAs has also been brought to the fore with single-nucleus RNA-seq (Ding et al., 2020; Grindberg et al., 2013; Kuo et al., 2024).

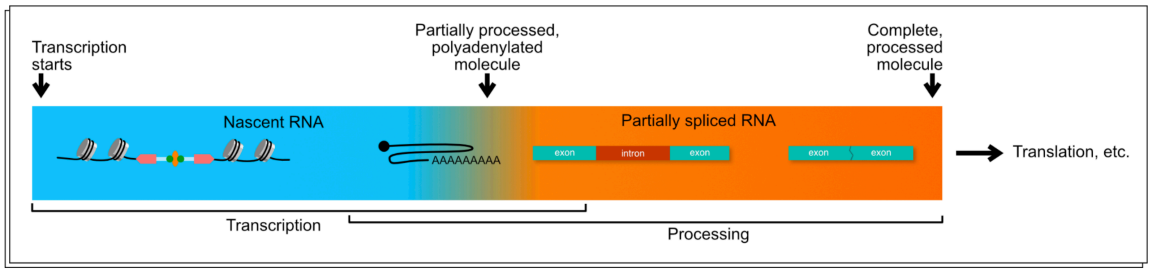



***Figure 3.1****: The maturation process of RNA transcripts.*

The traditional approach in quantifying RNA-seq has been to rely on a reference transcriptome that defines a "region of interest"—typically restricted to mature mRNA transcripts (i.e. no introns) for bulk RNA-seq analyses. This conventional focus has been adequate for the broad objectives of bulk sequencing but is insufficient for the more granular and precise requirements of single-cell and single-nucleus RNA-seq, where the

coexistence of nascent (unprocessed) and mature (processed) messenger RNA (mRNA) poses challenges in accurate read mapping and the interpretation of count matrices. Reads originating outside of the "region of interest" are prone to mismapping within this region, and additionally, such external reads cannot be matched to specific transcript targets. Expanding the "region of interest" to encompass both nascent and mature mRNA transcript targets provides a more comprehensive framework for RNA-seq analysis, enhancing the precision in quantifying both mature and nascent mRNA molecules, as well as in delineating reads of ambiguous status (He et al., 2023; Sullivan et al., 2025).

In a later section of this chapter, the concept of distinguishing flanking k-mers (DFKs) will be described in detail. DFKs are a minimal set of k-mers that can be used to distinguish whether a read that is mapped to a set of targets in the transcriptome index has its origin from within the transcriptome index or has an external origin. These DFKs can therefore address the problem of mismapping of external reads (Kaminow et al., 2021), by acting as a filter to prevent reads of external origin from being mismapped to the transcriptome index. In other words, these k-mers, if present in a read, will cause the read to be filtered out. We use the term D-list (distinguishing list) to denote the sequences from which DFKs are extracted based on the contents of the transcriptome index. By default, the D-list is set to the genome FASTA file. Therefore, hereinafter, specifying the usage of a D-list refers to supplying the genome FASTA file as the D-list. While using standard mature mRNA transcriptome index with a D-list can be used to improve the quantification of single-cell RNA-seq due to intronic and intergenic reads, still, only mRNA transcripts exist in the index and hence, only reads mapping to mature mRNA regions will be considered. While this is useful for certain applications of single-cell analyses such as cell type identification, having only a single-cell count matrix prevents the usage of biophysical models which jointly consider mature and nascent RNA quantifications (Carilli et al., 2024; Gorin and Pachter, 2022a). Thus, extending the index (He et al., 2022; Melsted et al., 2021; Soneson et al., 2021) to allow quantifications of RNA molecules at different stages of their processing is important, as will be discussed next.

To quantify nascent RNA transcripts, it is necessary to extend the transcriptome index to include such targets. That is, an index should be created that encompasses the nascent RNAs and the mature RNAs. While seemingly straightforward to construct such an index and to map reads against it, a difficulty arises from classifying individual reads, or individual unique molecular identifiers (UMIs), as being of "mature" or "nascent" status. This difficulty stems from the fact that sequenced reads are typically much shorter than transcripts, and therefore there can be ambiguity in classification of reads as "mature" or "nascent". Reads that span an exon–exon junction must originate from a completely or partially processed mRNA (which we call "mature"), whereas reads containing sequence unique to an intron must originate from a completely unprocessed or partially processed mRNA (which we call "nascent"). However there are many reads for which it is impossible to know whether they originated from an unprocessed or processed transcript (hence, are ambiguous) (Figure 3.2). Methods that rely on k-mer mapping must account for the distinction between k-mer ambiguity and read ambiguity, and this distinction has not been carefully accounted for in previous k-mer based single-cell RNA-seq pre-processing workflows (He et al., 2022; Melsted et al., 2021).

To classify reads as nascent, mature or ambiguous, we first pseudoalign reads using kallisto (Bray et al., 2016) against a kallisto index containing the mature mRNA (as used originally in pseudoalignment) and nascent mRNA. The nascent mRNA spans the full length of a gene and contains both the gene's exons and introns as a single contiguous sequence. This comprehensive representation, implemented as the nac index in kallisto (Sullivan et al., 2024), allows for accurate classification of reads as mature or nascent or ambiguous, as it properly accounts for the exon–intron boundary and acknowledges that exons are components of both nascent and mature mRNA (Figure 2). The developers of alevin-fry have also adopted this approach in the alevin-fry spliceu index (He et al., 2023), an updated version of the original alevin-fry splici index. However, other tools, such as STARsolo (Kaminow et al., 2021) and the popular Cell Ranger software (Zheng et al., 2017), do not produce such classifications.

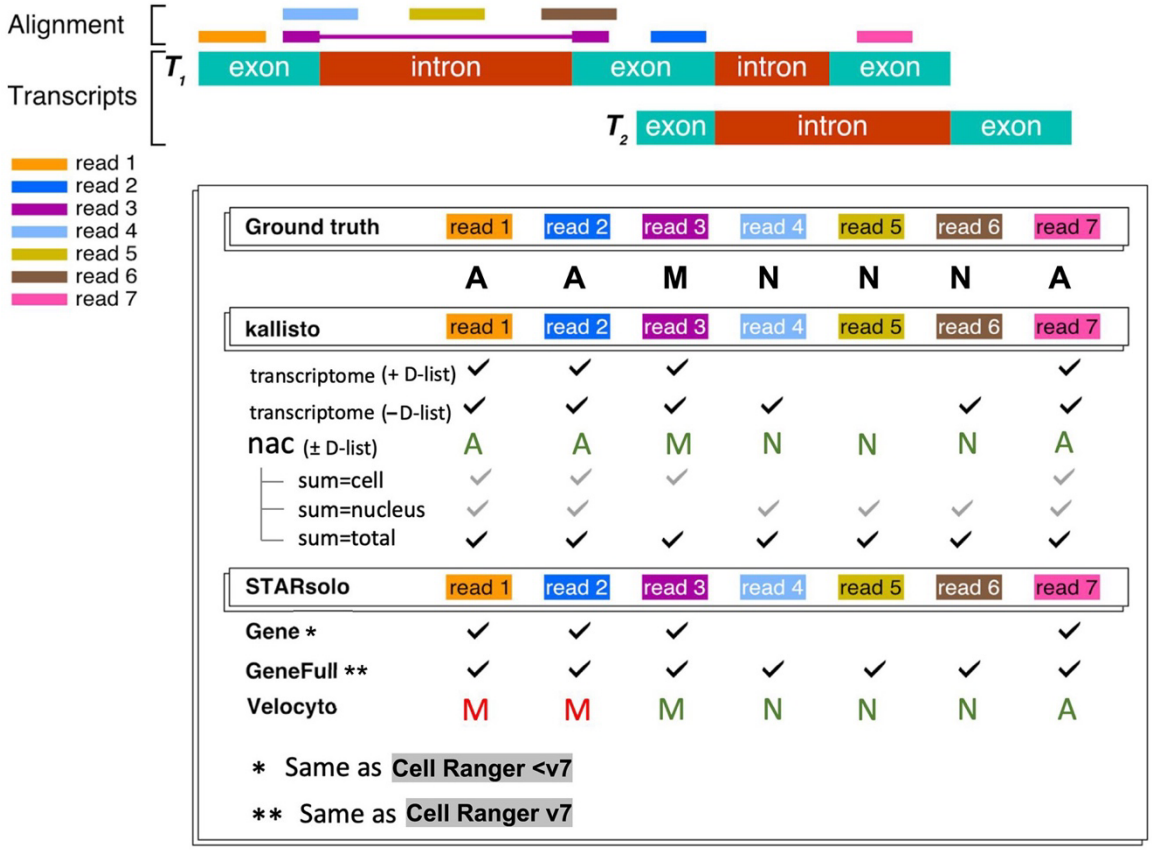



***Figure 3.2***: *Approach to read assignment and classification into nascent, mature, and ambiguous categories by kallisto, STARsolo, and Cell Ranger.*
*This classification of reads enables accurate classification of RNA species, enabling ambiguous (A) reads to be assigned in various ways based on context [e.g. ambiguous reads are allocated to "mature" (M) in single-cell RNA-seq splicing analysis or added to both "mature" (M) and "nascent" (N) in the case of quantifying "total" RNA content]. The kallisto nac index in this example produces the same classification with or without the D-list because no external reads (i.e. those existing outside annotated genomic regions) are present. The standard "transcriptome" index cannot resolve different RNA species and, without the D-list, will result in some reads originating from nascent transcripts (i.e. reads cross exon–intron boundaries) being mapped even though introns do not exist in the index; however, with the D-list, those intronic k-mers in the reads will map to DFKs in the index. The --sum options for the nac index represent the various ways the N, M and A matrices can be summed up (i.e. using M + A for "cell" and N + A for "nucleus" or N + A + M for "total"). Results for alevin-fry are not shown because its spliceu index produces classifications identical to kallisto. The checkmarks represent whether a given read will be counted and the letters M, N, and A represent the read classifications (with red letters denoting classifications that differ from the ground truth).*

While the nac index contains both mature and nascent mRNA, reads of external origin could still arise from intergenic regions of the genome being sequenced. These reads may still be erroneously mapped to this extended transcriptome index. To mitigate the possibility of such instances occurring, one would want to use DFKs by using the nac index with a D-list. This approach is implemented in kallisto by default when building the nac index. Altogether, the nac index, in conjunction with DFKs to mask out reads of external origin, enables the accurate quantification and classification of nascent, mature, and ambiguous mRNA.

*Nascent, mature, and ambiguous classifications:*

The extended transcriptome index (i.e. the nac index type), by virtue of indexing intron-containing nascent transcripts, enables the mapping of a substantial fraction of reads that would otherwise go unmapped when using the standard index type. Furthermore, the quantifications produced by the nac index can classify reads or UMIs as mature (M), nascent (N), or ambiguous (A). We assessed the classifications on both single-cell and single-nucleus data from mouse and human samples. As expected, single-nucleus data tends to have a higher ratio of nascent to mature RNA compared to single-cell data since RNA molecules that have been exported out of the nucleus have undergone splicing and maturation while 10x Genomics Visium spatial transcriptomics data has the lowest proportion of nascent RNA (<1%) due to the Visium kit's exon capture (Figure 3.3). Across the different count matrices, we observe that the total counts (N + M + A) are well-correlated with the ambiguous counts, implying that the results of a single-cell or single-nucleus RNA-seq analyses are largely driven by reads mapped solely within exons. Note that regardless of assay type, there tend to be more UMIs classified as nascent than mature, because introns have a much larger coverage over the genome than exon–exon SJs. The individual N, M, and A count matrices are poorly correlated with one another, reflecting that different information is present in each of those three matrices. As biophysical models of the RNA life cycle make use of nascent transcript counts and mature transcript counts, how to allocate those ambiguous counts to either nascent or mature remains a topic for

future research. For now, one might reasonably assume that the ambiguous counts in single-cell RNA-seq experiments originate from mature transcripts since, in such assays, it is expected that there will be more mature transcripts than nascent transcripts therefore a purely-exonic UMI is most likely to be mature. However, in the nucleus context, the likelihood of a purely-exonic UMI being mature is lower since there will be fewer mature transcripts, as evidenced by the much larger nascent-to-mature ratio in UMI classification. Developing methods to more accurately allocate ambiguous reads is an interesting topic to pursue, and there are now some efforts to do so (He et al., 2024).

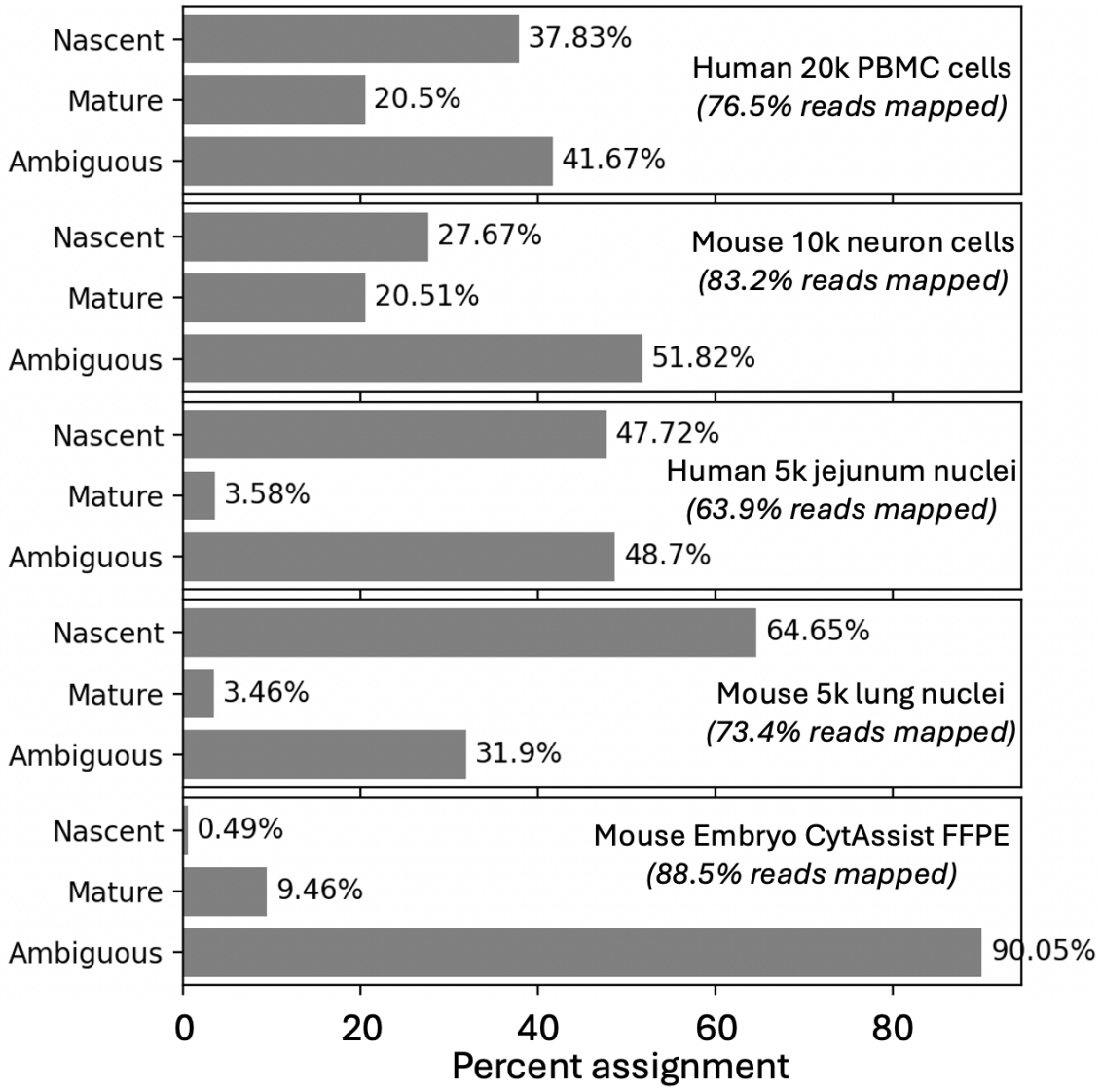



***Figure 3.3****: Quantification of mature and nascent RNA from single-cell and single-nucleus experiments.*
*Exploration of single-cell and single-nuclei count matrices from human and mouse samples (datasets from 10x Genomics). The bar plots show the percentage of UMIs assigned to the ambiguous, nascent and mature classifications. The datasets were downloaded from https://www.10xgenomics.com/ and are as follows: Human 20k PBMC cells (type: single-cell; name: 20k_PBMC_3p_HT_nextgem_Chromium_X; depth: 818,107,363 reads), Mouse 10k neuron cells (type: single-cell; name: SC3_v3_NextGem_SI_Neuron_10K; depth: 1,589,915,447 reads), Human 5k jejunum nuclei (type: single-nucleus; name: 5k_human_jejunum_CNIK_3pv3; depth: 121,378,620 reads), Mouse 5k lung nuclei (type: single-nucleus; name: 5k_mouse_lung_CNIK_3pv3; depth: 232,479,932 reads), Mouse embryo Visium CytAssist 11mm FFPE (type: spatial; name: CytAssist_11mm_FFPE_Mouse_Embryo; depth: 832,193,962 reads).*

Moreover, mature RNA can have multiple isoforms and a comprehensive analysis will identify not only whether a UMI originated from mature RNA produced by a gene but also which mature RNA of that gene the UMI originated from. As pseudoalignment works by identifying a set of targets that a UMI is compatible with, it is straightforward to determine whether those set of targets contain specific isoforms of a gene. To investigate how this may potentially be useful, we utilized SPLiT-seq (Rosenberg et al., 2018) data of mouse myoblasts (Rebboah et al., 2021); the SPLiT-seq data can be found at GEO accession identifier GSE168776 and all seven short read sequencing subpools within that dataset were used. The processing of SPLiT-seq data was performed as follows: Cell barcodes corresponding to C2C12 myoblast cells with at least 10 000 UMIs were extracted based on metadata obtained from the study which produced that dataset (Rebboah et al., 2021). The reads containing those barcodes were divided into oligo-dT reads and random hexamer reads based on the first round barcode sequence. These initial steps were performed using the splitcode program (Sullivan and Pachter, 2024). Next, to mitigate spurious read alignment to low complexity intronic sequences, bowtie2 (Langmead and Salzberg, 2012) was used to align the reads to an index of ribosomal RNAs, transfer RNAs, microRNAs, and repetitive elements, and those reads were removed with seqkit (Shen et al., 2016). STAR (Dobin et al., 2013) was used to align reads to the mouse reference genome to generate a BAM file, which was indexed with SAMtools (Li et al., 2009) and visualized with Integrative Genomics Viewer (Robinson et al., 2011). Kallisto | bustools was used to pseudoalign reads to the mouse nac index (with D-list) and to produce transcript compatibility counts (TCCs). Normalized counts were produced by CP10k normalization followed by log1p transformation and processed with Scanpy (Wolf et al., 2018).

We chose to analyze SPLiT-seq data because, in that technology, the same cell can be sequenced using an oligo-dT priming strategy and a random hexamer priming strategy. These two priming strategies will yield different isoform abundances as the oligo-dT primer selects for the polyA tail of mRNA while the random hexamer does not. An example from the gene Rplp0, which is present in both the oligo-dT library and the random hexamer library, albeit to a lesser extent in the latter, illustrates the difference (Figure 3.4A). As can

be shown via the integrative genomics viewer (IGV) software (Robinson et al., 2011), the random hexamer reads cover the entire gene body, including in intronic regions, while the oligo-dT reads are heavily localized to the 3′ end region of the gene with very few reads elsewhere or in introns (Figure 3.4B). Upon using the nac index to resolve nascent and isoform-level mature RNA (Figure 3.4C), we find that a large number of sequenced molecules from the oligo-dT library are of mature status and, specifically, belong to isoform ENSMUST00000086519, which is a transcript that extends to the 3′ end of the gene (Harrison et al., 2024). Nascent RNA and other isoforms primarily originate from the random hexamer library. Although this analysis was only done at the target-compatibility level (including both nascent and mature RNA as targets in contrast to previous approaches which only included mature RNA), one can use such TCCs directly in cell clustering analysis (Ntranos et al., 2019, 2016). While an expectation-maximization algorithm can attempt to probabilistically assign TCCs to transcript-level estimates, an identifiability problem due to a high degree of ambiguity may preclude robust estimates from being obtained when quantification relies on short reads. Finally, while some work has been done in jointly using nascent and mature RNA counts, as produced by kallisto for biophysically motivated cell clustering analysis (Chari et al., 2024), utilizing isoform-level mature RNA to further enhance such analysis is an avenue for future research (Gorin and Pachter, 2022b).

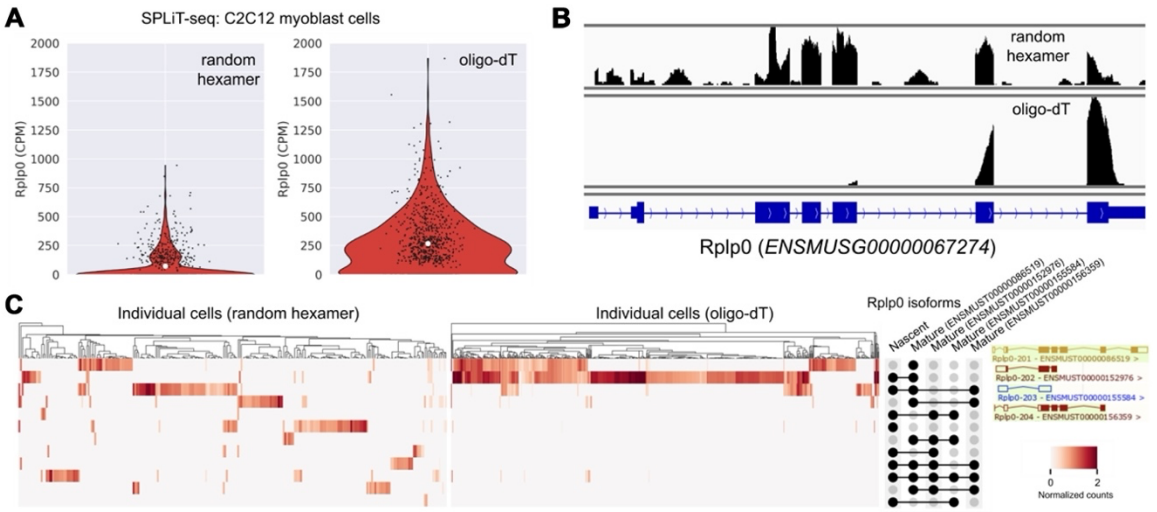



***Figure 3.4**: Isoform compatibility quantification of nascent and mature RNA.*
*(A) Rplp0 gene-level counts (nascent + mature + ambiguous) of mouse C2C12 myoblast cells from a SPLiT-seq single-cell RNA-seq assay, wherein reads from the random hexamer priming strategy were quantified separately from the reads with the oligo-dT priming strategy. The Rplp0 CPMs (counts per million) of individual cells are plotted. (B) Genome browser tracks of the reads aligned to the Rplp0 gene. (C) Normalized TCCs of UMIs assigned to the Rplp0 gene. Each row in the heatmap represents an EC (i.e. a set of transcripts that a UMI is compatible with) and each column represents an individual cell. ECs essentially capture UMI assignment ambiguity between isoforms and, when mapping reads to the nac index, between nascent and mature status. The transcripts that constitute each of the 12 ECs shown are presented in the UpSet plot labels to the right of the heatmap. Each UMI within a given cell is assigned to an EC and a transcript isoform can be present in multiple ECs. The isoform structures shown on the right were obtained from ENSEMBL.*

All in all, while the described scheme, implemented in the nac index, can provide accurate quantification of mature RNA transcripts, nascent RNA transcripts, and ambiguous RNA transcripts (i.e. transcripts that cannot be unambiguously resolved as nascent or mature), how to jointly utilize these three types of RNA transcripts remains an avenue for future research. One approach to "integrating" the nascent and mature modalities is via biophysical modeling of transcription (Carilli et al., 2024; Gorin et al., 2023; Gorin and Pachter, 2022a); however questions remain, such as how to best utilize reads that are ambiguous between the modalities. Importantly, there is not one single "count matrix"; rather, there are multiple count matrices that each lend value in single-cell and single-

nucleus RNA-seq analyses. The number of count matrices becomes even larger when considering technologies such as SPLiT-seq (Rosenberg et al., 2018), for which two different priming strategies (oligo-dT and random hexamer) exist for a single cell, or Smart-seq3 (Hagemann-Jensen et al., 2020), for which two complementary DNA (cDNA) fragment types (UMI and internal) exist, thus resulting in an additional set of count matrices. The ability to differentiate and quantify nascent, mature, and ambiguous transcripts offers a more nuanced view of gene expression, potentially enriching our understanding of RNA processing and transcriptome dynamics.

There are several limitations to the quantification framework we have proposed. In a cell, the set of unprocessed mRNAs at any given time is likely to include partially processed molecules (Pai et al., 2018; Pandya-Jones and Black, 2009), and in principle the complete splicing cascade must be understood and known in order to accurately quantify single-nucleus or single-cell RNA-seq data. Furthermore, the presence of ambiguous reads both for single-cell and single-nucleus RNA-seq is unsatisfactory. Ideally reads should be longer so that they can be uniquely classified, or they should be fractionally classified probabilistically.

Nevertheless, this work introduces a method for improving the accuracy of generating count matrices. It is anticipated that these improved quantifications and the multimodal nature of these quantifications will prove useful for multiple downstream applications, including both total gene expression quantification and the integration of multiple count matrices via biophysically informed models.

**Distinguishing flanking k-mers**

As mentioned in the previous section, DFKs are a minimal set of k-mers that can be used to distinguish whether a read that is mapped to a set of targets in the transcriptome index has its origin from within the transcriptome index or has an external origin. Essentially, they enable accurate quantification of RNA-seq reads in experiments where reads that are not an expression of the target transcriptome may still contain sequences which do occur in the target transcriptome (Sullivan et al., 2025). Without these DFKs, these reads may be erroneously quantified as transcripts in the target transcriptome (Kaminow et al., 2021), based on alignment of the common sequences. Thus, they serve as a sort of sophisticated "background filter". The D-list (distinguishing list) represents the sequences from which DFKs are extracted based on the contents of the transcriptome index. The transcriptome index is a colored de Bruijn graph (Iqbal et al., 2012) that sequencing reads can be mapped/aligned against (i.e. determining which transcript(s), represented as colors in the graph, that a read might have originated from); this mapping process is based on identifying k-mers shared between the sequencing reads and the de Bruijn graph via a process termed pseudoalignment (Bray et al., 2016). The D-list may contain any sequences that are not desired in the abundance matrix yielded by the quantification. Such sequences may include genomes of other organisms (Luebbert et al., 2025) to avoid mismapping due to sample contamination, they may consist of the genome from which the target transcriptome was made, or they may contain common transposable elements, such as Alu regions, which might confound analyses. The D-list is incorporated into the index by finding all sequences, k base-pairs or longer, that occur in both the D-list and the target transcriptome. The first k-mer upstream and the first k-mer downstream of each such common sequence in the D-list are added to the index-colored de Bruijn graph (dBG). We refer to these new vertices in the graph as DFKs. The DFK vertices are left uncolored in the index, such that during quantification, reads that contain them will be masked out, and go unaligned.

As an illustration of how DFKs work, consider a read containing both k-mers found only in intergenic RNA and k-mers found both in mRNA and the intergenic RNA. If that read

is mapped to an index built from mRNA transcripts, the mRNA k-mers will be found in the index, whereas the disambiguating genome k-mers will not. The whole read will be erroneously mapped based on the ambiguous k-mers (i.e. the k-mers found both in mRNA and the intergenic RNA) that are present in the index. By finding all ambiguous k-mers in the mRNA index, and adding any distinguishing flanking genome k-mers to the index, the read will be masked from mapping to an mRNA transcript (Figure 3.5).

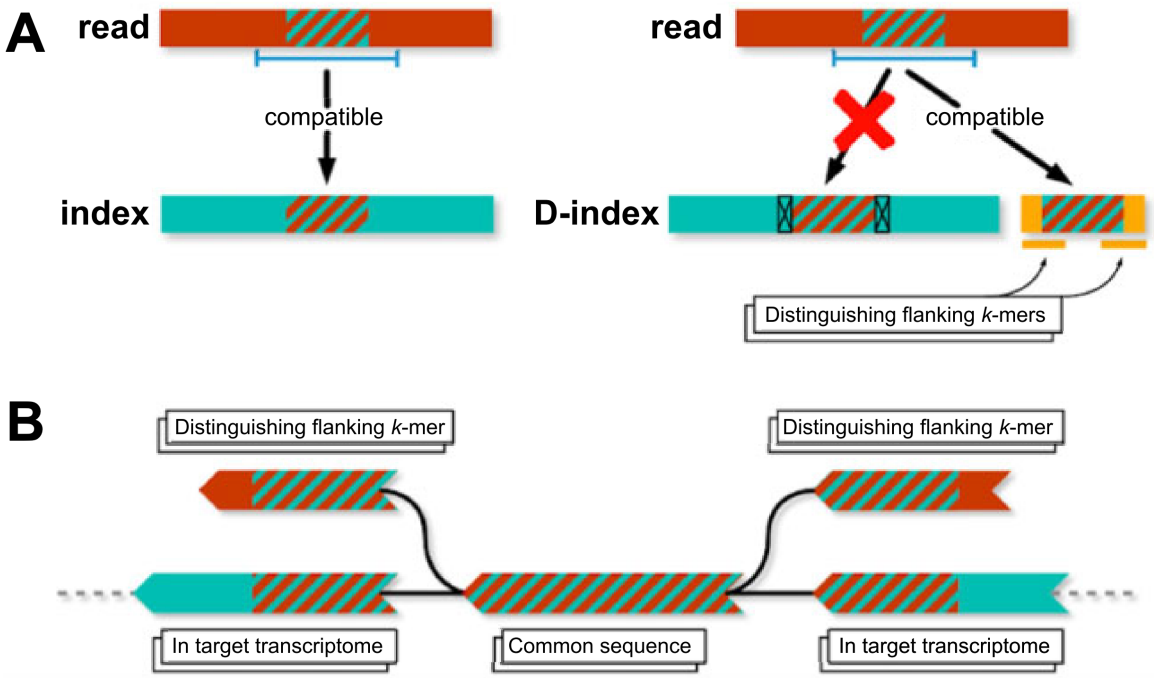



***Figure 3.5**: Overview of DFKs.*
*(A) A nontranscriptomic read containing a subsequence of length greater than k, which also occurs in a transcript in the target transcriptome index, will get attributed to that transcript. DFKs, here shown in the modified index (the D-index), can be used to determine whether a read compatible with a reference transcriptome may have originated from elsewhere in the genome. In this diagram, the D-index is one constructed with DFKs. The hatched region depicts the k-mers shared between the read and the index, and the line underneath the read shows the sequence stretch spanning both those shared k-mers plus the DFKs. (B) A dBG representation of DFKs.*

Recent papers have discussed various ways of reducing the number of false positives in RNA quantification through either including the entire genome or a subset of the genome in the index as a "decoy" or through alignment scoring (Srivastava et al., 2020). The D-list

method is distinct in that it incorporates only the minimum amount of data, required to disambiguate common sequences, into the index while still adhering strictly to the principles of k-mer based pseudoalignment. Therefore, the memory usage and runtime of using pseudoalignment using a D-list are on par with the memory usage and runtime without the use of a D-list. This method can scale favorably to larger genome size (or, more generally, larger D-lists) while the target sequences to map against remain small.

The algorithm on the next section details the procedure in more detail, and the lemma that follows demonstrates the space-efficiency of DFKs.

*Generating DFKs:*

A sequence $s$ is a string of symbols drawn from an alphabet $\Sigma = \{A, T, C, G\}$. The length of $s$ is denoted by $|s|$. A substring of $s$ is a string that occurs in $s$: it has (zero-indexed) start position $i$ and end position $j$ and is denoted by $s[i:j]$, therefore $|s[i:j]|$ is equal to $j - i$. In the case that $|s[i:j]|$ equals $k$-mer size $k$, $s[i:j]$ is $k$-mer. A compact *de Bruijn* graph (cdBG) is a dBG where all maximal nonbranching paths of vertices from a dBG, wherein each vertex is a $k$-mer, are merged into single vertices (21). Each vertex in a cdBG is a sequence called a unitig. We define a cdBG $U$ as a set where each element $u \in U$ is a unitig. The function Map($s$, $u$) takes in a $k$-mer $s$ and a unitig $u$, and returns the position of $s$ along $u$ if $s$ exists in $u$, or NULL otherwise. The following algorithm applies these definitions toward identifying *DFKs* from a D-list $D$, given a cdBG $U$ of $k$-mer size $k$ built over target sequences (e.g. a transcriptome). For expository purposes, the algorithm is described such that $U$ is a nonbidirected cdBG (i.e. the $k$-mers and their reverse complements are *not* represented identically). However, in practice each $k$-mer and its reverse complement are represented as a single canonical $k$-mer (the lexicographic minimum of the $k$-mer and its reverse complement). Additionally, for simplicity, we define DFKs and describe the algorithm only for single overhangs (i.e. the flanking sequences that make up the DFKs will not be more than one k-mer long).

**Algorithm** Generate distinguishing flanking k-mers from a D-list

**Input:** Set of sequences $D$ constituting the D-list, $k$-mer size $k$, cdBG $U$

```
1:  function GenerateDFKs(D, k, U)
2:      DFKs ← ∅
3:      for each seq ∈ D do
4:          pos ← 0
5:          lb ← −1          ▷ Lower bound of "common sequence" (position of leading DFK)
6:          ub ← −1          ▷ Upper bound of "common sequence" (position of trailing DFK)
7:          N ← |seq| − k + 1                                  ▷ Number of k-mers in seq
8:          while pos ≤ N do
9:              s ← NULL
10:             if pos ≠ N then
11:                 s ← seq[pos : pos + k]
12:             if s ≠ NULL ∧ ∃u ∈ U : Map(s, u) ≠ NULL then    ▷ If k-mer s present in cdBG
13:                 if lb = −1 then
14:                     lb ← pos − 1
15:                 η ← Map(s, u)
16:                 while s ≠ NULL ∧ Map(s, u) = η do
17:                     pos ← pos + 1, η ← η + 1    ▷ Extend mapping of sequence onto unitig
18:                     s ← NULL
19:                     if pos ≠ N then
20:                         s ← seq[pos : pos + k]
21:                 ub ← pos
22:             else
23:                 if 0 ≤ lb ≤ ub then
24:                     DFKs ← DFKs ∪ {seq[lb : lb + k]}          ▷ Add leading DFK
25:                 if lb < ub ≤ |seq| − k then
26:                     DFKs ← DFKs ∪ {seq[ub : ub + k]}          ▷ Add trailing DFK
27:                 pos ← pos + 1
28:                 lb ← −1, ub ← −1
29:     return DFKs
```

**Lemma**. The worst case space complexity of DFKs is $O(\min(N_k, M_k))$ where $N_k$ and $M_k$ are the number of unique $k$-mers in the de Bruijn graph (dBG) and D-list, respectively.

**Proof**. Considering the alphabet $\Sigma = \{A,T,C,G\}$, $\forall s \in$ dBG, the maximum number of flanking $k$-mers on each side of $s$ is $|\Sigma|$, permitting a flanking $k$-mer for each character in the alphabet. On each side of $s$, the maximum number of DFKs, which are the flanking $k$-mers in the D-list but not in the dBG, is $|\Sigma| - 1$ corresponding to the presence of one flanking $k$-mer that exists in the dBG and the remaining $|\Sigma| - 1$ $k$-mers being DFKs. Since $s$ has two sides (leading and trailing), the maximum number of DFKs becomes $2(|\Sigma| - 1) = 6$. In the worst-case scenario, $\forall s \in$ dBG, $s$ contains the maximum number of DFKs. Thus, $|\text{DFKs}| \leq 6N_k$ where $|\text{DFKs}|$ is the cardinality of the set of DFKs. The actual number of DFKs identified from the D-list is bounded by the number of unique $k$-mers in the D-list, denoted as $M_k$, i.e., $|\text{DFKs}| \leq M_k$. Since $|\text{DFKs}| \leq \min(6N_k, M_k)$, the space complexity for storing DFKs is $O(\min(N_k, M_k))$. ■

*Implementation of the D-list and DFKs in software:*

The D-list is implemented in kallisto version 0.50.1 (Bray et al., 2016; Sullivan et al., 2024). The kallisto index command contains a --d-list option that takes in, as an argument, the path to a FASTA file containing the D-list sequences for building an index with the D-list. The kallisto index command also contains a --d-list-overhang option for specifying longer overhangs (i.e. extending the flanking sequences that make up the DFKs). The kallisto bus command (20) contains a --dfk-onlist option that, when enabled, adds a D-list target to the equivalence class (i.e. the multi-set of transcripts associated with a read) for a given pseudoalignment if a DFK is encountered rather than discards the read; this option is useful for distinguishing reads that do not pseudoalign versus reads that are discarded due to a DFK. Finally, in kb-python (version 0.28.0), the kb ref command automatically uses the genome FASTA as the D-list when building the kallisto index—a behavior that can be overwritten by explicitly specifying --d-list in kb ref. As a minor nuance, the default genome FASTA D-list does not contain splice junctions (SJs); however, the number of additional DFKs that would be indexed with the inclusion of SJ-spanning sequences is miniscule since SJ-spanning contigs are only $k - 1$ number of k-mers in length. Therefore, including the spliced transcriptome in the D-list would be unlikely to make any difference in read mapping.

Additionally, in the updated kallisto software, the dBG implementation was replaced with Bifrost (Holley and Melsted, 2020), which employs a minimizer (Roberts et al., 2004) lookup table in lieu of a k-mer lookup table in order to achieve a lower memory footprint. Furthermore, since the set of minimizers in the graph is known at the time of quantification, we replaced the minimizer hash function with BBHash (Limasset et al., 2017), which implements a minimal perfect hash function. This enables kallisto to shrink the minimizer hash table to capacity, saving memory. Additionally, the equivalence class (EC) data structure was redone. Sets of transcripts are represented as Roaring bitmaps (Chambi et al., 2016) and a Robin Hood hash map (https://github.com/martinus/unordered_dense) is used for the inverted hash table mapping transcript sets to EC. The Robin Hood hash map data

structure is also for storing DFKs since, as the number of DFKs is relatively small, having a separate hash map to store DFKs occupies less memory, with only a small reduction in speed, compared with integrating the DFKs into the main dBG (Figure 3.6). These changes have resulted in an approximately 2× reduction in runtime and 4× reduction in memory consumption in kallisto v0.50.1 compared with kallisto version 0.48.0 when using the nac index type to map single-cell RNA-seq reads (Figure 3.6).

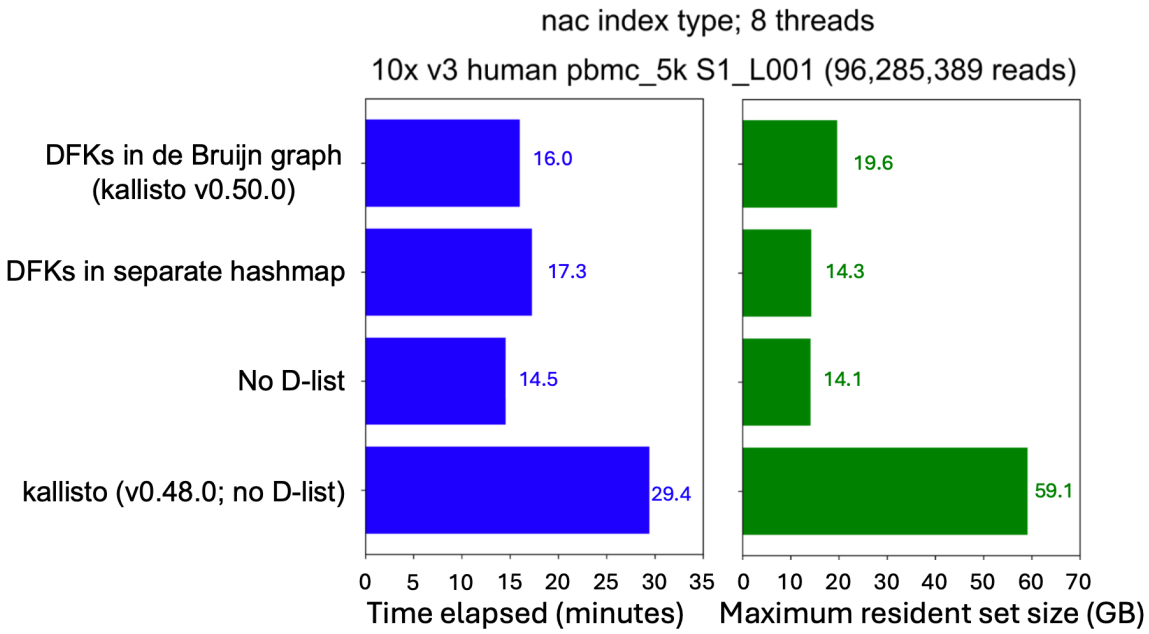



***Figure 3.6****: Performance comparison of different implementations of the kallisto nac index type (which contains both the nascent and mature transcriptomes), when assessed on sequencing reads on a dataset produced by 10x Genomics.*

The kallisto software is available under the BSD-2-Clause license and is available at https://github.com/pachterlab/kallisto.

*Benchmarking:*

We obtained the simulation framework developed by the authors of STARsolo (Kaminow et al., 2021) from https://github.com/dobinlab/STARsoloManuscript/ and ran the simulation as-is to generate a ground truth matrix. For the kallisto nac index type, the "mature" and "ambiguous" count matrices were summed up by using --sum=cell in kb

count and the resultant matrix was used for testing. For all tools, a predefined "on list" of barcodes (referred to in other tools as a whitelist or an unfiltered permit list) was supplied. The three simulated sequencing datasets used are as follows:

- No multigene: 339 million reads
- With multigene: 350 million reads
- Exon-only, no multigene: 189 million reads

The command /usr/bin/time -v which executes the GNU time program was used to obtain the elapsed (wall clock) time and the maximum resident set size for runtime and peak memory usage, respectively. All performance assessments were conducted on a server with x86-64 architecture, 88 CPUs (Intel Xeon Gold 6152 CPU @ 2.10GHz) and 768 GB of memory.

To evaluate the performance of a program's output gene count matrix $G_p \in \mathbb{R}^{n \times m}$ against a simulation's ground truth gene count matrix $G_s \in \mathbb{R}^{n \times m}$, where $n$ is the number of cells, $m$ is the number of genes, $\hat{y}_{ij}$ is the count of gene $j$ in cell $i$ in $G_p$, and $y_{ij}$ is the count of gene $j$ in cell $i$ in $G_s$, the following metrics are used:

Root mean squared error (RMSE):

$$RMSE = \sqrt{\frac{1}{nm}\sum_{i=1}^{n}\sum_{j=1}^{m}(y_{ij} - \hat{y}_{ij})^2}$$

False positive representation (FPR):

$$FPR = \frac{1}{nm}\sum_{i=1}^{n}\sum_{j=1}^{m}\begin{cases} 1 & \text{if } y_{ij} = 0 \text{ and } \hat{y}_{ij} > 0 \\ 0 & \text{otherwise} \end{cases}$$

False negative representation (FPR):

$$FNR = \frac{1}{nm} \sum_{i=1}^{n} \sum_{j=1}^{m} \begin{cases} 1 & \text{if } y_{ij} > 0 \text{ and } \hat{y}_{ij} = 0 \\ 0 & \text{otherwise} \end{cases}$$

Note that the FPR and FNR are defined such that the denominator is the total size of the matrix and therefore differ from the traditional false positive rate and false negative rate calculations.

Correlation coefficient: We use two per-cell correlation coefficients to assess the correlation between the "ground truth" simulated gene counts cell and the program's output gene counts for a given cell. The first, $r$, is the Pearson correlation computed across all genes within a given cell. The second, $\rho^*$, is a modified variant of the Spearman correlation in that the Spearman correlation is computed only using the genes that have a nonzero count in both the simulation and the program output within a given cell. This variant is the assessment used by the developers of the STARsolo simulations (Kaminow et al., 2021). The restriction to nonzero cells is necessary when using the Spearman correlation, as the zeroes cannot be ranked with respect to each other. However, we note that use of the Spearman (and therefore ignoring the zeroes) provides an assessment that is highly sensitive to low counts, especially the difference in a program reporting a one or a zero for a gene in a cell.

Pearson correlation for cell $i$ using all genes:

$$r_i = \text{pearson}\left(\{(y_{ij}, \hat{y}_{ij}),\ j = 1, \ldots, m\}\right)$$

Spearman correlation for cell $i$, using only genes with a nonzero count in both the simulation and the program output for that cell:

$$\rho_i* = \text{spearman}\left(\{(y_{ij}, \hat{y}_{ij}) \mid (y_{ij}, \hat{y}_{ij}) \neq (0, 0),\ j = 1, \ldots, m\}\right)$$

Further (downstream) analyses of count matrices were performed following methods described in other work, which should be referred to for a more detailed description (Rich et al., 2024). Briefly, after filtering the count matrix for a minimum of 3 cells per gene, a minimum of 200 genes per cell, and a maximum 20% mitochondrial gene content, count data were CP10k normalized then log1p transformed. Highly variable genes were selected for, then normalized gene counts were scaled to zero mean and unit variance. Nearest neighbor graphs were constructed from the cell coordinates on the top 50 principal component analysis (PCA) embeddings. Clusters were formed from the Leiden algorithm (Traag et al., 2019) and then visualized on alluvial plots and Uniform Manifold Approximation and Projection (UMAP) plots (Becht et al., 2019; McInnes et al., 2018). These processing steps, as well as selection of significant (adjusted p-value < 0.05) marker genes across all clusters, were performed using Scanpy (Wolf et al., 2018).

Code for the analysis is available at https://github.com/pachterlab/SHSOHMP_2024.

Unless stated otherwise, the software versions used are as follows: kallisto 0.50.1, bustools 0.43.2, kb-python 0.28.0, salmon 1.10.0, alevin-fry 0.8.2, simpleaf 0.15.1, Rsubread 2.12.3, Cell Ranger 7.0.1, STAR 2.7.9a, Bowtie2 2.5.3, seqkit 2.8.0, SAMtools 1.19.2, Scanpy 1.9.5 and splitcode 0.30.0. Additionally, Bandage version 0.8.1 (Wick et al., 2015) was used for rendering dBGs into ribbon-like representations. The human reference genome (GRCh38) used throughout is the same one used in the STARsolo simulations (Kaminow et al., 2021). The GRCh38 FASTA and GTF files used are available from the previously mentioned code repository. The mouse reference genome (GRCm39) used is the primary assembly FASTA file from Ensembl with the corresponding GTF annotation version 110, which was filtered to only include the gene_biotype values of protein_coding, lncRNA, lincRNA, and antisense. These references were used for the analyses throughout both this section and the previous section of this chapter of this dissertation.

*Results:*

To assess improvement of using pseudoalignment with DFKs on single-cell RNA-seq reads, we used the simulation framework developed by the authors of STARsolo (Kaminow et al., 2021). In that simulation framework, errors were introduced into reads at 0.5% mismatch error rate, and reads were simulated from both coding and noncoding genomic sequence to mimic the presence of both unprocessed, partially processed and completely processed transcripts in single-cell RNA-seq experiments. The top 5000 barcodes, based on UMI count from the simulated data, were used for analysis (Figure 3.7A). When quantifying simulated reads that only span exons with kallisto, the DFKs produced by a D-list do not considerably affect quantification accuracy (Figure 3.7B). However, upon including reads that span introns, the D-list improves the concordance between kallisto quantification count matrix and the simulated truth count matrix in both simulations that only include reads that map uniquely to one gene (Figure 3.7C) and in simulations that additionally include multigene reads (Figure 3.7D). Interestingly, although the nac index type includes nascent and mature transcripts, the quantification accuracy still improves slightly with the use of a D-list, likely due to filtering out reads that originate from outside annotated genic loci. The evaluation metrics are shown in Table 3.1. Note that, for the nac index type, UMIs assigned to nascent transcripts were not used in the quantification because the simulation truth matrix does not include nascent transcript counts. For the multigene case, bustools was run with the multimapping option enabled when counting UMIs following kallisto quantification. While this mode can identify nonuniquely mapped reads by dividing UMI counts uniformly amongst the genes that the UMI is assigned to, it results in counts that are not whole numbers; thus, the standard for the field has been to discard such UMIs.

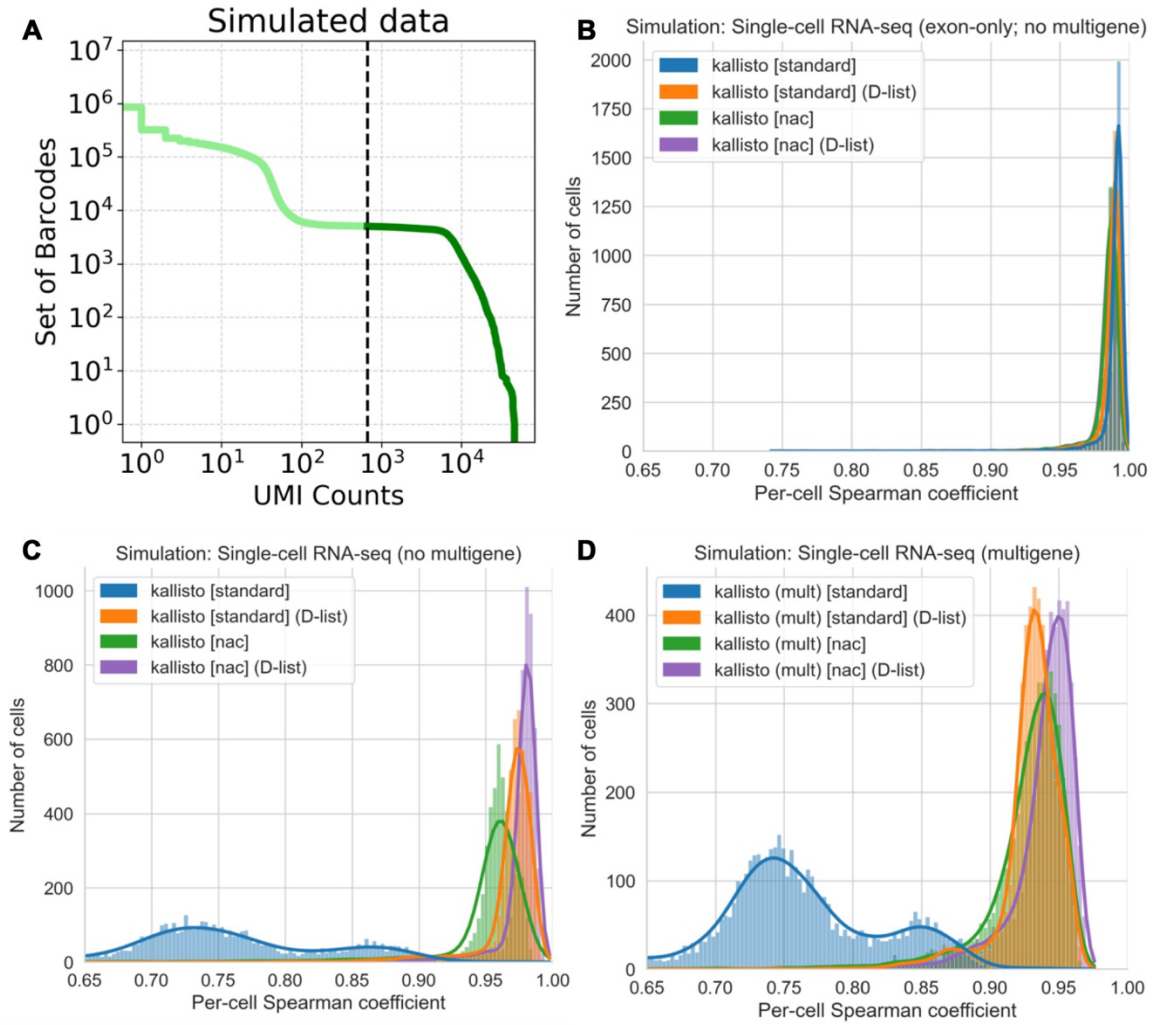



***Figure 3.7****: Assessment of the impact of DFKs on accuracy when tested on simulated data generated using the STARsolo simulation framework.*
*(A) Knee plot of the truth count matrix from the STARsolo single-cell RNA-seq simulation. These simulated data represent the "no multigene" simulation. The 5000 cell barcodes with the highest UMI counts were filtered for (corresponding to a UMI threshold of 667). These 5000 cell barcodes were used in downstream analysis of all STARsolo simulated data. (B) Correlation between kallisto quantifications versus simulated truth for reads only spanning exons. (C) Correlation between kallisto quantifications versus simulated truth for single-cell RNA-seq reads that map to a single gene. (D) Correlation between kallisto quantifications versus simulated truth for single-cell RNA-seq reads that include multigene reads. mult: the multimapping quantification mode is enabled. The per-cell spearman correlation, ρ*, between gene counts was determined by excluding genes that contain zero counts in both the kallisto quantification and in the simulation quantification for a given cell barcode.*

| Index type | D-list | mult | Median ρ* | Median r | RMSE | FPR | FNR | *k*-mers | DFKs |
|---|---|---|---|---|---|---|---|---|---|
| Simulation: Single-cell RNA-seq (exon-only; no multigene) | | | | | | | | | |
| standard | | | 0.991464 | 0.999953 | 0.04132 | 0.000056 | 0.000367 | 113,209,587 | 0 |
| standard | ✓ | | 0.988987 | 0.999922 | 0.052084 | 0.00004 | 0.000487 | 113,209,587 | 4,333,316 |
| nac | | | 0.986443 | 0.999888 | 0.062731 | 0.000038 | 0.000595 | 1,398,470,117 | 0 |
| nac | ✓ | | 0.985968 | 0.999875 | 0.066492 | 0.000038 | 0.000615 | 1,398,470,117 | 11,119,173 |
| Simulation: Single-cell RNA-seq (no multigene) | | | | | | | | | |
| standard | | | 0.742376 | 0.993061 | 0.626481 | 0.019403 | 0.000328 | 113,209,587 | 0 |
| standard | ✓ | | 0.973168 | 0.999815 | 0.081196 | 0.000808 | 0.000481 | 113,209,587 | 4,333,316 |
| nac | | | 0.959529 | 0.996173 | 0.484103 | 0.000646 | 0.000594 | 1,398,470,117 | 0 |
| nac | ✓ | | 0.980208 | 0.999809 | 0.084899 | 0.000206 | 0.000615 | 1,398,470,117 | 11,119,173 |
| Simulation: Single-cell RNA-seq (multigene) | | | | | | | | | |
| standard | | ✓ | 0.751159 | 0.991926 | 0.665475 | 0.051278 | 0.000265 | 113,209,587 | 0 |
| standard | ✓ | ✓ | 0.932976 | 0.999174 | 0.168532 | 0.00475 | 0.00048 | 113,209,587 | 4,333,316 |
| nac | | ✓ | 0.933767 | 0.995264 | 0.524677 | 0.037164 | 0.000568 | 1,398,470,117 | 0 |
| nac | ✓ | ✓ | 0.945177 | 0.999168 | 0.169827 | 0.032405 | 0.000595 | 1,398,470,117 | 11,119,173 |

***Table 3.1***. *Evaluation metrics of kallisto on simulated data generated using the STARsolo simulation framework.*

Since each DFK is only one k-mer flanking a unitig, we then sought to assess whether considering more k-mers flanking a unitig as DFKs (i.e. longer overhangs) would improve the accuracy of kallisto (Figure 3.8A). We found that the benefit of including longer overhangs is negligible (Figure 3.8B and 3.8C; Table 3.2); therefore, by default, we adhere to having exactly one DFK overhang.

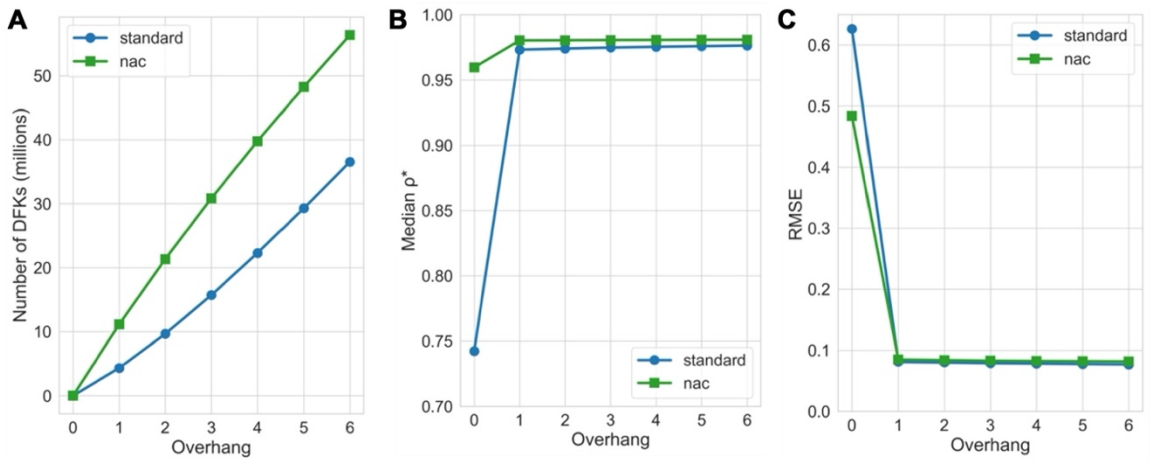



***Figure 3.8****: Assessment of the impact of longer overhang DFKs on accuracy when tested on simulated data generated using the STARsolo simulation framework.*
*(A) The number of DFKs at various overhang settings. An overhang of 0 means no DFKs were used. An overhang of 1 is the default setting for the D-list implementation. (B) Median correlation coefficient ρ* between kallisto quantifications at various D-list overhang settings versus simulated truth for the "single-cell RNA-seq (no multigene)" simulation. (C) RMSE between kallisto quantifications at various D-list overhang settings versus simulated truth for the "single-cell RNA-seq (no multigene)" simulation.*

| Index type | Overhang | mult | Median ρ* | Median r | RMSE | FPR | FNR | *k*-mers | DFKs |
|---|---|---|---|---|---|---|---|---|---|
| Simulation: Single-cell RNA-seq (no multigene) | | | | | | | | | |
| standard | 2 | | 0.973984 | 0.999819 | 0.080112 | 0.000739 | 0.000495 | 113,209,587 | 9,649,025 |
| standard | 3 | | 0.974748 | 0.999824 | 0.078881 | 0.000685 | 0.000506 | 113,209,587 | 15,696,903 |
| standard | 4 | | 0.97535 | 0.999826 | 0.07816 | 0.000643 | 0.000513 | 113,209,587 | 22,293,140 |
| standard | 5 | | 0.975842 | 0.99983 | 0.077472 | 0.000608 | 0.000519 | 113,209,587 | 29,278,838 |
| standard | 6 | | 0.976265 | 0.999833 | 0.076833 | 0.00058 | 0.000524 | 113,209,587 | 36,550,263 |
| nac | 2 | | 0.980336 | 0.999812 | 0.083905 | 0.000202 | 0.000618 | 1,398,470,117 | 21,338,179 |
| nac | 3 | | 0.980491 | 0.999814 | 0.08297 | 0.000197 | 0.000619 | 1,398,470,117 | 30,833,829 |
| nac | 4 | | 0.980585 | 0.999816 | 0.082474 | 0.000193 | 0.00062 | 1,398,470,117 | 39,760,674 |
| nac | 5 | | 0.980698 | 0.999817 | 0.082168 | 0.000189 | 0.000621 | 1,398,470,117 | 48,233,865 |
| nac | 6 | | 0.980771 | 0.999819 | 0.081806 | 0.000185 | 0.000622 | 1,398,470,117 | 56,344,346 |

mult: the multimapping quantification mode is enabled (not enabled for any of the runs here). Overhang: The number of *k*-mers flanking a unitig to be considered a DFK.
ρ*: Modified spearman correlation. r: Pearson correlation. RMSE: root mean squared error. FPR: false positive representation. FNR: false negative representation.

***Table 3.2**. Evaluation metrics of kallisto on simulated data as a function of DFK overhang.*

Next, we assessed the performance of other tools using the STARsolo simulation framework. Specifically, we assessed four tools: (i) STARsolo (Kaminow et al., 2021), a single-cell/nucleus RNA-seq tool built into the STAR aligner program (Dobin et al., 2013), (ii) Cell Ranger (Zheng et al., 2017), the pipeline implemented by 10x Genomics, (iii) cellCounts (Liao et al., 2023), a tool based on the Rsubread aligner (Liao et al., 2019) and the featureCounts program (Liao et al., 2014), and (iv) alevin-fry (He et al., 2022; He and Patro, 2023), a tool that leverages salmon (Patro et al., 2017) for pseudoalignment. We found that the tools produced quantifications that correlated well with the simulated ground truth for the simulated reads that only span exons (Figure 3.9A; Table 3.3). However, on

simulations including intronic reads, both alevin-fry, when executed in a standard pseudoalignment configuration against a spliced transcriptome, and cellCounts performed less well compared with STARsolo and Cell Ranger (Figure 3.9B; Table 3.3). In the case of alevin-fry, using an expanded index that includes introns eliminated this decrease in performance, which is consistent with prior reports (He et al., 2022). Enabling selective alignment (Srivastava et al., 2020) in alevin-fry resulted in further accuracy improvements, similar to the improvements yielded by the D-list, even when used with an expanded transcriptome index. As the same simulated data were used in Table 3.3 and Table 3.1, the results are directly comparable between kallisto (Table 3.1) and other software (Table 3.3).

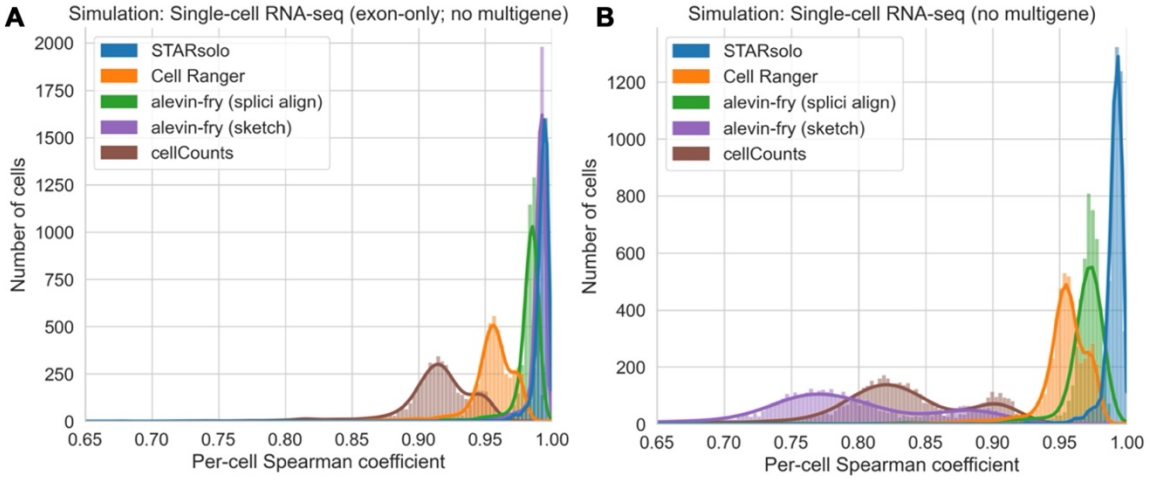



***Figure 3.9****: Assessment of different tools on simulated data generated using the STARsolo simulation framework.*
*(A) Correlation between quantifications produced by the tools versus simulated truth for reads only spanning exons. (B) Correlation between quantifications produced by the tools versus simulated truth for single-cell RNA-seq reads that map to a single gene. Evaluation against multigene reads was not performed because of different methods exposed by different tools to handle such reads. The per-cell spearman correlation, ρ*, between gene counts was determined by excluding genes that contain zero counts in both the tool's quantification and in the simulation quantification for a given cell barcode. splici align: Enabling the index used by alevin-fry that contains introns as well as selective alignment mode. sketch: Selective alignment disabled and index is a standard transcriptome index that does not include introns in alevin-fry. For Cell Ranger, version 7 was used with the include-introns option set to false in order to mimic the default behavior of older versions of Cell Ranger. For cellCounts, the featureType option was set to "exon" (which is the default option) rather than "gene" in order to exclude intronic read quantification.*

| Program | Run mode | Median ρ* | Median r | RMSE | FPR | FNR |
|---|---|---|---|---|---|---|
| Simulation: Single-cell RNA-seq (exon-only; no multigene) | | | | | | |
| STARsolo | | 0.993564 | 0.999968 | 0.030486 | 0.000053 | 0.000270 |
| Cell Ranger | --include-introns=false | 0.956866 | 0.999547 | 0.232195 | 0.000030 | 0.000343 |
| alevin-fry | splici align | 0.984427 | 0.999902 | 0.069541 | 0.000030 | 0.000709 |
| alevin-fry | splici sketch | 0.982900 | 0.999889 | 0.062949 | 0.000031 | 0.000743 |
| alevin-fry | align | 0.990880 | 0.999931 | 0.060543 | 0.000030 | 0.000466 |
| alevin-fry | sketch | 0.991528 | 0.999948 | 0.043601 | 0.000031 | 0.000393 |
| cellCounts | | 0.915796 | 0.994174 | 0.521436 | 0.000239 | 0.001337 |
| Simulation: Single-cell RNA-seq (no multigene) | | | | | | |
| STARsolo | | 0.991877 | 0.999940 | 0.042743 | 0.000140 | 0.000270 |
| Cell Ranger | --include-introns=false | 0.955377 | 0.999499 | 0.232933 | 0.000081 | 0.000343 |
| alevin-fry | splici align | 0.971626 | 0.998672 | 0.288308 | 0.000314 | 0.000702 |
| alevin-fry | splici sketch | 0.960025 | 0.997419 | 0.414070 | 0.000764 | 0.000739 |
| alevin-fry | align | 0.879684 | 0.998145 | 0.312199 | 0.005676 | 0.000446 |
| alevin-fry | sketch | 0.778933 | 0.995372 | 0.527728 | 0.016063 | 0.000357 |
| cellCounts | | 0.825302 | 0.993000 | 0.532384 | 0.005890 | 0.001299 |

ρ*: Modified spearman correlation. r: Pearson correlation. RMSE: root mean squared error. FPR: false positive representation. FNR: false negative representation.
alevin-fry options:

- splici align: Enabling the index used by alevin-fry that contains introns as well as selective alignment mode.
- splici sketch: Enabling the index used by alevin-fry that contains introns without selective alignment mode.
- align: Selective alignment enabled and index is a standard transcriptome index that does not include introns in alevin-fry.
- sketch: Selective alignment disabled and index is a standard transcriptome index that does not include introns in alevin-fry.

For Cell Ranger, version 7 was used with the include-introns option set to false in order to mimic the default behavior of versions 1,2,3,4,5, and 6.

***Table 3.3**. Evaluation of the STARsolo, Cell Ranger, alevin-fry, and cellCounts RNA-seq programs on simulated data generated using the STARsolo simulation framework.*

We assessed the impact of DFKs on memory usage and runtime when processing RNA-seq reads. Across single-cell and single-nucleus RNA-seq datasets from human and mouse tissue, DFKs resulted in only a minor increase in memory usage and runtime. Memory usage increased by <2% which is on the order of megabytes while runtime increased by <15% (Figure 3.10). On the other hand, mapping RNA-seq reads with the nac index type resulted in a much more substantial increase in memory usage and runtime compared with the standard index type. These results make sense as the nac index type is 10 times larger than (i.e. contains 10 times as many k-mers as) the standard index type, whereas the DFKs extracted from a D-list are only a small percentage (i.e. less than 5%) of the total number of k-mers. Thus, DFKs can substantially improve RNA-seq mapping accuracy without having a major impact on performance.

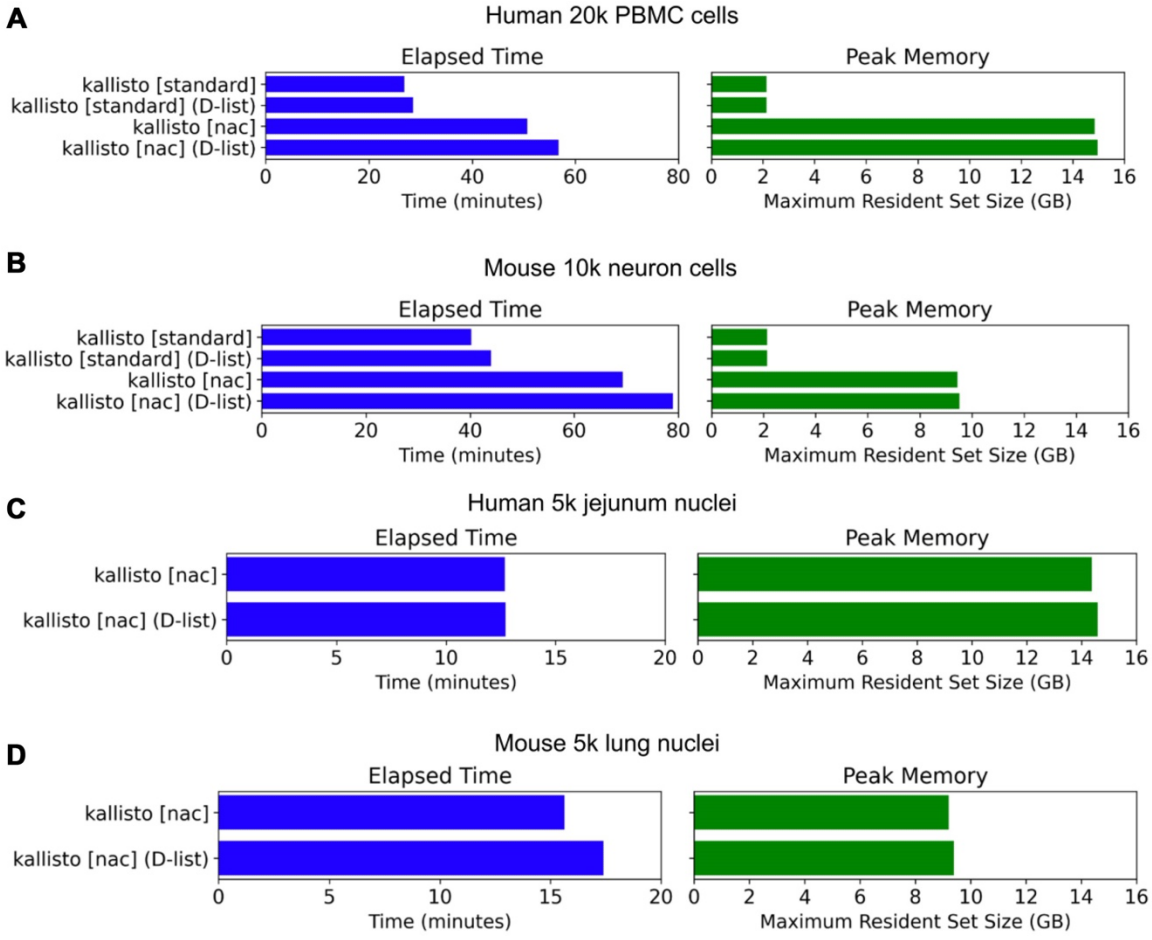


***Figure 3.10**: Runtime and memory usage of kallisto with different index types.*
*(A and B) Runtime (on 16 threads) and memory usage of the standard index type and the nac index type, created with and without a D-list, on single-cell RNA-seq data generated with 10x Genomics. (C and D) Runtime (on 16 threads) and memory usage of the nac index type, created with and without a D-list, on single-nucleus RNA-seq data generated with 10x Genomics. The standard index type was not employed for single-nucleus RNA-seq data because single-nucleus RNA-seq reads predominantly originate from intron-containing pre-mRNA.*

Since DFKs improve mapping specificity, a natural question that arises is whether the improvement in accuracy scales with higher sequencing error rates. Particularly, how do DFKs compare to alignment-based approaches in maintaining accuracy in the face of more sequencing errors? To address this, we introduced additional sequencing errors, consisting of a combination of mismatches, insertions, and deletions, into the STARsolo simulations. We found that the usage of DFKs always results in an improvement in accuracy, even with a high mismatch rate or a high indel rate within the simulated sequencing reads (Figure 3.11A). In contrast, while alignment-based methods tend to be robust to mismatch errors, they fall short with high indel rates (Figure 3.11B). In particular, the same selective alignment settings when executed on the original simulation and simulations where indels are introduced result in a substantial performance decrease on the indel simulations. Altogether, these results (Table 3.4) suggest that pseudoalignment with the incorporation of DFKs is more robust than alignment-based methods to indels. Such considerations may be important when mapping RNA-seq reads from technologies with higher indel rates, such as long-read RNA-seq (Delahaye and Nicolas, 2021; Zhang et al., 2020).

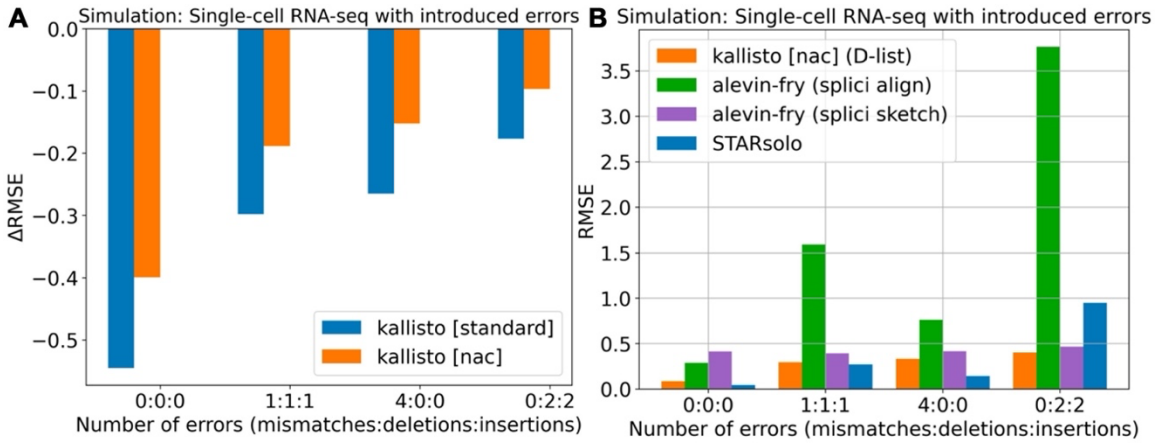



***Figure 3.11**: Assessment of different mapping modes on simulated data generated using the STARsolo simulation framework including the introduction of errors into the reads. (A) Reduction in quantification error, as measured by change in RMSE, by using a D-list to index DFKs compared with not using a D-list on simulated reads with mismatches, deletions, and insertions. (B) Quantification error of different tools on simulated reads with mismatches, deletions and insertions. splici align: enabling the index used by alevin-fry that contains introns as well as selective alignment mode. splici sketch: Same as "splici align" except selective alignment mode is disabled. The errors were introduced into the "single-cell RNA-seq (no multigene)" simulated reads.*

| Mismatches | Deletions | Insertions | Program | Run mode | Median ρ* | Median r | RMSE | FPR | FNR |
|---|---|---|---|---|---|---|---|---|---|
| Simulation: Single-cell RNA-seq (no multigene) | | | | | | | | | |
| 1 | 1 | 1 | kallisto | standard (no D-list) | 0.629277 | 0.993046 | 0.575156 | 0.030304 | 0.001565 |
| 1 | 1 | 1 | kallisto | standard (D-list) | 0.87452 | 0.998388 | 0.276873 | 0.003689 | 0.002376 |
| 1 | 1 | 1 | kallisto | nac (no D-list) | 0.890008 | 0.995764 | 0.484241 | 0.001393 | 0.002872 |
| 1 | 1 | 1 | kallisto | nac (D-list) | 0.900935 | 0.998259 | 0.29544 | 0.001023 | 0.002973 |
| 1 | 1 | 1 | alevin-fry | splici align | 0.673962 | 0.989232 | 1.592164 | 0.000077 | 0.018216 |
| 1 | 1 | 1 | alevin-fry | splici sketch | 0.899393 | 0.99736 | 0.39301 | 0.001391 | 0.002823 |
| 1 | 1 | 1 | STARsolo | | 0.911674 | 0.998962 | 0.271059 | 0.000888 | 0.002918 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 4 | 0 | 0 | kallisto | standard (no D-list) | 0.612826 | 0.992954 | 0.570622 | 0.030607 | 0.002184 |
| 4 | 0 | 0 | kallisto | standard (D-list) | 0.85271 | 0.998136 | 0.305361 | 0.004151 | 0.002991 |
| 4 | 0 | 0 | kallisto | nac (no D-list) | 0.870739 | 0.995728 | 0.483343 | 0.001446 | 0.003621 |
| 4 | 0 | 0 | kallisto | nac (D-list) | 0.881683 | 0.997985 | 0.330969 | 0.001098 | 0.003706 |
| 4 | 0 | 0 | alevin-fry | splici align | 0.819814 | 0.996343 | 0.76258 | 0.000141 | 0.008301 |
| 4 | 0 | 0 | alevin-fry | splici sketch | 0.878774 | 0.9971 | 0.415686 | 0.00136 | 0.003727 |
| 4 | 0 | 0 | STARsolo | | 0.948253 | 0.999501 | 0.141767 | 0.000947 | 0.001244 |
| 0 | 2 | 2 | kallisto | standard (no D-list) | 0.586625 | 0.993183 | 0.557911 | 0.030393 | 0.002935 |
| 0 | 2 | 2 | kallisto | standard (D-list) | 0.831982 | 0.997594 | 0.381203 | 0.00433 | 0.003804 |
| 0 | 2 | 2 | kallisto | nac (no D-list) | 0.851074 | 0.99581 | 0.500121 | 0.001891 | 0.004287 |
| 0 | 2 | 2 | kallisto | nac (D-list) | 0.861993 | 0.997502 | 0.403226 | 0.001458 | 0.004384 |
| 0 | 2 | 2 | alevin-fry | splici align | 0.412969 | 0.924791 | 3.764902 | 0.000033 | 0.053289 |
| 0 | 2 | 2 | alevin-fry | splici sketch | 0.852768 | 0.996873 | 0.466828 | 0.00178 | 0.004671 |
| 0 | 2 | 2 | STARsolo | | 0.763642 | 0.994432 | 0.948656 | 0.000552 | 0.010949 |

ρ*: Modified spearman correlation. r: Pearson correlation. RMSE: root mean squared error. FPR: false positive representation. FNR: false negative representation.
alevin-fry options:

- splici align: Enabling the index used by alevin-fry that contains introns as well as selective alignment mode.
- splici sketch: Enabling the index used by alevin-fry that contains introns without selective alignment mode.

***Table 3.4***. *Evaluation metrics of the kallisto, alevin-fry, and STARsolo single-cell RNA-seq programs on simulated data generated using the STARsolo simulation framework with errors introduced into sequencing reads.*

*Comparison of different index strategies:*

As mentioned in the first section of this thesis chapter, the extended transcriptome index (i.e. the nac index type), by virtue of indexing intron-containing nascent transcripts, enables the mapping of a substantial fraction of reads that would otherwise go unmapped when using the standard index type (Figure 3.12, Figure 3.13).

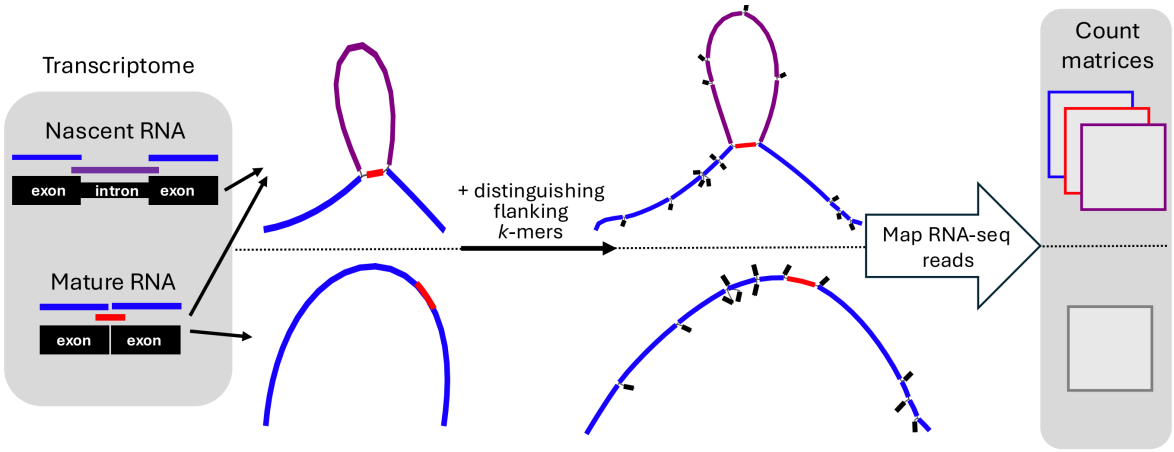



***Figure 3.12**: Transcriptome indices for accurately mapping RNA-seq reads.*
*The standard index is concise, as it only contains mature RNA; it is a suitable lightweight solution when one wishes to simply quantify mature RNA (as is typically the case with bulk RNA-seq analysis). The nac index contains both nascent and mature RNA, providing a more comprehensive framework for RNA-seq analysis. In either case, DFKs can be beneficial.*

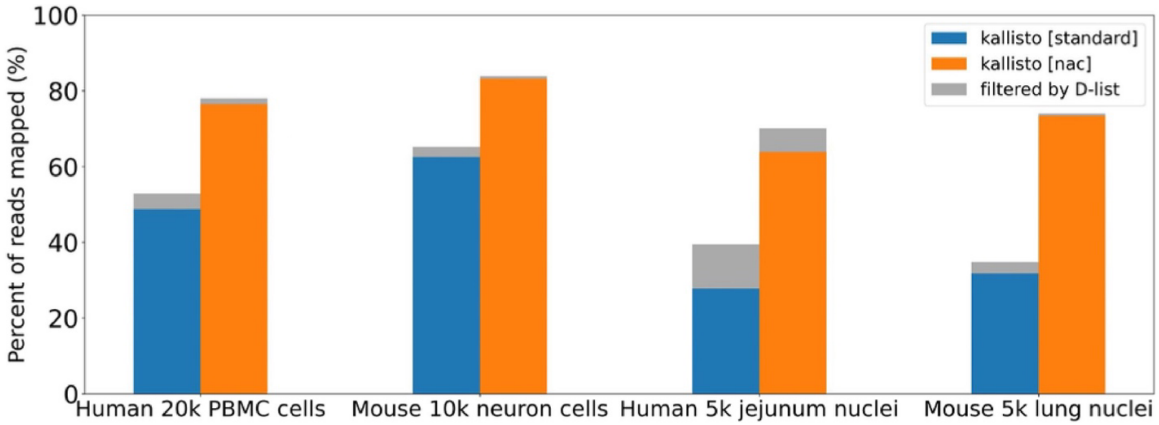



***Figure 3.13**: RNA-seq read mapping rate for different transcriptome indices.*
*The gray bar represents the reads that are excluded when the D-list is used. Of note, the standard index only has mature and ambiguous transcripts in the index (M + A) while the nac index has nascent, mature, and ambiguous transcripts in the index (N + M + A). The datasets used herein are described in the previous section of this thesis chapter.*

We explored how different index types and different count matrices may affect downstream clustering analysis and marker gene selection (Rich et al., 2024). Using the human 20k PBMC dataset (10x Genomics), we projected filtered count matrices (high-quality cells and highly variable genes) onto the first two principal components through PCA (Figure 3.14A). Applying a D-list to the standard index type affected cell projections, but the impact was mild, likely because the human reference genome is a comprehensive and well-annotated assembly, reducing the chance of reads from exonic regions being mismapped. The effect was even more subtle when applying a D-list to the nac index, which includes intronic regions. A more significant difference emerged when comparing analyses with and without nascent transcript quantification. When identifying marker genes through differential gene expression, application of the D-list led to a fraction of marker genes (2% for the standard index type; 14% for the nac index) being uniquely identified in one condition (D-list or no D-list) but not the other, while incorporating nascent transcript quantification resulted in 19% more marker genes being identified (Figure 3.14B).

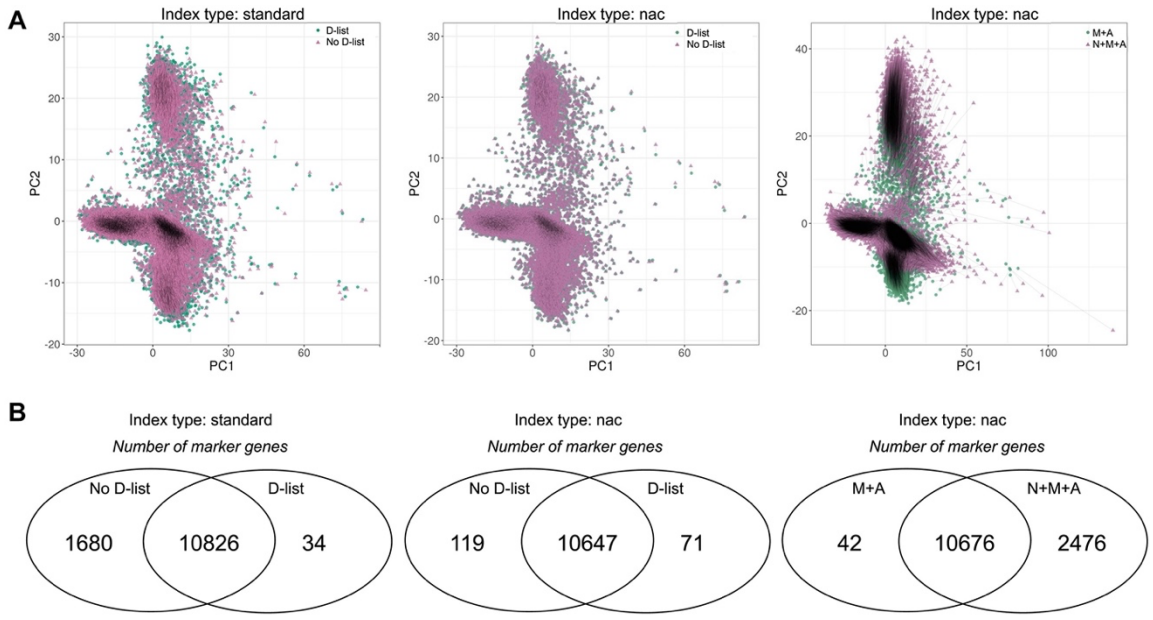



***Figure 3.14****: Effect of index strategy and count matrix type on single-cell RNA-seq analysis of the human 20k PBMC dataset (10x Genomics).*
*(A) PCA of cells from the human 20k PBMC dataset (10x Genomics) for count matrices generated in various ways. The first two principal components are shown. Black lines connect identical cell barcodes from each matrix used in pairwise comparisons. (B) Number of marker genes identified through differential gene expression analysis for count matrices generated in various ways. Left-hand panel: Count matrices were generated by mapping reads to the standard index type with and without a D-list. Middle panel: Count matrices (M + A) were generated by mapping reads to the nac index type with and without a D-list. Right panel: M + A and N + M + A count matrices were generated by mapping reads to the nac index type. N: nascent, M: nature, and A: ambiguous.*

Although, broadly, cluster analysis remained largely unaffected by these different strategies (Figure 3.15), the simulations earlier on showed that more pronounced differences can be observed at the individual cell and gene level, thus making the selection of index strategy an important consideration depending on the type of downstream analysis that is to be performed. Additionally, “mature” and “nascent” quantifications provide distinct insights into the cell’s profile. Although count matrices containing only “nascent” or “mature” gene counts can sometimes yield similar clusters, the cellular profile differs greatly, with many marker genes unique to each matrix (Figure 3.16).

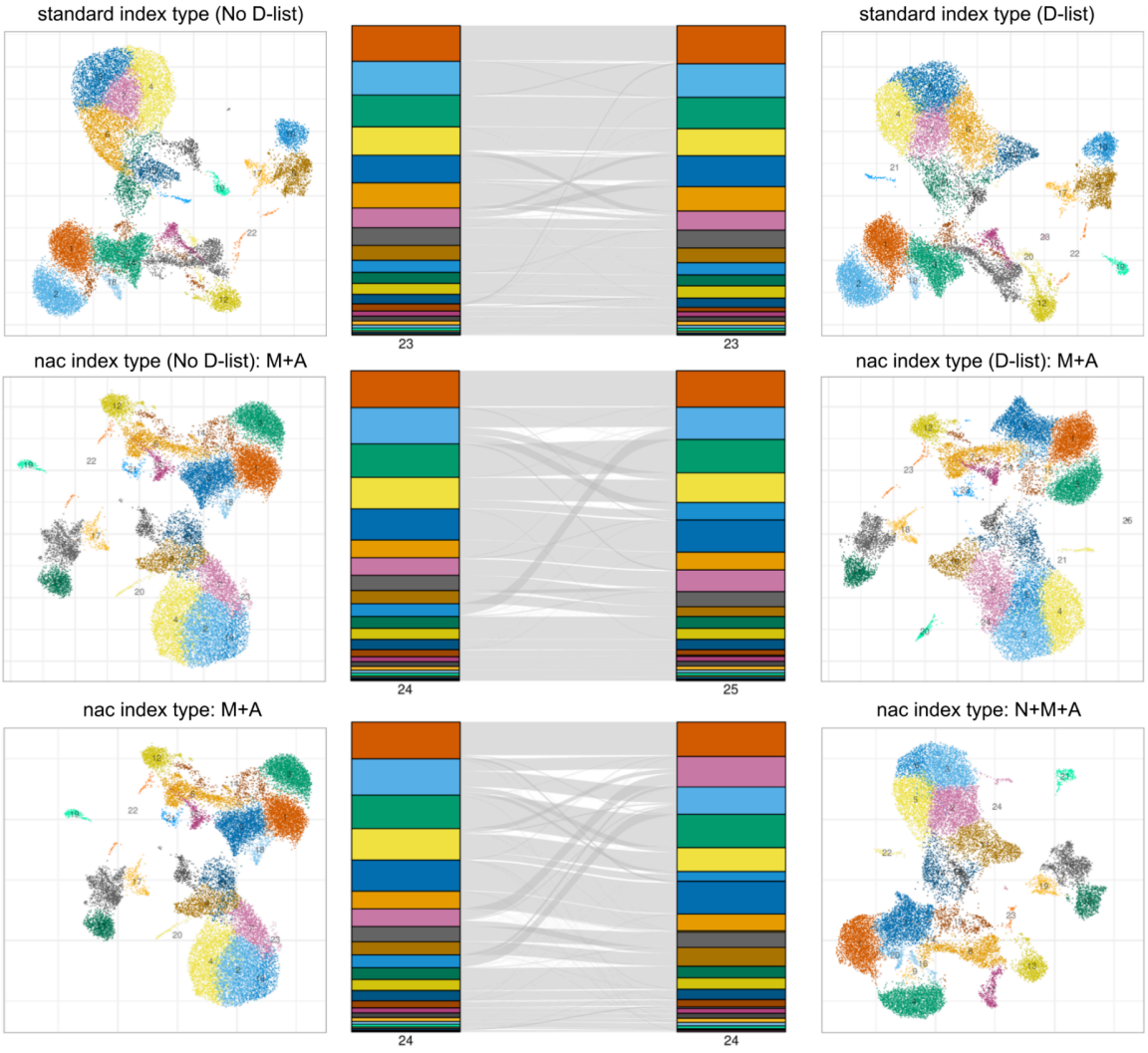



***Figure 3.15**: Effect of index strategy and count matrix type on clustering of the human 20k PBMC dataset (10x Genomics) single-cell RNA-seq dataset.*
*Alluvial plots of cluster assignment mapping alongside UMAP plots are shown for pairwise comparisons of count matrices generated by different index strategies in human 20k PBMC single-cell RNA-seq data. N: nascent, M: nature, and A: ambiguous.*

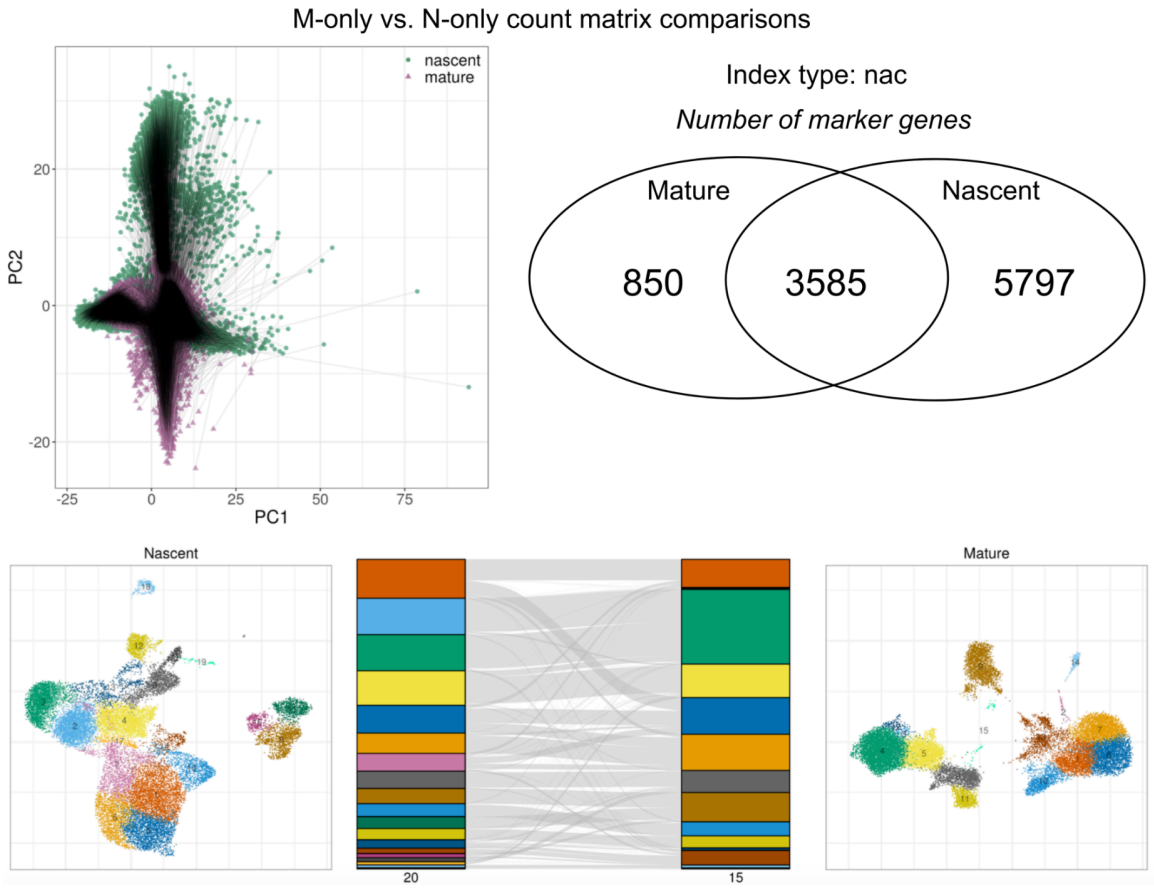



***Figure 3.16**: Effect of mature versus nascent count matrix types on clustering of the human 20k PBMC dataset (10x Genomics) single-cell RNA-seq dataset.*
*Alluvial plots of cluster assignment mapping, marker gene numbers, PCA plots, and UMAP plots are shown for comparing nascent and mature count matrices.*

This work introduces a combined approach of using DFKs with an extended transcriptome index (nac index) in single-cell and single-nucleus RNA-seq analysis (Figure 3.17). This method aims to address specific challenges in RNA-seq, particularly in the quantification of nascent and mature mRNA transcripts, and in reducing mismapping errors caused by reads originating outside of the targeted transcriptomic regions.

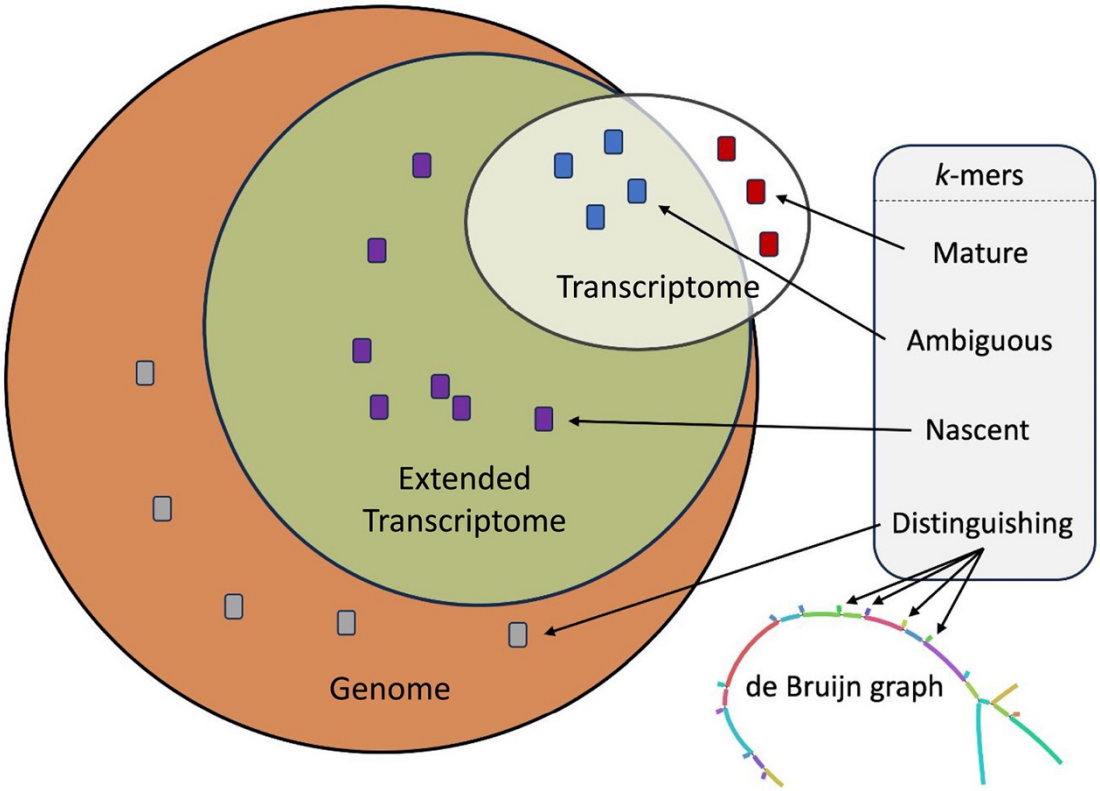



***Figure 3.17**: Summary of enhancements to read mapping and classification.*
*This figure shows k-mers originating from the standard transcriptome index, the extended transcriptome index containing nascent RNA transcripts, and the entire genome. The integration of DFKs into a de Bruijn graph is shown.*

While most mismapping errors that affect single-cell RNA-seq quantification are eliminated by extending the transcriptome index, DFKs provide further improvement to quantification accuracy. Specifically, DFKs can eliminate erroneous mapping of reads that originate from transcripts that appear outside even the extended transcriptome index. More importantly, DFKs provide high scalability. DFKs can scale to higher sequencing error rates as the accuracy gains of DFKs are not reversed when different sequencing error profiles are introduced. Moreover, DFKs can scale to size. When only a small specific set of targets is of interest but there are many known possible target sequences, those possible target sequences can simply be incorporated into the D-list. The resultant DFKs will

optimize mapping specificity making it unnecessary to index all the possible target sequences. Irrespective of whether the target sequences occupy a small proportion or a large proportion of the “background”, the DFKs will improve mapping specificity without any major impact on performance. Thus, the DFKs act as a space-efficient general “background filter”.

**Comprehensive pseudoalignment software protocol**

The term ‘RNA-seq’ refers to a collection of assays based on sequencing experiments that involve quantifying RNA species from bulk tissue, single cells, or single nuclei. The kallisto, bustools and kb-python programs are free, open-source software tools for performing this analysis that together can produce gene expression quantification from raw sequencing reads. The quantifications can be individualized for multiple cells, multiple samples or both. Additionally, these tools allow gene expression values to be classified as originating from nascent RNA species or mature RNA species, making this workflow amenable to both cell-based and nucleus-based assays. This protocol describes in detail how to use kallisto and bustools in conjunction with a wrapper, kb-python, to preprocess RNA-seq data. Execution of this protocol requires basic familiarity with a command line environment. With this protocol, quantification of a moderately sized RNA-seq dataset can be completed within minutes.

*Overview:*

The preprocessing step (Melsted et al., 2021; Tian et al., 2018) of RNA-seq experiments (Mortazavi et al., 2008) involves mapping reads to a reference genome or transcriptome, followed by gene expression or transcript abundance quantification (Conesa et al., 2016). Many open-source tools exist for bulk RNA-seq preprocessing (Anders et al., 2015; Bray et al., 2016; Dobin et al., 2013; Li and Dewey, 2011; Liao et al., 2019, 2014; Patro et al., 2017; Pertea et al., 2016; Roberts and Pachter, 2013; Trapnell et al., 2012) as well as single-cell RNA-seq preprocessing (Battenberg et al., 2022; He et al., 2022; He and Patro, 2023; Kaminow et al., 2021; Liao et al., 2023; Melsted et al., 2021; Niebler et al., 2020; Srivastava et al., 2019). Kallisto (Bray et al., 2016) introduced the pseudoalignment paradigm for improving the accuracy of alignment and reducing runtimes and memory footprint of bulk RNA-seq preprocessing and, with the development of bustools (Melsted et al., 2019), has been adapted for both single-cell RNA-seq quantification (Melsted et al., 2021) and single-nucleus RNA-seq quantification (Sullivan et al., 2025). The bustools suite of tools operates on the read mapping results of kallisto and processes them to generate

quantification results, which may involve unique molecular identifier (UMI) collapsing (Kivioja et al., 2012; Smith et al., 2017) and barcode error correction for single-cell and single-nucleus assays. While multiple steps are necessary to process input consisting of FASTQ sequencing files, a reference genome FASTA, and a GTF annotation (Kent et al., 2002; Reese et al., 2000), to an output of quantifications using kallisto and bustools, these steps are greatly facilitated by the wrapper tool, kb-python. kb-python can extract reference transcriptomes from reference genomes and run kallisto and bustools in workflows optimal for each assay type (Box 1) (Sullivan et al., 2024). The kb-python tool simplifies the running of kallisto and bustools to the extent that all of this can be done in two steps: **kb ref** for generating a kallisto index from an annotated reference genome and **kb count** for mapping and quantification. Thus, kallisto, bustools, and kb-python make RNA-seq preprocessing efficient, modular, flexible, and simple (Melsted et al., 2021).

**Box 1: Software tools and their description**

Software tools:

- **kallisto**: Performs pseudoalignment to a reference transcriptome and stores the mapping results in a BUS file.
- **bustools**: Processes the results in the BUS file to correct barcodes, deduplicate UMIs, and generate quantification files (e.g. count matrices).
- **kb-python**: A wrapper around kallisto and bustools that facilitates usage of those tools and facilitates the generation of a reference transcriptome. The kallisto and bustools binaries come packaged in kb-python.

Installation:

```
pip install kb_python
```

*Index construction*:

For RNA-seq read mapping, kallisto builds an *index* from a set of sequences, referred to as targets, representing the set of sequences that RNA-seq reads can be mapped to. In a standard analysis, these targets are usually transcript sequences (i.e. each individual target corresponds to one transcript). However, more generally, users can define targets from any

sets of sequences they wish to map their sequencing reads against. Since kallisto is a tool that leverages pseudoalignment, the mapping procedure relies on read assignment, such that each read is deemed to be compatible with a certain set of targets, rather than standard alignment. The kallisto index is based on the Bifrost (Holley and Melsted, 2020) implementation of the colored de Bruijn graph (Iqbal et al., 2012), which enables memory-efficient and rapid read assignment.

kb-python enables the construction of kallisto indices through the kb ref command (Figure 3.18). Different types of kallisto indices can be built by specifying the --workflow argument in kb ref, which selects the type of index to be constructed. The default setting is **--workflow=standard**, which creates an index suitable for bulk and single-cell RNA-seq quantification. It creates an index built from only the coding DNA sequences (the usage of coding DNA here follows that of Ensembl (Harrison et al., 2024), i.e., the sequences of the mature transcripts wherein introns are not included as they have been spliced out). The index created by **--workflow=nac** (nac: nascent and coding DNA) contains both the coding DNA and the nascent transcripts. The nascent transcript sequences consist of the full gene (both exons and introns). This nac index is suitable for single-nucleus RNA-seq as there exists a high abundance of non-mature transcripts captured in nucleus-based sequencing assays (Grindberg et al., 2013). Additionally, this nac index should be used for analyses that require jointly modeling nascent and mature RNA species (Carilli et al., 2024; Gorin et al., 2023, 2022a, 2022b; Gorin and Pachter, 2022a; La Manno et al., 2018). For both the standard and nac index types, a user supplies a genome FASTA and GTF annotation, which kb-python uses to extract the relevant sequences. Finally, if one wishes to index a custom set of targets or of k-mers, one can use **--workflow=custom** which builds an index from a FASTA file containing the target sequences of interest to be supplied.

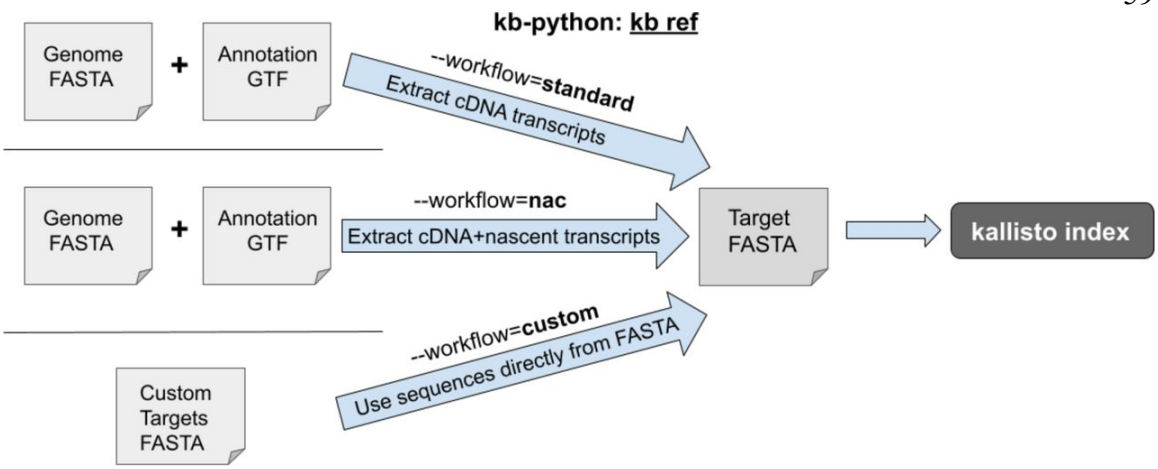



***Figure 3.18**: "kb ref" can be used to generate three different types of kallisto indices.*

Creating the index in kb-python invokes the **kallisto index** command in the kallisto program (Box 2). Indexing with kb-python has the advantage that a reference transcriptome is generated directly from a FASTA and GTF ensuring consistency between the transcriptome reference, its associated index, and the input FASTA and GTF.

Additionally, using kb-python (via the --include-attributes and --exclude-attributes options) allows specific biotypes to be selected from the GTF file, making possible filtering of entries such as pseudogenes, which can improve read mapping accuracy (Pool et al., 2023) and reduce memory usage. It is recommended to perform GTF filtering, especially for the nac index type where there will be many overlapping segments among annotated regions in the genome. While there is no universally defined best practice for GTF filtering, it is recommended that a user uses the CellRanger (Zheng et al., 2017) gene biotypes for standard single-cell and single-nucleus RNA-seq assays. More generally, if a user is unsure of what biotypes to include, it is recommended that the user selects only the specific biotypes that the user is interested in (e.g. selecting only protein coding genes if a user is only interested in protein coding genes).

**Box 2: kb ref**

Below, we show how to run kb ref using three different index types. Only the underlined files need to be supplied by the user; the other files are output files generated as part of the indexing process and may be necessary for the subsequent mapping and quantification step. The corresponding kallisto index commands that are invoked are shown beneath each kb ref call (note that, by default, the kallisto index command is invoked using 8 threads).

1. **standard** index type (default):

```
kb ref -i index.idx -g t2g.txt -f1 cdna.fasta genome.fasta genome.gtf
```

```
kallisto index -t 8 -i index.idx --d-list=genome.fasta cdna.fasta
```

2. **nac** index type:

```
kb ref --workflow=nac -i index.idx -g t2g.txt -c1 cdna.txt -c2 nascent.txt \
    -f1 cdna.fasta -f2 nascent.fasta genome.fasta genome.gtf
```

```
kallisto index -t 8 -i index.idx --d-list=genome.fasta cdna.fasta nascent.fasta
```

3. **custom** index type:

```
kb ref --workflow=custom -i index.idx custom.fasta
```

```
kallisto index -t 8 -i index.idx custom.fasta
```

Explanation of output files:

- index.idx: The kallisto index that is generated
- t2g.txt: The transcript-to-gene mapping file
- cdna.fasta: The generated FASTA file containing the extracted cDNA sequences
- nascent.fasta: The generated FASTA file with extracted nascent transcript sequences
- cdna.txt: The transcript names of the coding DNA sequences
- nascent.txt: The nascent transcript names (which are simply the gene names)

Finally, the kallisto index command has a --d-list option which improves the mapping specificity by isolating certain sequences, known as distinguishing flanking k-mers (DFKs), that may cause erroneous read mapping (Sullivan et al., 2025). The DFKs that are identified depend on the FASTA file supplied to the --d-list option. While the --d-list option can be entered by the user directly into kb ref, kb ref already by default calls kallisto index with the --d-list option set to the genome FASTA supplied but can be disabled by specifying --d-list=None in kb ref. For all analyses that involve RNA transcript quantification, it is recommended that the --d-list be set to the respective genome FASTA file to ensure good mapping specificity. This feature should typically only be used in any standard RNA-seq analysis (e.g. any usage with the standard index type or the nac index type produced by kb ref). This feature should not be used in other cases where custom non-transcript targets are indexed.

*Mapping and quantification*:

The **kb count** command within kb-python enables mapping and quantification of bulk, single-cell, and single-nucleus RNA-seq reads (Figure 3.19). As different sequencing assays have different read structures, strandedness, parity, and barcodes, one must provide the specifications for the technology which produced the sequencing reads.

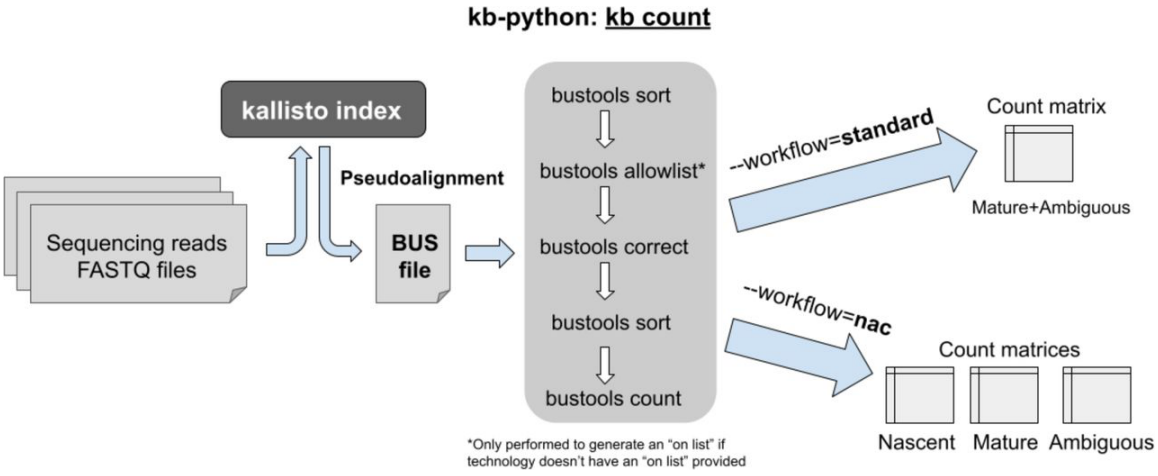



***Figure 3.19**: "kb count" can be used to produce quantifications in the form of count matrices for bulk, single-cell, and single-nucleus RNA-seq.*

The specifications for sequencing assay technology within kb-python are as follows:

- Technology string: A 'technology string' for a particular type of assay can be supplied via the **-x** option. The technology string can be used in one of two ways:
    - Option 1: several assays are predefined within the software (the list is viewable by calling **kb --list**) so one can name one of those directly (e.g. one can specify -x 10xv3).
    - Option 2: one can format their own custom technology string specifying the read locations of the barcodes, UMIs and the biological sequence that is to be mapped (Box 3).
- Strandedness: If a read (or the first read in the case of paired-end reads) is to be mapped in forward orientation, one should specify **--strand=forward**. If it is to be mapped in reverse orientation, one should specify **--strand=reverse**. If one does not want to map reads with strand-specificity, then one should specify the option as **--strand=unstranded**. If a predefined name is used in the technology string -x option (option 1), then kb-python uses a default stranded option for that technology (e.g., for 10xv3, the default is forward); otherwise, the default is unstranded. Setting the --strand option explicitly will overrule the default option.
- Parity: If the technology involves two biological read files that are derived from paired-end sequencing, as is the case with Smartseq2 (Picelli et al., 2013), Smartseq3 (Hagemann-Jensen et al., 2020), SPLiT-seq (Rosenberg et al., 2018), and many bulk RNA sequencing kits, one should specify **--parity=paired** to perform mapping that takes into account the fact that the reads are paired end. Otherwise, one can specify **--parity=single**. If a predefined name is used in the -x technology string option (option 1), then kb-python uses the default parity option for that technology (e.g., for -x Smartseq2, --parity=paired is already enabled by default).

- On list: For single-cell and single-nucleus sequencing assays, barcodes are used to identify each cell or nucleus. The ‘on list’ of barcodes represents the known barcode sequences that are included in the assay. Barcodes extracted from the sequencing reads will be error-tolerantly mapped to this list in a process known as barcode error correction. The ‘on list’ is a text file containing a list of barcode sequences and its filename can be specified with the **-w** option in kb count. If an on list is not provided or cannot be found for the given technology, then an on list is created by bustools via the **bustools allowlist** command, which identifies repeating barcodes in sequencing reads. If the technology does not include cell barcodes (as is the case in bulk RNA-seq), the ‘on list’ option is irrelevant and no barcode processing occurs, which should be the case for assays that do not include cell/nuclei barcodes (skipping barcode error correction can also be done by specifying **-w NONE**). If a predefined name is used in the -x technology string option (option 1), then kb-python uses the default on list option for that technology.

**Box 3: Custom technology string**

The custom technology string (supplied to -x) contains the format **barcode:UMI:DNA**, representing the locational information of the barcode, UMI, and the DNA (where DNA is the biological read to be mapped):

**-x a,b,c:d,e,f:g,h,i**

- a: barcode file number, b: barcode start position, c: barcode end position
- d: UMI file number, e: UMI start position, f: UMI end position
- g: DNA file number, h: DNA start position, i: DNA end position

Important notes: File numbers and positions are zero-indexed. If no specific end position exists (i.e. the end position is the very end of the read), the end position should be set to 0. If cell barcodes and/or UMIs are not supported by the technology, the barcode and/or UMI field can be set to -1,0,0.

Thus, for 10xv3:

**-x 0,0,16:0,16,28:1,0,0**

Sequences can be stitched together by specifying multiple locations; for example, a SPLiT-seq[45] assay, which contains three separate unlinked barcodes, each of length 8, and a UMI of length 10 in the second file and the DNA in the first file would look as follows:

**-x 1,10,18,1,48,56,1,78,86:1,0,10:0,0,0**

Final note about multiple locations: If the paired-end read mapping option is enabled, exactly two DNA locations should be specified (for the first and second read in the pair).

If a technology does not fit into this format (e.g. due to barcodes or UMIs of variable lengths and positions), preprocessing of the FASTQ file should be performed beforehand to reformat the reads into a structure that can be handled by this format.[46]

If a nac index was generated by kb ref, **--workflow=nac** should also be used in kb count so that the nascent and mature RNA species are quantified accurately; otherwise that option should be omitted or **--workflow=standard** (which is the default) can be explicitly specified. For the nac index type, one obtains three count matrices: (1) nascent, (2) mature, and (3) ambiguous. In most experiments, the plurality of reads will be "ambiguous" since they originate from exons, which are present in both nascent RNA and mature RNA. Therefore, it is desirable to generate additional matrices by adding the counts from those three matrices, which users can either do themselves or by using the --sum option (Sullivan et al., 2025). **--sum=total** adds all three matrices, **--sum=cell** adds the mature and ambiguous matrices, and **--sum=nucleus** adds the nascent and ambiguous matrices. Different matrices may be used for different types of analyses. For example, in single-cell RNA-seq analysis (where most "ambiguous" counts are probably of mature RNA origin), jointly modeling the mature + ambiguous count matrix (--sum=cell) with the nascent count matrix permits biophysical modeling of RNA processing (Gorin et al., 2023, 2022b). The kb-python, kallisto, and bustools commands for the standard and nac index types are presented in Box 4 and Box 5, respectively.

**Box 4: kb count (standard index type)**

Below, we show how to run kb count using the standard index type (which is the default used if no --workflow option is explicitly specified). The underlined files need to be supplied by the user; these include the files generated from the kb ref command as well as the FASTQ sequencing reads. The corresponding kallisto and bustools commands (as well as Unix commands to create and remove files/directories) that are called by kb count are shown beneath each kb count command (note that, by default, 8 threads and 2 gigabytes of memory are assigned).

**kb count -x <tech> -w onlist.txt -o output_dir -i index.idx -g t2g.txt **
**R1.fastq R2.fastq**

```
mkdir -p output_dir/tmp
mkdir -p output_dir
kallisto bus -x <tech> -i index.idx -o output_dir -t 8 R1.fastq R2.fastq
bustools sort -o output_dir/tmp/output.s.bus -T output_dir/tmp -t 8 -m 2G output_dir/output.bus
bustools inspect -o output_dir/inspect.json -w onlist.txt output_dir/tmp/output.s.bus
bustools correct -o output_dir/tmp/output.s.c.bus -w onlist.txt output_dir/tmp/output.s.bus
bustools sort -o output_dir/output.unfiltered.bus -T output_dir/tmp -t 8 -m 2G \
    output_dir/tmp/output.s.c.bus
mkdir -p output_dir/counts_unfiltered
bustools count -o output_dir/counts_unfiltered/cells_x_genes -g t2g.txt -e output_dir/matrix.ec \
    -t output_dir/transcripts.txt --genecounts --umi-gene output_dir/output.unfiltered.bus
rm -rf output_dir/tmp
```

- <tech>: The technology string
- onlist.txt: The name of the file containing the “on list” of barcodes
    - Specify NONE to skip barcode error correction, or omit completely to have bustools create its own “on list” for correction

Note: In the workflow above, the following options in kb count can be used:

- **--parity=single** or **--parity=paired**
- **--strand=forward** or **--strand=reverse** or **--strand=unstranded**

One can alternatively set those options at the end of <tech>, e.g. <tech>%forward%paired

The R1.fastq and R2.fastq inputs can be replaced with multiple sets of read files listed consecutively, as long as each pair is in order.

**Box 5: kb count (nac index type)**

Below, we show how to run kb count using the nac index type. The underlined files need to be supplied by the user;these include the files generated from the kb ref command using --workflow=nac as well as the FASTQ sequencing reads. The corresponding kallisto and bustools commands (as well as Unix commands to create and remove files/directories) that are called are shown beneath each kb count call (note that, by default, 8 threads and 4 gigabytes of memory are used).

**kb count -x <tech> --workflow=nac -w onlist.txt -o output_dir -i index.idx **
**-g t2g.txt -c1 cdna.txt -c2 nascent.txt --sum=<sum> R1.fastq R2.fastq**

```
mkdir -p output_dir/tmp
mkdir -p output_dir
kallisto bus -x <tech> -i index.idx -o output_dir -t 8 R1.fastq R2.fastq
bustools sort -o output_dir/tmp/output.s.bus -T output_dir/tmp -t 8 -m 4G output_dir/output.bus
bustools inspect -o output_dir/inspect.json -w onlist.txt output_dir/tmp/output.s.bus
bustools correct -o output_dir/tmp/output.s.c.bus -w onlist.txt output_dir/tmp/output.s.bus
bustools sort -o output_dir/output.unfiltered.bus -T output_dir/tmp -t 8 -m 4G \
    output_dir/tmp/output.s.c.bus
mkdir -p output_dir/counts_unfiltered
bustools count -o output_dir/counts_unfiltered/cells_x_genes -g t2g.txt -e output_dir/matrix.ec \
    -t output_dir/transcripts.txt --genecounts --umi-gene \
    -s nascent.txt output_dir/output.unfiltered.bus
mv output_dir/counts_unfiltered/cells_x_genes.mtx \
    output_dir/counts_unfiltered/cells_x_genes.mature.mtx
mv output_dir/counts_unfiltered/cells_x_genes.2.mtx \
    output_dir/counts_unfiltered/cells_x_genes.nascent.mtx
rm -rf output_dir/tmp
```

- <tech>: The technology string
- onlist.txt: The name of the file containing the “on list” of barcodes
    - Specify NONE to skip barcode error correction, or omit completely to have bustools create its own “on list” for correction
- <sum>: What additional matrix to create by adding up the output matrices (options: cell, nucleus, or total)

Note: In the workflow above, we can additionally set the following two options in kb count (otherwise, the defaults are chosen):

- **--parity=single** or **--parity=paired**
- **--strand=forward** or **--strand=reverse** or **--strand=unstranded**

One can alternatively set those options at the end of <tech>, e.g.: <tech>%forward%paired

In addition to single-cell and single-nucleus RNA-seq, kb count can be used for bulk RNA-seq. Bulk RNA-seq generally does not have UMIs or cell barcodes (although artificial unique sample-specific barcodes, i.e. pseudobarcodes, are used to identify each sample) and relies on cDNA mapping. With -x BULK as the technology string, a workflow specific for bulk RNA-seq quantification is executed (Box 6). This will produce both transcript-level and gene-level abundances that can be used by DESeq2 (Love et al., 2014; Soneson et al., 2015), sleuth (Pimentel et al., 2017), limma-voom (Law et al., 2014; Ritchie et al., 2015), edgeR (Baldoni et al., 2024; Chen et al., 2024; Robinson et al., 2010), and other differential gene expression programs.

**Box 6: kb count: bulk RNA-seq**

Below, we show how to run kb count for preprocessing bulk RNA-seq data. The procedure is similar to the preprocessing of single-cell RNA-seq (Box 4), but there are some differences in how quantification is performed and barcode error correction is not performed due to the lack of cell barcodes in bulk RNA-seq. **--tcc** specifies that estimated counts should be produced in accordance with the count estimation algorithm in the original kallisto publication and **--matrix-to-directories** means that those quantifications should be reformatted into directories of "abundance files" with each sample being a different directory. The abundance files can be directly used by downstream tools designed for bulk RNA-seq differential gene expression. Below is an example usage for a paired-end unstranded bulk RNA-seq experiment on one sample.

```
kb count -x BULK -o output_dir -i index.idx -g t2g.txt \
    --parity=paired --strand=unstranded \
    --tcc --matrix-to-directories R1.fastq R2.fastq
```

```
mkdir -p output_dir/tmp
mkdir -p output_dir
kallisto bus -x <tech> -i index.idx -o output_dir -t 8 --paired R1.fastq R2.fastq
bustools sort -o output_dir/tmp/output.s.bus -T output_dir/tmp -t 8 -m 2G output_dir/output.bus
bustools inspect -o output_dir/inspect.json output_dir/tmp/output.s.bus
mkdir -p output_dir/counts_unfiltered
mkdir -p output_dir/quant_unfiltered
bustools count -o output_dir/counts_unfiltered/cells_x_tcc -g t2g.txt -e output_dir/matrix.ec \
    -t output_dir/transcripts.txt --multimapping --cm output_dir/output.s.bus
kallisto quant-tcc -o output_dir/quant_unfiltered -i index.idx \
    -e output_dir/counts_unfiltered/cells_x_tcc.ec.txt -g t2g.txt -t 8 -f output_dir/flens.txt \
    --matrix-to-directories output_dir/counts_unfiltered/cells_x_tcc.mtx
rm -rf output_dir/tmp
```

To facilitate multi-sample analysis, artificial unique sample-specific barcodes can be created, and the resulting mapping between the artificially generated barcode and the sample ID is outputted. These sample-specific barcodes (i.e. ‘pseudobarcodes’) are 16-bp in length and are stored in the BUS file. Where there exists both a cell barcode (like in single-cell RNA-seq) and a sample-specific barcode, both sets of barcodes will be outputted so that each entry in the resulting output count matrix can be associated with a particular cell and a particular sample. To utilize the multi-sample workflow, a batch file containing the file names of the FASTQ files must be provided (Box 7). Each sample ID will then be assigned a ‘pseudobarcode’ upon running the program.

**Box 7: kb count (multi-sample analysis using the standard index type)**

Below, we show how to run kb count to perform an analysis of multiple samples using the standard (default) index type. Use of this index type facilitates a workflow that is similar to the single-sample standard workflow (Box 4). A batch file (**batch.txt**) should be provided, in lieu of FASTQ files, listing all the samples to be analyzed with the paths to their respective FASTQ files. The **--batch-barcodes** option is provided in order to store the sample-specific barcodes that are created in addition to the cell barcodes (without this option, only cell barcodes are stored). This option can be omitted in the case that no cell barcodes exist (as in bulk RNA-seq).

```
kb count -x <tech> -w onlist.txt -o output_dir -i index.idx -g t2g.txt \
    --batch-barcodes batch.txt
```

The only difference in the underlying kallisto command is in the *kallisto bus* command.

```
kallisto bus -x <tech> -i index.idx -o output_dir -t 8 --batch-barcodes --batch batch.txt
```

The batch.txt file looks as follows:

batch.txt

```
Sample1  sample1_R1.fastq.gz  sample1_R2.fastq.gz
Sample2  sample2_R1.fastq.gz  sample2_R2.fastq.gz
Sample3  sample3_R1.fastq.gz  sample3_R2.fastq.gz
```

The sample ID is in the first column. Multiple rows can be provided for the same sample ID (e.g. if the FASTQ files are divided across multiple lanes). The third column can be omitted if only one FASTQ file is specified by the technology.

In the output directory (output_dir), there will be two files: matrix.cells (which contains the sample ID) and matrix.sample.barcodes (which contains the 16-bp sample-specific barcodes, i.e. the pseudobarcodes). Each line in matrix.cells corresponds to the same line in matrix.sample.barcodes. In the example above, the files look as follows:

matrix.cells

```
Sample1
Sample2
Sample3
```

matrix.sample.barcodes

```
AAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAC
AAAAAAAAAAAAAAAG
```

The technical details of how kb count utilizes kallisto and bustools are detailed in the following paragraph. Note that the **--dry-run** option in kb count outputs the kallisto and bustools commands that will be run without actually running the programs. Also, the option **--verbose** in kb count is helpful for examining the kallisto and bustools commands that are being run as well as their output.

kb count first invokes the **kallisto bus** command within kallisto to produce a BUS file, which stores the read mapping information, and then uses bustools (Melsted et al., 2019) commands to process the BUS file. The **kallisto bus** command maps RNA-seq reads to a kallisto index, and the resultant BUS file stores the mapping information, including the barcode, unique molecular identifier (UMI), and the equivalence class representing the set of transcripts the read is compatible with (Melsted et al., 2019). In certain RNA-seq assays, barcodes and/or UMIs may not be present, and are therefore not considered when processing the BUS file. After the mapping step is complete, the BUS file is sorted via the **bustools sort** command to facilitate further processing. For single-cell and single-nucleus experiments with multiplexed barcodes in the sequencing reads, an ‘on list’ of barcodes, representing the known barcode sequences that are included in the assay, needs to be provided. If an ‘on list’ is unavailable, the **bustools allowlist** command can be used to construct one from a sorted BUS file. The barcodes in the sorted BUS file are error-corrected to the ‘on list’ via **bustools correct**, then the BUS file is sorted again with **bustools sort**. The final sorted, ‘on list’-corrected BUS file is then used to generate quantifications via count matrices through the **bustools count** command. At any point, a sorted BUS file can be inputted into **bustools compress** to create a compressed BUS file (a BUSZ file), which can be subsequently decompressed via **bustools decompress** (Einarsson and Melsted, 2023). There exist many other bustools features which enable more specialized workflows beyond what is provided by kb-python (Gustafsson et al., 2021; Melsted et al., 2021).

Quantification of RNA species can be performed in multiple ways as follows:

- Gene-level count matrices: in single-cell and single-nucleus RNA-seq, typically a gene-level count matrix is produced by collapsing UMIs to the gene level. Here, the bustools count command is run with the **--genecounts** option. The **--umi-gene** option may also be provided for sequencing technologies where the UMIs are not expected to be unique within each cell. This ensures that in a case where two reads with the same UMI sequence map to different genes, they are considered to be two distinct molecules which were unintentionally labeled with the same UMI, and hence each gene gets a count. Such instances occur very frequently when UMIs are short such as in CEL-Seq2 (Hashimshony et al., 2016). By default, UMIs assigned to multiple genes after collapsing are discarded in quantification; however, the option **--multimapping** retains such UMIs and distributes the count uniformly across the assigned genes. This option, while improving the sensitivity of gene detection, causes noninteger counts to be created and is therefore disabled by default, consistent with other single-cell RNA-seq software. Finally, if one wishes to not perform UMI collapsing (i.e., each mapped read is its own unique molecule regardless of the UMI sequence), one can supply the **--cm** option for quantification.

- Transcript-level count matrices: transcript compatibility counts (TCCs) are counts assigned to equivalence classes (ECs) where each EC is defined by a unique set of transcripts. For producing a matrix of TCCs, the **--genecounts** option is *not* provided, and **--multimapping** *is* provided to avoid discarding reads or collapsed UMIs that are assigned to multiple genes. If UMIs are not present in the sequencing technology, the **--cm** option is supplied to perform counting without UMI collapsing. While downstream analyses can be performed on TCCs (Ntranos et al., 2019, 2016), it is more often useful to produce transcript-level abundances from the TCCs for technologies where sequencing reads span segments of full-length RNA molecules, such as in bulk RNA-seq. In such cases, an expectation-maximization algorithm is typically performed to probabilistically estimate transcript abundances

(Li and Dewey, 2011; Pachter, 2011). The procedure to generate transcript-level abundance matrices is performed by running the **kallisto quant-tcc** command on the TCC matrices.

Now, we describe the quantification output of the kb count command. While the initial step of kb count uses kallisto to produce a BUS file located at output_dir/output.bus, the actual quantification results are located in matrices in subdirectories of output_dir/. All matrices have the extension .mtx and will be in a sparse matrix (Matrix Market) file format with the barcodes (i.e. the cells or samples) being the matrix rows and the genes (or transcripts or equivalence classes or other features (A Sina Booeshaghi et al., 2024)) being the matrix columns.

Gene-level counting to produce gene count matrices is the most common form of quantification for UMI-based single-cell and single-nucleus RNA-seq assays.

- The **output_dir/counts_unfiltered/** directory contains the following information for *gene count matrices* (these are the matrices that are most commonly used for single-cell and single-nucleus RNA-seq analysis):
    - standard index type:
        - **cells_x_genes.mtx**: The count matrix (in Matrix Market file format); only exonic reads are counted.
        - **cells_x_genes.barcodes.txt**: The barcodes representing the matrix row names.
        - **cells_x_genes.genes.txt**: The gene IDs representing the matrix column names.
        - **cells_x_genes.genes.names.txt**: Same as cells_x_genes.mtx except with gene names instead of gene IDs for the matrix columns.

        - **cells_x_genes.barcodes.prefix.txt**: If sample-specific barcodes are generated in addition to cell barcodes being recorded, then this file will be created and the sample-specific barcodes will be stored here. The lines of this file correspond to the lines in the cells_x_genes.barcodes.txt which contains the cell barcodes (both files will have the same number of lines). The sample-specific barcodes and cell barcodes can be joined together as a unique identifier for downstream analysis.

    - nac index type (same as the standard index type except the .mtx files produced are different):

        - **cells_x_genes.mature.mtx**: The *mature* RNA count matrix.

        - **cells_x_genes.ambiguous.mtx**: The *nascent* RNA count matrix.

        - **cells_x_genes.nascent.mtx**: The *ambiguous* RNA count matrix.

        - **cells_x_genes.cell.mtx**: The *mature+ambiguous* RNA count matrix (note: this is what is quantified in the count matrix with the standard index type workflow option).

        - **cells_x_genes.nucleus.mtx**: The *nascent+ambiguous* RNA count matrix.

        - **cells_x_genes.total.mtx**: The *mature+nascent+ambiguous* RNA count matrix.

For RNA-seq assays (e.g. bulk RNA-seq or Smartseq2) that profile the full length of transcripts in which case it is desirable to perform transcript-level quantification, the **--tcc** option is used.

- The first step to doing transcript-level quantification is to obtain transcript-compatibility counts (TCCs) over equivalence classes (ECs). The TCCs will be outputted into **output_dir/counts_unfiltered/** which contains the following files for the standard workflow:
    - **cells_x_tcc.mtx**: The count matrix containing the TCCs.
    - **cells_x_tcc.barcodes.txt**: The barcodes representing the matrix row names.
    - **cells_x_tcc.ec.txt**: The equivalence classes representing the matrix column names (note: this file has two columns—the first is the equivalence class numbers, which represent the column names, and the second is a comma-separated list of transcript numbers (0 based) for all transcripts within the equivalence class).
- The --tcc option will additionally produce transcript-level estimated counts which will be placed in the **output_dir/quant_unfiltered/** directory which contains the following:
    - **matrix.abundance.mtx**: The matrix containing the transcript-level estimated counts.
    - **matrix.abundance.tpm.mtx**: The matrix containing the TPM-normalized transcript-level abundances.
    - **matrix.efflens.mtx**: A matrix that contains the transcript effective length
    - **matrix.fld.tsv**: A file with two columns, containing the mean and standard deviation, respectively, of the fragment length distribution used to produce transcript-level abundances and effective lengths for each row of the matrix.
    - **matrix.abundance.gene.mtx**: A matrix that is the same as the matrix.abundance.mtx matrix except counts are aggregated to gene-level.

- **matrix.abundance.gene.tpm.mtx**: A matrix that is the same as the matrix.abundance.tpm.mtx matrix except TPMs are aggregated to gene-level.

- **transcripts.txt**: The transcript names representing the matrix column names for the transcript-level quantification matrices.

- **genes.txt**: The gene IDs representing the matrix column names for the gene-level aggregation quantification matrices.

- **transcript_lengths.txt**: The transcript names along with their lengths

*Note: The row names are the individual samples and will be the same as those in the output_dir/counts_unfiltered/cells_x_tcc.barcodes.txt file. The output_dir/matrix.cells and output_dir/matrix.sample.barcodes files provide a mapping between the sample name and the sample barcode.

*Note: The --matrix-to-directories option will output each row of the matrix into a separate subdirectory. In other words, using this option will produce multiple new directories within output_dir/quant_unfiltered/. Each one will be named abundance_{n} (where {n} is the sample number, corresponding to the rows in the matrix files). Within each subdirectory, an abundance.tsv text file and abundance.h5 HDF5 file will be created containing the quantifications for that particular sample. These abundance files are identical to the abundance files produced by the original version of kallisto for bulk RNA-seq.

To load the quantification results into SCANPY (Wolf et al., 2018) for downstream processing in Python, an anndata (Virshup et al., 2021) object needs to be created. A user can import the count matrices into an anndata object (Box 10), or can run kb count with the --h5ad option to generate the anndata object directly.

**Box 10: Loading count matrices into scanpy**

The standard index produces a single count matrix (in output_dir/counts_unfiltered/) which can be loaded into scanpy via an anndata object as follows:

```
import kb_python.utils as kb_utils

adata = kb_utils.import_matrix_as_anndata("cells_x_genes.mtx",
"cells_x_genes.barcodes.txt",
"cells_x_genes.genes.names.txt")
```

The nac index type produces multiple count matrices. If one wishes to investigate different RNA species separately, one can load multiple count matrices as *layers* into the anndata object. The first layer will always be named “spliced” and the second layer will always be named “unspliced”. Below, we load in the “spliced” layer (from the cells_x_genes.cell.mtx count matrix which represents mature+ambiguous counts) and the “unspliced” layer (from the cells_x_genes.nascent.mtx count matrix, which represents the nascent counts).

```
import kb_python.utils as kb_utils

adata_spliced =
kb_utils.import_matrix_as_anndata("cells_x_genes.cell.mtx",
"cells_x_genes.barcodes.txt",
"cells_x_genes.genes.names.txt")

adata_unspliced =
kb_utils.import_matrix_as_anndata("cells_x_genes.nascent.mtx",
"cells_x_genes.barcodes.txt",
"cells_x_genes.genes.names.txt")

adata = kb_utils.overlay_anndatas(adata_spliced, adata_unspliced)
```

Note: If sample-specific barcodes are specified in addition to cell barcodes, one can add `batch_barcodes_path="cells_x_genes.barcodes.prefix.txt"` to `import_matrix_as_anndata` to concatenate the two barcodes together.

If one runs kb count with the --h5ad option, the file adata.h5ad is created alongside the count matrix files. With the nac index, it will have three layers: nascent, mature, and ambiguous, containing those respective matrices. One can read the file in via:

```
adata = anndata.read_h5ad("output_dir/counts_unfiltered/adata.h5ad")
```

For downstream processing in R, one can load the quantification results into Seurat (Hao et al., 2021) (Box 11). Additionally, in R, one can create a Bioconductor SingleCellExperiment (Amezquita et al., 2020) object for use with single-cell analysis R packages such as scran (Lun et al., 2016) and scater (McCarthy et al., 2017) (Box 12).

**Box 11: Loading count matrices into Seurat**

The standard workflow produces a single count matrix (within the directory output_dir/counts_unfiltered/), which can be loaded into Seurat as follows:

```
library(Seurat)

expression_matrix <- ReadMtx(mtx="cells_x_genes.mtx",
features = "cells_x_genes.genes.names.txt",
cells = "cells_x_genes.barcodes.txt",
feature.column=1,
mtx.transpose = TRUE)
```

For the nac workflow, multiple count matrices are produced. For example, in this workflow, the matrix file named cells_x_genes.total.mtx can be used if one wants to consider the total counts (i.e. the sum of the nascent, mature, and ambiguous counts).

**Box 12: Loading count matrices into SingleCellExperiment**

Here, we show how to build a SingleCellExperiment object in R from the standard workflow output count matrix (in output_dir/counts_unfiltered/):

```
library(SingleCellExperiment)
library(Matrix)

counts <- Matrix::readMM("cells_x_genes.mtx")
gene_ids <- readLines("cells_x_genes.genes.txt")
gene_symbols <- readLines("cells_x_genes.genes.names.txt")
barcodes <- readLines("cells_x_genes.barcodes.txt")
sce <- SingleCellExperiment(list(counts=t(counts)),
colData=DataFrame(Barcode=barcodes),
rowData=DataFrame(ID=gene_ids,SYMBOL=gene_symbols))
rownames(sce) <- gene_ids
```

The count matrices are initially unfiltered, which makes them very large and inefficient to process. After filtering for cells with sufficient UMI counts (among other criteria), the matrices that are loaded in will become much smaller and more efficient to process.

*Materials:*

- A 64-bit computer running either macOS, Windows, or a Linux/Unix operating system.
- kb-python version 0.28.2 or later
    - kallisto version 0.50.1 or later (which comes packaged with kb-python)
    - bustools version 0.43.2 or later (which comes packaged with kb-python)
- Python 3.7 or later (for kb-python version 0.28.2)
- Bulk, single-cell, or single-nucleus RNA sequencing reads in (possibly gzip) FASTQ format.

*Timing:*

The runtime depends on the size of the reference being indexed, the number and length of the sequencing reads being processed, other properties of the dataset being quantified, system hardware and the number of threads allotted. The kb ref command only needs to be run once to create the index against which reads will be mapped. With 8 threads on a server with x86-64 architecture and 32 Intel Xeon CPUs (E5-2667 v3 @ 3.20GHz), kb ref, which by default uses the d-list option, takes ~15 min to generate a standard index from the GRCm39 mouse genome (using the respective raw unfiltered GTF file) and an hour to generate the nac index. For the preprocessing of 800 million Illumina sequencing reads (stored in a single pair of fastq.gz files) produced by single-cell RNA-seq from 10x Genomics, kb count with the nac workflow can take under an hour on 8 threads and under 40 min on 16 threads, with an even lower runtime for the standard workflow.

*Troubleshooting:*

The **--verbose** option in kb ref and kb count is helpful for examining the kallisto and bustools commands that are being run as well as their output. This can be used to troubleshoot errors.

The **--overwrite** option in kb ref and kb count can be used to regenerate output files and directories that were produced (or left over) from a previous kb-python run.

The output directory of a kb count run contains multiple JSON (Pezoa et al., 2016) files that contain quality control values such as the percentage of reads pseudoaligned.

When using kb ref to generate a kallisto index, a genome FASTA file (not a transcriptome FASTA file) should be supplied along with the genome annotation GTF file. A transcriptome file will automatically be generated by kb ref and be indexed by kallisto. In general, the Ensembl (Harrison et al., 2024) *.dna.toplevel.fa.gz* files or the GENCODE (Frankish et al., 2023) *.primary_assembly.genome.fa.gz* files should be used as the reference genome.

When using kb count, one should make sure that the value supplied to the -x technology string option matches the assay from which the sequencing reads were generated. Note that if the technology string begins with a -, for example: -1,0,0:0,0,5:0,5,0, one would need to write -x " -1,0,0:0,0,5:0,5,0" to avoid the string being misinterpreted as a command-line flag.

Additional troubleshooting information is shown in Table 3.5.

| Step | Problem | Possible reason | Solution |
|---|---|---|---|
| kb ref | Error: temporary directory 'tmp' exists! | Another instance of kb-python is running or the temporary directory 'tmp' already exists from a previous kb-python run that terminated prematurely | Use --tmp to specify a different temporary directory or delete the 'tmp' directory before rerunning kb-python |
| | SIGILL illegal Instruction | kallisto binary is incompatible with your system | Install kallisto from source and follow the instructions in Supplementary Note 1 |
| | Error: input file does not exist, is ill-formed or is not in FASTA/ FASTQ/ GFA format | Either the FASTA file or GTF file is empty, truncated or corrupted | Redownload the genome FASTA file and genome GTF file |
| | | | Note: this error can also arise if a transcriptome FASTA file was supplied to kb ref instead of a genome FASTA file, or if the FASTA file and GTF file are incompatible (e.g., one was downloaded from GENCODE and the other was downloaded from ENSEMBL). In these cases, the solution is also to download a correct pair of genome FASTA and GTF files |
| kb count | 'Error: incompatible indices' or 'Segmentation fault' at the 'kallisto bus' step | Either the index file being supplied is corrupted, is not an actual kallisto index file, or was an index file generated by a version of kallisto that utilized a different index format; kallisto version 0.50.1 utilizes a different index format than previous versions and future versions of kallisto may probably adopt a newer index format | The kallisto index should be regenerated |
| | Very low counts or very few reads being pseudoaligned | The strandedness setting is wrong | Rerun with --strand=unstranded |
| | | The technology specified is incorrect | Contact the source of the data to obtain the details about the assay, and then ensure that the technology specified via -x and that the on-list specified via -w are compatible with the FASTQ files produced by that assay |
| | | The index being used is wrong | Ensure that you are using the correct species' index (i.e., not using a mouse index to map human reads). Also ensure that, if you are quantifying data from nuclei, the nac index type is being used |
| | Error: temporary directory 'tmp' exists! | Another instance of kb-python is running or the temporary directory 'tmp' already exists from a previous kb-python run that terminated prematurely | Use --tmp to specify a different temporary directory or delete the 'tmp' directory before rerunning kb-python |
| | SIGILL illegal Instruction | kallisto binary is incompatible with your system | Install kallisto from source |
| | Program is hanging at the "bustools count" step | The t2g (transcripts-to-gene mapping) file created by kb ref should be the exact file used by kb count when running kb count on that index. All the transcripts in the t2g file must be exactly the same as the transcripts present in the kallisto index. Incompatibilities can lead to unpredictable behavior in the bustools quantification step | Fix the t2g file or make a new t2g file with a corresponding kallisto index by rerunning kb ref |

***Table 3.5****. Troubleshooting kallisto, bustools, and kb-python run issues.*

*Procedure:*

Here, we describe the procedures to use for mouse samples of paired-end bulk RNA-seq, 10x (version 3) single-cell RNA-seq, and 10x (version 3) single-nucleus RNA-seq.

Bulk RNA-seq

**Input:**

- Paired-end unstranded mouse RNA-seq reads (3 samples):
  sample1_R1.fastq.gz sample1_R2.fastq.gz
  sample2_R1.fastq.gz sample2_R2.fastq.gz
  sample3_R1.fastq.gz sample3_R2.fastq.gz

**1. Install kb-python**

```
pip install kb_python
```

**2. Download the mouse genome and annotation files**

```
wget ftp.ensembl.org/pub/release-108/fasta/mus_musculus/dna/Mus_musculus.GRCm39.dna.primary_assembly.fa.gz
wget ftp.ensembl.org/pub/release-108/gtf/mus_musculus/Mus_musculus.GRCm39.108.gtf.gz
```

**3. Build the index**

```
kb ref -i index.idx -g t2g.txt -f1 cdna.fasta \
    Mus_musculus.GRCm39.dna.primary_assembly.fa.gz \
    Mus_musculus.GRCm39.108.gtf.gz
```

**4. Map the input sequencing reads to the index**

```
kb count -x BULK -o output_dir -i index.idx -g t2g.txt \
    --parity=paired --strand=unstranded \
    --tcc --matrix-to-directories \
    sample1_R1.fastq.gz sample1_R2.fastq.gz \
    sample2_R1.fastq.gz sample2_R2.fastq.gz \
    sample3_R1.fastq.gz sample3_R2.fastq.gz
```

**5. Analyze the output**

Output for sample 1:
- output_dir/quant_unfiltered/abundance_1/abundance.tsv
- output_dir/quant_unfiltered/abundance_1/abundance.gene.tsv
- output_dir/quant_unfiltered/abundance_1/abundance.h5

Output for sample 2:
- output_dir/quant_unfiltered/abundance_2/abundance.tsv
- output_dir/quant_unfiltered/abundance_2/abundance.gene.tsv
- output_dir/quant_unfiltered/abundance_2/abundance.h5

Output for sample 3:
- output_dir/quant_unfiltered/abundance_3/abundance.tsv
- output_dir/quant_unfiltered/abundance_3/abundance.gene.tsv
- output_dir/quant_unfiltered/abundance_3/abundance.h5

The abundance.tsv files contain the transcript-level abundances. The abundance.h5 file contains the same information as the abundance.tsv files except in HDF5 format. The abundance.gene.tsv files contain the gene-level abundances (taken by summing up the transcript-level abundances for each gene). These files can be used in downstream differential gene expression programs.

Single-cell RNA-seq

**Input:**
- 10x version 3 single-cell RNA-seq reads: R1.fastq.gz and R2.fastq.gz

**1. Install kb-python**

```
pip install kb_python
```

**2. Download the mouse genome and annotation files**

```
wget ftp.ensembl.org/pub/release-108/fasta/mus_musculus/dna/Mus_musculus.GRCm39.dna.primary_assembly.fa.gz

wget ftp.ensembl.org/pub/release-108/gtf/mus_musculus/Mus_musculus.GRCm39.108.gtf.gz
```

**3. Build the index**

```
kb ref -i index.idx -g t2g.txt -f1 cdna.fasta \
     Mus_musculus.GRCm39.dna.primary_assembly.fa.gz \
     Mus_musculus.GRCm39.108.gtf.gz
```

**4. Map the input sequencing reads to the index**

```
kb count -x 10xv3 -o output_dir -i index.idx -g t2g.txt \
    R1.fastq.gz R2.fastq.gz
```

**5. Analyze the output**

Output:

- output_dir/counts_unfiltered/cells_x_genes.mtx
- output_dir/counts_unfiltered/cells_x_genes.barcodes.txt
- output_dir/counts_unfiltered/cells_x_genes.genes.txt
- output_dir/counts_unfiltered/cells_x_genes.genes.names.txt

The cells_x_genes.mtx is the count matrix file with the barcodes (the row names) listed in cells_x_genes.barcodes.txt and the gene names (the column names) listed in cells_x_genes.genes.names.txt (for gene IDs instead of gene names, use cells_x_genes.genes.txt).

Single-nucleus RNA-seq

**Input:**

- 10x version 3 single-nucleus RNA-seq reads: R1.fastq.gz and R2.fastq.gz

**1. Install kb-python**

```
pip install kb_python
```

**2. Download the mouse genome and annotation files**

```
wget ftp.ensembl.org/pub/release-108/fasta/mus_musculus/dna/Mus_musculus.GRCm39.dna.primary_assembly.fa.gz

wget ftp.ensembl.org/pub/release-108/gtf/mus_musculus/Mus_musculus.GRCm39.108.gtf.gz
```

**3. Build the index**

```
kb ref --workflow=nac -i index.idx -g t2g.txt \
    -c1 cdna.txt -c2 nascent.txt -f1 cdna.fasta -f2 nascent.fasta \
    Mus_musculus.GRCm39.dna.primary_assembly.fa.gz \
    Mus_musculus.GRCm39.108.gtf.gz
```

**4. Map the input sequencing reads to the index**

```
kb count -x 10xv3 --workflow=nac -o output_dir \
     -i index.idx -g t2g.txt -c1 cdna.txt -c2 nascent.txt \
     --sum=total R1.fastq.gz R2.fastq.gz
```

**5. Analyze the output**

Output:

- output_dir/counts_unfiltered/cells_x_genes.mature.mtx
- output_dir/counts_unfiltered/cells_x_genes.nascent.mtx
- output_dir/counts_unfiltered/cells_x_genes.ambiguous.mtx
- output_dir/counts_unfiltered/cells_x_genes.cell.mtx
- output_dir/counts_unfiltered/cells_x_genes.nucleus.mtx
- output_dir/counts_unfiltered/cells_x_genes.total.mtx
- output_dir/counts_unfiltered/cells_x_genes.barcodes.txt
- output_dir/counts_unfiltered/cells_x_genes.genes.txt
- output_dir/counts_unfiltered/cells_x_genes.genes.names.txt

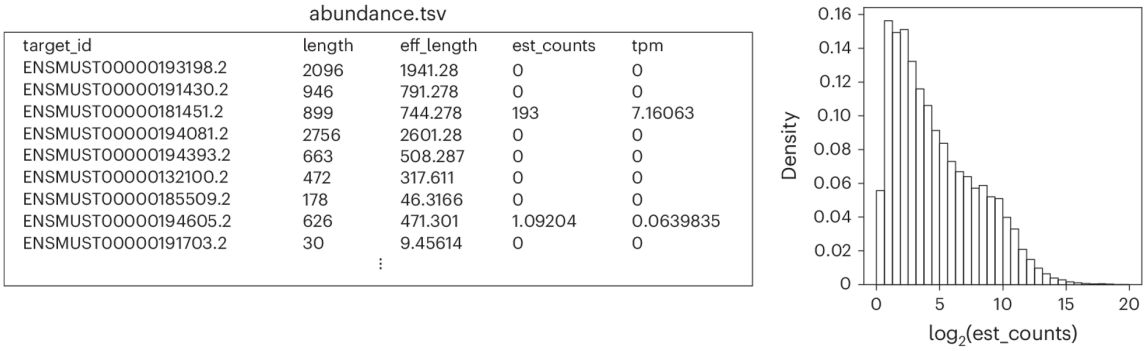
This workflow can be used for both single-cell RNA-seq and single-nucleus RNA-seq. Many count matrix files (.mtx files) are generated. For quantification of total RNA present in each cell or nucleus, one would want to use the cells_x_genes.total.mtx. For biophysical models that jointly consider spliced and unspliced transcripts, one may want to use cells_x_genes.cell.mtx (for the "spliced" transcripts) and cells_x_genes.nascent.mtx (for the "unspliced" transcripts).

The barcodes (the matrix row names) are listed in cells_x_genes.barcodes.txt and the gene names (the matrix column names) are listed in cells_x_genes.genes.names.txt (for gene IDs instead of gene names, use cells_x_genes.genes.txt).

***Additional note: A more straightforward way of quantifying single-nucleus RNA-seq (if one doesn't desire multiple count matrices) is to use kb count with the standard workflow against an index created via the nac workflow option of kb ref. This will give you the "total" counts in a single matrix file: output_dir/counts_unfiltered/cells_x_genes.mtx, rather than producing multiple count matrix files.***

*Example quantification files:*

Following a bulk RNA-seq analysis, one will obtain an abundance.tsv file as the main quantification output to be used in further analysis. An example of such an abundance.tsv file from an analysis run on a mouse liver tissue RNA-seq sample (Huntley et al., 2016), along with the distribution of counts from that analysis, in shown in Figure 3.20.



abundance.tsv

| target_id | length | eff_length | est_counts | tpm |
|---|---|---|---|---|
| ENSMUST00000193198.2 | 2096 | 1941.28 | 0 | 0 |
| ENSMUST00000191430.2 | 946 | 791.278 | 0 | 0 |
| ENSMUST00000181451.2 | 899 | 744.278 | 193 | 7.16063 |
| ENSMUST00000194081.2 | 2756 | 2601.28 | 0 | 0 |
| ENSMUST00000194393.2 | 663 | 508.287 | 0 | 0 |
| ENSMUST00000132100.2 | 472 | 317.611 | 0 | 0 |
| ENSMUST00000185509.2 | 178 | 46.3166 | 0 | 0 |
| ENSMUST00000194605.2 | 626 | 471.301 | 1.09204 | 0.0639835 |
| ENSMUST00000191703.2 | 30 | 9.45614 | 0 | 0 |



***Figure 3.20****: Example output of bulk RNA-seq quantification.*
*The abundance.tsv file was produced from running the protocol on a bulk RNA-seq sample (Gene Expression Omnibus accession ID: GSM1931645). The est_counts represents the estimated counts and the transcripts per million (tpm) represents the length- and depth-normalized abundance for each transcript ID. Additionally, transcript length and effective length (eff_length) information are provided. In the histogram, the distribution of estimated counts (est_counts; omitting transcripts with zero counts) is shown.*

For single-cell RNA-seq analysis, one will obtain a count matrix file as the main quantification output to be used in further analysis. Additionally, the files containing the row names (the cell barcodes) and the column names (the gene identifiers) will be outputted. The count matrix file from an analysis run on a mouse neuron single-cell RNA-seq sample prepared using 10x version 3 chemistry is shown in Figure 3.21, along with the row names and column names. The knee plot describing the distribution of UMI counts across barcodes for that analysis is also shown. Note that these results were produced using the standard index type; when using the nac index type (e.g., for single-nucleus RNA-seq), the output structure will be similar except multiple count matrix files will be produced (for nascent, mature, and ambiguous RNA).

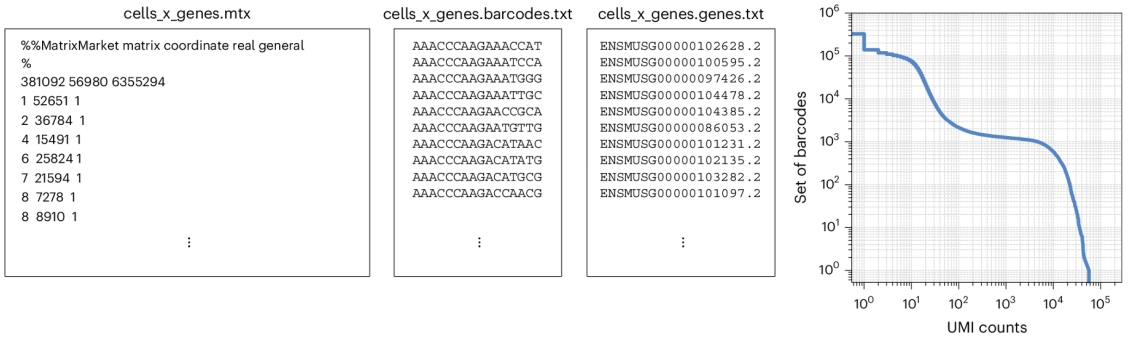



***Figure 3.21**: Example output of single-cell RNA-seq quantification.*
*The count matrix file, in sparse Matrix Market format, from a mouse neuron single-cell RNA-seq sample (neuron_1k_v3_fastqs, 10x Genomics) is shown (cells_x_genes.mtx). The count matrix contains 381,092 rows (i.e., cell barcodes), 56,980 columns (i.e., genes) and 6,355,294 nonzero entries. The cells_x_genes.barcodes.txt file containing the list of the 381,092 barcodes, corresponding to the rows of the matrix. The cells_x_genes.genes.txt file contains the list of the 56,980 genes (in Ensembl gene ID format), corresponding to the columns of the matrix. The knee plot shows the distribution of UMI counts across barcodes.*

*Supplementary Manual:*

Installation of kallisto and bustools from source or installation of specific versions.

**Installing kallisto and bustools from source**

kallisto (version 0.50.1):

```
git clone --branch v0.50.1 https://github.com/pachterlab/kallisto
cd kallisto
mkdir build
cd build
cmake ..
make
make install
```

bustools (version 0.43.2):

```
git clone --branch v0.43.2 https://github.com/BUStools/bustools
cd bustools
mkdir build
cd build
cmake ..
make
make install
```

Note: The --branch argument can be omitted to install the latest version of the software.

**Using kb_python with kallisto and bustools installed from source**

kb_python can be run with compiled binaries by supplying the paths to the binaries:

```
kb ref --kallisto=/path/to/kallisto --bustools=/path/to/bustools …
```

```
kb count --kallisto=/path/to/kallisto --bustools=/path/to/bustools …
```

**Installing a specific version of kb_python**

A specific version of kb_python (e.g. version 0.28.2) can installed as follows:

```
pip install kb_python==0.28.2
```

Indexing a custom set of k-mers.

**Indexing a custom set of k-mers**

When multiple sequences may belong to the same “target”, as is the case with genetic polymorphisms, it can be desirable to index k-mers distributed across multiple targets rather than across a single contiguous target sequence. The target names in the input FASTA file must be numbers (specifically, zero-indexed numerical identifiers). Each k-mer in the target sequence is associated with the target name specified in the header line. Indexing this FASTA file can then be accomplished with the custom index type using the **--distinguish** keyword.

**custom** workflow (--distinguish):

```
kb ref --workflow=custom -i index.idx --distinguish custom.fasta
```

```
kallisto index -t 8 -i index.idx --distinguish custom.fasta
```

Example custom.fasta file (with 3 targets):

```
>0
ACTCTATCATCATCTACTACTACTCGCAGCGACGACATCAGCTTTTTT
>1
GCGCGCCGCCGACGACACGCAGAGAAGAAAGCGCGACGAC
>2
TTATGTGTCGTGTAGTCGTAGTGTGTCGTGCCGCCGCGCGCAAA
>2
ATATACGATCATCAGCGACAGACTACTTCAGAAGACTATCA
>0
GTCGATCGGTGTCACATGCGCAAGCGTCAGCGACACGACTTCGG
```

**D-listing a custom set of k-mers**

When FASTA sequences are supplied to **--d-list**, distinguishing flanking k-mers (DFKs) are extracted from those sequences and placed in a D-list. Reads containing D-list k-mers will not be mapped. One can also specify a custom set of k-mers to be in the D-list, by using an empty sequence header. In the following example, since the header is absent, *all* k-mers in the sequence will be D-listed (if a header were present, only DFKs would be D-listed).

```
>
ACGCGACATAGCAGACTAGACATTATTTACGTATTATGATAGTAGAT
```

Filtering GTF entries when constructing the reference.

**kb ref: --include-attributes and --exclude-attributes to filter GTF entries**

Specific GTF entries can be included or excluded when building a reference transcriptome from a genome FASTA and GTF file. This can be done by using the following arguments to kb ref:

--include-attribute KEY:VALUE

--exclude-attribute KEY:VALUE

where KEY is the name of the field (e.g. gene_biotype) in the GTF file and the VALUE is the value of the field (e.g. protein_coding).

The box below shows an example of how to use --include-attribute to include only certain gene biotypes (the remaining gene biotypes present in the GTF file will not be included). Note that these are the same biotypes included in the Ensembl GRCh38 Cell Ranger reference (as of Cell Ranger version 7.1.0).

```
kb ref -i index.idx -g t2g.txt -f1 cdna.fasta \
    --include-attribute gene_biotype:protein_coding \
    --include-attribute gene_biotype:lncRNA \
    --include-attribute gene_biotype:lincRNA \
    --include-attribute gene_biotype:antisense \
    --include-attribute gene_biotype:IG_LV_gene \
    --include-attribute gene_biotype:IG_V_gene \
    --include-attribute gene_biotype:IG_V_pseudogene \
    --include-attribute gene_biotype:IG_D_gene \
    --include-attribute gene_biotype:IG_J_gene \
    --include-attribute gene_biotype:IG_J_pseudogene \
    --include-attribute gene_biotype:IG_C_gene \
    --include-attribute gene_biotype:IG_C_pseudogene \
    --include-attribute gene_biotype:TR_V_gene \
    --include-attribute gene_biotype:TR_V_pseudogene \
    --include-attribute gene_biotype:TR_D_gene \
    --include-attribute gene_biotype:TR_J_gene \
    --include-attribute gene_biotype:TR_J_pseudogene \
    --include-attribute gene_biotype:TR_C_gene \
    genome.fasta genome.gtf
```

**Supplementary Manual**: Reference for kallisto and bustools commands.

# 1. kallisto

Running kallisto usually involves two steps: 1) Indexing a FASTA file of target sequences via `kallisto index`, and 2) Mapping sequencing reads to kallisto index using `kallisto bus`.

## 1.1 kallisto index

Builds a kallisto index.

Usage: kallisto index [arguments] FASTA-files

```
Required argument:
-i, --index=STRING       Filename for the kallisto index to be constructed

Optional arguments:
-k, --kmer-size=INT      k-mer (odd) length (default: 31, max value: 31)

-t, --threads=INT        Number of threads to use (default: 1)

-d, --d-list=STRING      Path to a FASTA-file containing sequences to mask
                         from quantification (i.e. to extract distinguishing
                         flanking k-mers from).

--make-unique            Replace repeated target names with unique names

--aa                     Generate index from a FASTA-file containing
                         amino acid sequences

--distinguish            Generate index where sequences are distinguished
                         by the sequence name, for example, when indexing
                         k-mers distributed across multiple targets rather
                         than across a single contiguous target sequence.

-T, --tmp=STRING         Temporary directory (default: tmp)

-m, --min-size=INT       Length of minimizers (default: automatically chosen)

-e, --ec-max-size=INT    Maximum number of targets in an equivalence class
                         (default: no maximum)
```

Among the optional arguments in `kallisto index`, in a general use case, typically only `-i` (`--index`; to specify the name of the index output filename), `-t` (`--threads`; to specify the number of threads), and `-d` (`--d-list`; to specify the filename from which to extract distinguishing flanking k-mers) are used.

## 1.2 kallisto bus

Generates a BUS file containing the results from mapping sequencing reads to a kallisto index.

Usage:
kallisto bus [arguments] FASTQ-files
kallisto bus [arguments] --batch=batch.txt

```
Required arguments:
-i, --index=STRING            Filename for the kallisto index to be used for
                              pseudoalignment

-o, --output-dir=STRING       Directory to write output to

-x, --technology=STRING       The "technology" string for the sequencing
                              technology used

Other arguments:
-l, --list                    List the technologies that are hard-coded into
                              kallisto so the name of the technology can
                              simply be supplied as the technology string

-B, --batch=FILE              Path to a batch file. The batch file is a text
                              file listing all the samples to be analyzed
                              with the paths to their respective FASTQ files.
                              If a batch file is supplied, then one shouldn't
                              supply FASTQ files on the command line.

-t, --threads=INT             Number of threads to use (default: 1)

-b, --bam                     Input file is a BAM file rather than a set of
                              FASTQ files. Note: This is a nonstandard
                              workflow. It is strongly recommended to supply
                              FASTQ files rather than use this option and not
                              all technologies are supported by this option.

-n, --num                     Output read number in flag column of BUS file
                              The read number is zero-indexed. One can view
                              the read numbers by inspecting the BUS file
                              using bustools text. This option is useful for
                              pulling specific mapped reads out of the FASTQ
                              file or for examining which reads did not end
                              up being mapped by kallisto. (Important note:
                              BUS files with read numbers in the flag column
                              can NOT be used in quantification tasks with
                              bustools). (Note: incompatible with --bam)

-N, --numReads=INT            Maximum number of reads to process from
                              supplied input. This is useful for processing
                              a small subset of reads from a large sequencing
                              experiment as a quick quality control.
                              Moreover, the program returns 1 if the number
```

```
                              of reads processed from the input is less than
                              the number supplied here. This is useful for
                              catching errors when we expect a certain number
                              of reads to be present in the input but not all
                              the reads end up being there.

-T, --tag=STRING              5’ tag sequence to identify UMI reads for
                              certain technologies. This is useful for
                              smart-seq3 where the UMI-containing reads have
                              an 11-bp tag sequence (ATTGCGCAATG) located at
                              the beginning of the UMI location. If this tag
                              sequence is present immediately before the UMI
                              location, then the UMI is processed into the
                              output BUS file; for all other sequences, the
                              UMI field in the BUS file is left empty (the
                              field is populated with the value -1 in binary
                              format).
                              Note: Matching the tag sequence is done with
                              a hamming distance error tolerance of 1 if the
                              tag is longer than 5 nucleotides. Otherwise,
                              no error tolerance is permitted.
                              Note: If strand-specificity is enabled, it will
                              only be applied to the UMI-containing reads.

    --fr-stranded             Strand specific reads, first read forward

    --rf-stranded             Strand specific reads, first read reverse

    --unstranded              Treat all read as non-strand-specific

    --paired                  Treat reads as paired (i.e. if two biological
                              read sequences are present across two FASTQ
                              files, they will be mapped taking into account
                              their paired-endness: fragment length
                              distribution will be estimated for the read
                              pairs, and only one read in the pair needs to
                              map successfully in order to be considered
                              successful pseudoalignment)

    --aa                      Align to index generated from a FASTA-file
                              containing amino acid sequences

    --inleaved                Specifies that input is an interleaved FASTQ
                              file. That is, only one FASTQ file is supplied
                              and the sequences are interleaved. For example,
                              instead of an R1 and R2 FASTQ file, a single
                              FASTQ file can be supplied where the reads are
                              listed in order of each R2 read immediately
                              following each R1 read. This is also useful
                              when piping interleaved output generated by
                              another program directly into
                              kallisto bus which can be done by
                              supplying - as the input file in lieu of
                              FASTQ file names.
```

```
--batch-barcodes        Records both the generated sample-specific
                        barcodes as well as the cell barcodes extracted
                        from the reads in the output BUS file. If not
                        supplied, then the sample-specific barcodes are
                        not recorded.
```

In the output directory specified by `-o` or `--output-dir`, the following files are made:

- output.bus: A BUS file containing the mapped reads information, which will be further processed using bustools.
- transcripts.txt: A text file containing a list of the names of the targets or transcripts used.
- matrix.ec: A text file containing the equivalence classes. The equivalence class number (zero-indexed) is in the first column and a comma-separated list of target or transcript IDs belonging to that equivalence class are in the second column. The transcript IDs are numbers (zero-indexed) that correspond to the line numbers (zero-indexed) in the transcripts.txt file.
- run_info.json: Contains information about the run, including percent of reads pseudoaligned, number of reads processed, index version, etc.
- flens.txt: Only produced when using paired-end mapping. Contains the fragment length distribution, which can be used by `kallisto quant-tcc` to produce TPM abundance values.

## 1.3 kallisto quant-tcc

Quantifies abundance from pre-computed transcript-compatibility counts. It takes in a transcript compatibility counts (TCC) matrix outputted by `bustools count` and runs an expectation-maximization (EM) algorithm to produce transcript abundances. This is useful for producing TPM values from bulk RNA-seq and smart-seq2 RNA-seq data. The output files can be used by bulk RNA-seq differential gene expression programs.

Usage: kallisto quant-tcc [arguments] transcript-compatibility-counts-file

```
Required arguments:
-o, --output-dir=STRING       Directory to write output to.


-e, --ec-file=FILE            File containing equivalence classes
                              (the equivalence class file in the same
                              directory as the output matrix file
                              should be used).

Other arguments:
-i, --index=STRING            Filename for the kallisto index
                              to be used
                              (required if --txnames is not supplied
                              or if any of the fragment length
                              options: -f, -l, -s, is supplied
                              since the index contains transcript
                              lengths, which is necessary for length
                              normalization).

-T, --txnames=STRING          File with names of transcripts
                              (required if index file not supplied).

-f, --fragment-file=FILE      File containing fragment length
                              distribution (flens.txt outputted by
                              kallisto).

-l, --fragment-length=DOUBLE  Estimated average fragment length.

-s, --sd=DOUBLE               Estimated standard deviation of fragment
                              length
                              (note: -l, -s values only should be
                              supplied when effective length
                              normalization needs to be performed
                              but --fragment-file is not specified).

                              Note: If none of the fragment length
                              options -f -l -s are supplied, then
                              effective length normalization is not
                              performed (i.e. transcript length isn't
                              taken into account when quantification
                              is performed).

-p, --priors=FILE             Priors for the EM algorithm, either as
```

```
                              raw counts or as probabilities.
                              Pseudocounts are added to raw counts to
                              prevent zero valued priors. Supplied in
                              the same order as the transcripts in the
                              transcriptome (e.g. in --txnames).

-t, --threads=INT             Number of threads to use (default: 1).

-g, --genemap=FILE            File for mapping transcripts to genes
                              (this is the t2g.txt file produced by
                              kb ref in kb-python and is required for
                              obtaining gene-level abundances).

-G, --gtf=FILE                GTF file for transcriptome information
                              (can be used instead of --genemap for
                              obtaining gene-level abundances).

-b, --bootstrap-samples=INT   Number of bootstrap samples (default: 0)
                              Bootstrap samples are useful for
                              obtaining inferential variance which can
                              be used by programs such as sleuth.

    --matrix-to-files         Reorganize matrix output into abundance
                              tsv files.

    --matrix-to-directories   Reorganize matrix output into abundance
                              tsv files across multiple directories.

    --seed=INT                Seed for the bootstrap sampling
                              (default: 42).

    --plaintext               Output plaintext only, not HDF5
                              (When --matrix-to-directories or
                              --matrix-to-files are supplied, HDF5
                              files are outputted by default, in
                              addition to the plaintext abundance tsv
                              files since HDF5 files containing
                              abundance information are used by
                              programs such as sleuth; this option
                              disables that).
```

In the output directory specified by `-o` or `--output-dir`, the following files are made:

- matrix.abundance.mtx: A sample-by-transcript (or cell-by-transcript) MatrixMarket sparse matrix file containing the estimated transcript counts.
- matrix.abundance.gene.mtx: A sample-by-gene (or cell-by-gene) MatrixMarket sparse matrix file containing the estimated transcript counts summed up to gene-level. Only made if a transcript-to-gene mapping was provided.
- matrix.abundance.tpm.mtx: A sample-by-transcript (or cell-by-transcript) MatrixMarket sparse matrix file containing the normalized transcript abundances (if effective length normalization is performed, then the results are in length-normalized TPM units; otherwise the results are in CPM [counts-per-million] units

wherein each value is normalized by the sum of all counts for that particular sample or cell).

- matrix.abundance.gene.tpm.mtx: A sample-by-gene (or cell-by-gene) MatrixMarket sparse matrix file containing the same information as matrix.abundance.tpm.mtx except summed up to gene-level if a transcript-to-gene mapping was provided.
- transcripts.txt: A text file containing a list of the names of the targets or transcripts used (not made if a transcripts file was already provided via `--txnames`). These transcripts correspond to the columns of transcripts in the matrix abundance output files.
- genes.txt: A text file containing a list of genes, if a transcript-to-gene mapping was provided. These genes correspond to the columns of genes in the matrix abundance output files.
- --matrix-to-files: If this option is provided, the abundance output files will be named abundance_{n}.tsv and abundance_{n}.h5 (hdf5 format) where {n} is the sample number or cell number (which corresponds to the rows in the matrix files). If bootstrapping is enabled, additional abundance tsv files (starting with the prefix bs_abundance_{n}_) will be created for each bootstrap sample. If a transcript-to-gene mapping is provided, abundance.gene_{n}.tsv files will be created as well with the gene-level quantification.
- --matrix-to-directories: If this option is provided, directories named abundance_{n} (where {n} is the sample number or cell number, corresponding to the rows in the matrix files) will be created. Within each directory, an abundance.tsv text file and abundance.h5 HDF5 file will be created containing the quantifications for that particular sample or cell. If bootstrapping is enabled, additional abundance tsv files (starting with the prefix bs_abundance_) will be created for each bootstrap sample. If a transcript-to-gene mapping is provided, an abundance.gene.tsv file will be created within each directory with the gene-level quantification.

The first few lines of an abundance tsv file looks as follows:

```
target_id           length   eff_length est_counts tpm
ENST00000641515.2   2618     2349.39     0          0
ENST00000426406.4   939      670.39      0          0
ENST00000332831.4   995      726.39      0          0
ENST00000616016.5   3465     3196.39     5.68407    0.128913
ENST00000618323.5   3468     3199.39     1.83535    0.041586
```

## 1.4 kallisto quant

`kallisto quant` is an old usage of kallisto when kallisto was first developed for bulk RNA-seq quantification. It is now recommended that users use the `kallisto bus` command instead.

As such, documentation for the old `kallisto quant` is not within the scope of this protocol.

## 1.5 kallisto inspect

Inspects and gives information about an index. The index can be loaded more quickly by using multiple threads, which can be specified by the **-t** option.

Example usage:

```
kallisto inspect -t 8 /path/to/kallisto/index.idx
```

Sample output:

```
[index] k-mer length: 31
[index] number of targets: 252,301
[index] number of k-mers: 155,644,518
[index] number of distinguishing flanking k-mers: 7,425,493
[inspect] Index version number = 12
[inspect] number of unitigs = 9411252
[inspect] minimizer length = 23
[inspect] max EC size = 3873
[inspect] number of ECs discarded = 0
```

## 1.6 kallisto version

Prints out the version of the kallisto software that is being used

## 1.7 kallisto cite

Prints out citation information

# 2. bustools

Bustools is run on BUS files generated by the `kallisto bus` command. The first step in working with BUS files is usually to sort the BUS file using `bustools sort`. This will organize the BUS file, making it suitable for use with other bustools commands. In a standard workflow, the sorted BUS file is error-corrected to a barcode on list via `bustools correct`, then sorted again, then quantified into count matrices via `bustools count`. There are many bustools commands, some of which are outside the scope of this protocol and some of which are in development; therefore only the bustools commands relevant to most RNA-seq analyses are presented here.

Many of the bustools commands can read from the standard input (stdin), by specifying `-` as the input file and write to standard output (stdout) using the `-p` flag if available.

## 2.1 bustools sort

Sorts a BUS file. `bustools sort` (using the default options) should always be done before any additional processing of the BUS file following generation of the BUS file from the `kallisto bus` command. Many bustools commands will not work properly with an unsorted BUS file. Increasing the number of threads and maximum memory will speed up sorting.

The default behavior is to sort by barcode, UMI, equivalence class (ec), then the flag column.

Usage: bustools sort [options] bus-files

```
Arguments:
-t, --threads=INT         Number of threads to use (default: 1).

-m, --memory=STRING       Maximum memory used (default: 4G).

-T, --temp=STRING         Location and prefix for temporary files
                          (required if using -p, otherwise
                          defaults to output).

-o, --output=STRING       Filename to output sorted BUS file into.

-p, --pipe                Write to standard output.


    --umi                 Sort by UMI, barcode, then ec.

    --count               Sort by multiplicity (count), barcode, UMI, then ec.

    --flags               Sort by flag, ec, barcode, then UMI.

    --flags-bc            Sort by flag, barcode, UMI, then ec.
```

```
    --no-flags            Ignore and reset the flag column while sorting.
                          If read numbers are present in the flag column of
                          the BUS file, sorting using this option renders
                          BUS file suitable for use in generating
                          count matrices.
```

## 2.2 bustools correct

Error-corrects the barcodes in a BUS file to an “on list”.

Error correction is done based on a hamming distance 1 mismatch between each BUS file barcode sequence and each “on list” sequence. For barcode error correction, the “on list” file simply contains a list of sequences in the “on list”.

Another operation supported is the replacement operation: Each “on list” sequence (in the first column of the “on list” file) has a replacement sequence (in the second column of the “on list” file) designated therefore if a BUS file barcode has an exact match to one of those “on list” sequences, it is replaced with its replacement sequence.

Note: The input BUS file need not be sorted.

Usage: bustools correct [options] bus-files

```
Arguments:
-o, --output=STRING      Filename to output barcode-corrected BUS file into.

-w, --onlist=FILE        File containing the “on list” sequences.

-p, --pipe               Write to standard output.

-r, --replace            Perform the replacement operation rather than the
                         barcode error correction operation for the file
                         supplied in the -w option.
```

## 2.3 bustools count

Generates count matrices from BUS files that have been sorted and barcode-error-corrected.

Usage: bustools count [options] sorted-bus-files

```
Arguments:
-o, --output=STRING      The prefix of the output files for count matrices.

-g, --genemap=FILE       File for mapping transcripts to genes
                         (when using kb ref in kb-python, this is the
                         t2g.txt file produced by kb ref).

-e, --ecmap=FILE         File for mapping equivalence classes to transcripts.

-t, --txnames=FILE       File with names of transcripts.

    --genecounts         Aggregate counts to genes only.
                         This option generates a gene count matrix; if this
                         option is not supplied, a transcript-compatibility
                         counts (TCC) matrix (where each equivalence class
                         gets a count) is generated instead.

    --umi-gene           Handles cases of UMI collisions. For example, a case
                         may be where two reads with the same UMI
                         sequence and the same barcode map to different
                         genes. With this option enabled, those reads are
                         considered to be two distinct molecules which were
                         unintentionally labeled with the same UMI, and hence
                         each gene gets a count.

    --cm                 Counts multiplicities rather than UMIs. In other
                         words, no UMI collapsing is performed and each
                         mapped read is its own unique molecule regardless of
                         the UMI sequence (i.e. the UMI sequence is ignored).

-m, --multimapping       Include bus records that map to multiple genes.
                         When --genecounts is enabled, this option causes
                         counts to be distributed uniformly across all the
                         mapped genes (for example, if a read multimaps to
                         two genes, each gene will get a count of 0.5).

-s, --split=FILE         Split output matrix in two (plus ambiguous) based on
                         the list of transcript names supplied in this file.
                         If a UMI (after collapsing) or a read maps to
                         transcripts found in this file, the count is entered
                         into a matrix file with the extension .2.mtx; if it
                         maps to transcripts not in this file, the count is
                         entered into a separate matrix file with the
                         extension .mtx; if it maps to some transcripts in
                         this file and some transcripts not in this file, the
                         count is entered into a third matrix file with the
                         extension .ambiguous.mtx.
```

```
When quantifying nascent, ambiguous, and mature RNA
species, the nascent transcript names (which will
actually simply be the gene IDs themselves) will
be listed in the file supplied to --split so that
the .mtx file contains the mature RNA counts, the
.2.mtx file contains the nascent RNA counts, and the
.ambiguous.mtx file contains the ambiguous RNA
counts. Note that kb-python renames .mtx to
.mature.mtx and renames 2.mtx to .nascent.mtx.
```

Output:
Each output file is prefixed with what is supplied to the `--output` option. In kb count within kb-python, the prefix is cells_x_genes. Thus, the files outputted (when generating a gene count matrix via --genecounts) will be cells_x_genes.mtx (the matrix file), cells_x_genes.barcodes.txt (the barcodes, i.e. the rows of the matrix), and cells_x_genes.genes.txt (the genes, i.e. the columns of the matrix). When generating a TCC matrix, cells_x_genes.ec.txt will be generated in lieu of cells_x_genes.genes.txt as the columns of the matrix will be equivalence classes (ECs) rather than genes. If both sample-specific barcodes and cell barcodes are supplied (as is the case when one uses `--batch-barcodes` in `kallisto bus`), then an additional cells_x_genes.barcodes.prefix.txt file will be created containing the sample-specific barcodes. The lines of this file correspond to the lines in the cells_x_genes.barcodes.txt (both files will have the same number of lines). Finally, when `--split` is supplied, additional .mtx matrix files will be generated (see the `--split` option described above).

## 2.4 bustools inspect

Produces a report summarizing the contents of a sorted BUS file. The report can be output either to standard output or to a JSON file.

Usage: bustools inspect [options] sorted-bus-file

```
Arguments:
-o, --output=STRING       Filename to output sorted BUS file into

-e, --ecmap=FILE          File for mapping equivalence classes to transcripts

-w, --onlist=FILE         File containing the barcodes “on list”

-p, --pipe                Write to standard output
```

Sample report output in standard output (using `-p`):

```
Read in 3148815 BUS records
Total number of reads: 3431849

Number of distinct barcodes: 162360
Median number of reads per barcode: 1.000000
Mean number of reads per barcode: 21.137281

Number of distinct UMIs: 966593
Number of distinct barcode-UMI pairs: 3062719
Median number of UMIs per barcode: 1.000000
Mean number of UMIs per barcode: 18.863753

Estimated number of new records at 2x sequencing depth:
2719327

Number of distinct targets detected: 70492
Median number of targets per set: 2.000000
Mean number of targets per set: 3.091267

Number of reads with singleton target: 1233940

Estimated number of new targets at 2x seuqencing depth:
6168

Number of barcodes in agreement with on-list: 92889
(57.211752%)
Number of reads with barcode in agreement with on-list:
3281671 (95.623992%)
```

Sample report output in JSON format:


```
{
 "numRecords": 3148815,
 "numReads": 3431849,
 "numBarcodes": 162360,
 "medianReadsPerBarcode": 1.000000,
 "meanReadsPerBarcode": 21.137281,
 "numUMIs": 966593,
 "numBarcodeUMIs": 3062719,
 "medianUMIsPerBarcode": 1.000000,
 "meanUMIsPerBarcode": 18.863753,
 "gtRecords": 2719327,
 "numTargets": 70492,
 "medianTargetsPerSet": 2.000000,
 "meanTargetsPerSet": 3.091267,
 "numSingleton": 1233940,
 "gtTargets": 6168,
 "numBarcodesOnOnlist": 92889,
 "percentageBarcodesOnOnlist": 0.57211752,
 "numReadsOnOnlist": 3281671,
 "percentageReadsOnOnlist": 0.95623992
}
```


Note: The numTargets, medianTargetsPerSet, meanTargetsPerSet, numSingleton, and gtTargets values are only generated if the `--ecmap` option is provided. The numBarcodesOnOnlist, percentageBarcodesOnOnlist, numReadsOnOnlist, percentageReadsOnOnlist values are only generated if the `--onlist` is provided.

## 2.5 bustools allowlist

Generates an “on list” based on the barcodes in a sorted BUS file. This is a way of generating an on list that the barcodes in the BUS file will be corrected to, for technologies that don’t provide an on list.

Usage: bustools allowlist [options] sorted-bus-file

```
Arguments:
-o, --output=STRING          Filename to output the “on list” into.

-f, --threshold=INT          A highly optional parameter specifying the
                             minimum number of times a barcode must appear
                             to be included in “on list”. If not provided, a
                             threshold will be determined based on the first
                             200 to 100200 BUS records.
```

## 2.6 bustools capture

Separates a BUS file into multiple files according to the capture criteria.

Usage: bustools capture [options] bus-files

```
Capture options:
-F, --flags                Capture list is a list of flags to capture

-s, --transcripts          Capture list is a list of transcripts to capture

-u, --umis                 Capture list is a list of UMI sequences to capture

-b, --barcode              Capture list is a list of barcodes to capture

Arguments:
-o, --output=STRING        Name of file for the captured BUS output

-x, --complement           Take complement of captured set
                           (i.e. output all BUS records that do NOT
                           match an entry in the capture list)

-c, --capture=FILE         File containing the “capture list”
                           (i.e. list of transcripts, transcripts, flags,
                           UMI sequences, or barcode sequences)

-e, --ecmap=FILE           File for mapping equivalence classes to
                           transcripts (required for --transcripts)

-t, --txnames=FILE         File with names of transcripts
                           (required for --transcripts)

-p, --pipe                 Write to standard output
```

Note: If you use the `-b` (`--barcode`) option and want to capture all records containing a sample-specific barcode from running `--batch-barcodes` in `kallisto bus`, in the “capture list” file, enter the 16-bp sample-specific barcode followed by a * character (e.g. AAAAAAAAAAAAACT*).

## 2.7 bustools text

Converts a binary BUS file into its plaintext representation. The plaintext will have the columns (in order): barcode, UMI, equivalence class, count, flag, and pad. Note: The last two columns will only be outputted if the respective option is specified by the user.

Usage: bustools text [options] bus-files

```
Arguments:
-o, --output=STRING       Filename of the output text file.

-f, --flags               Write the flag column.

-d, --pad                 Write the pad column
                          (the "pad" column is an additional 32-bit field
                          in the BUS file, in case one would like to use the
                          BUS format to store additional data for each BUS
                          record; this column is typically not used).

-p, --pipe                Write to standard output.

-a, --showAll             Show all 32 bases in the barcodes field (e.g. if
                          --batch-barcodes is specified in kallisto bus, the
                          cell barcodes are stored in barcodes field and are
                          used for bustools barcode correction to an on-list;
                          however, the artificial sample-specific barcodes
                          are stored as an additional "hidden" field in the
                          barcodes column, immediately preceding the cell
                          barcodes, and may be truncated or left-padded with
                          A's to fill the 32 bases. For example, if the cell
                          barcode is 12 bases, there will be 4 A's followed
                          by the 16-bp sample-specific barcode followed by
                          the 12-base cell barcode. If the cell barcode is 26
                          bases, the last 6 bases of the sample-specific
                          barcode will be shown followed by the 26-base cell
                          barcode).
```

An example of the plaintext output of a BUS file (with the flag column):

```
AAAAGATCACTATGCACTATCATCGCAAAACCTT 156 2 0
AAAAGATCAGATCGCACACTTTCATAGAGTAACC 438 3 0
AAAAGATCAGATCGCAGCTCTACTTTAGGTATAG 18081 0
AAAAGATCAGCACCTCCTGACTTCAATCGGCATT 44811 0
```

If one runs `kallisto bus` with the `-n` (`--num`) option, the read number (zero-indexed) of the mapped reads will be stored in the flags column (i.e. the fifth column). One can view those read numbers using `bustools text` to identify which reads in the input FASTQ files mapped (and which reads were unmapped).

## 2.8 bustools fromtext

Converts a plaintext representation of a BUS file to a binary BUS file. The plaintext input file should have four columns: barcode, UMI, equivalence class, and count. Optionally, a fifth column (the flags column) can be supplied.

Usage: bustools fromtext [options] text-files

```
Arguments:
-o, --output=STRING           Filename to write the output BUS file

-p, --pipe                    Write to standard output
```

## 2.9 bustools extract

Extracts the successfully mapped sequencing reads from the input FASTQ files that were processed with `kallisto bus` with the `-n` (`--num`) option, which places the read number (zero-indexed) in the flags column of the BUS file. Although BUS files with read numbers present in the flags column should not be used for downstream quantification, they can be used by `bustools extract` to extract the original sequencing reads (as well as by `bustools text` to view the sequencing read number along with the barcode, UMI, and equivalence class).

Note: The BUS file must be sorted by flag. The output BUS file directly from kallisto should already be sorted by flag, but, if not, one can use apply `bustools sort --flag` on the BUS file.

Usage: bustools extract [options] sorted-by-flag-bus-file

```
Arguments:
-o, --output=STRING   Directory that the output FASTQ files will be stored in.

-f, --fastq=STRING    FASTQ file(s) from which to extract reads
                      (comma-separated list). These should be the same files
                      used as input to kallisto bus.

-N, --nFastqs=INT     Number of FASTQ file(s) per run. For example, in 10xv3
                      where there are two FASTQ files (and R1 and R2 file),
                      --nFastqs=2 should be set.
```

Bustools extract is especially useful to use in conjunction with `bustools capture` when one wishes to extract specific reads (e.g. reads that contain a certain barcode or reads whose equivalence class contains a certain transcript). Below, we show an example of how to extract reads from two input files: R1.fastq.gz and R2.fastq.gz entered into a kallisto bus run with results outputted into a directory named output_dir. We'll extract reads that are compatible with either the transcript ENSMUST00000171143.2 or ENSMUST00000131532.2.

Create a file called capture.txt containing the following two lines:

```
ENSMUST00000171143.2
ENSMUST00000131532.2
```

Run the following:

```
bustools capture -c capture.txt --transcripts \
--ecmap=output_dir/matrix.ec \
--txnames=output_dir/transcripts.txt -p \
output_dir/output.bus | bustools extract --nFastqs=2 \
--fastq=R1.fastq.gz,R2.fastq.gz -o extracted_output -
```

The capture results are directly piped into the extract command, and the extracted FASTQ sequencing reads output are placed into the paths extracted_output/1.fastq.gz and extracted_output/2.fastq.gz (for the input files R1.fastq.gz and R2.fastq.gz, respectively).

`bustools extract` does not work when you have sample-specific barcodes in your BUS file because each sample's read number (as recorded in the flags column of the BUS file) starts from 0. To work around this, you should first use `bustools capture` to isolate a specific sample and then supply that specific sample's FASTQ file(s).

## 2.10 bustools umicorrect

Implements the UMI correction algorithm of UMI-tools and outputs a BUS file with the corrected UMIs.

Usage: bustools umicorrect [options] sorted-bus-file

```
Arguments:
-o, --output=STRING      Filename of the output BUS file with UMIs corrected

-p, --pipe               Write to standard output

-g, --genemap=FILE       File for mapping transcripts to genes
                         (when using kb ref in kb-python, this is the
                         t2g.txt file produced by kb ref)

-e, --ecmap=FILE         File for mapping equivalence classes to transcripts

-t, --txnames=FILE       File with names of transcripts
```

## 2.11 bustools compress

Takes in a BUS file, sorted by barcode-umi-ec (i.e. the default option for `bustools sort`), and compresses it.

Usage: bustools compress [options] sorted-bus-file

```
Arguments:
-N, --chunk-size=INT           Number of rows to compress as a single
block

-o, --output=STRING            Filename for the output compressed BUS
file

-p, --pipe                     Write to standard output
```

## 2.12 bustools decompress

Takes in a compressed BUS file and inflates (i.e. decompresses) it.

Usage: bustools decompress [options] compressed-bus-file

```
Arguments:
-o, --output=STRING            Filename for the output decompressed BUS
file

-p, --pipe                     Write to standard output
```

## 2.13 bustools version

Prints out the version of the bustools software that is being used.

## 2.14 bustools cite

Prints out citation information.

An example mouse multiplexed single-nucleus SPLiT-seq preprocessing workflow.

Here we describe how to process a mouse multiplexed single-nucleus SPLiT-seq assay. The input FASTQ files are split across multiple subpools such that two cells may have the same cell barcode but be in different subpools. The SPLiT-seq assay uses both oligo-dT and random hexamer primers (which are represented in the third component of the cell barcode, corresponding to the first round of split pooling). As a result, two sets of matrices will be produced: One with both the oligo-dT and random hexamer barcodes in the same count matrix and one with the oligo-dT barcodes converted into the random hexamer barcodes (so that each barcode is unique to one nucleus). This facilitates investigation of each library type separately (should one wish to generate an “oligo-dT” count matrix and a “random hexamer” count matrix) as well as of the two library types combined together.

**1. Install kb-python.**

```
pip install kb_python
```

**2. Download the mouse genome and annotation files.**

```
wget ftp.ensembl.org/pub/release-108/fasta/mus_musculus/dna/Mus_musculus.GRCm39.dna.primary_assembly.fa.gz

wget ftp.ensembl.org/pub/release-108/gtf/mus_musculus/Mus_musculus.GRCm39.108.gtf.gz
```

**3. Build the index.**

To illustrate index generation with GTF filtering we show below how to filter the GTF file to only keep the relevant biotypes (the same ones that are used in the CellRanger reference). This can improve both accuracy and efficiency. Additional methods to optimize the GTF file can also be used such as the one proposed in Pool et al., 2023[38] which can greatly increase gene detection sensitivity.

```
kb ref --workflow=nac -i index.idx -g t2g.txt \
     -c1 cdna.txt -c2 nascent.txt -f1 cdna.fasta -f2 nascent.fasta \
     --include-attribute gene_biotype:protein_coding \
     --include-attribute gene_biotype:lncRNA \
     --include-attribute gene_biotype:lincRNA \
     --include-attribute gene_biotype:antisense \
     --include-attribute gene_biotype:IG_LV_gene \
     --include-attribute gene_biotype:IG_V_gene \
     --include-attribute gene_biotype:IG_V_pseudogene \
     --include-attribute gene_biotype:IG_D_gene \
     --include-attribute gene_biotype:IG_J_gene \
     --include-attribute gene_biotype:IG_J_pseudogene \
     --include-attribute gene_biotype:IG_C_gene \
     --include-attribute gene_biotype:IG_C_pseudogene \
     --include-attribute gene_biotype:TR_V_gene \
     --include-attribute gene_biotype:TR_V_pseudogene \
     --include-attribute gene_biotype:TR_D_gene \
     --include-attribute gene_biotype:TR_J_gene \
     --include-attribute gene_biotype:TR_J_pseudogene \
     --include-attribute gene_biotype:TR_C_gene \
     Mus_musculus.GRCm39.dna.primary_assembly.fa.gz \
     Mus_musculus.GRCm39.108.gtf.gz
```

**4. Map the input sequencing reads to the index.**

This assay has multiple FASTQ files across multiple subpools as well as two primer types. To process this, we supply a batch.txt file containing the FASTQ files along with their designated subpool, a barcodes.txt file containing the three barcode components (since the assay contains three 8-bp barcodes, each separated by a linker, in the first read file), and a replace.txt file designating how to convert the random hexamer barcodes to the oligo-dT barcodes for the “combined” matrix. The final command to run with these files is as follows:

```
kb count --strand=forward -r replace.txt -w barcodes.txt \
     --workflow=nac -i index.idx -g t2g.txt -c1 cdna.txt \
     -c2 nascent.txt -x 1,10,18,1,48,56,1,78,86:1,0,10:0,0,0 \
     --sum=total -o output_dir --batch-barcodes batch.txt
```

**5. Analyze the output.**

Output (both the oligo-dT and random hexamer barcodes in the same count matrix):

- output_dir/counts_unfiltered/cells_x_genes.mature.mtx
- output_dir/counts_unfiltered/cells_x_genes.nascent.mtx
- output_dir/counts_unfiltered/cells_x_genes.ambiguous.mtx
- output_dir/counts_unfiltered/cells_x_genes.cell.mtx
- output_dir/counts_unfiltered/cells_x_genes.nucleus.mtx
- output_dir/counts_unfiltered/cells_x_genes.total.mtx
- output_dir/counts_unfiltered/cells_x_genes.barcodes.txt
- output_dir/counts_unfiltered/cells_x_genes.barcodes.prefix.txt
- output_dir/counts_unfiltered/cells_x_genes.genes.txt
- output_dir/counts_unfiltered/cells_x_genes.genes.names.txt

Output (the oligo-dT and random hexamer barcodes are combined):

- output_dir/counts_unfiltered_modified/cells_x_genes.mature.mtx
- output_dir/counts_unfiltered_modified/cells_x_genes.nascent.mtx
- output_dir/counts_unfiltered_modified/cells_x_genes.ambiguous.mtx
- output_dir/counts_unfiltered_modified/cells_x_genes.cell.mtx
- output_dir/counts_unfiltered_modified/cells_x_genes.nucleus.mtx
- output_dir/counts_unfiltered_modified/cells_x_genes.total.mtx
- output_dir/counts_unfiltered_modified/cells_x_genes.barcodes.txt
- output_dir/counts_unfiltered_modified/cells_x_genes.barcodes.prefix.txt
- output_dir/counts_unfiltered_modified/cells_x_genes.genes.txt
- output_dir/counts_unfiltered_modified/cells_x_genes.genes.names.txt

Note that the cells_x_genes.barcodes.prefix.txt will contain a unique identifier for each subpool.

**Information about batch.txt, barcodes.txt, and replace.txt files:**

**batch.txt:**

Example with three subpools, each sequenced on four lanes:

```
subpool_1    S1_lane1_R1.fastq.gz    S1_lane1_R2.fastq.gz
subpool_1    S1_lane2_R1.fastq.gz    S1_lane2_R2.fastq.gz
subpool_1    S1_lane3_R1.fastq.gz    S1_lane3_R2.fastq.gz
subpool_1    S1_lane4_R1.fastq.gz    S1_lane4_R2.fastq.gz
subpool_2    S2_lane1_R1.fastq.gz    S2_lane1_R2.fastq.gz
subpool_2    S2_lane2_R1.fastq.gz    S2_lane2_R2.fastq.gz
subpool_2    S2_lane3_R1.fastq.gz    S2_lane3_R2.fastq.gz
subpool_2    S2_lane4_R1.fastq.gz    S2_lane4_R2.fastq.gz
subpool_3    S3_lane1_R1.fastq.gz    S3_lane1_R2.fastq.gz
subpool_3    S3_lane2_R1.fastq.gz    S3_lane2_R2.fastq.gz
subpool_3    S3_lane3_R1.fastq.gz    S3_lane3_R2.fastq.gz
subpool_3    S3_lane4_R1.fastq.gz    S3_lane4_R2.fastq.gz
```

In this configuration, subpool_1 will have the sample-specific barcode AAAAAAAAAAAAAAAA, subpool_2 will have the sample-specific barcode AAAAAAAAAAAAAAAC, and subpool_3 will have the sample-specific barcode AAAAAAAAAAAAAAAG. This mapping can be found in the output_dir/matrix.cells and output_dir/matrix.sample.barcodes files. These sample-specific barcodes are found in cells_x_genes.barcodes.prefix.txt to identify the subpool a specific cell barcode originated from when inspecting the count matrices.

**barcodes.txt:**

The cell barcodes contain three 8-bp components so we should correct each component individually to its own “on list”. This can be done by having multiple columns in the barcodes.txt file. Note that the first two columns have 96 barcodes and the third column has 192 barcodes.

```
AACGTGAT AACGTGAT CATTCCTA
AAACATCG AAACATCG CTTCATCA
ATGCCTAA ATGCCTAA CCTATATC
AGTGGTCA AGTGGTCA ACATTTAC
ACCACTGT ACCACTGT ACTTAGCT
ACATTGGC ACATTGGC CCAATTCT
CAGATCTG CAGATCTG GCCTATCT
CATCAAGT CATCAAGT ATGCTGCT
CGCTGATC CGCTGATC CATTTACA
ACAAGCTA ACAAGCTA ACTCGTAA
CTGTAGCC CTGTAGCC CCTTTGCA
AGTACAAG AGTACAAG ACTCCTGC
AACAACCA AACAACCA ATTTGGCA
AACCGAGA AACCGAGA TTATTCTG
AACGCTTA AACGCTTA TCATGCTC
```

AAGACGGA AAGACGGA CATACTTC
AAGGTACA AAGGTACA CCGTTCTA
ACACAGAA ACACAGAA GCTTCATA
ACAGCAGA ACAGCAGA CTCTGTGC
ACCTCCAA ACCTCCAA CCCTTATA
ACGCTCGA ACGCTCGA ACTGCTCT
ACGTATCA ACGTATCA CTCTAATC
ACTATGCA ACTATGCA ACCCTTGC
AGAGTCAA AGAGTCAA ATCTTAGG
AGATCGCA AGATCGCA CATGTCTC
AGCAGGAA AGCAGGAA TCATTGCA
AGTCACTA AGTCACTA ACACCTTT
ATCCTGTA ATCCTGTA AATTTCTC
ATTGAGGA ATTGAGGA ATTCATGG
CAACCACA CAACCACA ACTTTACC
GACTAGTA GACTAGTA CTTCTAAC
CAATGGAA CAATGGAA CTATTTCA
CACTTCGA CACTTCGA TCTCATGC
CAGCGTTA CAGCGTTA ATCCTTAC
CATACCAA CATACCAA TAAATATC
CCAGTTCA CCAGTTCA TTACCTGC
CCGAAGTA CCGAAGTA CACTTTCA
CCGTGAGA CCGTGAGA CACCTTTA
CCTCCTGA CCTCCTGA CTGACTTC
CGAACTTA CGAACTTA CATTTGGA
CGACTGGA CGACTGGA GCTCTACT
CGCATACA CGCATACA GTTACGTA
CTCAATGA CTCAATGA CCTGTTGC
CTGAGCCA CTGAGCCA CTATCATC
CTGGCATA CTGGCATA GCTATCAT
GAATCTGA GAATCTGA ACATTCAT
CAAGACTA CAAGACTA TTCGCTAC
GAGCTGAA GAGCTGAA CATTCTAC
GATAGACA GATAGACA CACTTATC
GCCACATA GCCACATA ATAAGCTC
GCGAGTAA GCGAGTAA TCATCCTG
GCTAACGA GCTAACGA CCTGGTAT
GCTCGGTA GCTCGGTA TGGTATAC
GGAGAACA GGAGAACA TTGGGAGA
GGTGCGAA GGTGCGAA ACTTCATC
GTACGCAA GTACGCAA TCTCTAGC
GTCGTAGA GTCGTAGA ATGCCCTT
GTCTGTCA GTCTGTCA CCCAATTT
GTGTTCTA GTGTTCTA ACTATATA
TAGGATGA TAGGATGA CTCTATAC
TATCAGCA TATCAGCA CTGTCTCA
TCCGTCTA TCCGTCTA GACCTTTC
TCTTCACA TCTTCACA GATTTGGC
TGAAGAGA TGAAGAGA CGTCTAGG
TGGAACAA TGGAACAA TACTCGAA
TGGCTTCA TGGCTTCA CAGCCTTT
TGGTGGTA TGGTGGTA CCTCATTA
TTCACGCA TTCACGCA CTTATACC
AACTCACC AACTCACC TCTATTAC
AAGAGATC AAGAGATC CCTGCATT
AAGGACAC AAGGACAC CAATCCTT
AATCCGTC AATCCGTC TTGTCTTA
AATGTTGC AATGTTGC TCACTTTA
ACACGACC ACACGACC TGCTTGGG

| | | |
|---|---|---|
| ACAGATTC | ACAGATTC | CGCTCATT |
| AGATGTAC | AGATGTAC | GCCTCTAT |
| AGCACCTC | AGCACCTC | GAGCACAA |
| AGCCATGC | AGCCATGC | CTCTTAAC |
| AGGCTAAC | AGGCTAAC | TCTAGGCT |
| ATAGCGAC | ATAGCGAC | AATTCTGC |
| ATCATTCC | ATCATTCC | CATTCTCA |
| ATTGGCTC | ATTGGCTC | ACTTGCCT |
| CAAGGAGC | CAAGGAGC | ATCATTGC |
| CACCTTAC | CACCTTAC | GTTCAACA |
| CCATCCTC | CCATCCTC | CCATTTGC |
| CCGACAAC | CCGACAAC | GACTTTGC |
| CCTAATCC | CCTAATCC | ATTGGCTC |
| CCTCTATC | CCTCTATC | GTGCTAGC |
| CGACACAC | CGACACAC | CTTTCAAC |
| CGGATTGC | CGGATTGC | ACTATTGC |
| CTAAGGTC | CTAAGGTC | ACTGGCTT |
| GAACAGGC | GAACAGGC | ATTAGGCT |
| GACAGTGC | GACAGTGC | GCCTTTCA |
| GAGTTAGC | GAGTTAGC | ATTCTAGG |
| GATGAATC | GATGAATC | CCTTACAT |
| GCCAAGAC | GCCAAGAC | ACATTTGG |
| - | - | CATCATCC |
| - | - | CTGCTTTG |
| - | - | CTAAGGGA |
| - | - | GCTTATAG |
| - | - | TCTGATCC |
| - | - | TCTCTTGG |
| - | - | CAATTTCC |
| - | - | AGTCTCTT |
| - | - | TGCTGCTC |
| - | - | GTATTTCC |
| - | - | TTCCTGTG |
| - | - | GCTGCTTC |
| - | - | TATGTGTC |
| - | - | CAATTCTC |
| - | - | TGGTCTCC |
| - | - | GCTCTTTA |
| - | - | GCTGCATG |
| - | - | ACTCATTT |
| - | - | AGTCTTGG |
| - | - | GGTTCTTC |
| - | - | TCATGTTG |
| - | - | ATTTTGCC |
| - | - | CTTCTGTA |
| - | - | GTCCATCT |
| - | - | GCTATCTC |
| - | - | TAGTTTCC |
| - | - | TCCATTAT |
| - | - | AGGATTAA |
| - | - | AATCTTTC |
| - | - | GTCATATG |
| - | - | GTGCTTCC |
| - | - | ATGTGTTG |
| - | - | CCATCTTG |
| - | - | TACTGTCT |
| - | - | TTCATCGC |
| - | - | ACTGTGGG |
| - | - | TCTGTGCC |

| | | |
|---|---|---|
| - | - | TCAATCTC |
| - | - | GTCCTCTG |
| - | - | TTACATTC |
| - | - | ATTCTGTC |
| - | - | TGTGTATG |
| - | - | TCCATTTG |
| - | - | TTAGCTTC |
| - | - | GTGCTTGA |
| - | - | GTTTGTGA |
| - | - | GAAATTAG |
| - | - | GCAAATTC |
| - | - | GAGGTTGA |
| - | - | CCTGTCTG |
| - | - | GTGGGTTC |
| - | - | TTTGCATC |
| - | - | AGGTAATA |
| - | - | GTGCCTTC |
| - | - | ATGTTTCC |
| - | - | CTTAATTC |
| - | - | TCTGGCTC |
| - | - | CATCATTT |
| - | - | GTTGTCTC |
| - | - | ATCTTCTG |
| - | - | TGTTTGCC |
| - | - | TTCTGTCA |
| - | - | ACGGACTC |
| - | - | TTTGGTCA |
| - | - | TATCCGGG |
| - | - | TGTCATTC |
| - | - | ATTCTCTG |
| - | - | TGGCTTCC |
| - | - | TTGTTGCC |
| - | - | GTCATCTC |
| - | - | TTGCTCAT |
| - | - | CTGTCTGC |
| - | - | TATATTCC |
| - | - | ATATTGGC |
| - | - | GTGTCCTC |
| - | - | ATCTTCAT |
| - | - | CGTGGTTG |
| - | - | TTGCATCC |
| - | - | TCTTAATC |
| - | - | TGCATTTC |
| - | - | GATGTTTC |
| - | - | ATCTTGTC |
| - | - | TCATATTC |
| - | - | TGGCCTCT |
| - | - | CGTTGTCT |
| - | - | TCTTGTCA |
| - | - | TATTCCTG |
| - | - | TCCATGTC |
| - | - | TTGTCATC |
| - | - | ATTTCCTG |
| - | - | GTGTCTCC |
| - | - | GTGTGTGT |
| - | - | TATGCTTC |
| - | - | ATGGTGTT |
| - | - | GAATAATG |
| - | - | CCTCTGTG |

**replace.txt:**

This file contains the instructions on how to produce the “modified” count matrix in output_dir/counts_unfiltered_modified/ – the output directory which contains the combined oligo-dT and random hexamer barcodes wherein the random hexamer barcodes (first column of the file) are converted to their oligo-dT counterparts (second column of the file). These barcodes, being the third component of the barcode, occur at the end of the final barcode string. The asterisk (*) at the beginning of the replacement string tells bustools to convert the nucleotides at the end of the barcode sequence. As an example, the barcode sequence AACAACCATGAAGAGACATCATCC will be converted into AACAACCATGAAGAGACATTCCTA in the final output in the output_dir/counts_unfiltered_modified/ directory.

```
CATCATCC *CATTCCTA
CTGCTTTG *CTTCATCA
CTAAGGGA *CCTATATC
GCTTATAG *ACATTTAC
TCTGATCC *ACTTAGCT
TCTCTTGG *CCAATTCT
CAATTTCC *GCCTATCT
AGTCTCTT *ATGCTGCT
TGCTGCTC *CATTTACA
GTATTTCC *ACTCGTAA
TTCCTGTG *CCTTTGCA
GCTGCTTC *ACTCCTGC
TATGTGTC *ATTTGGCA
CAATTCTC *TTATTCTG
TGGTCTCC *TCATGCTC
GCTCTTTA *CATACTTC
GCTGCATG *CCGTTCTA
ACTCATTT *GCTTCATA
AGTCTTGG *CTCTGTGC
GGTTCTTC *CCCTTATA
TCATGTTG *ACTGCTCT
ATTTTGCC *CTCTAATC
CTTCTGTA *ACCCTTGC
GTCCATCT *ATCTTAGG
GCTATCTC *CATGTCTC
TAGTTTCC *TCATTGCA
TCCATTAT *ACACCTTT
AGGATTAA *AATTTCTC
AATCTTTC *ATTCATGG
GTCATATG *ACTTTACC
GTGCTTCC *CTTCTAAC
ATGTGTTG *CTATTTCA
CCATCTTG *TCTCATGC
TACTGTCT *ATCCTTAC
TTCATCGC *TAAATATC
ACTGTGGG *TTACCTGC
TCTGTGCC *CACTTTCA
TCAATCTC *CACCTTTA
GTCCTCTG *CTGACTTC
TTACATTC *CATTTGGA
ATTCTGTC *GCTCTACT
TGTGTATG *GTTACGTA
TCCATTTG *CCTGTTGC
```

TTAGCTTC *CTATCATC
GTGCTTGA *GCTATCAT
GTTTGTGA *ACATTCAT
GAAATTAG *TTCGCTAC
GCAAATTC *CATTCTAC
GAGGTTGA *CACTTATC
CCTGTCTG *ATAAGCTC
GTGGGTTC *TCATCCTG
TTTGCATC *CCTGGTAT
AGGTAATA *TGGTATAC
GTGCCTTC *TTGGGAGA
ATGTTTCC *ACTTCATC
CTTAATTC *TCTCTAGC
TCTGGCTC *ATGCCCTT
CATCATTT *CCCAATTT
GTTGTCTC *ACTATATA
ATCTTCTG *CTCTATAC
TGTTTGCC *CTGTCTCA
TTCTGTCA *GACCTTTC
ACGGACTC *GATTTGGC
TTTGGTCA *CGTCTAGG
TATCCGGG *TACTCGAA
TGTCATTC *CAGCCTTT
ATTCTCTG *CCTCATTA
TGGCTTCC *CTTATACC
TTGTTGCC *TCTATTAC
GTCATCTC *CCTGCATT
TTGCTCAT *CAATCCTT
CTGTCTGC *TTGTCTTA
TATATTCC *TCACTTTA
ATATTGGC *TGCTTGGG
GTGTCCTC *CGCTCATT
ATCTTCAT *GCCTCTAT
CGTGGTTG *GAGCACAA
TTGCATCC *CTCTTAAC
TCTTAATC *TCTAGGCT
TGCATTTC *AATTCTGC
GATGTTTC *CATTCTCA
ATCTTGTC *ACTTGCCT
TCATATTC *ATCATTGC
TGGCCTCT *GTTCAACA
CGTTGTCT *CCATTTGC
TCTTGTCA *GACTTTGC
TATTCCTG *ATTGGCTC
TCCATGTC *GTGCTAGC
TTGTCATC *CTTTCAAC
ATTTCCTG *ACTATTGC
GTGTCTCC *ACTGGCTT
GTGTGTGT *ATTAGGCT
TATGCTTC *GCCTTTCA
ATGGTGTT *ATTCTAGG
GAATAATG *CCTTACAT
CCTCTGTG *ACATTTGG

**The commands run by kb count in this example:**

```
mkdir -p output_dir/tmp

mkdir -p output_dir

kallisto bus -i index.idx -o output_dir -x
1,10,18,1,48,56,1,78,86:1,0,10:0,0,0 -t 8 --fr-stranded
--batch-barcodes --batch batch.txt

bustools sort -o output_dir/tmp/output.s.bus -T output_dir/tmp -t 8
-m 4G output_dir/output.bus

bustools inspect -o output_dir/inspect.json -w barcodes.txt
output_dir/tmp/output.s.bus

bustools correct -o output_dir/tmp/output.s.c.bus -w barcodes.txt
output_dir/tmp/output.s.bus

bustools sort -o output_dir/output.unfiltered.bus -T output_dir/tmp
-t 8 -m 4G output_dir/tmp/output.s.c.bus

mkdir -p output_dir/counts_unfiltered

bustools count -o output_dir/counts_unfiltered/cells_x_genes -g
t2g.txt -e output_dir/matrix.ec -t output_dir/transcripts.txt -s
nascent.txt --genecounts --umi-gene output_dir/output.unfiltered.bus

mv output_dir/counts_unfiltered/cells_x_genes.mtx
output_dir/counts_unfiltered/cells_x_genes.mature.mtx

mv output_dir/counts_unfiltered/cells_x_genes.2.mtx
output_dir/counts_unfiltered/cells_x_genes.nascent.mtx

bustools correct -o output_dir/tmp/output.unfiltered.c.bus -w
replace.txt output_dir/output.unfiltered.bus --replace
bustools sort -o output_dir/output_modified.unfiltered.bus -T
output_dir/tmp -t 8 -m 4G output_dir/tmp/output.unfiltered.c.bus

mkdir -p output_dir/counts_unfiltered_modified

bustools count -o output_dir/counts_unfiltered_modified/cells_x_genes
-g t2g.txt -e output_dir/matrix.ec -t output_dir/transcripts.txt -s
nascent.txt --genecounts --umi-gene
output_dir/output_modified.unfiltered.bus
```

```
mv output_dir/counts_unfiltered_modified/cells_x_genes.mtx
output_dir/counts_unfiltered_modified/cells_x_genes.mature.mtx

mv output_dir/counts_unfiltered_modified/cells_x_genes.2.mtx
output_dir/counts_unfiltered_modified/cells_x_genes.nascent.mtx

rm -rf output_dir/tmp
```

*Chapter 4*

# PSEUDOASSEMBLY OF K-MERS

## Beyond annotated reference mapping

In the preceding chapters, we focused on methods that map sequencing reads to annotated reference sequences. In these methods, each k-mer, indexed within a colored de Bruijn graph, originates from one or more annotated target sequences. This framework enables rapid and lightweight mapping (Almodaresi et al., 2021, 2018; Bray et al., 2016; Patro et al., 2017, 2014) by leveraging known transcript or genome annotations. However, in Chapter 3, we briefly introduced an extension of this framework: the use of unannotated k-mers, called distinguishing flanking k-mers (Sullivan et al., 2025), which are extracted directly from raw sequences (e.g. the genome assembly) without regards to annotation. These k-mers proved valuable in improving the accuracy of pseudoalignment.

In this chapter, we explore a more generalized and systematic extension of that idea. Rather than relying on known annotations, we now build target sequences de novo, derived from k-mers discovered directly from raw sequences (Sullivan et al., 2025). We refer to this approach as pseudoassembly, in the spirit of the term pseudoalignment. Unlike full assembly, which reconstructs entire transcripts or genomes, pseudoassembly focuses on indexing only those k-mers that are relevant for downstream read assignment. This targeted construction avoids the complexities of full assembly while retaining the ability to detect biologically meaningful sequence variation.

Pseudoassembly is particularly valuable for studying genetic variation. Because k-mers can uniquely represent variant sequences, they are well-suited for identifying sample-specific or condition-specific differences in sequence content. One common strategy for analyzing known variants involves tiling k-mer windows around known polymorphisms, such as SNPs, structural variants, or splice junctions, and indexing them for downstream analysis. This approach depends on prior knowledge of variant locations. In contrast,

pseudoassembly proceeds de novo from raw sequences in FASTQ or FASTA format. In this mode, we extract k-mers that are uniquely found in an "experimental" sample (i.e. sequences absent from control datasets or the reference genome). This is particularly relevant in settings such as cancer, where mutations are highly variable in both structure and genomic location. The flexibility and reference-agnostic nature of k-mers make them ideal for capturing such unstructured variation.

Several prior tools have leveraged this idea. For example, DE-kupl focuses on differential k-mer abundance between conditions and has demonstrated high sensitivity in detecting novel biological variation (Audoux et al., 2017). Differential k-mers have also been applied in association mapping (Rahman et al., 2018). Another more recent method, SPLASH, uses "anchor" and "target" k-mers, where a sequence anchor may be associated with multiple different targets (each representing a variant), to quantify variation (Chaung et al., 2023).

Building on these ideas, we developed klue (k-mer based local uniqueness exploration), a general-purpose tool for discovering, organizing, and extracting informative k-mers. klue accepts both FASTQ and FASTA files as input and can be applied to RNA-seq data, DNA-seq data, or sequence assemblies. k-mers are extracted in the form of longer contiguous sequences (contigs). More details of klue will be provided in the subsequent section.

Following contig extraction (where the contigs represent variant-specific k-mers), we proceed to pseudoassembly. This involves mapping the contigs of interest—those extracted by klue—to known reference sequences. For example, if the input k-mer size was 31, we might use smaller k-mer matches (e.g. 29-mers) to find a partial match between a contig and annotated transcript sequences. When a contig is mapped to a target sequence, it inherits the color of that target in the de Bruijn graph (which is built by kallisto over the transcripts). However, to distinguish it as a variant-derived sequence, the color is modified into a uniquely shaded version. Each distinct contig mapped to a target gets its own unique shade. If a contig maps to multiple target sequences, it is assigned a shade for each.

This process gives rise to an ornamental de Bruijn graph (Figure 4.1), a concept introduced in earlier work and implemented in the tool Ornaments (Adduri and Kim, 2024). In this setup, pseudoalignment proceeds as usual via set intersection of k-mer colors. However, a set union is performed for the shades that are encountered in a read. The resulting equivalence class will contain both the parent colors and the shades.

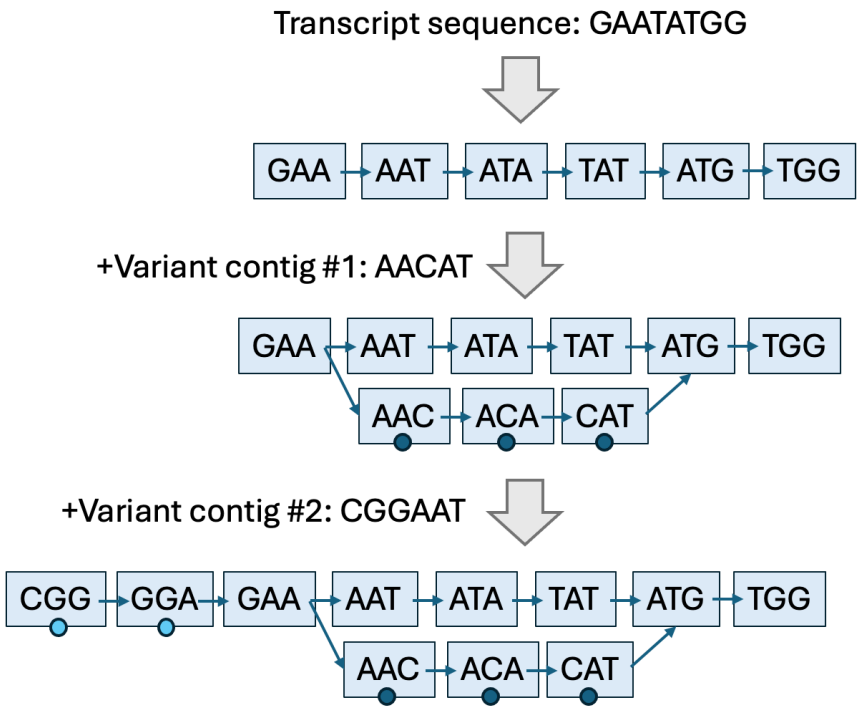



***Figure 4.1****: Building a de Bruijn graph with shades.*
*One transcript sequence (parent color) and two variant contigs (shades, shown as circles) are represented. In the end, 11 k-mers (k=3) are present, all with the same parent color, two with one shade of that color, and three with another shade of that color.*

Next, we provide a formal definition of a shade.

Let:

- $\mathcal{T} = \{T_1, \ldots, T_n\}$ be the set of annotated transcripts
- $[n] = \{1, 2, \ldots, n\}$ be the set of *canonical* colors (like those used in standard pseudoalignment).
- $\mathcal{S}$ be a disjoint set of *shade* colors.
- $\pi : \mathcal{S} \longrightarrow [n]$ be a parent function, assigning each shade $s \in \mathcal{S}$ exactly one canonical parent color $\pi(s)$.
- $\mathcal{A} \coloneqq [n] \cup \mathcal{S}$ be the universe of all colors.

A colored de Bruijn graph of order $k$ is a directed graph $G = (V, E, \mathcal{C})$ with:

- Each vertex $v \in V$ representing a unique $k$-mer over the alpha $\Sigma = \{A, T, C, G\}$.
- A directed edge $(v, w) \in E$ whenever the $(k-1)$-suffix of $v$'s $k$-mer equals the $(k-1)$-prefix of $w$'s $k$-mer.
- $\mathcal{C}(v) \subseteq \mathcal{A}$ as the color set of each vertex $v$.

We call $s \in S$ a shade based on the following:

- Parent relationship: Each shade $s$ is associated with exactly one canonical transcript color via a parent mapping $\pi(s) \in [n]$; that is, $s$ has exactly one parent color.
- Parent inclusion: A shade never appears without its parent color on any $k$-mer:

$$\forall v \in V, s \in \mathcal{C}(v) \implies \pi(s) \in \mathcal{C}(v).$$

When adding a shade to a transcript de Bruijn graph, the shade is added if and only if the $k$-mer does not exist in the original graph with its parent color. Note: Each shade implicitly adds its parent color to the $k$-mer color set. See the following algorithm.

---

**Algorithm** Add a new sequence to the colored de Bruijn graph as a shade

---

**Input:** Set of $k$-mers from new sequence $u$, shade color $s$,
colored de Bruijn graph (cdBG) $G = (V, E, C)$,
original cdBG before any shades were added $G_{orig} = (V_{orig}, E_{orig}, C_{orig})$
*(Note: G is a supergraph of $G_{orig}$)*

1: **function** AddShadeToGraph($u, s, G, G_{orig}$)
2: **for each** $k$-mer $x \in u$ **do**
3: **if** $x \in G_{orig}$ **then**
4: **if** $\pi(s) \notin C_{orig}(x)$ **then**
5: $C(x) \leftarrow C(x) \cup \{\pi(s), s\}$
6: **else**
7: $V \leftarrow V \cup \{x\}$
8: $C(x) \leftarrow C(x) \cup \{\pi(s), s\}$

---

To create equivalence classes, first set-intersection is done among the parent colors encountered in the read then set-union is performed among the shade colors corresponding to those parent colors. See the following algorithm.

---

**Algorithm** Assign equivalence class to read

---

**Input:** Set of $k$-mers in read $v$, set of canonical colors $[n]$, set of shades $S$,
colored de Bruijn graph $G = (V, E, C)$

1: **function** AssignEquivalenceClass($v, [n], S, G$)
2: Compute the canonical intersection (i.e. standard pseudoalignment):

$$C_{\text{canon}} := \bigcap_{i=1}^{|v|} \Big( \mathcal{C}(v_i) \cap [n] \Big)$$

3: Collect valid shades:

$$C_{\text{shade}} := \left\{ s \in \bigcup_{i=1}^{|v|} \mathcal{C}(v_i) \cap \mathcal{S} \;\middle|\; \pi(s) \in C_{\text{canon}} \right\}$$

4: Return equivalence class:

return $C_{\text{canon}} \cup C_{\text{shade}}$

---

In the remainder of this chapter, we demonstrate two applications of this framework. First, we describe klue and show an example use of klue in mouse strain demultiplexing by extracting strain-specific sequences. Second, we apply pseudoassembly to RNA-seq data from melanoma samples, using variant-specific contigs to profile tumor-specific mutations. These case studies illustrate the power of k-mer-centric approaches in analyzing complex, unstructured, and unannotated biological variation.

**A general-purpose k-mer toolkit**

Klue is a general-purpose k-mer toolkit that internally uses a colored compacted de Bruijn graph (ccdBG), implemented via the C++ Bifrost library (Holley and Melsted, 2020). Each input file is assigned a unique color, encoding sample identity in the graph. A given k-mer may appear in multiple samples and thus exhibit a multi-color profile. The ccdBG compacts adjacent k-mers into longer contiguous sequences called unitigs, conserving memory and enabling analysis over longer sequence contexts.

A core feature of klue is its ability to extract contigs—unitigs, or contiguous substrings of unitigs, that share the same color profile. For instance, given three input samples colored red, blue, and green, klue can extract red-only contigs, red+blue shared contigs, contigs found in any two colors, and so on. A comprehensive suite of set operations over input samples is supported by klue. Figure 4.2 shows an example of extracting contigs based on color profile, wherein monochromatic contigs (those unique to one sample) are extracted. These contigs can represent tumor-specific mutations, species-specific sequences, or treatment-induced transcripts. The resulting contigs are written to FASTA files, where they can be subjected to further downstream analyses such as mapping, annotation, or pseudoassembly.

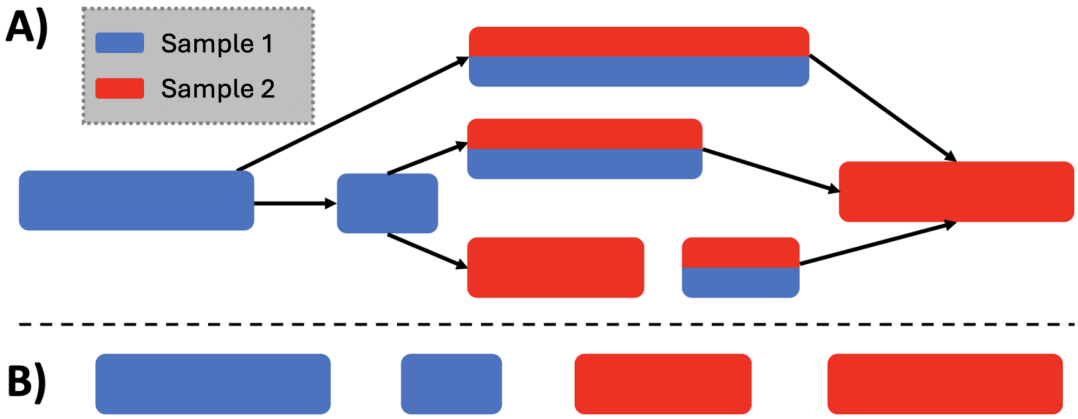



***Figure 4.2**: Partitioning the colored compacted de Bruijn graph.*
*(A) The structure of the graph. The boxes shown are the nodes of the graph and represent colored contigs. (B) The monochromatic contigs that are extracted.*

To illustrate klue in practice, we applied klue to various different mouse genome assemblies (Ferraj et al., 2023), including the standard C57BL/6J mouse reference genome assembly (GRCm39). As expected, the genomes of the five inbred laboratory mouse strains (C57BL/6J, A/J, 129S1/SvImJ, NOD/ShiLtJ, NZO/HlLtJ) are more similar to each other than to the genomes of the three inbred wild-derived strains (WSB/EiJ, PWK/PhJ, CAST/EiJ) based on overlap of k-mer content from the klue-extracted k-mers (Figure 4.3), recapitulating expected phylogeny (Morgan and Welsh, 2015). Thus, klue can be used as a convenient tool for extracting shared and unique k-mers.

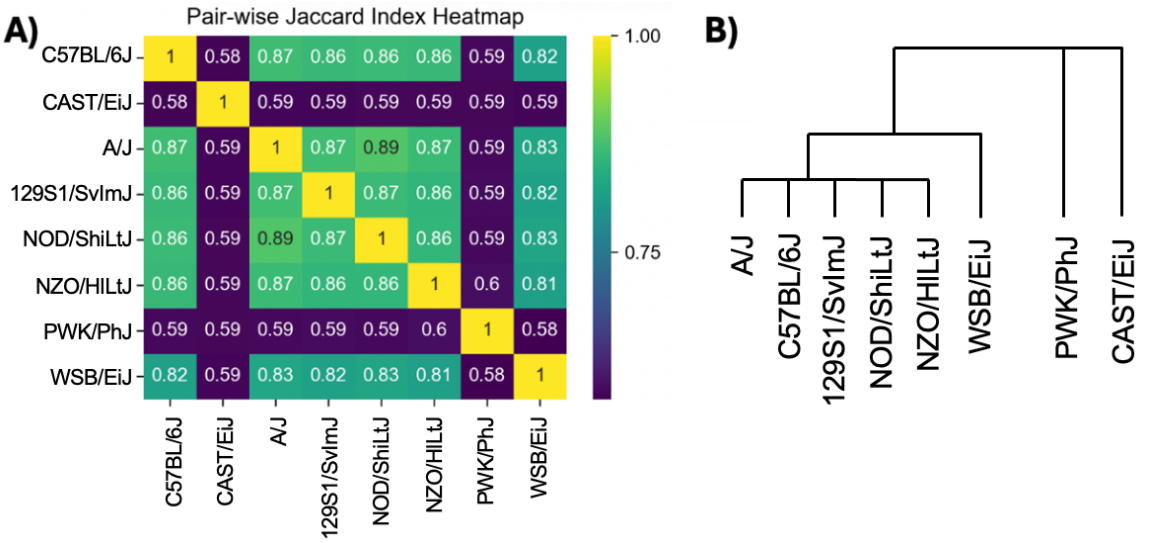



***Figure 4.3**: Relationship between different mouse strains.*
*(A) Jaccard similarity index of k-mers between different mouse strains as determined by klue: the cardinality of the set intersection, X, represents the number of k-mers shared between two mouse strains while the cardinality of the set union, Y, represents all the number of k-mers present in either or both of two mouse strains; the Jaccard index is the ratio of X to Y. (B) The ground truth phylogenetic relationship between the 8 mouse strains.*

Next, we evaluated klue's ability to infer cell identity using data from single-nucleus RNA-seq experiments conducted on multiple tissues from multiple mouse strains (Rebboah et al., 2025) as part of work of the IGVF Consortium (IGVF Consortium, 2024). Specifically, we examined kidney tissue from eight A/J mice and eight PWK/PhJ mice. Contigs unique to each mouse strain genome assembly were extracted, and only 61-bp contigs were

retained, as these correspond to SNPs when using a k-mer size of 31. We focused on k-mers overlapping SNPs rather than all strain-specific k-mers because these "SNP contigs" include both the variant and a conserved flanking region. These common anchors helped reduce noise from spurious k-mers that may arise due to artifacts from genome assembly or differences in sequencing quality between strains. Kallisto was then used to map reads to the strain-specific contigs, and cell-level strain assignments were determined based on the number of UMIs mapping to each strain (Figure 4.4). This klue+kallisto demultiplexing method could successfully resolve cells from PWK/PhJ mice versus A/J mice (Figure 4.5).

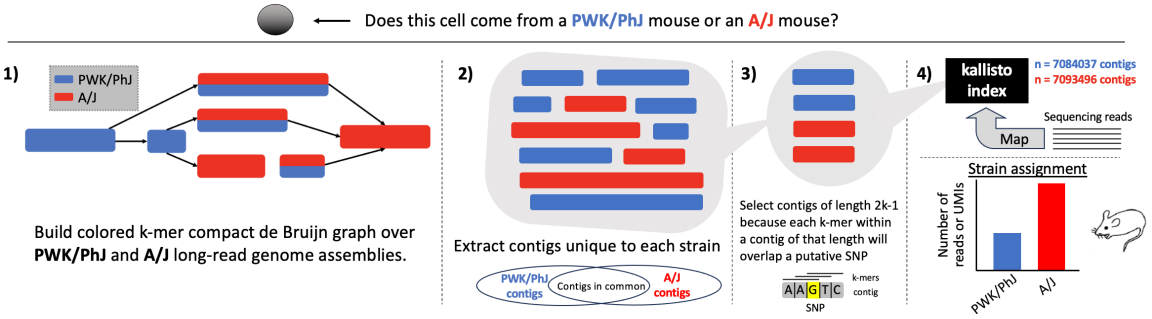



***Figure 4.4**: klue for mouse strain demultiplexing.*
*Overview of the workflow used for strain demultiplexing with klue and kallisto.*

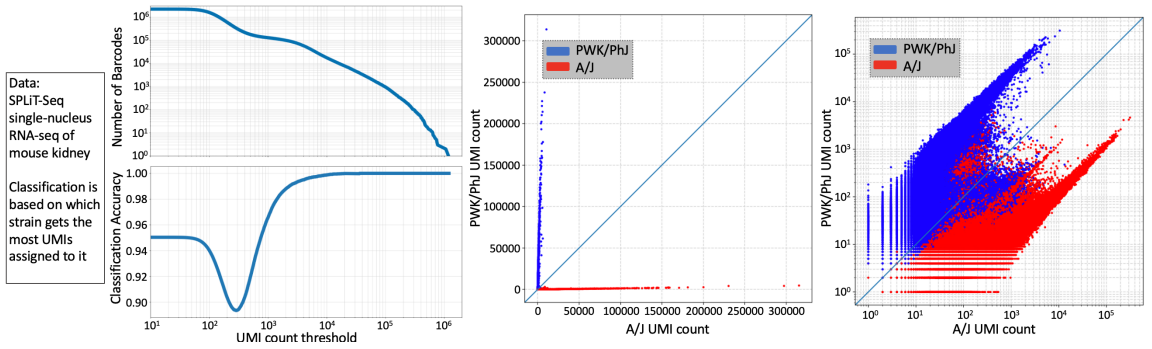



***Figure 4.5**: Results of klue+kallisto demultiplexing of the PWK/PhJ and A/J mouse cells. Accuracy could be determined because the PWK/PhJ cells and A/J mouse cells were placed into separate wells at the initial step of the split-pool barcoding, therefore the well barcode could serve as a ground truth label for each cell.*

**Application to cancer genomics**

*A melanoma mouse model:*

We performed pseudoassembly on single-cell data from a melanoma mouse model in order to identify cell type-specific mutations. The dataset we used (Sun et al., 2019) featured 10x Genomics (version 2 chemistry) single-cell RNA-seq of mouse melanocyte stem cells (McSCs) and melanomas arising from the McSCs. In that study, McSCs, derived from transgenic Tyr-CreER:Braf:Pten:Tomato mice, were transplanted onto immunodeficient nude mice and tumorigenesis was achieved by tamoxifen induction. Control McSCs were obtained from the telogen back skin of Tyr-CreER:R26R-Tomato mice. Single-cell RNA-seq was performed on both the tumors (melanomas) and the control McSCs, with three biological replicates per condition; replicates were pooled and not distinguished in downstream analyses. Single-end bulk RNA-seq of control McSCs was also performed.

klue was applied to four files:

1. A FASTA file containing both the human genome FASTA (from the T2T-CHM13v2.0 assembly) and a transcriptome FASTA derived from the GRCh38 assembly.

2. Bulk RNA-seq FASTQ file for control McSCs.

3. Single-cell RNA-seq FASTQ file for control McSCs.

4. Single-cell RNA-seq FASTA file for McSC-derived melanomas.

The reason for including all these datasets was to identify melanoma-unique contigs (containing 31-mers unique to file #4), that would not include barcodes, adapters, or other sequences common to 10x experiments (removed by inclusion of file #3), any existing sequences in the genome (removed by inclusion of file #1), or sequences from exon-exon junctions (removed by inclusion of file #2 and #3). The resultant contigs were mapped to

the human transcriptome using kallisto (with a k-mer size of 29) to assign "shades" (Figure 4.5). Finally, the melanoma single-cell RNA-seq reads were mapped (using kallisto version 0.51.1) to an index containing the human transcriptome targets along with their shades.

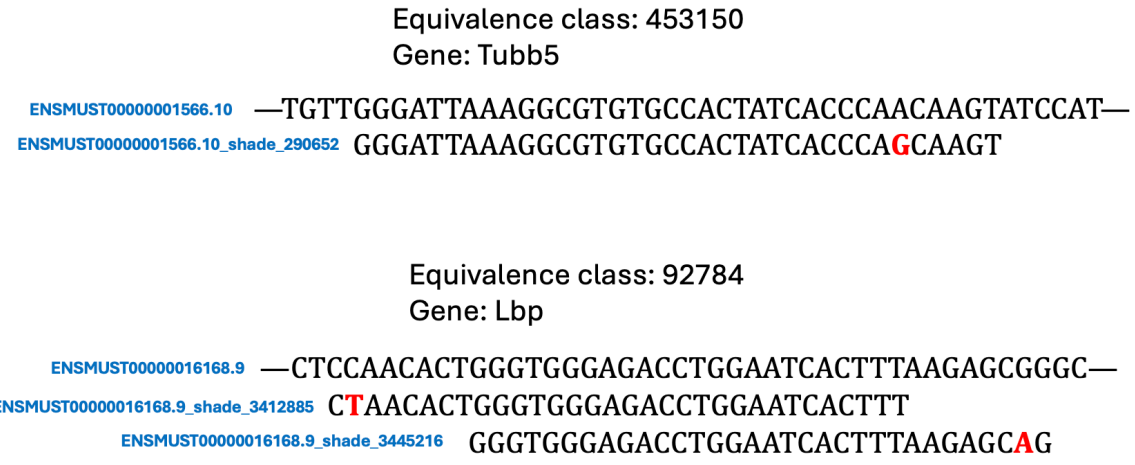



***Figure 4.5****: Examples of equivalence classes containing shades.*
*The equivalence classes corresponding to two different genes are shown, each containing a standard transcriptome target and one or more "shade" targets.*

To obtain cell clusters, the melanoma single-cell RNA-seq reads were also mapped to a standard kallisto index of the human transcriptome. After quantification of the data with kallisto | bustools (Melsted et al., 2021; Sullivan et al., 2024) to generate a cell-by-gene count matrix, cells with at least 5000 UMIs were retained. The counts were then normalized with CP10k normalization followed by log1p transformation. Highly variable genes were identified and then nearest neighbor graphs were constructed from the cell coordinates on the top 40 principal component analysis (PCA) embeddings in Scanpy (Wolf et al., 2018). The Leiden algorithm (Traag et al., 2019) was performed in Scanpy, resulting in 15 clusters. Clusters with fewer than 50 cells were excluded, resulting in a final count of 2776 cells distributed across 10 clusters. As the number of clusters obtained here was larger than that obtained in the original study which produced this dataset (Sun et al., 2019), we merged related clusters to create more coarse-grained groupings. The original study performed

pseudotemporal trajectory analysis (Trapnell et al., 2014) to define the branching transition between McSC cells to either a mesenchymal-like cell type or a neural crest/neuronal-like cell type. We therefore merged two clusters with high expression of mesenchymal markers (Dcn, Col1a1, Col1a2), merged five clusters with high expression of neural crest/neuronal genes (Nes, Foxd3, L1cam, Ngfr), and then merged the remaining three clusters which had high expression of the genes Fosb, Klf4, Serpine2, Cdkn1a from a published metastatic melanoma gene set (Perego et al., 2018). These correspond to the "mesenchymal-like" cluster, the "neural crest/neuronal-like" cluster, and the "intermediate" cluster, respectively, from the original publication (Figure 4.6). The original study also identified two additional clusters. One was a neural crest/neuronal-like cluster characterized by high expression of proliferation genes (Mki67, Cdk1), which we merged into the broader neural crest/neuronal-like cluster. The other was composed of control McSC cells; however, since we did not include control McSC data in our cell type clustering analysis, and the few McSC cells present in the melanoma dataset were likely removed when filtering out clusters with low cell counts, this cluster was not represented in our analysis.

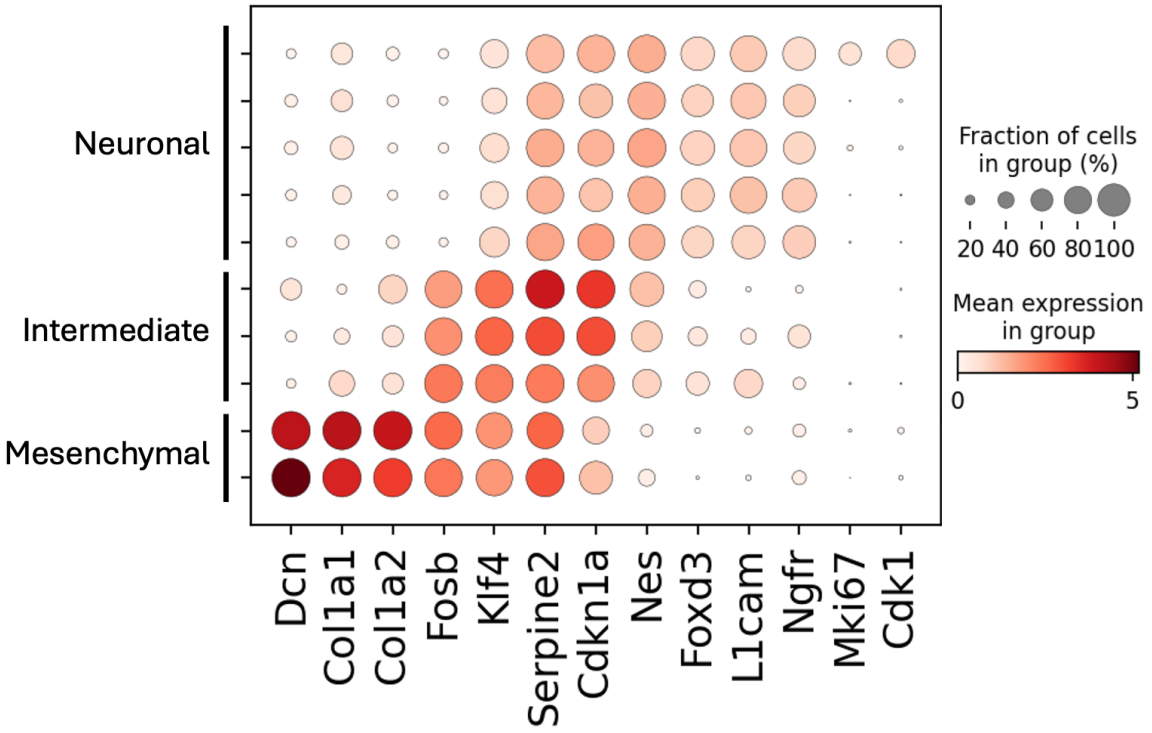



***Figure 4.6**: Cell type clustering of melanoma single-cell RNA-seq.*
*Based on expression of marker genes, three coarse-grained clusters of cells with a mesenchymal-like signature, cells with a neural crest/neuronal-like signature, and intermediate cells with a signature between neuronal and mesenchymal could be resolved.*

Next, we further analyzed those clusters, defined by gene-level counts, using the transcript compatibility counts (TCCs) from read mapping to the index containing the shades. We sought to identify mutations that are expressed uniquely in certain clusters (i.e. mutation cell-type specificity). We first filtered the cell-by-TCC count matrix to only contain equivalence classes (ECs) which contain a shade target mapping and which are present in at least 10 cells. All ECs corresponding to multiple genes, unannotated genes, or pseudogenes were excluded. For differential expression testing of shade counts, a 2x2 contingency table was built for each shade-containing EC as follows:

| | In cluster | Outside cluster |
|---|---|---|
| Shade-containing EC for a gene | $a$ | $b$ |
| All regular ECs for that gene | $c$ | $d$ |

Here, $a$, $b$, $c$, and $d$ represent cell numbers (i.e. the number of cells that contain at least one count for the EC). The odds ratio is then calculated as follows, with higher values indicating greater specificity of the shade (i.e. the putative mutation) for the cluster being evaluated.

$$OR = \frac{\frac{a}{c}}{\frac{b}{d}} = \frac{ad}{bc}$$

For each EC, a p-value was determined by Fisher's exact test. A volcano plot depicting the results of differential expression testing of shade (mutation) counts is shown in Figure 4.7.

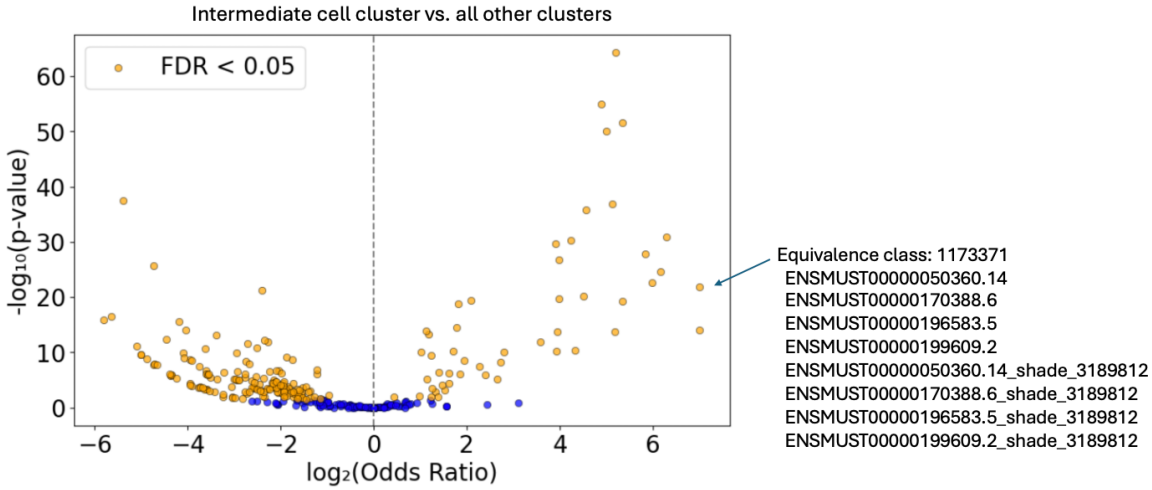



***Figure 4.7****: Differential expression testing of shade (mutation) counts.*
*Each point on the volcano plot represents a shade-containing equivalence class (EC). An example of such an EC (EC number 1173371), corresponding to the P2ry12 gene, is shown on the right. 50 ECs had an odds ratio greater than 1 and a Benjamini-Hochberg FDR-adjusted p-value less than 0.05, while 278 ECs had an odds ratio less than 1 with an adjusted p-value below the same threshold.*

The method was able to identify many examples in which a shade for a given gene is abundant in one melanoma cluster but not the others (Figure 4.8). Specifically, the P2ry12 gene appeared to contain a differential variant in the intermediate cluster, and the Tubb5 gene was revealed to contain a differential variant in the mesenchymal cluster. These observations would not have been possible within the standard single-cell RNA-seq analysis framework (Chen et al., 2016).

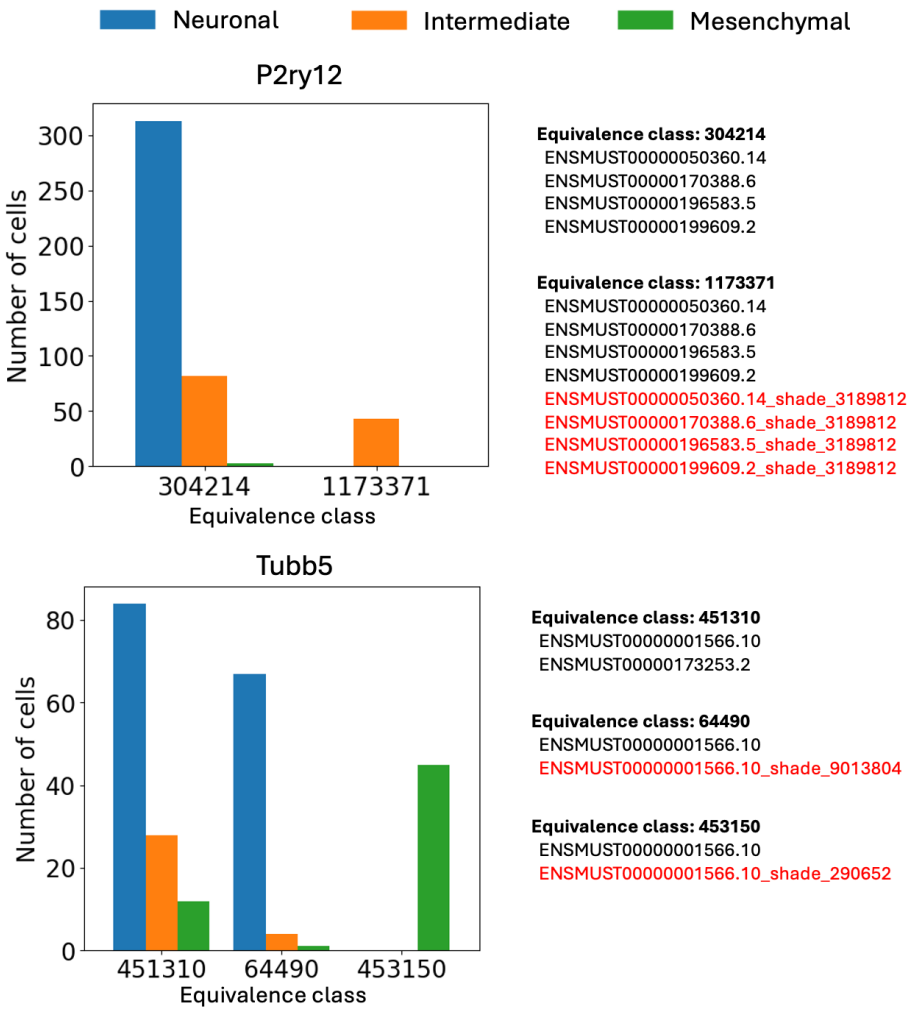



***Figure 4.8****: Examples of genes where the associated equivalence classes display different patterns of cluster-specific expression.*
*Shaded targets are marked in red. A P2ry12 mutation is found specifically in the intermediate cluster; a Tubb5 mutation is found specifically in the mesenchymal cluster. The y-axis represents the number of cells for which at least one UMI is identified as being associated with the given equivalence class.*

This melanoma dataset was well-suited for identifying mutation cell type specificity due to two key factors: the inclusion of a control sample and the nature of 10x Genomics chemistry, which sequences only the ends of transcripts. The control sample is essential to distinguish true mutations from common genetic variants such as SNPs, which would otherwise be picked up as "shades". While sequencing only the 3′ ends of transcripts limits coverage and may miss mutations located in the middle of genes, it helps reduce false signals. For instance, incomplete coverage might result in certain transcript regions being captured only in the disease group but not in the control group—leading to spurious disease-unique contigs. Taken together, the melanoma datasets (Sun et al., 2019) provide a compelling argument for pseudoassembly to discover cell-type specific mutations. The generality and flexibility of using klue with kallisto makes it suitable for any (single-cell) genomics datasets, and it should prove to be a useful complement to standard assembly algorithms.

*Code*:

Code for the analysis is available at https://github.com/pachterlab/SBP_2025.

The klue program is available at https://github.com/pachterlab/klue.

*Chapter 5*

# EFFICIENT AND SCALABLE SINGLE-CELL TRANSCRIPTOMICS

## Scaling single-cell transcriptomics

Single-cell RNA-seq has several limitations: it is expensive, it requires complex library preparation (compared to bulk RNA-seq), and it often relies on proprietary protocols. A key tradeoff exists between the number of cells captured and sequencing depth. Droplet-based technologies like 10x Genomics allow high-throughput cell capture but at low per-cell depth, whereas plate-based methods like Smart-seq achieve high sequencing depth per cell but are limited to hundreds or, at most, thousands of cells (Ding et al., 2020). Scaling single-cell RNA-seq to be cost-effective, time-efficient, and capable of capturing both a large number of cells and high per-cell depth is therefore a challenge.

SPLiT-seq, when introduced, boasted the features of "low-cost", "hundreds of thousands of fixed cells or nuclei in a single experiment", and "consists just of pipetting steps and no complex instruments are needed" (Rosenberg et al., 2018). This method, based on *in situ* barcoding via split-and-pool ligation chemistry, has since been commercialized by Parse Biosciences as part of its proprietary Evercode technology. By virtue of its split-pool barcoding strategy, SPLiT-seq made high-throughput, high-depth single-cell transcriptomics feasible.

Here, we introduce a split-and-pool barcoding-based technology, which we named SWIFT-seq (Single-cell With Iterative Fast Transcriptome-sequencing), that features the following: 1) High depth per cell, 2) High cell throughput, 3) Cost-effectiveness, 4) Open source accessibility, and 5) A simple, fast-executing protocol. These characteristics enable scalability not only in terms of cell numbers and sequencing depth but also in cost and ease of use, allowing for rapid, affordable experiments (Figure 5.1) . Like other split-pool barcoding technologies, SWIFT-seq supports highly multiplexed experiments, includes

UMIs to mitigate PCR amplification bias, provides strand specificity, and offers full-length gene body coverage, enabling isoform and allele detection.

A full cost breakdown of SWIFT-seq, when preparing one million cells for sequencing on an AVITI System Sequencing Instrument (Element Biosciences) (Arslan et al., 2024) as was done in all experiments described in this dissertation, is displayed in Figure 5.2. The total cost per cell ends up being approximately $0.00028 ($0.0012 if sequencing is included).

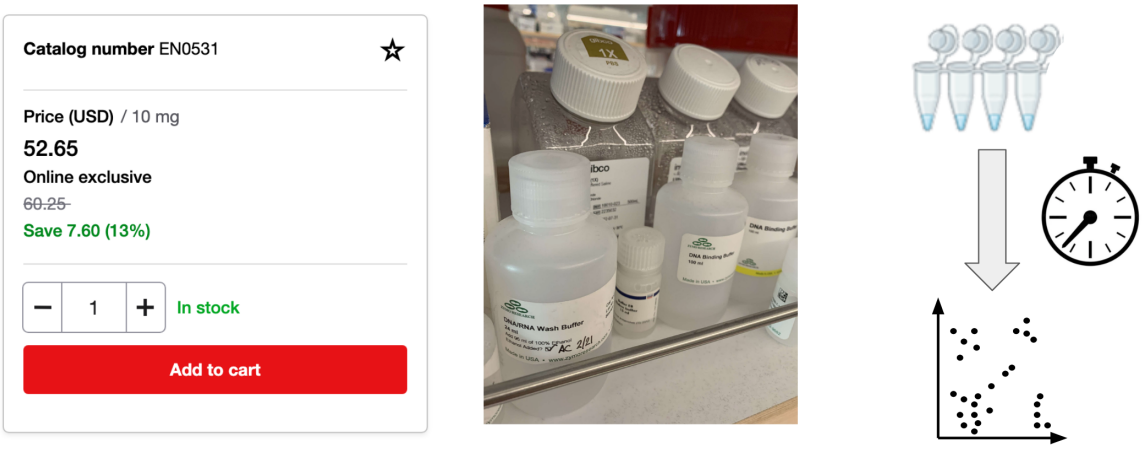



***Figure 5.1**: SWIFT-seq is cost-effective and time-effective.*
*By relying solely on in-house reagents with minimal clean-up and wash steps, SWIFT-seq is affordable, open-source, and fast.*

| change paramaters in green boxes: | | |
|---|---|---|
| How many cells crosslinked (millions): | 1 | |
| How many cells thawed to wash (millions): | 1 | |
| How many RTBCs: | 24 | |
| How many ODD wells: | 96 | |
| How many NY wells: | 96 | |
| How many PCR aliquots: | 10 | |
| How many cells: | 1,000,000 | |
| How many reads on Aviti (millions) | 500 | |
| | | |
| sum of everything, cost of this SWIFT seq | how many cells | **cost per cell ($)** |
| 287.3723794 | 1,000,000 | **0.0002873723794** |
| | | |
| sum of everything, **PLUS sequencing**, cost of this SWIFT seq | how many cells | **cost per cell ($) INCLUDING Aviti** |
| 1214.372379 | 1,000,000 | **0.001214372379** |

| | reagents (in order of appearance) | per one reaction aka | cost w Guttman lab pricing | volume purchased (uL) | cost per uL | cost per uL * volume per rxn | | calculator input |
|---|---|---|---|---|---|---|---|---|
| fixation | formaldehyde | 62.5 | 51.32 | 10,000 | 0.005132 | 0.32075 | | |
| | 1x PBS | 3937.5 | 162.5 | 5,000,000 | 0.0000325 | 0.12796875 | 0.44871875 | 0.44871875 |
| wash/perm | 1x PBS | 100000 | 162.5 | 5,000,000 | 0.0000325 | 3.25 | | |
| | triton | 5 | 56.1 | 100000 | 0.000561 | 0.002805 | | |
| | tween | 5 | 54.9 | 100000 | 0.000549 | 0.002745 | | |
| | Ribolock (in the washes) | 10 | 5440 | 14000 | 0.3885714286 | 3.885714286 | 7.141264286 | 7.141264286 |
| RT | Ribolock | 1 | 5440 | 14000 | 0.3885714286 | 0.3885714286 | | |
| | RT barcoded 9mer (400uM) | 4 | 120 | 3360 | 0.03571428571 | 0.1428571429 | | |
| | 5x RT buffer | 4 | *included in price for maxima RT (purchased together, one cost) | | | | | |
| | 10mM dNTP | 2 | 230.4 | 4000 | 0.0576 | 0.1152 | | |
| | Maxima RT | 1 | 555.52 | 250 | 2.22208 | 2.22208 | | |
| | ExoI | 5 | 222.6 | 750 | 0.2968 | 1.484 | | |
| | EDTA | 1 | 54.65 | 100000 | 0.0005465 | 0.0005465 | 4.353255071 | 104.4781217 |
| split pool ODD | homemade ligation mix (table below) | 6 | *cost from table below | | 0.08483895 | 0.5090337 | | |
| | **odd plate (45uM, 1 well)** | 3 | 220 | 10080 | 0.02182539683 | 0.06547619048 | 0.5745098905 | 55.15294949 |
| **split pool NY** | homemade ligation mix (table below) | 6 | | | 0.08483895 | 0.5090337 | | |
| | **NY odd plate (45uM, 1 well)** | 3 | 220 | 10080 | 0.02182539683 | 0.06547619048 | 0.5745098905 | 55.15294949 |
| PCR reaction | Thermolabile Pro K | 1 | 141.6 | 250 | 0.5664 | 0.5664 | | |
| | 5x RT buffer | | | | | 0 | | |
| | 10mM dNTP | 1 | 230.4 | 4000 | 0.0576 | 0.0576 | | |
| | Maxima RT | 1 | 555.52 | 250 | 2.22208 | 2.22208 | | |
| | Rnase H | 2 | 230.3 | 250 | 0.9212 | 1.8424 | | |
| | Rnase cocktail | 1 | 112.62 | 1000 | 0.11262 | 0.11262 | | |
| | UMI splint | 0.45 | 233.47 | 3500 | 0.06670571429 | 0.03001757143 | | |
| | Silane beads | 12 | 195 | 5000 | 0.039 | 0.468 | | |
| | RLT | 120 | 20 | 120000 | 0.0001666666667 | 0.02 | | |
| | Q5 Master Mix | 25 | 584 | 25000 | 0.02336 | 0.584 | | |
| | i5 primer (12.5uM) | 25 | 75.18 | 12500 | 0.0060144 | 0.15036 | | |
| | i7 primer (12.5uM) | 25 | 75.18 | 12500 | 0.0060144 | 0.15036 | | |
| | spri beads | 40 | 3700 | 500000 | 0.0074 | 0.296 | 6.499837571 | 64.99837571 |
| | **more costs to consider** | | | | | | | |
| | tape station one lane and reagents | | | | | | | |
| | gel clean | | | | | | | |
| | qubit reagents and tubes | | | | | | | |

| | | how many reads asking for | price of full run | one billion reads total | cost per one read | total cost for this Aviti run | | |
|---|---|---|---|---|---|---|---|---|
| | Aviti Run | 500,000,000 | 1854 | 1,000,000,000 | 0.000001854 | 927 | | |

| | homemade ligation mix | amount (uL) | cost w Guttman lab pricing | volume purchased (uL) | cost per uL | cost per 1 SWIFT seq | | |
|---|---|---|---|---|---|---|---|---|
| | Instant Sticky Master Mix | 1000 | 303.8 | 1250 | 0.24304 | 243.04 | | |
| | 1,2 Propane diol | 600 | 41.86 | 25000 | 0.0016744 | 1.00464 | | |
| | 5x quick ligation buffer | 1600 | 34.3 | 2,000 | 0.01715 | 27.44 | | |
| | total volume | 3200 | 379.96 | | cost for master mix | 271.48464 | | |
| | | | | | cost per uL MM | 0.08483895 | | |

| more things to consider adding |
|---|
| tris-HCL |
| jumpcode kit |
| cost of annealing buffer included in plates |

***Figure 5.2**: Cost breakdown of a SWIFT-seq experiment.*
*Spreadsheet showing the cost per cell for a SWIFT-seq experiment.*

**The SWIFT-seq protocol**

*Experimental protocol:*

SWIFT-seq, our single-cell split-pool-barcoding approach, works by sequentially tagging individual cells with unique barcode combinations in a simple, streamlined three day workflow (Figure 5.3), as follows:

5. Day 1: First, formaldehyde-crosslinked cells are gently permeabilized and subjected to first-strand cDNA synthesis using barcoded reverse transcription (RT) primers, forming the first barcode in the combinatorial barcoding scheme. Next, the cells are distributed into wells of a 96-well plate, each well containing well-specific barcodes that are ligated onto the cDNA. Cells are then pooled and re-distributed for additional barcoding rounds. Typically, three sets of barcodes provide sufficient combinatorial complexity to resolve single cells, but, as the need arises (i.e. scaling to tens of millions of cells), more barcoding rounds can easily be incorporated.

6. Day 2: Crosslinks are reversed, and a second RT reaction extends partially transcribed cDNA molecules from the first RT step. RNA is then digested via RNase treatment, followed by the incorporation of a UMI sequence through splint ligation.

7. Day 3: The cDNA is isolated and PCR-amplified, simultaneously adding sequencing adapters to generate the final double-stranded cDNA library (Figure 5.4). The product is then cleaned with SPRI beads, size-selected via gel electrophoresis, quality-checked on an Agilent TapeStation, and prepared for sequencing.

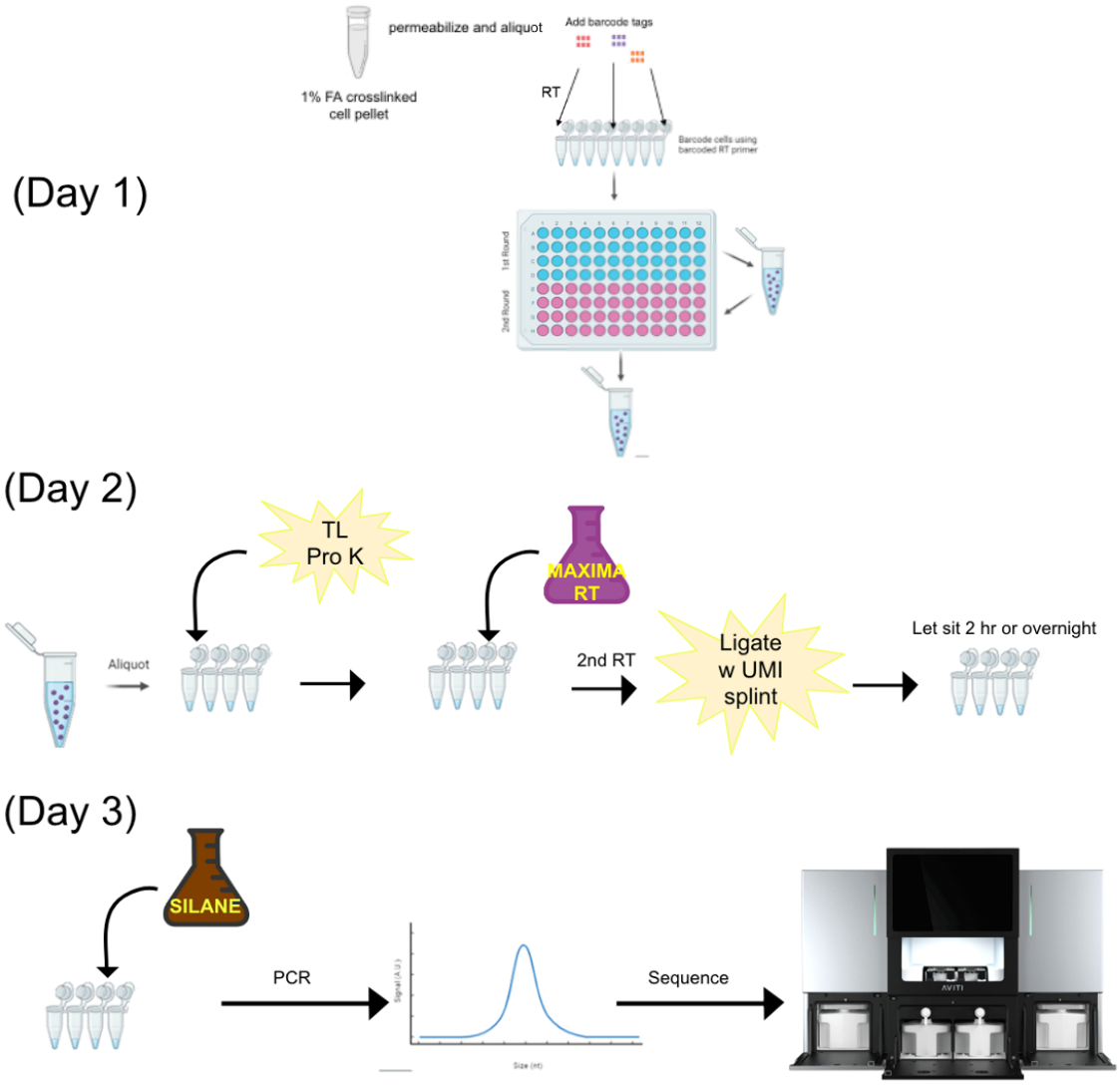



***Figure 5.3**: SWIFT-seq library preparation.*
*SWIFT-seq uses an efficient three day library preparation strategy.*

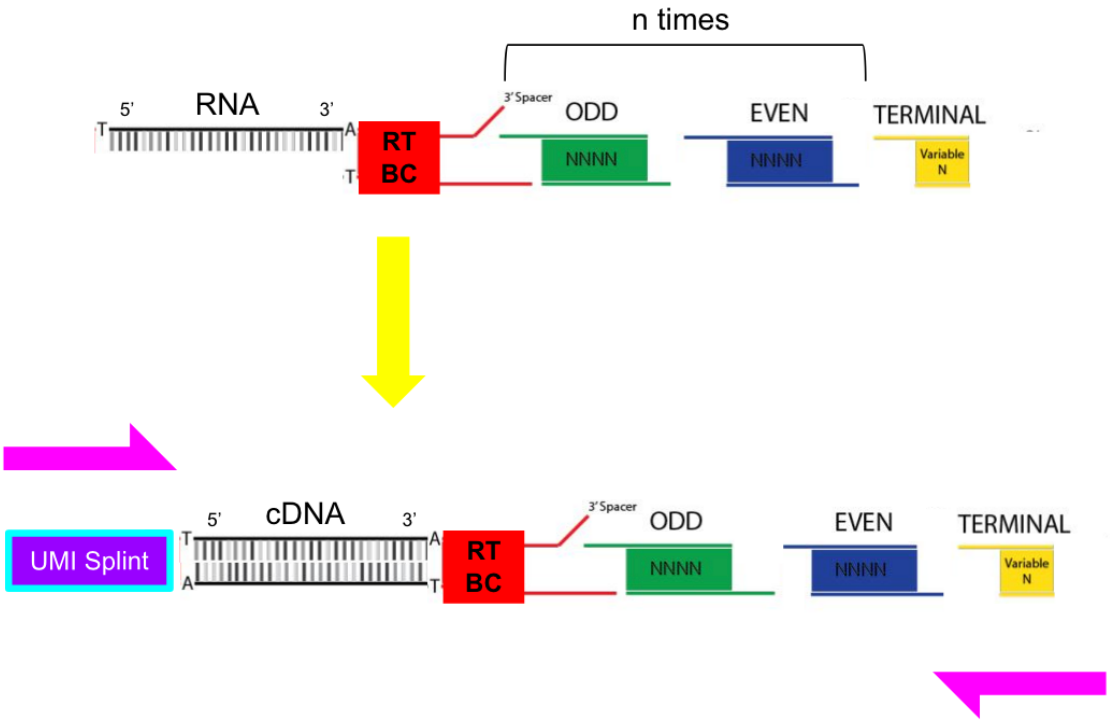



***Figure 5.4**: SWIFT-seq library structure.*
*The R1 sequencing reads begin at the UMI sequence while the R2 sequencing reads begin at the final round barcode tag and extend through all preceding barcode sequences.*

SWIFT-seq reagents include:

- Buffers:
  - 1X PBS (RNase/DNase free)
  - 1X PBS (RNase/DNase free) + 0.125% Triton X-100
  - 1X PBS (RNase/DNase free) + 0.1% Tween
  - 1X PBS (RNase/DNase free) + 0.1% Tween + 50 mM EDTA
- Magnetic beads:
  - MyOne Silane Beads
    [Catalog Number: 37005D]
  - Ampure SPRI Beads
    [Bulldog Bio Catalog Number: CNGS500]
- Enzymes:
  - RNA integrity:
    - Ribolock RNase Inhibitor
      [ThermoFisher Catalog Number: EO038C005]
  - Ligation:
    - NEB 2X Instant Sticky Master Mix (ISMM)
      [NEB Catalog Number: M0370L]
  - Reverse Transcription:
    - 5X RT MasterMix Buffer & Maxima RT H minus
      [LifeTech Catalog Number: EP0753]
    - 10 mM dNTP mix
      [Catalog Number:N0447L]
    - NEB Exonuclease I
      [Catalog Number: M0293L]
  - Cell Lysis and Splint Ligation:
    - 10X RNase H Buffer
      [NEB Catalog Number: B0297S]
    - NEB Thermolabile ProK
      [NEB Catalog Number: P8111S]
    - NEB RNase H
      [Catalog Number: M0297L]
    - NEB RNase A/T1 cocktail
      [LifeTech Catalog Number: AM2286]
  - PCR:
    - 2X Q5 Mastermix
      [NEB Catalog Number: M0492L]

*Computational protocol:*

SWIFT-seq data (in the form of paired-end FASTQ files) can be processed via a Snakemake workflow (Mölder et al., 2021) (Figure 5.5). This pipeline includes steps such as adapter trimming, barcode identification, ribosomal RNA removal, and alignment.


***Figure 5.5**: SWIFT-seq computational pipeline.*
*The pipeline to process SWIFT-seq data is shown as dependency graph of rules as part of the Snakemake workflow management system.*

The workflow is executed based on settings specified in a YAML (Ben-Kiki et al., 2004) configuration file where one can specify such options as the path to the input FASTQ files, the adapter sequences, the path to the splitcode config file (Sullivan and Pachter, 2024) (for processing the cell barcodes), the path to a file of ribosomal RNA sequences that are to be excluded, and aligner settings (e.g. the path to the index for read alignment).

Main components of the workflow are detailed as follows:

- adaptor_trimming_pe: The first processing step of the workflow. Adapters are trimmed from the reads using cutadapt (Martin, 2011).

- splitcode_barcodeID: Processes the adapter-trimmed reads using splitcode (Sullivan and Pachter, 2024), which assigns unique pseudobarcodes to each barcode combination and reformats the reads into the following format:

    - R1: 20-bp pseudobarcode followed by 10-bp UMI followed by the sequence to be aligned.

    - R2: The sequence to be aligned from the other mate.

    (By default, the alignment sequence is limited to 65 bp to prevent reading into the split-pool barcode tags, but this setting can be adjusted).

- set_barcode_mapping_file: Prepares a "barcode mapping file" from splitcode output. This file contains the encoding necessary to convert the 20-bp pseudobarcodes back to the original combination of split-pool barcoding tags.

- starsolo_align: Runs STARsolo (Kaminow et al., 2021) to align the reformatted reads to the reference genome and to generate a count matrix.

- decode_barcodes_starsolo: Decodes the pseudobarcodes of the count matrix into the original combination of split-pool barcoding tags using the encoding specified in the "barcode mapping file".

- generate_anndata_starsolo: An anndata object (Virshup et al., 2021) to store the output count matrix for downstream processing in Python.

- merge_anndata: In case multiple samples or aliquots are analyzed, the multiple count matrices are merged together into a single anndata object.

While the workflow described above uses STARsolo as the aligner, the workflow also supports kallisto | bustools (Melsted et al., 2021; Sullivan et al., 2024) for generating output count matrices. As the contents of this dissertation primarily focus on kallisto, STARsolo was chosen to highlight the pipeline's flexibility and compatibility with different short-read aligners. The pros and cons of each alignment strategy (i.e. full genome alignment vs. pseudoalignment) are the same as with any other single-cell RNA-seq technology.

The SWIFT-seq pipeline also has components used for ribosomal RNA removal via bowtie2 (Langmead and Salzberg, 2012):

- create_exclusion_index: Prepares a bowtie2 index from a path to a FASTA file containing the sequences (e.g. ribosomal RNAs) that are to be removed.

- get_exclusion_reads: Performs the bowtie2 alignment in order to identify reads that align to the bowtie2 index.

- exclude_reads_r1: Uses seqkit (Shen et al., 2016) to remove R1 reads that correspond to the bowtie2 alignments.

- exclusion_reads_r2: Uses seqkit to remove R2 reads that correspond to the bowtie2 alignments (note: paired-end alignment is done with bowtie2 so the list of alignments supplied to seqkit is the same for R1 and R2, ensuring that they remain synchronized).

The final components of the SWIFT-seq pipeline are the generation of QC reports:

- initial_fastqc: Generates QC statistics from running the FastQC program (Babraham Bioinformatics; program obtained from the following URL: https://www.bioinformatics.babraham.ac.uk/projects/fastqc/) on the raw FASTQ files from a SWIFT-seq run.

- post_trim_fastqc: Generates QC statistics from running the FastQC program on the FASTQ files after adapter trimming has been performed.

- concatenate_exclusion_results: Produces a QC report of the reads aligned (via bowtie2) to the "exclusion" sequences (e.g. ribosomal RNA).

- calc_ligation_efficiency: Calculates metrics on how many reads contain the full set of barcodes, a partial set of barcodes, or no barcodes. This lends insight into the ligation efficiency of the barcoding reactions.

- concatenate_ligeff_results: Generates a QC report of the above barcode metrics.

- log_config: Simply records the configuration file used for the execution of the Snakemake pipeline.

In addition to standard quantifications, one can also quantify allele-specific expression using the SWIFT-seq Snakemake pipeline. One can supply a VCF file (Danecek et al., 2011), which annotates single nucleotide polymorphisms (SNPs) in the reference genome. For the STARsolo step, the WASP algorithm will be executed (Asiimwe and Alexander, 2024; Van De Geijn et al., 2015), enabling allele-specific read alignment with reduced reference mapping bias. Alternately, for a kallisto-based pseudoalignment approach, one can make a kallisto transcriptome index from a hybrid genome (i.e. containing the two haplotypes if the organism is diploid).

**Analysis of sequenced SWIFT-seq libraries**

*Human-mouse mixing experiment:*

A human-mouse mixing experiment was performed to determine whether each combination of barcodes represents a unique cell (as opposed to doublets). To do this, cross-linked human HEK293T cells and mouse embryonic stem cells (TX1072 cells) were mixed prior to split-pooling. The resulting SWIFT-seq libraries were then sequenced and the sequencing reads were aligned to a combined human and mouse reference genome. The UMIs that could be uniquely assigned to either a human gene or a mouse gene were counted (Figure 5.6). Overall, there was a very low mixing rate (under 1%) suggesting that each combination of barcodes represents a single cell (Figure 5.7).

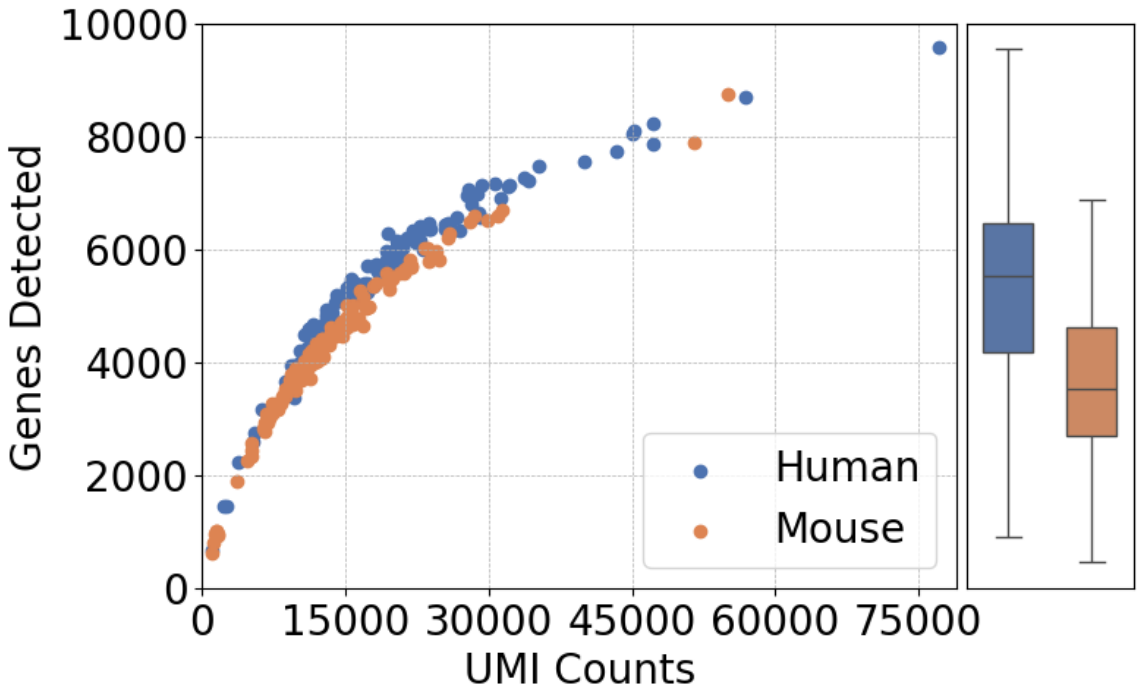



***Figure 5.6**: UMI and gene detection from a species-mixing experiment.*
*UMIs mapped uniquely to either a mouse gene or a human gene were counted in a species mixing experiment consisting of mouse embryonic stem cells (TX1072) and human cells (HEK293T).*

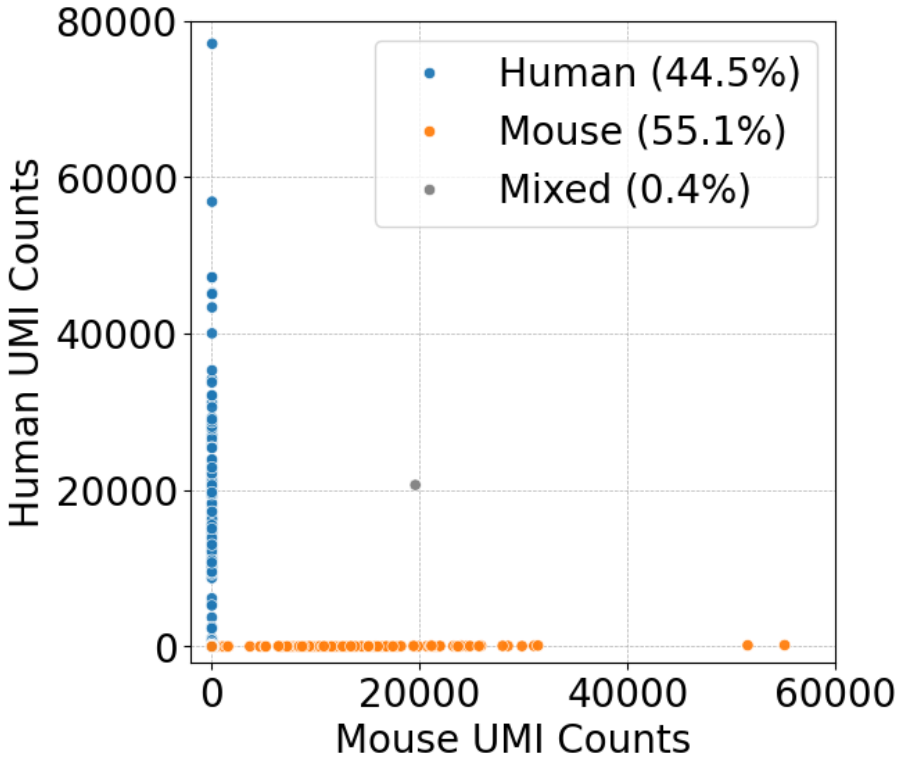



***Figure 5.7**: Species-mixing experiment shows minimal mixing.*
*Mouse (TX1072) vs. human (HEK293T) cells are identified based on ≥ 1000 UMI counts and ≥ 95% UMIs assigned to mouse or human.*

*Gene body coverage:*

To assess whether SWIFT-seq provides full coverage over the gene body (as opposed to, say, only the 5′ end or the 3′ end of genes), the mouse embryonic stem cells (TX1072) from the species-mixing experiment were further analyzed. The program deepTools2 (Ramírez et al., 2016) was used to create metagene plots (i.e. coverage profiles) from the BAM file of aligned reads. Overall, SWIFT-seq appears to provide comprehensive gene length coverage (Figure 5.8).

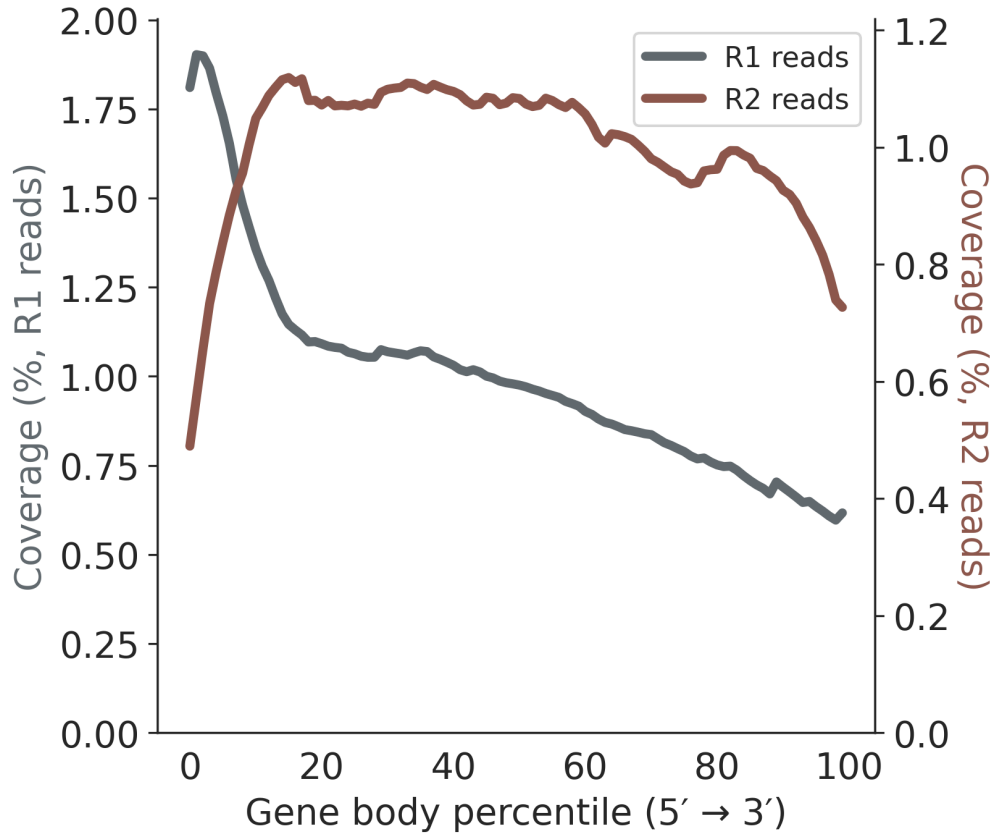



***Figure 5.8****: SWIFT-seq shows comprehensive gene length coverage.*
*Coverages over R1 and R2 reads of mouse embryonic stem cells (TX1072) are shown.*

*Allele-specific expression:*

To validate SWIFT-seq's ability to resolve allele-specific expression, we performed SWIFT-seq experiments on pSM44 mouse embryonic stem cells, which are 129S1/SvImJ × CAST hybrids carrying a doxycycline-inducible Xist gene from the 129S1/SvImJ allele, and induced Xist expression with the addition of doxycycline. As expected, UMIs for the Xist gene in individual cells were predominantly assigned to the 129S1/SvImJ allele (Figure 5.9).

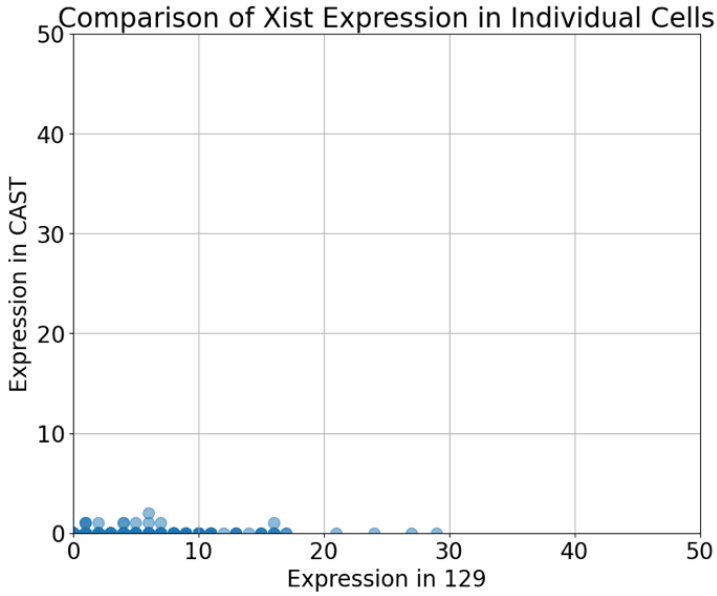



***Figure 5.9**: Example of allele-specific expression resolvable by SWIFT-seq.*
*For the Xist gene, most UMI counts in each cell are derived from the 129S1/SvImJ allele, the inducible allele in pSM44 mouse embryonic stem cells.*

*Cell type classification of a mixed population of blood cells:*

Next, to assess the ability of SWIFT-seq to resolve cell types from a complex population of cells, human peripheral blood mononuclear cells (PBMCs) were used. After filtering for cells with at least 1000 UMIs, the counts were normalized with CP10k normalization followed by log1p transformation. Highly variable genes were identified then nearest neighbor graphs were constructed from the cell coordinates on the top 40 principal component analysis (PCA) embeddings in Scanpy (Wolf et al., 2018). Cell type annotation was then performed with CellTypist (Domínguez Conde et al., 2022; Xu et al., 2023), with the option 'model' set to "Immune_All_Low.pkl" and the option 'majority_voting' set to True, enabling cell type annotation of immune cells with the aid of Leiden clustering. Multiple subtypes of B cells, T cells, natural killer cells, and macrophages were identified, along with confirmed expression of known marker genes, demonstrating that SWIFT-seq can effectively be used to characterize diverse cell types (Figure 5.10).

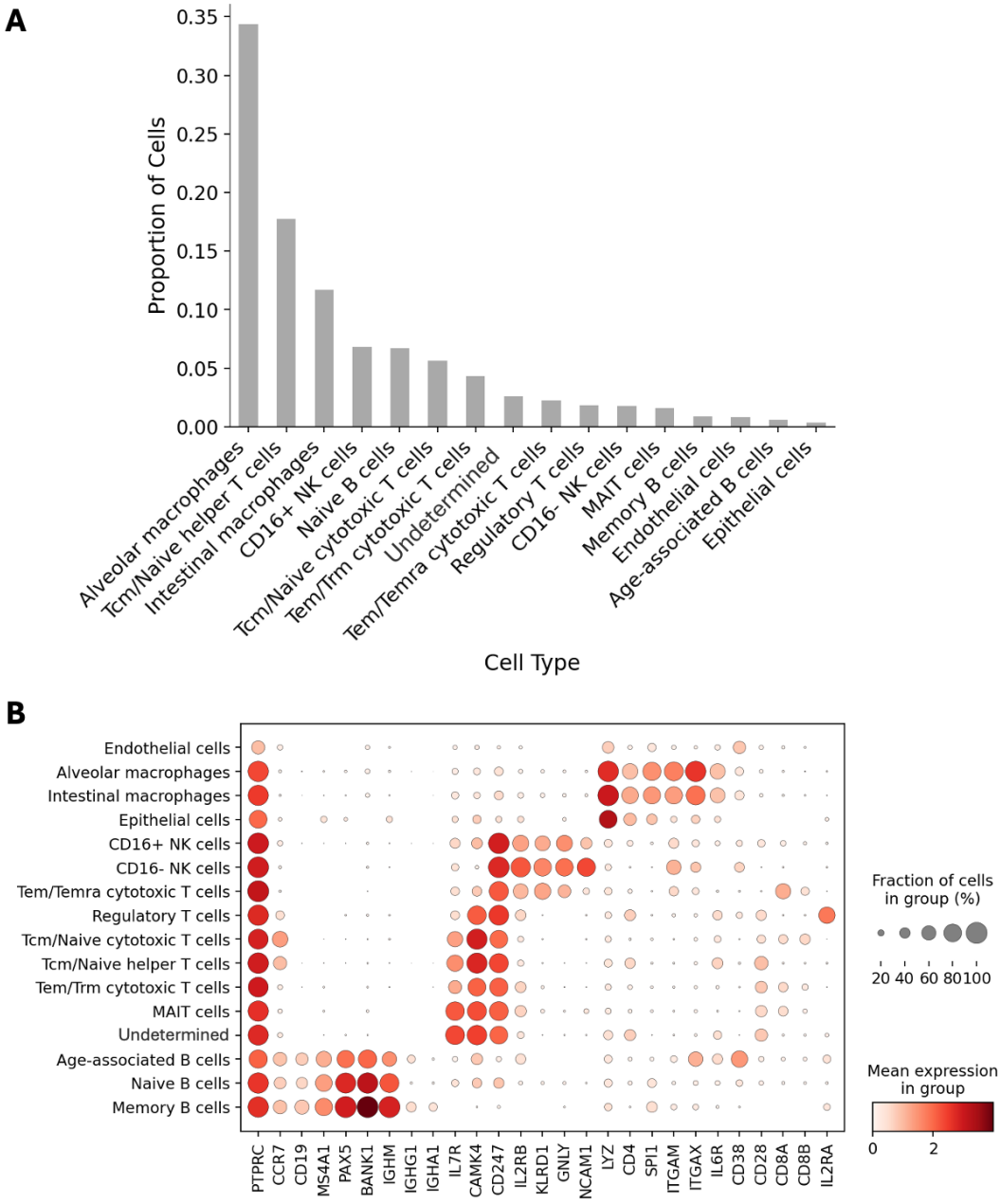



***Figure 5.10**: Cell Type Classification of a mixed population (PBMCs) using SWIFT-seq. (A) Proportions of each cell type identified within a PBMC sample. (B) Heatmap of immune marker gene expressions across identified cell types.*

*Ribosomal RNA content:*

Ribosomal RNAs (rRNAs) are extremely abundant in cells and it is expected that the majority reads will originate from rRNAs, which is why the SWIFT-seq pipeline explicitly removes alignments to rRNAs. One solution to remove rRNAs before sequencing is to use a recombinant Cas9 enzyme complexed with a library of guide RNAs targeting ribosomal RNA sequences. This method, called Depletion of Abundant Sequences by Hybridization (DASH) (Gu et al., 2016), can deplete rRNAs directly from a sequencing library and is thus an attractive option to add to SWIFT-seq. A proprietary Cas9-based rRNA depletion approach has been commercialized into the CRISPRclean kit (Jumpcode Genomics). We tested the kit and observed over a 3-fold reduction in rRNA content in our SWIFT-seq libraries (Figure 5.11). Work is ongoing to develop an affordable, open-source approach for depleting rRNA directly from sequencing libraries.

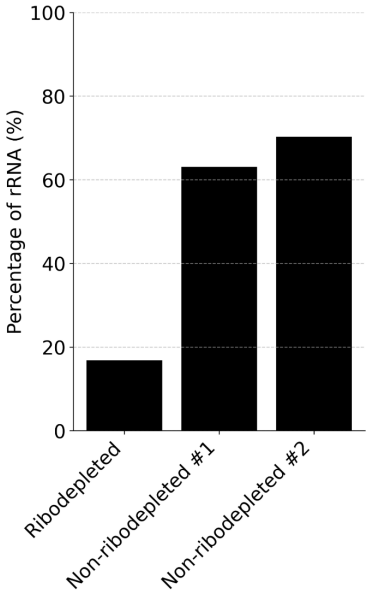



***Figure 5.11****: Ribodepletion of SWIFT-seq libraries.*
*Using a ribosome depletion kit can greatly reduce the rRNA content of SWIFT-seq libraries. Ribodepleted: A SWIFT-seq library with the CRISPRclean kit used. Non-ribodepleted #1 and Non-ribodepleted #2: Two SWIFT-seq libraries without any rRNA depletion procedures used.*

*Comparisons against other technologies:*

To validate SWIFT-seq, we compared SWIFT-seq with other technologies to determine whether SWIFT-seq is concordant with other technologies, such as Smart-seq3, SPLiT-seq (Parse Biosciences), and 10x Genomics. SWIFT-seq, like other technologies, can capture a considerable amount of both nascent and mature RNA for single-cell RNA-seq (Figure 5.12). Moreover, HEK cells from SWIFT-seq are correlated well with HEK cells sequenced from other technologies (Figure 5.13). Altogether, SWIFT-seq appears to be concordant with other technologies.

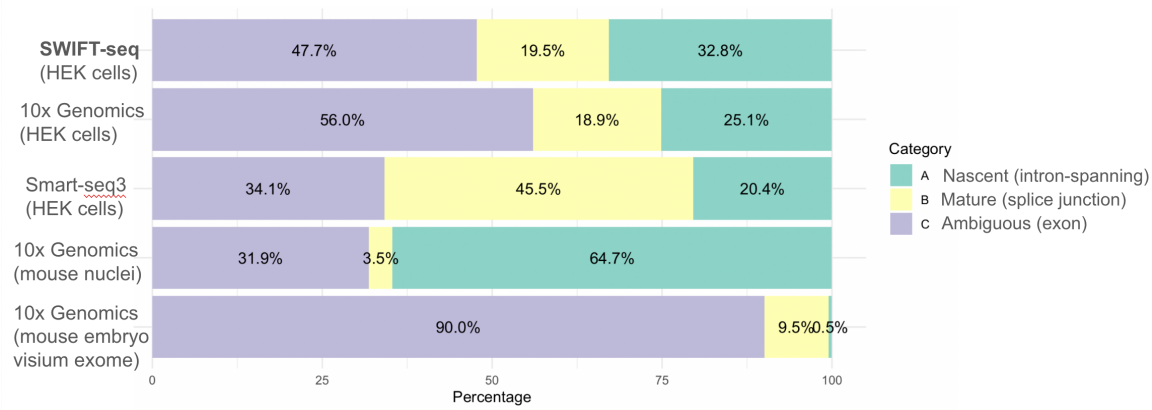



***Figure 5.12**: Distribution of RNA splicing status across different technologies. Technologies are compared on the basis of nascent, mature, and ambiguous RNA abundance.*

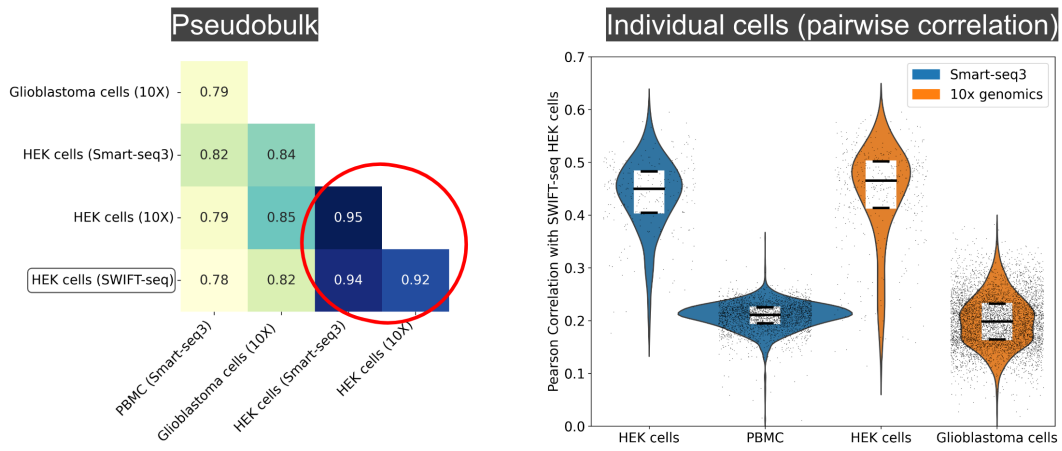



***Figure 5.13**: Correlation between different single-cell RNA-seq technologies. HEK cells profiled with SWIFT-seq show strong concordance with those from Smart-seq3 and 10x Genomics, as demonstrated at both the pseudobulk and single-cell level.*

*Conclusion:*

In conclusion, SWIFT-seq is an affordable, open-source technology designed for both small and large-scale single-cell RNA-seq experiments. By combining high-throughput cell capture with deep, full-length transcript coverage, it enables accurate detection of allelic and isoform-level differences across many cells. With its low cost and ease of use, SWIFT-seq provides researchers a practical, scalable approach for achieving both depth and breadth in single-cell transcriptomic studies.

# Bibliography

Adduri, A., Kim, S., 2024. Ornaments for efficient allele-specific expression estimation with bias correction. The American Journal of Human Genetics 111, 1770–1781. https://doi.org/10.1016/j.ajhg.2024.06.014

Ahmed, Z., Zeeshan, S., Dandekar, T., 2014. Developing sustainable software solutions for bioinformatics by the "Butterfly" paradigm. F1000Res 3, 71. https://doi.org/10.12688/f1000research.3681.1

Aldridge, S., Teichmann, S.A., 2020. Single cell transcriptomics comes of age. Nat Commun 11, 4307. https://doi.org/10.1038/s41467-020-18158-5

Almodaresi, F., Sarkar, H., Srivastava, A., Patro, R., 2018. A space and time-efficient index for the compacted colored de Bruijn graph. Bioinformatics 34, i169–i177. https://doi.org/10.1093/bioinformatics/bty292

Almodaresi, F., Zakeri, M., Patro, R., 2021. PuffAligner: a fast, efficient and accurate aligner based on the Pufferfish index. Bioinformatics 37, 4048–4055. https://doi.org/10.1093/bioinformatics/btab408

Amezquita, R.A., Lun, A.T.L., Becht, E., Carey, V.J., Carpp, L.N., Geistlinger, L., Marini, F., Rue-Albrecht, K., Risso, D., Soneson, C., Waldron, L., Pagès, H., Smith, M.L., Huber, W., Morgan, M., Gottardo, R., Hicks, S.C., 2020. Orchestrating single-cell analysis with Bioconductor. Nat Methods 17, 137–145. https://doi.org/10.1038/s41592-019-0654-x

Anders, S., Pyl, P.T., Huber, W., 2015. HTSeq--a Python framework to work with high-throughput sequencing data. Bioinformatics 31, 166–169. https://doi.org/10.1093/bioinformatics/btu638

Arslan, S., Garcia, F.J., Guo, M., Kellinger, M.W., Kruglyak, S., LeVieux, J.A., Mah, A.H., Wang, H., Zhao, J., Zhou, C., Altomare, A., Bailey, J., Byrne, M.B., Chang, C., Chen, S.X., Cho, B., Dennler, C.N., Dien, V.T., Fuller, D., Kelley, R., Khandan, O., Klein, M.G., Kim, M., Lajoie, B.R., Lin, B., Liu, Y., Lopez, T., Mains, P.T., Price, A.D., Robertson, S.R., Taylor-Weiner, H., Tippana, R., Tomaney, A.B., Zhang, S., Abtahi, M., Ambroso, M.R., Bajari, R., Bellizzi, A.M., Benitez, C.B., Berard, D.R., Berti, L., Blease, K.N., Blum, A.P., Boddicker, A.M., Bondar, L., Brown, C., Bui, C.A., Calleja-Aguirre, J., Cappa, K., Chan, J., Chang, V.W., Charov, K., Chen, X., Constandse, R.M., Damron, W., Dawood, M., DeBuono, N., Dimalanta, J.D., Edoli, L., Elango, K., Faustino, N., Feng, C., Ferrari, M., Frankie, K., Fries, A., Galloway, A., Gavrila, V., Gemmen, G.J., Ghadiali, J., Ghorbani, A., Goddard, L.A., Guetter, A.R., Hendricks, G.L., Hentschel, J., Honigfort, D.J., Hsieh, Y.-T., Hwang Fu, Y.-H., Im, S.K., Jin, C., Kabu, S., Kincade, D.E., Levy, S., Li, Y., Liang, V.K., Light, W.H., Lipsher, J.B., Liu, T., Long, G., Ma, R., Mailloux, J.M., Mandla, K.A., Martinez, A.R., Mass, M., McKean, D.T., Meron, M., Miller, E.A., Moh, C.S., Moore, R.K., Moreno, J., Neysmith, J.M., Niman, C.S., Nunez, J.M., Ojeda, M.T., Ortiz, S.E., Owens, J., Piland, G., Proctor, D.J., Purba, J.B., Ray, M., Rong, D., Saade, V.M., Saha, S.,

Tomas, G.S., Scheidler, N., Sirajudeen, L.H., Snow, S., Stengel, G., Stinson, R., Stone, M.J., Sundseth, K.J., Thai, E., Thompson, C.J., Tjioe, M., Trejo, C.L., Trieger, G., Truong, D.N., Tse, B., Voiles, B., Vuong, H., Wong, J.C., Wu, C.-T., Yu, H., Yu, Y., Yu, M., Zhang, X., Zhao, D., Zheng, G., He, M., Previte, M., 2024. Sequencing by avidity enables high accuracy with low reagent consumption. Nat Biotechnol 42, 132–138. https://doi.org/10.1038/s41587-023-01750-7

Asiimwe, R., Alexander, D., 2024. STAR+WASP reduces reference bias in the allele-specific mapping of RNA-seq reads. https://doi.org/10.1101/2024.01.21.576391

Audoux, J., Philippe, N., Chikhi, R., Salson, M., Gallopin, M., Gabriel, M., Le Coz, J., Drouineau, E., Commes, T., Gautheret, D., 2017. DE-kupl: exhaustive capture of biological variation in RNA-seq data through k-mer decomposition. Genome Biol 18, 243. https://doi.org/10.1186/s13059-017-1372-2

Baldoni, P.L., Chen, Y., Hediyeh-zadeh, S., Liao, Y., Dong, X., Ritchie, M.E., Shi, W., Smyth, G.K., 2024. Dividing out quantification uncertainty allows efficient assessment of differential transcript expression with edgeR. Nucleic Acids Research 52, e13–e13. https://doi.org/10.1093/nar/gkad1167

Battenberg, K., Kelly, S.T., Ras, R.A., Hetherington, N.A., Hayashi, M., Minoda, A., 2022. A flexible cross-platform single-cell data processing pipeline. Nat Commun 13, 6847. https://doi.org/10.1038/s41467-022-34681-z

Becht, E., McInnes, L., Healy, J., Dutertre, C.-A., Kwok, I.W.H., Ng, L.G., Ginhoux, F., Newell, E.W., 2019. Dimensionality reduction for visualizing single-cell data using UMAP. Nat Biotechnol 37, 38–44. https://doi.org/10.1038/nbt.4314

Ben-Kiki, O., Evans, C., Ingerson, B., 2004. Yaml ain't markup language (yaml™) version 1.1.

Boisvert, R.F., Pozo, R., Remington, K.A., 1996. The matrix market exchange formats: Initial design. US Department of Commerce, National Institute of Standards and Technology 5935.

Bolchini, D., Finkelstein, A., Perrone, V., Nagl, S., 2009. Better bioinformatics through usability analysis. Bioinformatics 25, 406–412. https://doi.org/10.1093/bioinformatics/btn633

Bolger, A.M., Lohse, M., Usadel, B., 2014. Trimmomatic: a flexible trimmer for Illumina sequence data. Bioinformatics 30, 2114–2120. https://doi.org/10.1093/bioinformatics/btu170

Booeshaghi, A.S., Min, K.H. (Joseph), Gehring, J., Pachter, L., 2024. Quantifying orthogonal barcodes for sequence census assays. Bioinformatics Advances 4, vbad181. https://doi.org/10.1093/bioadv/vbad181

Booeshaghi, A.S., Chen, X., Pachter, L., 2024. A machine-readable specification for genomics assays. Bioinformatics 40, btae168. https://doi.org/10.1093/bioinformatics/btae168

Bray, N.L., Pimentel, H., Melsted, P., Pachter, L., 2016. Near-optimal probabilistic RNA-seq quantification. Nat Biotechnol 34, 525–527. https://doi.org/10.1038/nbt.3519

Bushnell, B., Rood, J., Singer, E., 2017. BBMerge – Accurate paired shotgun read merging via overlap. PLoS ONE 12, e0185056. https://doi.org/10.1371/journal.pone.0185056

Carilli, M., Gorin, G., Choi, Y., Chari, T., Pachter, L., 2024. Biophysical modeling with variational autoencoders for bimodal, single-cell RNA sequencing data. Nat Methods 21, 1466–1469. https://doi.org/10.1038/s41592-024-02365-9

Chambi, S., Lemire, D., Kaser, O., Godin, R., 2016. Better bitmap performance with Roaring bitmaps. Softw Pract Exp 46, 709–719. https://doi.org/10.1002/spe.2325

Chari, T., Gorin, G., Pachter, L., 2024. Biophysically interpretable inference of cell types from multimodal sequencing data. Nat Comput Sci 4, 677–689. https://doi.org/10.1038/s43588-024-00689-2

Chaung, K., Baharav, T.Z., Henderson, G., Zheludev, I.N., Wang, P.L., Salzman, J., 2023. SPLASH: A statistical, reference-free genomic algorithm unifies biological discovery. Cell 186, 5440-5456.e26. https://doi.org/10.1016/j.cell.2023.10.028

Chen, S., Zhou, Y., Chen, Y., Gu, J., 2018. fastp: an ultra-fast all-in-one FASTQ preprocessor. Bioinformatics 34, i884–i890. https://doi.org/10.1093/bioinformatics/bty560

Chen, X., Love, J.C., Navin, N.E., Pachter, L., Stubbington, M.J.T., Svensson, V., Sweedler, J.V., Teichmann, S.A., 2016. Single-cell analysis at the threshold. Nat Biotechnol 34, 1111–1118. https://doi.org/10.1038/nbt.3721

Chen, Y., Chen, L., Lun, A.T.L., Baldoni, P.L., Smyth, G.K., 2024. edgeR v4: powerful differential analysis of sequencing data with expanded functionality and improved support for small counts and larger datasets. https://doi.org/10.1101/2024.01.21.576131

Cheng, O., Ling, M.H., Wang, C., Wu, S., Ritchie, M.E., Göke, J., Amin, N., Davidson, N.M., 2024. Flexiplex: a versatile demultiplexer and search tool for omics data. Bioinformatics 40, btae102. https://doi.org/10.1093/bioinformatics/btae102

Conesa, A., Madrigal, P., Tarazona, S., Gomez-Cabrero, D., Cervera, A., McPherson, A., Szcześniak, M.W., Gaffney, D.J., Elo, L.L., Zhang, X., Mortazavi, A., 2016. A survey of best practices for RNA-seq data analysis. Genome Biol 17, 13. https://doi.org/10.1186/s13059-016-0881-8

Crick, F.H.C., 1958. On protein synthesis. Symp Soc Exp Biol. 12, 138–163.

Danecek, P., Auton, A., Abecasis, G., Albers, C.A., Banks, E., DePristo, M.A., Handsaker, R.E., Lunter, G., Marth, G.T., Sherry, S.T., McVean, G., Durbin, R., 1000 Genomes Project Analysis Group, 2011. The variant call format and VCFtools. Bioinformatics 27, 2156–2158. https://doi.org/10.1093/bioinformatics/btr330

Davis, M.P.A., Van Dongen, S., Abreu-Goodger, C., Bartonicek, N., Enright, A.J., 2013. Kraken: A set of tools for quality control and analysis of high-throughput sequence data. Methods 63, 41–49. https://doi.org/10.1016/j.ymeth.2013.06.027

Delahaye, C., Nicolas, J., 2021. Sequencing DNA with nanopores: Troubles and biases. PLoS One 16, e0257521. https://doi.org/10.1371/journal.pone.0257521

Ding, J., Adiconis, X., Simmons, S.K., Kowalczyk, M.S., Hession, C.C., Marjanovic, N.D., Hughes, T.K., Wadsworth, M.H., Burks, T., Nguyen, L.T., Kwon, J.Y.H.,

Barak, B., Ge, W., Kedaigle, A.J., Carroll, S., Li, S., Hacohen, N., Rozenblatt-Rosen, O., Shalek, A.K., Villani, A.-C., Regev, A., Levin, J.Z., 2020. Systematic comparison of single-cell and single-nucleus RNA-sequencing methods. Nat Biotechnol 38, 737–746. https://doi.org/10.1038/s41587-020-0465-8

Dobin, A., Davis, C.A., Schlesinger, F., Drenkow, J., Zaleski, C., Jha, S., Batut, P., Chaisson, M., Gingeras, T.R., 2013. STAR: ultrafast universal RNA-seq aligner. Bioinformatics 29, 15–21. https://doi.org/10.1093/bioinformatics/bts635

Domínguez Conde, C., Xu, C., Jarvis, L.B., Rainbow, D.B., Wells, S.B., Gomes, T., Howlett, S.K., Suchanek, O., Polanski, K., King, H.W., Mamanova, L., Huang, N., Szabo, P.A., Richardson, L., Bolt, L., Fasouli, E.S., Mahbubani, K.T., Prete, M., Tuck, L., Richoz, N., Tuong, Z.K., Campos, L., Mousa, H.S., Needham, E.J., Pritchard, S., Li, T., Elmentaite, R., Park, J., Rahmani, E., Chen, D., Menon, D.K., Bayraktar, O.A., James, L.K., Meyer, K.B., Yosef, N., Clatworthy, M.R., Sims, P.A., Farber, D.L., Saeb-Parsy, K., Jones, J.L., Teichmann, S.A., 2022. Cross-tissue immune cell analysis reveals tissue-specific features in humans. Science 376, eabl5197. https://doi.org/10.1126/science.abl5197

Einarsson, P.H., Melsted, P., 2023. BUSZ: compressed BUS files. Bioinformatics 39, btad295. https://doi.org/10.1093/bioinformatics/btad295

Ferraj, A., Audano, P.A., Balachandran, P., Czechanski, A., Flores, J.I., Radecki, A.A., Mosur, V., Gordon, D.S., Walawalkar, I.A., Eichler, E.E., Reinholdt, L.G., Beck, C.R., 2023. Resolution of structural variation in diverse mouse genomes reveals chromatin remodeling due to transposable elements. Cell Genomics 3, 100291. https://doi.org/10.1016/j.xgen.2023.100291

Frankish, A., Carbonell-Sala, S., Diekhans, M., Jungreis, I., Loveland, J.E., Mudge, J.M., Sisu, C., Wright, J.C., Arnan, C., Barnes, I., Banerjee, A., Bennett, R., Berry, A., Bignell, A., Boix, C., Calvet, F., Cerdán-Vélez, D., Cunningham, F., Davidson, C., Donaldson, S., Dursun, C., Fatima, R., Giorgetti, S., Giron, C.G., Gonzalez, J.M., Hardy, M., Harrison, P.W., Hourlier, T., Hollis, Z., Hunt, T., James, B., Jiang, Y., Johnson, R., Kay, M., Lagarde, J., Martin, F.J., Gómez, L.M., Nair, S., Ni, P., Pozo, F., Ramalingam, V., Ruffier, M., Schmitt, B.M., Schreiber, J.M., Steed, E., Suner, M.-M., Sumathipala, D., Sycheva, I., Uszczynska-Ratajczak, B., Wass, E., Yang, Y.T., Yates, A., Zafrulla, Z., Choudhary, J.S., Gerstein, M., Guigo, R., Hubbard, T.J.P., Kellis, M., Kundaje, A., Paten, B., Tress, M.L., Flicek, P., 2023. GENCODE: reference annotation for the human and mouse genomes in 2023. Nucleic Acids Res 51, D942–D949. https://doi.org/10.1093/nar/gkac1071

Gorin, G., Fang, M., Chari, T., Pachter, L., 2022a. RNA velocity unraveled. PLoS Comput Biol 18, e1010492. https://doi.org/10.1371/journal.pcbi.1010492

Gorin, G., Pachter, L., 2022a. Distinguishing biophysical stochasticity from technical noise in single-cell RNA sequencing using *Monod*. https://doi.org/10.1101/2022.06.11.495771

Gorin, G., Pachter, L., 2022b. Modeling bursty transcription and splicing with the chemical master equation. Biophys J 121, 1056–1069. https://doi.org/10.1016/j.bpj.2022.02.004

Gorin, G., Vastola, J.J., Fang, M., Pachter, L., 2022b. Interpretable and tractable models of transcriptional noise for the rational design of single-molecule quantification experiments. Nat Commun 13, 7620. https://doi.org/10.1038/s41467-022-34857-7

Gorin, G., Vastola, J.J., Pachter, L., 2023. Studying stochastic systems biology of the cell with single-cell genomics data. Cell Systems 14, 822-843.e22. https://doi.org/10.1016/j.cels.2023.08.004

Griffith, M., Griffith, O.L., Mwenifumbo, J., Goya, R., Morrissy, A.S., Morin, R.D., Corbett, R., Tang, M.J., Hou, Y.-C., Pugh, T.J., Robertson, G., Chittaranjan, S., Ally, A., Asano, J.K., Chan, S.Y., Li, H.I., McDonald, H., Teague, K., Zhao, Y., Zeng, T., Delaney, A., Hirst, M., Morin, G.B., Jones, S.J.M., Tai, I.T., Marra, M.A., 2010. Alternative expression analysis by RNA sequencing. Nat Methods 7, 843–847. https://doi.org/10.1038/nmeth.1503

Grindberg, R.V., Yee-Greenbaum, J.L., McConnell, M.J., Novotny, M., O'Shaughnessy, A.L., Lambert, G.M., Araúzo-Bravo, M.J., Lee, J., Fishman, M., Robbins, G.E., Lin, X., Venepally, P., Badger, J.H., Galbraith, D.W., Gage, F.H., Lasken, R.S., 2013. RNA-sequencing from single nuclei. Proc Natl Acad Sci U S A 110, 19802–19807. https://doi.org/10.1073/pnas.1319700110

Gu, W., Crawford, E.D., O'Donovan, B.D., Wilson, M.R., Chow, E.D., Retallack, H., DeRisi, J.L., 2016. Depletion of Abundant Sequences by Hybridization (DASH): using Cas9 to remove unwanted high-abundance species in sequencing libraries and molecular counting applications. Genome Biol 17, 41. https://doi.org/10.1186/s13059-016-0904-5

Gustafsson, J., Robinson, J., Nielsen, J., Pachter, L., 2021. BUTTERFLY: addressing the pooled amplification paradox with unique molecular identifiers in single-cell RNA-seq. Genome Biol 22, 174. https://doi.org/10.1186/s13059-021-02386-z

Guttman, M., Garber, M., Levin, J.Z., Donaghey, J., Robinson, J., Adiconis, X., Fan, L., Koziol, M.J., Gnirke, A., Nusbaum, C., Rinn, J.L., Lander, E.S., Regev, A., 2010. Ab initio reconstruction of cell type–specific transcriptomes in mouse reveals the conserved multi-exonic structure of lincRNAs. Nat Biotechnol 28, 503–510. https://doi.org/10.1038/nbt.1633

Hagemann-Jensen, M., Ziegenhain, C., Chen, P., Ramsköld, D., Hendriks, G.-J., Larsson, A.J.M., Faridani, O.R., Sandberg, R., 2020. Single-cell RNA counting at allele and isoform resolution using Smart-seq3. Nat Biotechnol 38, 708–714. https://doi.org/10.1038/s41587-020-0497-0

Hao, Y., Hao, S., Andersen-Nissen, E., Mauck, W.M., Zheng, S., Butler, A., Lee, M.J., Wilk, A.J., Darby, C., Zager, M., Hoffman, P., Stoeckius, M., Papalexi, E., Mimitou, E.P., Jain, J., Srivastava, A., Stuart, T., Fleming, L.M., Yeung, B., Rogers, A.J., McElrath, J.M., Blish, C.A., Gottardo, R., Smibert, P., Satija, R., 2021. Integrated analysis of multimodal single-cell data. Cell 184, 3573-3587.e29. https://doi.org/10.1016/j.cell.2021.04.048

Harrison, P.W., Amode, M.R., Austine-Orimoloye, O., Azov, A.G., Barba, M., Barnes, I., Becker, A., Bennett, R., Berry, A., Bhai, J., Bhurji, S.K., Boddu, S., Branco Lins, P.R., Brooks, L., Ramaraju, S.B., Campbell, L.I., Martinez, M.C., Charkhchi, M.,

Chougule, K., Cockburn, A., Davidson, C., De Silva, N.H., Dodiya, K., Donaldson, S., El Houdaigui, B., Naboulsi, T.E., Fatima, R., Giron, C.G., Genez, T., Grigoriadis, D., Ghattaoraya, G.S., Martinez, J.G., Gurbich, T.A., Hardy, M., Hollis, Z., Hourlier, T., Hunt, T., Kay, M., Kaykala, V., Le, T., Lemos, D., Lodha, D., Marques-Coelho, D., Maslen, G., Merino, G.A., Mirabueno, L.P., Mushtaq, A., Hossain, S.N., Ogeh, D.N., Sakthivel, M.P., Parker, A., Perry, M., Piližota, I., Poppleton, D., Prosovetskaia, I., Raj, S., Pérez-Silva, J.G., Salam, A.I.A., Saraf, S., Saraiva-Agostinho, N., Sheppard, D., Sinha, S., Sipos, B., Sitnik, V., Stark, W., Steed, E., Suner, M.-M., Surapaneni, L., Sutinen, K., Tricomi, F.F., Urbina-Gómez, D., Veidenberg, A., Walsh, T.A., Ware, D., Wass, E., Willhoft, N.L., Allen, J., Alvarez-Jarreta, J., Chakiachvili, M., Flint, B., Giorgetti, S., Haggerty, L., Ilsley, G.R., Keatley, J., Loveland, J.E., Moore, B., Mudge, J.M., Naamati, G., Tate, J., Trevanion, S.J., Winterbottom, A., Frankish, A., Hunt, S.E., Cunningham, F., Dyer, S., Finn, R.D., Martin, F.J., Yates, A.D., 2024. Ensembl 2024. Nucleic Acids Res 52, D891–D899. https://doi.org/10.1093/nar/gkad1049

Hashimshony, T., Senderovich, N., Avital, G., Klochendler, A., de Leeuw, Y., Anavy, L., Gennert, D., Li, S., Livak, K.J., Rozenblatt-Rosen, O., Dor, Y., Regev, A., Yanai, I., 2016. CEL-Seq2: sensitive highly-multiplexed single-cell RNA-Seq. Genome Biol 17, 77. https://doi.org/10.1186/s13059-016-0938-8

He, D., Gao, Y., Chan, S.S., Quintana-Parrilla, N., Patro, R., 2024. Forseti : a mechanistic and predictive model of the splicing status of scRNA-seq reads. Bioinformatics 40, i297–i306. https://doi.org/10.1093/bioinformatics/btae207

He, D., Patro, R., 2023. simpleaf: a simple, flexible, and scalable framework for single-cell data processing using alevin-fry. Bioinformatics 39, btad614. https://doi.org/10.1093/bioinformatics/btad614

He, D., Soneson, C., Patro, R., 2023. Understanding and evaluating ambiguity in single-cell and single-nucleus RNA-sequencing. https://doi.org/10.1101/2023.01.04.522742

He, D., Zakeri, M., Sarkar, H., Soneson, C., Srivastava, A., Patro, R., 2022. Alevin-fry unlocks rapid, accurate and memory-frugal quantification of single-cell RNA-seq data. Nat Methods 19, 316–322. https://doi.org/10.1038/s41592-022-01408-3

Holley, G., Melsted, P., 2020. Bifrost: highly parallel construction and indexing of colored and compacted de Bruijn graphs. Genome Biol 21, 249. https://doi.org/10.1186/s13059-020-02135-8

Hoon, S., Ratnapu, K.K., Chia, J., Kumarasamy, B., Juguang, X., Clamp, M., Stabenau, A., Potter, S., Clarke, L., Stupka, E., 2003. Biopipe: A Flexible Framework for Protocol-Based Bioinformatics Analysis. Genome Res. 13, 1904–1915. https://doi.org/10.1101/gr.1363103

Huntley, M.A., Lou, M., Goldstein, L.D., Lawrence, M., Dijkgraaf, G.J.P., Kaminker, J.S., Gentleman, R., 2016. Complex regulation of ADAR-mediated RNA-editing across tissues. BMC Genomics 17, 61. https://doi.org/10.1186/s12864-015-2291-9

IGVF Consortium, 2024. Deciphering the impact of genomic variation on function. Nature 633, 47–57. https://doi.org/10.1038/s41586-024-07510-0

Iqbal, Z., Caccamo, M., Turner, I., Flicek, P., McVean, G., 2012. De novo assembly and genotyping of variants using colored de Bruijn graphs. Nat Genet 44, 226–232. https://doi.org/10.1038/ng.1028

Johnson, M.S., Venkataram, S., Kryazhimskiy, S., 2023. Best Practices in Designing, Sequencing, and Identifying Random DNA Barcodes. J Mol Evol 91, 263–280. https://doi.org/10.1007/s00239-022-10083-z

Kaminow, B., Yunusov, D., Dobin, A., 2021. STARsolo: accurate, fast and versatile mapping/quantification of single-cell and single-nucleus RNA-seq data. https://doi.org/10.1101/2021.05.05.442755

Karimzadeh, M., Hoffman, M.M., 2018. Top considerations for creating bioinformatics software documentation. Briefings in Bioinformatics 19, 693–699. https://doi.org/10.1093/bib/bbw134

Kebschull, J.M., Zador, A.M., 2018. Cellular barcoding: lineage tracing, screening and beyond. Nat Methods 15, 871–879. https://doi.org/10.1038/s41592-018-0185-x

Kent, W.J., Sugnet, C.W., Furey, T.S., Roskin, K.M., Pringle, T.H., Zahler, A.M., Haussler, A.D., 2002. The Human Genome Browser at UCSC. Genome Res. 12, 996–1006. https://doi.org/10.1101/gr.229102

Kijima, Y., Evans-Yamamoto, D., Toyoshima, H., Yachie, N., 2023. A universal sequencing read interpreter. Sci. Adv. 9, eadd2793. https://doi.org/10.1126/sciadv.add2793

Kivioja, T., Vähärautio, A., Karlsson, K., Bonke, M., Enge, M., Linnarsson, S., Taipale, J., 2012. Counting absolute numbers of molecules using unique molecular identifiers. Nat Methods 9, 72–74. https://doi.org/10.1038/nmeth.1778

Kong, Y., 2011. Btrim: a fast, lightweight adapter and quality trimming program for next-generation sequencing technologies. Genomics 98, 152–153. https://doi.org/10.1016/j.ygeno.2011.05.009

Kumar, S., Dudley, J., 2007. Bioinformatics software for biologists in the genomics era. Bioinformatics 23, 1713–1717. https://doi.org/10.1093/bioinformatics/btm239

Kuo, A., Hansen, K.D., Hicks, S.C., 2024. Quantification and statistical modeling of droplet-based single-nucleus RNA-sequencing data. Biostatistics 25, 801–817. https://doi.org/10.1093/biostatistics/kxad010

La Manno, G., Soldatov, R., Zeisel, A., Braun, E., Hochgerner, H., Petukhov, V., Lidschreiber, K., Kastriti, M.E., Lönnerberg, P., Furlan, A., Fan, J., Borm, L.E., Liu, Z., Van Bruggen, D., Guo, J., He, X., Barker, R., Sundström, E., Castelo-Branco, G., Cramer, P., Adameyko, I., Linnarsson, S., Kharchenko, P.V., 2018. RNA velocity of single cells. Nature 560, 494–498. https://doi.org/10.1038/s41586-018-0414-6

Langmead, B., Salzberg, S.L., 2012. Fast gapped-read alignment with Bowtie 2. Nat Methods 9, 357–359. https://doi.org/10.1038/nmeth.1923

Law, C.W., Chen, Y., Shi, W., Smyth, G.K., 2014. voom: precision weights unlock linear model analysis tools for RNA-seq read counts. Genome Biol 15, R29. https://doi.org/10.1186/gb-2014-15-2-r29

Li, B., Dewey, C.N., 2011. RSEM: accurate transcript quantification from RNA-Seq data with or without a reference genome. BMC Bioinformatics 12, 323. https://doi.org/10.1186/1471-2105-12-323

Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., Marth, G., Abecasis, G., Durbin, R., 1000 Genome Project Data Processing Subgroup, 2009. The Sequence Alignment/Map format and SAMtools. Bioinformatics 25, 2078–2079. https://doi.org/10.1093/bioinformatics/btp352

Liao, Y., Raghu, D., Pal, B., Mielke, L.A., Shi, W., 2023. cellCounts: an R function for quantifying 10x Chromium single-cell RNA sequencing data. Bioinformatics 39, btad439. https://doi.org/10.1093/bioinformatics/btad439

Liao, Y., Smyth, G.K., Shi, W., 2019. The R package Rsubread is easier, faster, cheaper and better for alignment and quantification of RNA sequencing reads. Nucleic Acids Res 47, e47. https://doi.org/10.1093/nar/gkz114

Liao, Y., Smyth, G.K., Shi, W., 2014. featureCounts: an efficient general purpose program for assigning sequence reads to genomic features. Bioinformatics 30, 923–930. https://doi.org/10.1093/bioinformatics/btt656

Limasset, A., Rizk, G., Chikhi, R., Peterlongo, P., 2017. Fast and scalable minimal perfect hashing for massive key sets. https://doi.org/10.48550/ARXIV.1702.03154

List, M., Ebert, P., Albrecht, F., 2017. Ten Simple Rules for Developing Usable Software in Computational Biology. PLoS Comput Biol 13, e1005265. https://doi.org/10.1371/journal.pcbi.1005265

Liu, D., 2019. Fuzzysplit: demultiplexing and trimming sequenced DNA with a declarative language. PeerJ 7, e7170. https://doi.org/10.7717/peerj.7170

Love, M.I., Huber, W., Anders, S., 2014. Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. Genome Biol 15, 550. https://doi.org/10.1186/s13059-014-0550-8

Luebbert, L., Sullivan, D.K., Carilli, M., Eldjárn Hjörleifsson, K., Winnett, A.V., Chari, T., Pachter, L., 2025. Detection of viral sequences at single-cell resolution identifies novel viruses associated with host gene expression changes. Nat Biotechnol. https://doi.org/10.1038/s41587-025-02614-y

Lun, A.T.L., McCarthy, D.J., Marioni, J.C., 2016. A step-by-step workflow for low-level analysis of single-cell RNA-seq data with Bioconductor. F1000Res 5, 2122. https://doi.org/10.12688/f1000research.9501.2

Martin, M., 2011. Cutadapt removes adapter sequences from high-throughput sequencing reads. EMBnet j. 17, 10. https://doi.org/10.14806/ej.17.1.200

McCarthy, D.J., Campbell, K.R., Lun, A.T.L., Wills, Q.F., 2017. Scater: pre-processing, quality control, normalization and visualization of single-cell RNA-seq data in R. Bioinformatics 33, 1179–1186. https://doi.org/10.1093/bioinformatics/btw777

McIlroy, M.D., Pinson, E.N., Tague, B.A., 1978. UNIX Time-Sharing System: Foreword. Bell System Technical Journal 57, 1899–1904. https://doi.org/10.1002/j.1538-7305.1978.tb02135.x

McInnes, L., Healy, J., Melville, J., 2018. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. https://doi.org/10.48550/ARXIV.1802.03426

Melsted, P., Booeshaghi, A.S., Liu, L., Gao, F., Lu, L., Min, K.H.J., da Veiga Beltrame, E., Hjörleifsson, K.E., Gehring, J., Pachter, L., 2021. Modular, efficient and constant-memory single-cell RNA-seq preprocessing. Nat Biotechnol 39, 813–818. https://doi.org/10.1038/s41587-021-00870-2

Melsted, P., Ntranos, V., Pachter, L., 2019. The barcode, UMI, set format and BUStools. Bioinformatics 35, 4472–4473. https://doi.org/10.1093/bioinformatics/btz279

Merkel, D., 2014. Docker: lightweight Linux containers for consistent development and deployment. Linux Journal.

Mölder, F., Jablonski, K.P., Letcher, B., Hall, M.B., Tomkins-Tinch, C.H., Sochat, V., Forster, J., Lee, S., Twardziok, S.O., Kanitz, A., Wilm, A., Holtgrewe, M., Rahmann, S., Nahnsen, S., Köster, J., 2021. Sustainable data analysis with Snakemake. F1000Res 10, 33. https://doi.org/10.12688/f1000research.29032.2

Morgan, A.P., Welsh, C.E., 2015. Informatics resources for the Collaborative Cross and related mouse populations. Mamm Genome 26, 521–539. https://doi.org/10.1007/s00335-015-9581-z

Mortazavi, A., Williams, B.A., McCue, K., Schaeffer, L., Wold, B., 2008. Mapping and quantifying mammalian transcriptomes by RNA-Seq. Nat Methods 5, 621–628. https://doi.org/10.1038/nmeth.1226

Niebler, S., Müller, A., Hankeln, T., Schmidt, B., 2020. RainDrop: Rapid activation matrix computation for droplet-based single-cell RNA-seq reads. BMC Bioinformatics 21, 274. https://doi.org/10.1186/s12859-020-03593-4

Ntranos, V., Kamath, G.M., Zhang, J.M., Pachter, L., Tse, D.N., 2016. Fast and accurate single-cell RNA-seq analysis by clustering of transcript-compatibility counts. Genome Biol 17, 112. https://doi.org/10.1186/s13059-016-0970-8

Ntranos, V., Yi, L., Melsted, P., Pachter, L., 2019. A discriminative learning approach to differential expression analysis for single-cell RNA-seq. Nat Methods 16, 163–166. https://doi.org/10.1038/s41592-018-0303-9

Oshlack, A., Robinson, M.D., Young, M.D., 2010. From RNA-seq reads to differential expression results. Genome Biol 11, 220. https://doi.org/10.1186/gb-2010-11-12-220

Pachter, L., 2011. Models for transcript quantification from RNA-Seq. https://doi.org/10.48550/ARXIV.1104.3889

Pai, A.A., Paggi, J.M., Yan, P., Adelman, K., Burge, C.B., 2018. Numerous recursive sites contribute to accuracy of splicing in long introns in flies. PLoS Genet 14, e1007588. https://doi.org/10.1371/journal.pgen.1007588

Pandya-Jones, A., Black, D.L., 2009. Co-transcriptional splicing of constitutive and alternative exons. RNA 15, 1896–1908. https://doi.org/10.1261/rna.1714509

Parekh, S., Ziegenhain, C., Vieth, B., Enard, W., Hellmann, I., 2018. zUMIs - A fast and flexible pipeline to process RNA sequencing data with UMIs. GigaScience 7, giy059. https://doi.org/10.1093/gigascience/giy059

Patro, R., Duggal, G., Love, M.I., Irizarry, R.A., Kingsford, C., 2017. Salmon provides fast and bias-aware quantification of transcript expression. Nat Methods 14, 417–419. https://doi.org/10.1038/nmeth.4197

Patro, R., Mount, S.M., Kingsford, C., 2014. Sailfish enables alignment-free isoform quantification from RNA-seq reads using lightweight algorithms. Nat Biotechnol 32, 462–464. https://doi.org/10.1038/nbt.2862

Pavelin, K., Cham, J.A., De Matos, P., Brooksbank, C., Cameron, G., Steinbeck, C., 2012. Bioinformatics Meets User-Centred Design: A Perspective. PLoS Comput Biol 8, e1002554. https://doi.org/10.1371/journal.pcbi.1002554

Pearson, W.R., Lipman, D.J., 1988. Improved tools for biological sequence comparison. Proc. Natl. Acad. Sci. U.S.A. 85, 2444–2448. https://doi.org/10.1073/pnas.85.8.2444

Perego, M., Maurer, M., Wang, J.X., Shaffer, S., Müller, A.C., Parapatics, K., Li, L., Hristova, D., Shin, S., Keeney, F., Liu, S., Xu, X., Raj, A., Jensen, J.K., Bennett, K.L., Wagner, S.N., Somasundaram, R., Herlyn, M., 2018. A slow-cycling subpopulation of melanoma cells with highly invasive properties. Oncogene 37, 302–312. https://doi.org/10.1038/onc.2017.341

Pertea, M., Kim, D., Pertea, G.M., Leek, J.T., Salzberg, S.L., 2016. Transcript-level expression analysis of RNA-seq experiments with HISAT, StringTie and Ballgown. Nat Protoc 11, 1650–1667. https://doi.org/10.1038/nprot.2016.095

Pezoa, F., Reutter, J.L., Suarez, F., Ugarte, M., Vrgoč, D., 2016. Foundations of JSON Schema, in: Proceedings of the 25th International Conference on World Wide Web. Presented at the WWW '16: 25th International World Wide Web Conference, International World Wide Web Conferences Steering Committee, Montréal Québec Canada, pp. 263–273. https://doi.org/10.1145/2872427.2883029

Picelli, S., Björklund, Å.K., Faridani, O.R., Sagasser, S., Winberg, G., Sandberg, R., 2013. Smart-seq2 for sensitive full-length transcriptome profiling in single cells. Nat Methods 10, 1096–1098. https://doi.org/10.1038/nmeth.2639

Pickrell, J.K., Marioni, J.C., Pai, A.A., Degner, J.F., Engelhardt, B.E., Nkadori, E., Veyrieras, J.-B., Stephens, M., Gilad, Y., Pritchard, J.K., 2010. Understanding mechanisms underlying human gene expression variation with RNA sequencing. Nature 464, 768–772. https://doi.org/10.1038/nature08872

Pimentel, H., Bray, N.L., Puente, S., Melsted, P., Pachter, L., 2017. Differential analysis of RNA-seq incorporating quantification uncertainty. Nat Methods 14, 687–690. https://doi.org/10.1038/nmeth.4324

Pool, A.-H., Poldsam, H., Chen, S., Thomson, M., Oka, Y., 2023. Recovery of missing single-cell RNA-sequencing data with optimized transcriptomic references. Nat Methods 20, 1506–1515. https://doi.org/10.1038/s41592-023-02003-w

Quinodoz, S.A., Bhat, P., Chovanec, P., Jachowicz, J.W., Ollikainen, N., Detmar, E., Soehalim, E., Guttman, M., 2022. SPRITE: a genome-wide method for mapping higher-order 3D interactions in the nucleus using combinatorial split-and-pool barcoding. Nat Protoc 17, 36–75. https://doi.org/10.1038/s41596-021-00633-y

Quinodoz, S.A., Ollikainen, N., Tabak, B., Palla, A., Schmidt, J.M., Detmar, E., Lai, M.M., Shishkin, A.A., Bhat, P., Takei, Y., Trinh, V., Aznauryan, E., Russell, P.,

Cheng, C., Jovanovic, M., Chow, A., Cai, L., McDonel, P., Garber, M., Guttman, M., 2018. Higher-Order Inter-chromosomal Hubs Shape 3D Genome Organization in the Nucleus. Cell 174, 744-757.e24. https://doi.org/10.1016/j.cell.2018.05.024

Rahman, A., Hallgrímsdóttir, I., Eisen, M., Pachter, L., 2018. Association mapping from sequencing reads using k-mers. Elife 7, e32920. https://doi.org/10.7554/eLife.32920

Ramírez, F., Ryan, D.P., Grüning, B., Bhardwaj, V., Kilpert, F., Richter, A.S., Heyne, S., Dündar, F., Manke, T., 2016. deepTools2: a next generation web server for deep-sequencing data analysis. Nucleic Acids Res 44, W160–W165. https://doi.org/10.1093/nar/gkw257

Rebboah, E., Reese, F., Williams, K., Balderrama-Gutierrez, G., McGill, C., Trout, D., Rodriguez, I., Liang, H., Wold, B.J., Mortazavi, A., 2021. Mapping and modeling the genomic basis of differential RNA isoform expression at single-cell resolution with LR-Split-seq. Genome Biol 22, 286. https://doi.org/10.1186/s13059-021-02505-w

Rebboah, E., Weber, R., Abdollahzadeh, E., Swarna, N., Sullivan, D.K., Trout, D., Reese, F., Liang, H.Y., Filimban, G., Mahdipoor, P., Duffield, M., Mojaverzargar, R., Taghizadeh, E., Fattahi, N., Mojgani, N., Zhang, H., Loving, R.K., Carilli, M., Booeshaghi, A.S., Kawauchi, S., Hallgrímsdóttir, I.B., Williams, B.A., MacGregor, G.R., Pachter, L., Wold, B.J., Mortazavi, A., 2025. Systematic cell-type resolved transcriptomes of 8 tissues in 8 lab and wild-derived mouse strains captures global and local expression variation. https://doi.org/10.1101/2025.04.21.649844

Reese, M.G., Hartzell, G., Harris, N.L., Ohler, U., Abril, J.F., Lewis, S.E., 2000. Genome annotation assessment in Drosophila melanogaster. Genome Res 10, 483–501. https://doi.org/10.1101/gr.10.4.483

Rich, J.M., Moses, L., Einarsson, P.H., Jackson, K., Luebbert, L., Booeshaghi, A.S., Antonsson, S., Sullivan, D.K., Bray, N., Melsted, P., Pachter, L., 2024. The impact of package selection and versioning on single-cell RNA-seq analysis. bioRxiv 2024.04.04.588111. https://doi.org/10.1101/2024.04.04.588111

Ritchie, M.E., Phipson, B., Wu, D., Hu, Y., Law, C.W., Shi, W., Smyth, G.K., 2015. limma powers differential expression analyses for RNA-sequencing and microarray studies. Nucleic Acids Res 43, e47. https://doi.org/10.1093/nar/gkv007

Roberts, A., Pachter, L., 2013. Streaming fragment assignment for real-time analysis of sequencing experiments. Nat Methods 10, 71–73. https://doi.org/10.1038/nmeth.2251

Roberts, M., Hayes, W., Hunt, B.R., Mount, S.M., Yorke, J.A., 2004. Reducing storage requirements for biological sequence comparison. Bioinformatics 20, 3363–3369. https://doi.org/10.1093/bioinformatics/bth408

Robinson, J.T., Thorvaldsdóttir, H., Winckler, W., Guttman, M., Lander, E.S., Getz, G., Mesirov, J.P., 2011. Integrative genomics viewer. Nat Biotechnol 29, 24–26. https://doi.org/10.1038/nbt.1754

Robinson, M.D., McCarthy, D.J., Smyth, G.K., 2010. edgeR : a Bioconductor package for differential expression analysis of digital gene expression data. Bioinformatics 26, 139–140. https://doi.org/10.1093/bioinformatics/btp616

Roehr, J.T., Dieterich, C., Reinert, K., 2017. Flexbar 3.0 – SIMD and multicore parallelization. Bioinformatics 33, 2941–2942. https://doi.org/10.1093/bioinformatics/btx330

Rosenberg, A.B., Roco, C.M., Muscat, R.A., Kuchina, A., Sample, P., Yao, Z., Graybuck, L.T., Peeler, D.J., Mukherjee, S., Chen, W., Pun, S.H., Sellers, D.L., Tasic, B., Seelig, G., 2018. Single-cell profiling of the developing mouse brain and spinal cord with split-pool barcoding. Science 360, 176–182. https://doi.org/10.1126/science.aam8999

Roy, S., Coldren, C., Karunamurthy, A., Kip, N.S., Klee, E.W., Lincoln, S.E., Leon, A., Pullambhatla, M., Temple-Smolkin, R.L., Voelkerding, K.V., Wang, C., Carter, A.B., 2018. Standards and Guidelines for Validating Next-Generation Sequencing Bioinformatics Pipelines. The Journal of Molecular Diagnostics 20, 4–27. https://doi.org/10.1016/j.jmoldx.2017.11.003

Schena, M., Shalon, D., Davis, R.W., Brown, P.O., 1995. Quantitative Monitoring of Gene Expression Patterns with a Complementary DNA Microarray. Science 270, 467–470. https://doi.org/10.1126/science.270.5235.467

Shen, W., Le, S., Li, Y., Hu, F., 2016. SeqKit: A Cross-Platform and Ultrafast Toolkit for FASTA/Q File Manipulation. PLoS One 11, e0163962. https://doi.org/10.1371/journal.pone.0163962

Smith, T., Heger, A., Sudbery, I., 2017. UMI-tools: modeling sequencing errors in Unique Molecular Identifiers to improve quantification accuracy. Genome Res. 27, 491–499. https://doi.org/10.1101/gr.209601.116

Soneson, C., Love, M.I., Robinson, M.D., 2015. Differential analyses for RNA-seq: transcript-level estimates improve gene-level inferences. F1000Res 4, 1521. https://doi.org/10.12688/f1000research.7563.2

Soneson, C., Srivastava, A., Patro, R., Stadler, M.B., 2021. Preprocessing choices affect RNA velocity results for droplet scRNA-seq data. PLoS Comput Biol 17, e1008585. https://doi.org/10.1371/journal.pcbi.1008585

Srivastava, A., Malik, L., Sarkar, H., Zakeri, M., Almodaresi, F., Soneson, C., Love, M.I., Kingsford, C., Patro, R., 2020. Alignment and mapping methodology influence transcript abundance estimation. Genome Biol 21, 239. https://doi.org/10.1186/s13059-020-02151-8

Srivastava, A., Malik, L., Smith, T., Sudbery, I., Patro, R., 2019. Alevin efficiently estimates accurate gene abundances from dscRNA-seq data. Genome Biol 20, 65. https://doi.org/10.1186/s13059-019-1670-y

Sullivan, D.K., Hjörleifsson, K.E., Swarna, N.P., Oakes, C., Holley, G., Melsted, P., Pachter, L., 2025. Accurate quantification of nascent and mature RNAs from single-cell and single-nucleus RNA-seq. Nucleic Acids Research 53, gkae1137. https://doi.org/10.1093/nar/gkae1137

Sullivan, D.K., Min, K.H., Hjörleifsson, K.E., Luebbert, L., Holley, G., Moses, L., Gustafsson, J., Bray, N.L., Pimentel, H., Booeshaghi, A.S., Melsted, P., Pachter,

L., 2024. kallisto, bustools and kb-python for quantifying bulk, single-cell and single-nucleus RNA-seq. Nat Protoc. https://doi.org/10.1038/s41596-024-01057-0
Sullivan, D.K., Pachter, L., 2024. Flexible parsing, interpretation, and editing of technical sequences with *splitcode*. Bioinformatics 40, btae331. https://doi.org/10.1093/bioinformatics/btae331
Sullivan, D.K., Boffelli, M., Pachter, L., 2025. Pseudoassembly of k-mers. bioRxiv 2025.05.11.653354. https://doi.org/10.1101/2025.05.11.653354
Sun, Q., Lee, W., Mohri, Y., Takeo, M., Lim, C.H., Xu, X., Myung, P., Atit, R.P., Taketo, M.M., Moubarak, R.S., Schober, M., Osman, I., Gay, D.L., Saur, D., Nishimura, E.K., Ito, M., 2019. A novel mouse model demonstrates that oncogenic melanocyte stem cells engender melanoma resembling human disease. Nat Commun 10, 5023. https://doi.org/10.1038/s41467-019-12733-1
Tian, L., Su, S., Dong, X., Amann-Zalcenstein, D., Biben, C., Seidi, A., Hilton, D.J., Naik, S.H., Ritchie, M.E., 2018. scPipe: A flexible R/Bioconductor preprocessing pipeline for single-cell RNA-sequencing data. PLoS Comput Biol 14, e1006361. https://doi.org/10.1371/journal.pcbi.1006361
Traag, V.A., Waltman, L., van Eck, N.J., 2019. From Louvain to Leiden: guaranteeing well-connected communities. Sci Rep 9, 5233. https://doi.org/10.1038/s41598-019-41695-z
Trapnell, C., 2015. Defining cell types and states with single-cell genomics. Genome Res. 25, 1491–1498. https://doi.org/10.1101/gr.190595.115
Trapnell, C., Cacchiarelli, D., Grimsby, J., Pokharel, P., Li, S., Morse, M., Lennon, N.J., Livak, K.J., Mikkelsen, T.S., Rinn, J.L., 2014. The dynamics and regulators of cell fate decisions are revealed by pseudotemporal ordering of single cells. Nat Biotechnol 32, 381–386. https://doi.org/10.1038/nbt.2859
Trapnell, C., Roberts, A., Goff, L., Pertea, G., Kim, D., Kelley, D.R., Pimentel, H., Salzberg, S.L., Rinn, J.L., Pachter, L., 2012. Differential gene and transcript expression analysis of RNA-seq experiments with TopHat and Cufflinks. Nat Protoc 7, 562–578. https://doi.org/10.1038/nprot.2012.016
Trapnell, C., Williams, B.A., Pertea, G., Mortazavi, A., Kwan, G., Van Baren, M.J., Salzberg, S.L., Wold, B.J., Pachter, L., 2010. Transcript assembly and quantification by RNA-Seq reveals unannotated transcripts and isoform switching during cell differentiation. Nat Biotechnol 28, 511–515. https://doi.org/10.1038/nbt.1621
Van De Geijn, B., McVicker, G., Gilad, Y., Pritchard, J.K., 2015. WASP: allele-specific software for robust molecular quantitative trait locus discovery. Nat Methods 12, 1061–1063. https://doi.org/10.1038/nmeth.3582
Virshup, I., Rybakov, S., Theis, F.J., Angerer, P., Wolf, F.A., 2021. anndata: Annotated data. https://doi.org/10.1101/2021.12.16.473007
Wang, A.M., Doyle, M.V., Mark, D.F., 1989. Quantitation of mRNA by the polymerase chain reaction. Proc. Natl. Acad. Sci. U.S.A. 86, 9717–9721. https://doi.org/10.1073/pnas.86.24.9717

Wick, R.R., Schultz, M.B., Zobel, J., Holt, K.E., 2015. Bandage: interactive visualization of *de novo* genome assemblies. Bioinformatics 31, 3350–3352. https://doi.org/10.1093/bioinformatics/btv383

Wold, B., Myers, R.M., 2008. Sequence census methods for functional genomics. Nat Methods 5, 19–21. https://doi.org/10.1038/nmeth1157

Wolf, F.A., Angerer, P., Theis, F.J., 2018. SCANPY: large-scale single-cell gene expression data analysis. Genome Biol 19, 15. https://doi.org/10.1186/s13059-017-1382-0

Xu, C., Prete, M., Webb, S., Jardine, L., Stewart, B.J., Hoo, R., He, P., Meyer, K.B., Teichmann, S.A., 2023. Automatic cell-type harmonization and integration across Human Cell Atlas datasets. Cell 186, 5876-5891.e20. https://doi.org/10.1016/j.cell.2023.11.026

Zeng, H., 2022. What is a cell type and how to define it? Cell 185, 2739–2755. https://doi.org/10.1016/j.cell.2022.06.031

Zhang, H., Jain, C., Aluru, S., 2020. A comprehensive evaluation of long read error correction methods. BMC Genomics 21, 889. https://doi.org/10.1186/s12864-020-07227-0

Zheng, G.X.Y., Terry, J.M., Belgrader, P., Ryvkin, P., Bent, Z.W., Wilson, R., Ziraldo, S.B., Wheeler, T.D., McDermott, G.P., Zhu, J., Gregory, M.T., Shuga, J., Montesclaros, L., Underwood, J.G., Masquelier, D.A., Nishimura, S.Y., Schnall-Levin, M., Wyatt, P.W., Hindson, C.M., Bharadwaj, R., Wong, A., Ness, K.D., Beppu, L.W., Deeg, H.J., McFarland, C., Loeb, K.R., Valente, W.J., Ericson, N.G., Stevens, E.A., Radich, J.P., Mikkelsen, T.S., Hindson, B.J., Bielas, J.H., 2017. Massively parallel digital transcriptional profiling of single cells. Nat Commun 8, 14049. https://doi.org/10.1038/ncomms14049