

ABSTRACT

Title of Dissertation: **METHODS FOR EFFICIENT PROCESSING
AND UNCERTAINTY-AWARE ANALYSIS OF
HIGH THROUGHPUT GENOMICS AND
TRANSCRIPTOMICS DATA**
Noor Pratap Singh
Doctor of Philosophy, 2025

Dissertation Directed by: **Dr. Rob Patro, Associate Professor**
Department of Computer Science

The advent of modern high-throughput sequencing technologies has empowered researchers to rapidly and cost-effectively sequence large amounts of DNA/RNA fragments from biological samples. A crucial step following sequencing is the estimation of feature abundances, which requires mapping/aligning reads to a reference that far surpasses the length of the individual reads. We thus require efficient algorithms for read mapping, especially considering that many datasets comprise millions of reads. However, even with efficient mapping, overlapping sequences among features introduce uncertainty in determining the true locus of origin for a read. This dissertation seeks to address these challenges in the context of bulk RNA-Seq and single-cell ATAC-Seq data, with applications in transcriptomics and genomics.

First, we introduce TreeTerminus, a tree-based framework designed to incorporate uncertainty in abundance estimates for RNA-seq data. TreeTerminus constructs hierarchical trees from the samples in an RNA-Seq experiment, where the leaf nodes represent the individual transcripts, and

the inner nodes represent aggregated transcript groups. The uncertainty decreases as one ascends the tree. The tree provides the flexibility to analyze data at nodes that are at different levels of resolution in the tree, adjustable based on the analysis of interest.

Next, we present mehenDi, a method for uncertainty-aware differential analysis that leverages the tree generated from TreeTerminus. mehenDi identifies differential candidate nodes that can include both transcripts and inner nodes, maximizing signal detection while controlling for uncertainty in inference in a data-driven manner. Applying mehenDi to both simulated and experimental datasets, we discovered inner nodes with a strong differential signal that would have been overlooked when analyzing the individual transcripts alone.

Finally, we introduce alevin-fry-atac, a method for processing and mapping single-cell ATAC-Seq data. We propose a novel pseudoalignment algorithm and a caching scheme that enables fast and memory-efficient read mapping to the genome utilizing the piscem index. Alevin-fry-atac is currently three times faster while using three times less memory compared to Chromap - the only fully open-source alternative—which itself was significantly faster than Cell-Ranger ATAC, a method developed by 10X Genomics. With the introduction of alevin-fry-atac, we establish the first fully open-source ecosystem capable of processing both single-cell RNA-Seq and ATAC-Seq data, facilitating seamless multi-omic analysis.

METHODS FOR EFFICIENT PROCESSING AND UNCERTAINTY-AWARE
ANALYSIS OF HIGH-THROUGHPUT GENOMICs AND
TRANSCRIPTOMICs DATA

by

Noor Pratap Singh

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfilment
of the requirements for the degree of
Doctor of Philosophy
2025

Advisory Committee:

Dr. Rob Patro, Chair/Advisor
Dr. Steve Mount, Dean's Representative
Dr. Mihai Pop
Dr. Erin Molloy
Dr. Heng Huang

© Copyright by
Noor Pratap Singh
2025

Preface

The algorithms, tools, methods, and analysis presented in this thesis have been published in peer-reviewed journals, published as preprints, or are currently under revision/review. I am extremely thankful to all my collaborators and co-authors over the years on all of my projects.

- **Chapter 2** - Noor Pratap Singh, Michael I.Love, and Rob Patro. "TreeTerminus—creating transcript trees using inferential replicate counts." *Iscience* 26.6 (2023). (RECOMB-Seq 2023). <https://doi.org/10.1016/j.isci.2023.106961>
- **Chapter 3** - Noor Pratap Singh, Euphy Y.u, Jason Fan, Michael I.Love, and Rob Patro. "Tree-based differential testing using inferential uncertainty for RNA-Seq." *bioRxiv* (2023) (Under Revision). <https://doi.org/10.1101/2023.12.25.573288>
- **Chapter 4** - Noor Pratap Singh, Jamshed Khan, and Rob Patro. "Alevin-fry-atac enables rapid and memory frugal mapping of single-cell ATAC-seq data using virtual colors for accurate genomic pseudoalignment." *bioRxiv* (2024): 2024-11. (Under Review). <https://doi.org/10.1101/2024.11.27.625771>

In addition to the papers described in the thesis, I have also contributed to the following papers,

- Euphy Y.Wu, Noor Pratap Singh, Kwangborn Choi, Mohsen Zakeri, Matthew Vincent, Gary A. Churchill, Cheryl L.Ackert-Bicknell, Rob Patro and Michael I.Love "SEESAW: detecting isoform-level allelic imbalance accounting for inferential uncertainty." *Genome biology* 24.1 (2023): 165. <https://doi.org/10.1186/s13059-023-03003-x>

- Jason Fan, Noor Pratap Singh, Jamshed Khan, Giulio Ermanno Pibiri, and Rob Patro. "Fulgor: a fast and compact k-mer index for large-scale matching and color queries." *Algorithms for Molecular Biology* 19.1 (2024): 3. (WABI 2023). <https://doi.org/10.1186/s13015-024-00251-9>
- Zare Z.Jousheghani, Noor Pratap Singh, and Rob Patro. "A replicable and modular benchmark for long-read transcript quantification methods". *bioRxiv* (2024): 2024-07. <https://doi.org/10.1101/2024.07.30.605821>
- Zare Z.Jousheghani, Noor Pratap Singh, and Rob Patro. "Oarfish: Enhanced probabilistic modeling leads to improved accuracy in long read transcriptome quantification." *bioRxiv* (2024). <https://doi.org/10.1101/2024.02.28.582591>

Dedication

To Mamu and Nanaji....

Acknowledgments

Waheguru has been incredibly kind.

There are countless people who have supported me through different stages of life, and without them, this thesis would not have been possible. While it is impossible to fully capture my gratitude in words, I will do my best. My journey through graduate school has been an experience I will cherish for a lifetime, largely because of the incredible people I had the fortune to interact with and learn from.

First and foremost, I would like to express my deepest gratitude to my advisor, Dr. Rob Patro. He took me into his lab at the end of my first Ph.D. year and gave me the invaluable opportunity to work on intellectually challenging and exciting projects. Our regular one-on-one meetings were a constant source of motivation and enthusiasm, leaving me inspired every time. His mentorship has shaped my scientific thinking and enabled me to tackle problem domains that once seemed unthinkable. I will always be grateful for his guidance and the impact he has had on my academic journey.

I would also like to extend my heartfelt thanks to Dr. Michael Love, with whom I had the privilege of closely collaborating during my Ph.D. His expertise in statistics and sequencing data is unmatched, and without his insights, this thesis would have remained a distant dream. I am also deeply grateful to my thesis committee members, Dr. Mihai Pop, Dr. Erin Molloy, Dr. Heng Huang, and Dr. Steve Mount, for taking the time to review my work and provide invaluable

feedback.

Being part of CBCB has been an incredibly rewarding experience. The camaraderie among fellow graduate students created a warm and welcoming environment from the very beginning. Many of them have become close friends, offering support in ways big and small—whether it was manuscript feedback, code reviews, insightful discussions on sequencing technologies, algorithms, or biology, or simply sharing programming hacks. Beyond academics, I will always cherish the hikes, happy hours, and impromptu conversations that made this journey even more special. Thank you, Jamshed, Dongze, Hari, Yuelin, Jason, MG, Naky, Nathalie, Zinat, Zahra, Mohamed, Nidhi, and Kiran, for being part of this incredible experience.

I was also fortunate to build lasting friendships with fellow graduate students across UMD, who provided unwavering emotional support through this journey. Thank you, Vasu, Vaishnavi, Komal, Sharmila, Samya, Sneha, Sai, Shishira, Neha, Naman, Pulkit, Pooja, Usama, Manas, Adesh, and Sushmija, for always being there.

Above all, I owe my deepest gratitude to my family—my mother, uncle, grandmother, and my late grandfather—who have always stood by me, encouraged me, and guided me through every stage of my life. Their unwavering belief in me has been my greatest source of strength. Losing my grandfather during COVID was an immense loss, but I hope he is proud of me today.

There are so many others who have played a role in my journey, and while I cannot name everyone, I am deeply thankful to all who have supported me along the way.

Table of Contents

Preface	iv
Dedication	iv
Acknowledgements	v
Table of Contents	vii
List of Tables	x
List of Figures	xiii
Chapter 1: Introduction	1
1.1 Preliminaries for Chapter 2 and 3	7
1.1.1 RNA-seq basics	7
1.1.2 Equivalence classes	8
1.1.3 Terminus	8
1.2 Preliminaries for Chapter 4	10
1.2.1 Basics of ATAC-Seq and the 10x ATAC-Seq protocol	10
1.2.2 de Bruijn graph	12
1.2.3 Pseudoalignment	12
Chapter 2: TreeTerminus — Creating transcript trees using inferential replicate counts	14
2.1 Introduction	14
2.2 Methods	17
2.2.1 Overview of TreeTerminus	17
2.2.2 Mean Tree	18
2.2.3 Consensus tree	18
2.2.4 Creating a unified tree from the output of TreeTerminus	22
2.2.5 Solving objective functions to obtain discrete inferential units	22
2.2.6 Datasets	27
2.2.7 Construction of anti-correlation tree	28
2.2.8 Quantification pipeline	29
2.2.9 Mapping to gene families	29
2.2.10 Differential expression analysis	30
2.3 Results	31
2.3.1 Overview of TreeTerminus	31

2.3.2	Experimental Setup	33
2.3.3	TreeTerminus nodes have the lowest mean inferential variance	35
2.3.4	TreeTerminus nodes map to relatively fewer genes and gene families	36
2.3.5	Cuts obtained from TreeTerminus trees are among the top performers when optimizing for the different objective functions	38
2.3.6	An optimal performance is observed for differential expression analysis on the cuts obtained from TreeTerminus trees	42
2.4	Discussion	43
Chapter 3:	Tree-based differential testing using inferential uncertainty for RNA-seq	46
3.1	Introduction	46
3.2	Methods	51
3.2.1	Overview	51
3.2.2	Problem Statement	52
3.2.3	mehenDi description	54
3.2.4	Experimental Setup	57
3.3	Results	60
3.3.1	Running mehenDi on null simulation	60
3.3.2	Comparing the methods on simulated datasets with signal	61
3.3.3	Analyzing the MouseMuscle dataset	68
3.3.4	Analyzing the ChimpBrain dataset	71
3.4	Discussion	73
Chapter 4:	Alevin-fry-atac enables rapid and memory frugal mapping of single- cell ATAC-seq data using virtual colors for accurate genomic pseudoalignment	77
4.1	Introduction	77
4.1.1	Contributions	80
4.2	Material and Methods	81
4.2.1	Preliminaries	81
4.2.2	Mapping and Pseudoalignment using virtual colors	82
4.2.3	Speeding up k -mer query	87
4.2.4	Alevin-fry-atac pipeline	90
4.2.5	Dataset and the analysis pipeline	91
4.3	Results	92
4.3.1	Mapping accuracy on the simulated data	92
4.3.2	Experimental datasets	94
4.3.3	Memory and Running Time	97
4.4	Discussion	99
Chapter 5:	Conclusion and Future Directions	101
Appendix A:	Supplementary Material for "TreeTerminus"	106
A.1	Supplementary material for Chapter 2	106
A.1.1	Tables	106
A.1.2	Figures	110

Appendix B: Supplementary Material for "Tree-based differential testing using inferential uncertainty for RNA-Seq"	116
B.1 Methods	116
B.1.1 Median-ratio scaled counts for the nodes in the tree	116
B.1.2 Distance between nodes	118
B.2 Tables	119
B.3 Figures	124
Appendix C: Supplementary Material for "Alevin-fry-atac enables rapid and memory frugal mapping of single-cell ATAC-seq data using virtual colors for accurate genomic pseudoalignment"	146
C.1 Methods	146
C.1.1 Mapping and Pseudoalignment	146
C.1.2 Analysis pipeline	147
C.2 Results	149
C.2.1 Mapping accuracy for alevin-fry-atac	149
C.3 Tables	150
C.4 Figures	151
Bibliography	169

List of Tables

2.1	Peak memory usage and running time for the different steps of Terminus and TreeTerminus (both Mean and Cons modes) on BrSimNorm dataset.	33
2.2	True Positive Rate and False Discovery Rate for the different methods at nominal FDR cutoffs 0.01, 0.05, 0.10 for the BrSimLow Dataset. The cuts are obtained by optimizing the metric <code>irv_height_desc</code> at different γ values and <code>lfc_desc</code> on the Cons tree. The performance is also computed when the inferential units consist of genes, transcripts and terminus groups.	43
3.1	Number of <code>mehenDi</code> nodes that map to a significant gene but the gene doesn't consist of any significant transcripts for the <code>MouseMuscle</code> and the <code>ChimpBrain</code> dataset respectively.	71
4.1	Mapping accuracy on the simulated data for the different methods	94
4.2	Percentage overlap between the peaks pairwise for the different methods across the datasets. Depending on which method is used as a denominator, percentage overlap can change. The values in the parenthesis denote the overlap when the second method was used as the denominator. N/A entries denote that <code>Cell Ranger ATAC</code> was not run on that dataset.	96
4.3	Evaluating the concordance between the clusters obtained for the different methods pairwise across the datasets. The two entries per cell separated by comma denote Adjusted Rand Index (ARI) and Normalized Mutual Information (NMI). N/A entries denote that <code>Cell Ranger ATAC</code> was not run on that dataset.	97
4.4	caption for per	98
A1	Accession IDs for the tissue samples for the <code>MouseMuscle</code> dataset that have been used for analysis in this paper	106
A2	Peak memory usage and running time for the different steps of Terminus and TreeTerminus (both Mean and Cons modes) on <code>MouseMuscle</code> dataset.	107
A3	Total number of transcripts covered by different methods across the datasets.	107
A4	Median of mean inferential variance (MIRV) of the inner nodes for different trees stratified by their height for the BrSimNorm dataset. All nodes with height larger than 5 have been labelled as 5.	107

A5	Median of mean inferential variance (MIRV) of the inner nodes for different trees stratified by their height for the <code>BrSimLow</code> dataset. All nodes with height larger than 5 have been labelled as 5.	108
A6	Median of mean inferential variance (MIRV) of the inner nodes for different trees stratified by their height for the <code>MouseMuscle</code> dataset. All nodes with height larger than 5 have been labelled as 5.	108
A7	Median of mean inferential variance (MIRV) of the inner nodes for different trees stratified by their height for the <code>ChimpBrain</code> dataset. All nodes with height larger than 5 have been labelled as 5	108
A8	Total number of inner nodes mapping to more than 100 genes for different datasets.	109
A9	True Positive Rate and False Discovery Rate for the different methods at nominal FDR cutoffs 0.01, 0.05, 0.10 for the <code>BrSimNorm</code> Dataset. The cuts are obtained by optimizing the metric <code>irv_height_desc</code> at different γ values and <code>lfc_desc</code> on the <code>Cons</code> tree. The performance is also computed when the inferential units consist of genes, transcripts and terminus groups.	109
B1	Distribution of the height of the selected nodes output by <code>mehenDi</code> and candidate nodes by <code>treeclimbR</code> for the <code>BrSimNorm</code> dataset.	119
B2	Proportion of unique nodes that are observed for <code>mehenDi</code> , <code>treeclimbR</code> when doing a pairwise comparison for the <code>BrSimNorm</code> dataset at different nominal FDR. The number inside the parentheses represents the total number of nodes. . .	119
B3	Proportion of unique nodes that are observed for <code>Txps</code> , <code>treeclimbR</code> when doing a pairwise comparison for the <code>BrSimNorm</code> dataset at different nominal FDR. The number inside the parentheses represents the total number of nodes. . .	120
B4	Number of differentially expressed genes obtained at the different nominal thresholds for the <code>MouseMuscle</code> dataset between EDL and MAST.	120
B5	Number of differentially expressed transcripts obtained at the different nominal FDR thresholds for the <code>MouseMuscle</code> dataset between EDL and MAST.	120
B6	Distribution of the height of the <code>mehenDi</code> nodes for the <code>MouseMuscle</code> dataset.	120
B7	Distribution of the total number of unique genes, the transcripts covered by <code>mehenDi</code> nodes map to for the <code>MouseMuscle</code> dataset.	121
B8	Number of differentially expressed genes (DEGs) that don't contain a single underlying transcript that is called significant at the different nominal FDR thresholds for the <code>MouseMuscle</code> dataset.	121
B9	Number of differentially expressed transcripts (DETs) that don't map to a single significant gene at the different nominal FDR thresholds for the <code>MouseMuscle</code> dataset.	121
B10	Number of <code>mehenDi</code> nodes that neither map to a significant gene nor contain a single significant transcript for the <code>MouseMuscle</code> dataset.	121
B11	Number of differentially expressed genes (DEGs) at the different nominal FDR thresholds for the <code>ChimpBrain</code> dataset.	121
B12	Number of differentially expressed transcripts (DETs) at the different nominal FDR thresholds for the <code>ChimpBrain</code> dataset.	122
B13	Distribution of the height of the <code>mehenDi</code> nodes for the <code>ChimpBrain</code> dataset. .	122

B14	Distribution of the total number of unique genes, the transcripts covered by <code>mehenDi</code> nodes map to for the <code>ChimpBrain</code> dataset.	122
B15	Number of differentially expressed genes (DEGs) that don't contain a single underlying transcript that is called significant at the different nominal FDR thresholds for the <code>ChimpBrain</code> dataset.	122
B16	Number of differentially expressed transcripts (DETs) that don't map to significant genes for the <code>ChimpBrain</code> dataset.	122
B17	Number of <code>mehenDi</code> nodes that neither map to a significant gene nor contain a single significant transcript for the <code>ChimpBrain</code> dataset.	123
C1	URLs for the datasets used in the manuscript for analysis.	150
C2	The total number of bases (MB) covered by the different methods across the different datasets. N/A entries denotes that <code>Cell Ranger ATAC</code> was not run on that dataset. <code>Cell Ranger ATAC (MACS2)</code> and <code>Cell Ranger ATAC (Custom)</code> denote that peaks were generated on the <code>Cell Ranger ATAC</code> fragments using <code>MACS2</code> and the inbuilt peak caller of <code>Cell Ranger ATAC</code> respectively. .	150

List of Figures

1.1	Main steps in the ATAC-Seq protocol adapted from [1]	11
1.2	Summary of the 10X protocol, the figure is adapted from 10X Website(V2)	12
2.1	Example of a sample group to demonstrate why consensus step cannot be directly applied on the trees obtained from <code>group</code> step for samples and the modifications made in order to apply the consensus tree algorithm. a Trees across the 5 different samples that contain overlapping transcripts. Each tree is labeled as T_{gm} , where m denotes the sample and g denotes a group in that sample. b Updated group that is a superset of all the transcripts in the individual trees across samples. c Tree created for each sample w.r.t updated group g' , with $\Lambda(T'_g) = \Lambda(g')$	20
2.2	Demonstration of creating a unified tree from the forest that is obtained as the output of TreeTerminus. The unified tree can then be used for optimizing an objective function of interest in order to obtain a cut for downstream analysis. . .	26
2.3	a) Schematic overview of TreeTerminus pipeline. Taking an RNA-Seq experiment as an input, that has been quantified with Salmon and for which inferential replicates have been generated, TreeTerminus outputs a forest of trees. b) Toy example demonstrating how from the transcript-graph (G), tree $B(G)$ shall be generated by TreeTerminus for a single sample. A forest will be the output when the graph consists of disconnected components or when some of the necessary conditions that are required for aggregating transcripts are not met. The colour red indicates the edge with the lowest weight which is chosen for aggregation at each iteration. c) When there are multiple samples - TreeTerminus provides two modes to generate the forest. In the Consensus mode, trees corresponding to each transcript group are first extracted and then passed as input into a majority rule extended consensus algorithm. In the toy example, there is only 1 transcript group with the forest consisting of a single tree. For the Mean mode, the graph $M(G)$ is constructed, where the edge weight is given by the mean of $s(t_i, t_j)$ across all samples. The forest is then obtained by applying the procedure B on $M(G)$ as demonstrated in part b of the figure.	32
2.4	Distribution of $\log_2$ MIRV(mean inferential relative variance) across the inner nodes stratified by their height for different trees for the BrSimNorm Dataset, with the total number of inner nodes belonging to a method at a given height written on top of the violin plot. Also plotted for comparison at each height is the distribution of $\lg$ of MIRV for the transcripts and genes.	36

2.5	Comparison of different methods with respect to the number of genes to which an inner node in the tree maps for the <code>BrSimNorm</code> dataset, stratified by their height. The x-axis represents the number of unique genes that transcripts belonging to the inner nodes map to and y-axis represents the frequency of such mappings at a given height for a tree. For all the inner nodes located at a height greater than or equal to 5, the number of unique genes was binned using the set $\{1, 2, 4, 16, 128, 1024, 16384\}$, with the bin representing a number of unique genes less than or equal to the bin but larger than the bin left to it.	37
2.6	Comparison of different tree methods with respect to the number of gene families to which an inner node maps. The number of gene families that map to an inner node has been binned into the set $\{1, 2, 4, 10, 100, 500, 1000, 12000\}$, with the bin representing the number of unique gene families less than or equal to the bin but larger than the bin left to it. The x-axis represents the bin and the y-axis represents the total frequency of inner nodes mapping to the gene families in that bin for a given tree. This has been plotted for A) <code>MouseMuscle Dataset</code> and B) <code>ChimpBrain Dataset</code>	39
2.7	Barplot representing the values for the objective function on the cut obtained by minimizing the sum of metric - the sum of the metric (<code>irv_height_desc</code>) on the different datasets. For each dataset, optimal values of the objective function are compared across the methods for a range of γ values. Also plotted for comparison is the optimal value obtained by using transcripts(<code>Txp</code>) as the cut.	40
2.8	Comparing the performance on the cuts across different trees obtained by solving the objective function that maximizes the sum of metric CPM log fold change multiplied by the number of descendant leaves (<code>lfc_desc</code>) for the nodes in a cut on the different datasets. The performance is also compared when the transcripts and <code>Terminus</code> groups are taken as the cut. A) Value of the Objective Function. B) Size of the cut.	41
3.1	Schematic overview of <code>mehenDi</code> . A) Given a set of RNA-seq samples, <code>Tree Terminus</code> creates a tree that encodes the uncertainty structure by using the Gibbs/Bootstrap samples generated by Salmon to compute uncertainty. B) Given a tree, <code>mehenDi</code> outputs a set of selected nodes. The selected are differentially expressed between the conditions of interest and can consist of transcripts and inner nodes, with n. Using an example, with a tree having 12 leaves to demonstrate <code>mehenDi</code> . A highlighted node implies that it is selected by <code>mehenDi</code> , with a red outlined node indicating that the node has a significant p-value. The $+$, $-$ on a node indicates the direction of sign change between the conditions, whereas $+-$ indicates we are unsure about the direction. The values plotted along the nodes represent their mean inferential relative variance across the samples (<code>meanInfRV</code>). The <code>meanInfRV</code> threshold in this example is 0.4. The nodes N1 and N2 are chosen instead of node N18 since the <code>meanInfRV</code> of one of its children, node N2, is below the threshold of 0.4. The node N5 is chosen instead of the node N19 since the children have opposite sign changes. Node N20 is the only significant node in its branch and meets all the required criteria. Node N21 is picked since it has at least one non-significant child node.	53

3.2	A) Distribution of p-values and test-statistic for Swish for the null brain simulation. i) Distribution of the p-values on the entire node set over the tree when all hypotheses are tested together using Swish. ii) Distribution of the p-values over the leaf nodes. iii) Distribution of the p-values over the inner nodes. iv) Distribution of the test-statistic on the inner and leaf nodes, which is used by Swish to generate the p-values. B) False Positive Rate (FPR) obtained for the null simulations by varying the p-value threshold. The bar along the y-axis represents the variation observed across the 10 simulations. C) True Positive Rates and Empirical False Discovery Rates at the different nominal FDR thresholds across the different methods and entities for the <code>BrSimNorm</code> and <code>BrSimLow</code> Dataset. Both the metrics have been rounded to 3 decimal places.	62
3.3	A) Example of a <code>mehenDi</code> node that is overaggregated in <code>Terminus</code> for the <code>BrSimNorm</code> dataset. i) Subtree representing the transcripts covered by the <code>Terminus</code> group. ii) Inferential replicates for the <code>Terminus</code> group. iii) Inferential replicates for the selected node output by <code>mehenDi</code> . B) Example of a <code>mehenDi</code> node that is not aggregated enough in <code>Terminus</code> for the <code>BrSimNorm</code> dataset. i) Subtree representing the transcripts covered by the <code>mehenDi</code> group. ii) Inferential replicates for the <code>Terminus</code> group. iii) Inferential replicates for the selected node output by <code>mehenDi</code>	66
3.4	Examination of error metrics for the unique nodes obtained for <code>mehenDi</code> when doing <code>mehenDi</code> vs <code>treeclimbR</code> analysis at the different nominal FDR thresholds for the <code>BrSimNorm</code> dataset. We vary the magnitude of log fold change (LFC) and plot the empirical FDR and the total number of nodes that are left after filtering the unique nodes based on LFC.	68
3.5	Examining the transcript profile for the gene <i>Prrgl</i> (A) and <i>Tec</i> (B) in the <code>MouseMuscle</code> dataset. i) Transcripts in a pileup style. ii) Tree representing the transcripts covered by the gene, with the red node representing the transcripts covered by the <code>mehenDi</code> selected node. iii) Inferential replicates for the transcript which had the lowest p-value among all the transcripts in the tree. iv) Inferential replicates for the <code>mehenDi</code> selected node.	71
3.6	Examining the transcript profile for the gene <i>DNAH9</i> and <i>EYA4</i> in the <code>ChimpBrain</code> dataset. i) Transcripts in a pileup style. ii) Tree representing the transcripts covered by the gene, with the red node representing the transcripts covered by the <code>mehenDi</code> selected node. iii) Inferential replicates for the transcript ENSMUST00000104666, which had the lowest p-value among all the transcripts in the tree. iv) Inferential replicates for the <code>mehenDi</code> selected node.	73

4.1	Toy example demonstrating how traditional pseudoalignment on genomic references could lead to spurious mapping or completely missing the correct mapping. In the example, we have two k -mers, which map to multiple references, with k -mer 1 mapping to multiple locations within chromosome 1. The individual mapped positions within a reference are denoted by p_{ij} , with i denoting the k -mer id and j denoting the order of the positions globally for that k -mer. PSA(Read) denotes the reference ids and the relative positions on the reference along which Read maps using pseudoalignment. The true mapping for the read in the example is (Chr1, p12). A) Using traditional pseudoalignment both Chr1 and Chr2 are selected as references from which the read originates. The position to which a read maps in pseudoalignment is governed by where the left-most k -mer of a read maps. Thus, for reference Chr2, p14 is picked as a position for the hit. If a k -mer maps to multiple positions within a reference, then by design the position with the smallest coordinate is picked, which for Chr1 is p11. We thus have two false positive mappings (Chr1, p11), (Chr2, p14) and a false negative since (Chr1, p11) is not reported. B) When the references are projected onto a set of virtual colors whose boundary spaces do not overlap, we miss the mapping if the read spans across the boundaries (b2, b3), creating an $\emptyset$ set when taking the intersection across the colors. C) When the boundaries of virtual colors overlap, each k -mer is assigned two colors if it falls within the overlapping region. This ensures that the read gets mapped after pseudoalignment.	83
4.2	Evaluating accuracy across different read lengths for <code>alevin-fry-atac</code> by varying A) ℓ_{vcol} for $k = 25, \tau = 0.7$. The x-axis label <i>chr</i> implies mapping is done using chromosome sequence as colors (removing the impact of virtual colors on mapping procedure).	93
4.3	Comparing the time taken and maximum resident memory consumed by <code>alevin-fry-atac</code> and <code>Chromap</code> for the Human 10K PBMC dataset across the different threads.	98
A1	Distribution of $\log_2$ MIRV (mean inferential variance) across samples for the inner nodes stratified by their height for different trees for the <code>BrSimLow</code> Dataset, with the total number of inner nodes belonging to a method at a given height written on top of the violin plot. Also plotted for comparison at each height is the distribution of MIRV for the transcripts and genes.	110
A2	Distribution of $\log_2$ MIRV (mean inferential variance) across the inner nodes stratified by their height for different trees for the <code>MouseMuscle</code> Dataset, with the total number of inner nodes belonging to a method at a given height written on top of the violin plot. Also plotted for comparison at each height is the distribution of $\lg$ of MIRV for the transcripts and genes.	111
A3	Distribution of $\log_2$ MIRV (mean inferential variance) across samples for the inner nodes stratified by their height for different trees for the <code>ChimpBrain</code> Dataset, with the total number of inner nodes belonging to a method at a given height written on top of the violin plot. Also plotted for comparison at each height is the distribution of MIRV for the transcripts and genes.	111

A4	Comparison of different tree methods with respect to the number of genes to which an inner node in the tree maps for the <code>BrSimLow</code> dataset stratified by their height. The x-axis represents the number of unique genes that transcripts belonging to the inner nodes map to and y-axis represents the frequency of such mappings at a given height for a tree. For all the inner nodes located at a height greater than or equal to 5, number of unique genes were binned using the set $\{1, 2, 4, 16, 128, 1024, 16384\}$, with the bin representing number of unique genes less than or equal to the bin but larger than the bin left to it.	112
A5	Comparison of different tree methods with respect to the number of genes an to which an inner node in the tree maps for the <code>MouseMuscle</code> dataset stratified by their height. The x-axis represents the number of unique genes that transcripts belonging to the inner nodes map to and y-axis represents the frequency of such mappings at a given height for a tree. For all the inner nodes located at a height greater than or equal to 5, number of unique genes were binned using the set $\{1, 2, 4, 16, 128, 1024, 16384\}$, with the bin representing number of unique genes less than or equal to the bin but larger than the bin left to it.	113
A6	Comparison of different tree methods with respect to the number of genes to which an inner node in the tree maps for the <code>ChimpBrain</code> dataset stratified by their height. The x-axis represents the number of unique genes that transcripts belonging to the inner nodes map to and y-axis represents the frequency of such mappings at a given height for a tree. For all the inner nodes located at a height greater than or equal to 5, number of unique genes were binned using the set $\{1, 2, 4, 16, 128, 1024, 16384\}$, with the bin representing number of unique genes less than or equal to the bin but larger than the bin left to it.	114
A7	Distribution of the size of cuts on the different datasets obtained after solving for the objective function that minimizes the sum of metric - weighted sum of mean inferential variance and height multiplied by the number of descendant leaves (<code>irv_height_desc</code>) for the nodes in a cut. For each method, the distribution is plotted for a range of γ values. Also plotted for comparison are the total number of transcripts/leaves <code>Txp</code>	115
A8	Distribution of the <code>lg</code> of the metric - weighted sum of mean inferential variance and height multiplied by the number of descendant leaves (<code>irv_height_desc</code>) for the nodes in the cut obtained after minimizing for the objective function using <code>irv_height_desc</code> as the underlying metric across trees on the different datasets. The metric has been computed using $\gamma = 0.1$. Also plotted for comparison is the distribution of <code>irv_height_desc</code> for transcripts.	115
B1	Distribution of the p-values for the leaf and inner nodes on the null simulations, when the hypothesis testing is carried out separately.	124
B2	UpSet plot covering the number of true positive transcripts that are covered by the nodes that are output by the different methods for the <code>BrSimNorm</code> dataset nominal FDR.	124
B3	UpSet plot covering the number of true positive transcripts that are covered by the nodes that are output by the different methods for the <code>BrSimLow</code> dataset at the 0.01 nominal FDR.	125

B4	True Positive Rates and Empirical False Discovery Rates at the different nominal FDR thresholds by individually varying the parameters <i>minP</i> and <i>mIrvThresh</i> for the BrSimNorm dataset. Both the metrics have been rounded to 3 decimal places	125
B5	Example of a mehenDi node that is overaggregated in Terminus for the BrSimNorm dataset. A) Subtree representing the transcripts covered by the Terminus group. B) Inferential replicates for the Terminus group. C) Inferential replicates for the selected node output by mehenDi.	126
B6	Example of a mehenDi node that is overaggregated in Terminus for the BrSimNorm dataset. A) Subtree representing the transcripts covered by the Terminus group. B) Inferential replicates for the Terminus group. C) Inferential replicates for the selected node output by mehenDi.	127
B7	Example of a mehenDi node that is not aggregated enough in Terminus for the BrSimNorm dataset. A) Subtree representing the transcripts covered by the mehenDi group. B) Inferential replicates for the Terminus group. C) Inferential replicates for the selected node output by mehenDi.	128
B8	Example of a mehenDi node that is not aggregated enough in Terminus for the BrSimNorm dataset. A) Subtree representing the transcripts covered by the mehenDi group. B) Inferential replicates for the Terminus group. C) Inferential replicates for the selected node output by mehenDi.	129
B9	Examination of error metrics for the unique nodes obtained for treeclimbR when doing treeclimbR vs Txps analysis at the different nominal FDR thresholds. We vary the magnitude of log fold change (LFC) and plot the empirical FDR and the total number of nodes that are left after filtering the unique nodes based on LFC.	130
B10	Examination of error metrics for the unique nodes obtained for Txps when doing treeclimbR vs Txps analysis at the different nominal FDR thresholds. We vary the magnitude of log fold change (LFC) and plot the empirical FDR and the total number of nodes that are left after filtering the unique nodes based on LFC. .	131
B11	The first two dimensions of the PCA using the top 1000 variable features for the MouseMuscle dataset	132
B12	The average distance between the nodes obtained for mehenDi using default parameters and varying <i>minP</i> and <i>mIrvThresh</i> individually for the MouseMuscle dataset.	133
B13	UpSet plot covering the number of transcripts that are covered by the nodes that are output by the different methods for the MouseMuscle dataset.	133
B14	Examining the transcript profile for the gene <i>Hmcn2</i> in the MouseMuscle dataset. A) Transcripts in a pileup style. B) Tree representing the transcripts covered by the gene <i>Hmcn2</i> , with the red node representing the transcripts covered by the mehenDi selected node. C) Inferential replicates for the transcript ENSMUST00000138821.7, which had the lowest p-value among all the transcripts in the tree. D) Inferential replicates for the mehenDi selected node.	134

B15	Examining the transcript profile for the gene <i>Emid1</i> in the <code>MouseMuscle</code> dataset. A) Transcripts in a pileup style. B) Tree representing the transcripts covered by the gene <i>Emid1</i> , with the red node representing the transcripts covered by the <code>mehenDi</code> selected node. C) Inferential replicates for the transcript ENSMUST00000151906.7, which had the lowest p-value among all the transcripts in the tree. D) Inferential replicates for the <code>mehenDi</code> selected node.	135
B16	Examining the transcript profile for the gene <i>Prss55</i> in the <code>MouseMuscle</code> dataset. A) Transcripts in a pileup style. B) Tree representing the transcripts covered by the gene <i>Prss55</i> , with the red node representing the transcripts covered by the <code>mehenDi</code> selected node. C) Inferential replicates for the transcript ENSMUST00000171503.7, which had the lowest p-value among all the transcripts in the tree. D) Inferential replicates for the <code>mehenDi</code> selected node.	136
B17	Examining the transcript profile for the gene <i>Stard10</i> in the <code>MouseMuscle</code> dataset. A) Transcripts in a pileup style. B) Tree representing the transcripts covered by the gene <i>Stard10</i> , with the red node representing the transcripts covered by the <code>mehenDi</code> selected node. C) Inferential replicates for the transcript ENSMUST000000032927.13 which had the lowest p-value among all the transcripts in the tree. D) Inferential replicates for the <code>mehenDi</code> selected node.	137
B18	Examining the transcript profile for the gene <i>Syk</i> in the <code>MouseMuscle</code> dataset. A) Transcripts in a pileup style. B) Tree representing the transcripts covered by the gene <i>SYK</i> , with the red node representing the transcripts covered by the <code>mehenDi</code> selected node. C) Inferential replicates for the transcript ENSMUST00000055087.6, which had the lowest p-value among all the transcripts in the tree. D) Inferential replicates for the <code>mehenDi</code> selected node.	138
B19	UpSet plot covering the number of transcripts that are covered by the nodes that are output by the different methods for the <code>ChimpBrain</code> dataset.	139
B20	Examining the transcript profile for the gene <i>CABIN1</i> in the <code>ChimpBrain</code> dataset. A) Transcripts in a pileup style. B) Tree representing the transcripts covered by the gene <i>CABIN1</i> , with the red node representing the transcripts covered by the <code>mehenDi</code> selected node. C) Inferential replicates for the transcript ENSMUST00000103768, which had the lowest p-value among all the transcripts in the tree. D) Inferential replicates for the <code>mehenDi</code> selected node.	140
B21	Examining the transcript profile for the gene <i>CATSPERG</i> in the <code>ChimpBrain</code> dataset. A) Transcripts in a pileup style. B) Tree representing the transcripts covered by the gene <i>CATSPERG</i> , with the red node representing the transcripts covered by the <code>mehenDi</code> selected node. C) Inferential replicates for the transcript ENSMUST00000020240, which had the lowest p-value among all the transcripts in the tree. D) Inferential replicates for the <code>mehenDi</code> selected node.	141
B22	Examining the transcript profile for the gene <i>EYAI</i> in the <code>ChimpBrain</code> dataset. A) Transcripts in a pileup style. B) Tree representing the transcripts covered by the gene <i>EYAI</i> , with the red node representing the transcripts covered by the <code>mehenDi</code> selected node. C) Inferential replicates for the transcript ENSMUST00000097751, which had the lowest p-value among all the transcripts in the tree. D) Inferential replicates for the <code>mehenDi</code> selected node.	142

B23	Examining the transcript profile for the gene <i>MCF2</i> in the ChimpBrain dataset. A) Transcripts in a pileup style. B) Tree representing the transcripts covered by the gene <i>MCF2</i> , with the red node representing the transcripts covered by the <i>mehenDi</i> selected node. C) Inferential replicates for the transcript ENSMUST00000105901, which had the lowest p-value among all the transcripts in the tree. D) Inferential replicates for the <i>mehenDi</i> selected node.	143
B24	Examining the transcript profile for the gene <i>MYO5C</i> in the ChimpBrain dataset. A) Transcripts in a pileup style. B) Tree representing the transcripts covered by the gene <i>MYO5C</i> , with the red node representing the transcripts covered by the <i>mehenDi</i> selected node. C) Inferential replicates for the transcript ENSMUST0000013074, which had the lowest p-value among all the transcripts in the tree. D) Inferential replicates for the <i>mehenDi</i> selected node.	144
B25	Examining the transcript profile for the gene <i>PIEZO2</i> in the ChimpBrain dataset. A) Transcripts in a pileup style. B) Tree representing the transcripts covered by the gene <i>PIEZO2</i> , with the red node representing the transcripts covered by the <i>mehenDi</i> selected node. C) Inferential replicates for the transcript ENSMUST00000094046, which had the lowest p-value among all the transcripts in the tree. D) Inferential replicates for the <i>mehenDi</i> selected node.	145
C1	Evaluating accuracy across different read lengths for <i>alevin-fry-atac</i> by varying ℓ_{vcol} at $k = 23$, with panels representing different τ values. The label <i>chr</i> on the x-axis implies mapping is done using chromosome sequence as colors (removing the impact of virtual colors on mapping procedure).	151
C2	Evaluating accuracy across different read lengths for <i>alevin-fry-atac</i> by varying ℓ_{vcol} at $k = 25$, with panels representing different τ values. The label <i>chr</i> on the x-axis implies mapping is done using chromosome sequence as colors (removing the impact of virtual colors on mapping procedure).	152
C3	Evaluating accuracy across different read lengths for <i>alevin-fry-atac</i> by varying ℓ_{vcol} for $k = 27$ with panels representing different τ values. The label <i>chr</i> on the x-axis implies mapping is done using chromosome sequence as colors (removing the impact of virtual colors on mapping procedure).	153
C4	Evaluating accuracy across different read lengths for <i>alevin-fry-atac</i> by varying ℓ_{vcol} for $k = 29$ with panels representing different τ values. The label <i>chr</i> on the x-axis implies mapping is done using chromosome sequence as colors (removing the impact of virtual colors on mapping procedure).	154
C5	Evaluating accuracy across different read lengths for <i>alevin-fry-atac</i> by varying the pseudoalignment thresholds for $k = 23$ with panels representing different ℓ_{vcol} values. The panel with ℓ_{vcol} <i>lchr</i> implies mapping is done using chromosome sequence as colors (removing the impact of virtual colors on mapping procedure).	155
C6	Evaluating accuracy across different read lengths for <i>alevin-fry-atac</i> by varying the pseudoalignment thresholds for $k = 25$ with panels representing different ℓ_{vcol} values. The panel with ℓ_{vcol} <i>chr</i> implies mapping is done using chromosome sequence as colors (removing the impact of virtual colors on mapping procedure).	156

C7	Evaluating accuracy across different read lengths for <code>alevin-fry-atac</code> by varying the pseudoalignment thresholds for $k = 27$ with panels representing different ℓ_{vcol} values. The panel with $\ell_{\text{vcol}} \text{ chr}$ implies mapping is done using chromosome sequence as colors (removing the impact of virtual colors on mapping procedure).	157
C8	Evaluating accuracy across different read lengths for <code>alevin-fry-atac</code> by varying the pseudoalignment thresholds for $k = 29$ with panels representing different ℓ_{vcol} values. The panel with bin length chr implies mapping is done using chromosome sequence as colors (removing the impact of virtual colors on mapping procedure).	158
C9	Evaluating accuracy across different read lengths for <code>alevin-fry-atac</code> by varying the k -mer size for $\tau = 0.6$ with panels representing different ℓ_{vcol} values. The panel with $\ell_{\text{vcol}} \text{ chr}$ implies mapping is done using chromosome sequence as colors (removing the impact of virtual colors on mapping procedure).	159
C10	Evaluating accuracy across different read lengths for <code>alevin-fry-atac</code> by varying the k -mer size for $\tau = 0.7$ with panels representing different ℓ_{vcol} values. The panel with $\ell_{\text{vcol}} \text{ chr}$ implies mapping is done using chromosome sequence as colors (removing the impact of virtual colors on mapping procedure).	160
C11	Evaluating accuracy across different read lengths for <code>alevin-fry-atac</code> by varying the k -mer size for $\tau = 0.8$ with panels representing different ℓ_{vcol} values. The panel with $\ell_{\text{vcol}} \text{ chr}$ implies mapping is done using chromosome sequence as colors (removing the impact of virtual colors on mapping procedure).	161
C12	Evaluating accuracy across different read lengths for <code>alevin-fry-atac</code> by varying the k -mer size for $\tau = 1$ with panels representing different ℓ_{vcol} values. The panel with $\ell_{\text{vcol}} \text{ chr}$ implies mapping is done using chromosome sequence as colors (removing the impact of virtual colors on mapping procedure).	162
C13	Evaluating the mapping rate across different k -mer size for <code>alevin-fry-atac</code> by varying ℓ_{vcol} for A) $\tau = 0.7$, B) $\tau = 1$ on the Human 10K PBMC, Mouse 8K Cortex and Human 3K Brain datasets respectively. The x-axis labels chr implies mapping is done using chromosome sequence as colors (removing the impact of virtual colors on mapping procedure).	163
C14	Pairwise Sankey plot on the clusters obtained across the different methods for Human 10K PBMC dataset.	164
C15	Pairwise Sankey plot on the clusters obtained across the different methods for Mouse 8K Cortex and Human 3K Brain dataset.	164
C16	UMAP projections with the cluster labels obtained for the different methods on the Human 10K PBMC dataset.	165
C17	UMAP projections with the cluster labels obtained for the different methods on the Mouse 8K Cortex dataset.	166
C18	UMAP projections with the cluster labels obtained for the different methods on the Human 3K Brain dataset.	166
C19	Comparing the time taken and maximum resident memory consumed by <code>alevin-fry-atac</code> and <code>Chromap</code> for the Mouse 8K Cortex dataset across the different threads.	167

C20	Comparing the time taken and maximum resident memory consumed by <code>alevin-fry-atac</code> and <code>Chromap</code> for the Human 10K PBMC dataset across the different threads to map the reads. <code>Chromap(Map Only)</code> refers to when <code>Chromap</code> is run without the <code>--preset</code> arguments.	167
C21	Comparing the time taken and maximum resident memory consumed by <code>alevin-fry-atac</code> and <code>Chromap</code> for the Mouse 8K Cortex dataset across the different threads to map the reads. <code>Chromap(Map Only)</code> refers to when <code>Chromap</code> is run without the <code>--preset</code> arguments.	168

Chapter 1: Introduction

The rise of modern high-throughput sequencing technologies has transformed biological research, allowing scientists to rapidly and cost-effectively sample vast amounts of DNA and RNA from biological samples. These technologies generate short sequences of nucleotides, known as reads, which serve as the foundation for most genomic analyses. Depending on the research goals, specific methods and protocols can be used to target distinct genomic or transcriptomic features [2–5]. These features may include RNAs, DNA fragments from DNA-binding sites, open-chromatin regions, protein-coding regions, or genomic DNA in general, among others [6, 7]. Recent technological advances have further extended our ability to access this information at the level of single-cells [8,9]. These breakthroughs in sequencing technologies have not only deepened our understanding of biological phenomena but also open new avenues for developing targeted therapeutics [10–13].

Once sequencing is complete, the next critical step is to quantify the abundance of targeted features. This step helps researchers compare gene expression across different conditions—like healthy vs. diseased tissues—or identify cell-type-specific expression patterns, among other analysis. However, the exact origin of each read within the reference genome or transcriptome is often unknown, making accurate mapping essential.

Mapping these short reads—often 80 to 300 base pairs long—to a reference genome,

which may span billions of base pairs, is a significant computational challenge. For context, the human genome spans around 3.05 billion base pairs and contains around 233,000 transcripts, with average transcript length ranging from 1500-2000 nucleotides [14]. The genomes and transcriptomes of many other organisms can be even larger and more complex compared to humans, further complicating the mapping process. Most sequencing datasets often contain **millions** of reads.

A key step is thus to map the reads efficiently to the reference before the abundances of the features can be estimated or any downstream analysis such as differential testing [15] can be initiated. Using directly the brute-force strategies or dynamic programming-based approaches, such as Smith-Waterman [16] or Needleman-Wunsch [17], for mapping each read to the reference is generally impractical for most large-scale datasets. Over the years, various specialized methods have been developed to address this challenge [18]. Since in most cases, all the reads are mapped to a fixed set of reference sequence(s), a common approach in these methods is to use specialized indexing strategies, such as FM-indexes, suffix-arrays or hash-tables, to store the redundant subsequences in the reference efficiently [19–27]. There are two primary approaches for associating reads with reference sequences:

- Alignment-based methods: These provide base-level resolution, pinpointing where each nucleotide in a read aligns with the reference, including mismatches or gaps. Mapping-based methods: These identify the likely origin of a read without determining the exact nucleotide-level alignment, offering significant speed improvements while maintaining accuracy for specific applications.

These strategies differ in how the individual mapped seeds from a read are aggregated, with the

alignment-based approaches typically applying a dynamic programming based approach to the mapped seeds. We refer to the seeds/subsequences of fixed-length k as k -mers.

For applications like RNA-seq, where the reference may consist of a transcriptome, mapping-based approaches such as pseudoalignment [25, 28] provide a compelling alternative. These methods achieve a comparable accuracy to alignment-based techniques while being significantly faster. However, when the reference is the entire genome, typically alignment-based methods are used, as techniques like pseudoalignment are not designed for genome-wide mapping. We will explore this topic in more detail later in this chapter.

Even when reads can be mapped efficiently, a significant challenge arises in determining the true locus of origin for a read, particularly when it comes from the overlapping sequences between features. This issue is often prevalent in RNA-seq, where reads often originate from shared regions between transcripts. This ambiguity introduces uncertainty in inferring the abundance estimates of these features, which, in turn, can propagate and affect any downstream analyses that rely on these estimates. As a result, it is important to develop methods for downstream analysis that account for this uncertainty.

In this thesis, we address two main challenges:

1. Design a framework that incorporates uncertainty associated with inferring the abundance estimates for RNA-seq data and develop methods that enable downstream analysis using this framework. Specifically, we develop a differential testing method on top of this framework.
2. Design and develop a method for processing and mapping single-cell ATAC-seq data, by introducing a new pseudoalignment-based approach for mapping to the genome.

In chapter 2, we address part one of the first challenge by introducing `TreeTerminus`

[29], which provides a framework for incorporating uncertainty associated with RNA-seq abundance estimation. We first review some of the existing work in RNA-seq quantification, focusing on the commonly used generative model that can be employed to model the sequencing process. The output of these quantification methods is typically a point estimate for each transcript. We then explain the underlying reasons for the uncertainty inherent in these point estimates and explore methods for quantifying it. In general, there is less uncertainty associated with gene-level abundances compared to transcripts, as genes have less sequence overlap than individual transcripts. The uncertainty for a transcript primarily depends on the number of reads that map uniquely to it relative to the total number of reads mapped to it. We also discuss the existing approaches of grouping transcripts into discrete groups, and highlight their limitations, particularly with respect to the downstream analysis.

`TreeTerminus` builds on top of `Terminus` [30], which groups transcripts based on uncertainty. However, instead of providing discrete groups as output, `TreeTerminus` organizes transcripts into a tree-like structure for a set of samples in an RNA-seq experiment in a data-driven manner. The leaves represent the transcripts, and the inner nodes represent the aggregated group of transcripts, with uncertainty decreasing across the samples as we climb up the tree. We describe the method for creating transcript-trees for each sample and provide two different approaches for creating trees across a set of samples, which enables summarizing the uncertainty structure in the RNA-seq experiment. The tree provides the flexibility to analyze data at different layers of resolution depending on the downstream analysis of interest, providing a middle-ground between gene and transcript-level analysis. Additionally, we present a dynamic programming solution to obtain a cut from the tree that optimizes an objective function. We demonstrate its utility by providing two objective functions and using the resulting cuts for differential

analysis. To the best of our knowledge, `TreeTerminus` is currently the only method that constructs transcript-trees.

In chapter 3, we continue addressing the first challenge, with a focus on downstream analysis for accounting uncertainty, and introduce `mehenDi` [31], a method designed for differential testing. `mehenDi` leverages the tree structure obtained from `TreeTerminus` and outputs a set of nodes from the tree that are differentially expressed between the conditions of interest, ensuring no two nodes share an ancestor-descendant relationship. The output nodes can consist of inner nodes—particularly in cases where uncertainty masks the signal of individual transcripts but can be preserved through aggregation at an internal node. In this chapter, we first provide a detailed overview of the existing methods developed for tree-based differential testing, categorizing them based on their objectives and applications. We highlight the limitations of these approaches in addressing our specific use case and why they are not directly applicable to the trees generated by `TreeTerminus`.

Next, we present a detailed description of `mehenDi`, motivating each of its design principles. `mehenDi` has been designed to maximize the signal that can be extracted from the data while accounting for uncertainty. We then rigorously evaluate `mehenDi` using various simulated and experimental datasets, applying both quantitative and qualitative metrics. Its performance is compared with other relevant tree-based differential testing methods, as well as with established gene- and transcript-level differential analysis techniques. On the simulated datasets, `mehenDi` effectively control the false positives while demonstrating superior sensitivity. On the experimental datasets, `mehenDi` identifies numerous transcript groups where aggregation recovers signals that would have otherwise been masked by uncertainty. Notably, many of these groups correspond to genes that show differential expression at the gene level but do not contain any transcripts deemed

significant in traditional transcript-level analyses.

In chapter 4, we address the second challenge by introducing `alevin-fry-atac` [32], our method for mapping and processing single-cell ATAC-seq data. In the previous section, we had briefly discussed the process of alignment and noted that for mapping reads to the genome, alignment-based strategies are used, as mapping-based strategies are not directly applicable. For single-cell ATAC-seq data, accurate read mapping to the genome is essential. This chapter begins with an overview of the existing methods and pipelines that are used for this task in single-cell ATAC-seq data, which primarily rely on alignment-based techniques. We also review mapping-based strategies and their adaptations in transcriptomics and metagenomics while addressing its limitations for genomics. Next, we introduce our modified pseudoalignment-based approach, which uses virtual colors to map reads to the genome, providing a viable alternative to traditional alignment-based strategies. This method is overlaid on the top of the existing index, eliminating the need for index reconstruction when the sizes of virtual colors are varied. It also does not add any computational overhead. To further improve query speed, we implement a caching scheme that leverages the fact that most reads in single-cell ATAC-seq data tend to map to similar genomic positions. This optimization significantly improves performance, aiding in making `alevin-fry-atac` faster than existing methods.

`Alevin-fry-atac` utilizes the `piscem` index [33], which has also been employed for mapping the single-cell RNA-seq data in `alevin-fry` [34,35]. Building on top of the existing `alevin-fry` framework, we extended its methodology to process single-cell ATAC-seq data and produce outputs compatible with downstream analysis tools. Additionally, we introduced a new sorting procedure into the pipeline, enhancing its overall efficiency. This unified `alevin-fry` framework simplifies multi-omic data analysis by providing a consistent and streamlined

processing pipeline. We evaluated `alevin-fry-atac` on both simulated and experimental datasets, assessing both mapping accuracy and computational efficiency. On simulated datasets, it achieved accuracy comparable to alignment-based methods, while on experimental datasets, the resulting clusters showed high concordance with those obtained from other state-of-the-art methods. Importantly, `alevin-fry-atac` demonstrated significant performance improvements, running three times faster and using three times less memory compared to existing methods.

Below, we provide a primer on some relevant topics that will serve as a foundation for understanding the methods discussed in the following chapters.

1.1 Preliminaries for Chapter 2 and 3

1.1.1 RNA-seq basics

RNA-seq is a widely used next-generation sequencing technique that enables the quantification of RNA molecules present within a sample [36, 37]. It can be used to infer and compare gene and transcript expression both within and between samples. The RNA fragments are isolated from the sample but cannot be directly sequenced with the most commonly used sequencing technologies. To prepare RNA for sequencing, the extracted RNA is first fragmented into smaller pieces. The fragments are then reverse-transcribed into complementary DNA (cDNA) using reverse transcriptase. The cDNA fragments undergo further processing in a step called library preparation, where sequence adaptors are ligated into cDNA. The library is then amplified by PCR to generate enough material for sequencing. Finally, the prepared library is sequenced using next-generation sequencing platforms such as Illumina.

1.1.2 Equivalence classes

In the context of RNA-seq quantification, collapsing of fragments/reads into *equivalence classes* is a common practice with different notions defined across different methods. We use the notion defined in `Salmon` [28] or `kallisto` [25]. An equivalence class refers to a grouping of reads that can be mapped to the same set of transcripts. In RNA-seq, when a read aligns to multiple transcripts that share overlapping regions, the read is assigned to an equivalence class representing all the transcripts in that set. Equivalence classes capture the ambiguity in read mapping, particularly when multiple transcripts share sequence similarities or overlap, making it difficult to unambiguously assign a read to a single transcript. A range-factorized equivalence class [38] in addition also encodes the mapping quality, with a single class constituting a set of pairs (t_i, w_i) rather than just set of t_i , where t_i denotes the transcript and w_i represents the average conditional probability with which the fragments in the equivalence class arose from that transcript.

1.1.3 Terminus

`Terminus` [30] is a method used for grouping transcripts such that the uncertainty for the transcript-group is less than the individual transcripts. It provides two modes, namely, `group` and `consensus`. The `group` step provides transcript groups for each RNA-seq sample whereas `consensus` step outputs transcript groups for a set of RNA-seq samples. The `group` step of `Terminus` takes as input an individual `Salmon` [28] quantified sample from an RNA-Seq experiment and outputs a set of transcript groups. It makes use of the range-factorized equivalence classes ξ [38], and inferential replicates $\mathcal{P}$ obtained after running `Salmon` with

the appropriate flags. An equivalence class denotes an association from a set of transcripts to a set of reads, that are mapped to all the transcripts in that set. A `range-factorized` equivalence class in addition also encodes the mapping quality, with a single class constituting a set of pairs (t_i, w_i) rather than just a set of t_i , where t_i denotes the transcript and w_i represents the average conditional probability with which the fragments in the equivalence class arose from that transcript. The `Terminus` groups consist of transcripts that share large numbers of ambiguously-mapped fragments. It employs a union-find data structure, with the first step being to scan over ξ and group the transcripts that appear in the same set of equivalence classes and have near-identical conditional probability vectors using the `Union` operation. This ensures that these transcripts will belong to the same partition when checked through the `Find` operation. It then constructs a graph with the transcripts as nodes, where an edge between any two nodes v_i, v_j implies that they co-occur in at least one equivalence class and have the edge score $s(v_i, v_j) \leq \tau$ where

$$s(v_i, v_j) = \text{infRV}_{i+j} - \frac{\text{infRV}_i + \text{infRV}_j}{2} \quad (1.1)$$

$$\text{infRV}_i = \frac{\max(\sigma_{p_i}^2 - \mu_{p_i}, 0)}{\mu_{p_i} + pc} + d \quad (1.2)$$

p_i are the posterior (`Gibbs`) replicates for transcript i , $\sigma_{p_i}^2$ and μ_{p_i} are the variance and mean over the posterior replicates, pc is the pseudocount (default is 5), d is a small global shift (default is 0.01) and τ is a threshold of difference in inferential relative variance. A min heap H is constructed over the edges keyed by $s(v_i, v_j)$, and at each iteration t an edge with the lowest score is popped out till the iteration H becomes empty. If the partition (group) of any endpoint vertex for the popped edge has been modified but its corresponding score was not updated, then the edge

is called stale. If the edge is not stale then the corresponding nodes v_i, v_j are grouped using the `Union` operation and the posterior samples for v_i are updated with $p_i = p_i + p_j$, assuming $i < j$. The score $s(v_i, x)$ is recomputed for all the vertices $\{x | x \in \text{adj}(v_i) \cup \text{adj}(v_j) - \{v_i, v_j\}\}$ with $\text{adj}(v_i)$ denoting the neighbours of vertex i in the graph at iteration t . The edge with the updated $s(v_i, x)$ is pushed to H , if $s(v_i, x) \leq \tau$.

The `consensus` step outputs a set of transcript groups across samples using the groups obtained for the individual samples. The procedure involves creating a connected undirected graph on the transcripts, with an edge between any pair of transcripts denoting that they co-occur in a group in any sample and the edge weight denotes the count of the total number of samples in which the pair co-occurs in a group. To create the consensus groups, it finds the connected components on the updated graph obtained after removing the edges that occur in at least a certain proportion of samples. This proportion is a user-defined parameter and by default has been set to 0.5.

1.2 Preliminaries for Chapter 4

1.2.1 Basics of ATAC-Seq and the 10x ATAC-Seq protocol

ATAC-Seq is a technique that is used to assess the open-chromatin regions in the genome [4]. The samples are incubated with the transposase enzyme Tn5 which is loaded with the sequencing adapters on its ends. The Tn5 transposase then cuts the open chromatin regions, attaching adapters to the cut fragments, doing both fragmentation and tagmentation simultaneously. The obtained library then can be amplified and sequenced using Illumina sequencing in Figure 1.1.

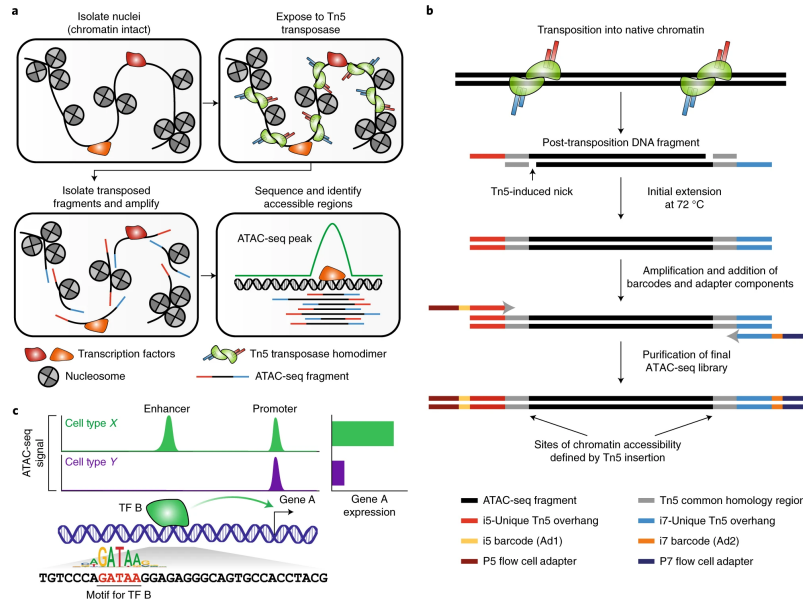


Figure 1.1: Main steps in the ATAC-Seq protocol adapted from [1]

The platform provided by 10X genomics is currently the most widely used and available platform for single-cell ATAC-seq. We briefly describe the platform below, which is shown in (Figure 1.2) for the version V2 chemistry. First, the nuclei are transposed in a bulk solution, with the adapter sequences added to the ends of DNA fragments. GEMs are generated by combining barcoded Gel Beads, transposed nuclei and partitioning oil on a Chromium Chip. While most GEMs(90-99%) don't contain a single nuclei, the remainder largely contain a single nucleus. Once the GEM is generated, the Gel Bead is dissolved, releasing oligonucleotide containing Illumina P5 sequence, 16 nt 10X barcode and Read 1N. All the fragments belonging to the same cell will have the same 10X barcode. Thermal cycling of the GEMs produces 10x barcoded single-stranded DNA. The GEM is broken, and the fragments are then amplified by PCR and sequenced using Illumina. It is important to note that the order and length of the nucleotides that are attached to GEM can vary depending on the protocol and chemistry.

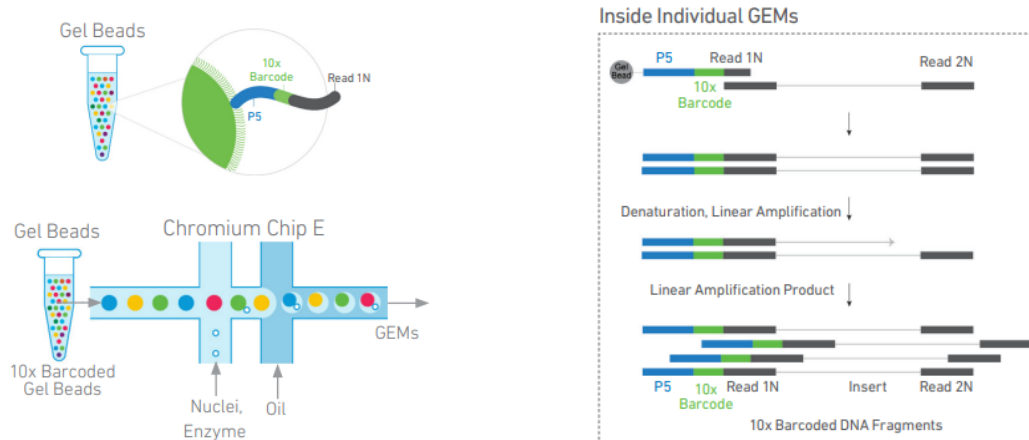


Figure 1.2: Summary of the 10X protocol, the figure is adapted from [10X Website\(V2\)](#)

1.2.2 de Bruijn graph

De Bruijn graph is a popularly used graph-based data structure in genomics with wide applications in bioinformatics, including genome assembly [39, 40], and read mapping [25, 26, 41], to name a few. De Bruijn is used to represent a set of sequences, where the nodes represent k -mers in the sequences and the edges represent a $k + 1$ -mers, with $k - 1$ overlap between the nodes. To reduce the space occupied by the graph, the vertices across each maximal non-branching path are collapsed into a single node, forming the compacted de Bruijn graph [42]. When each vertex is associated with color(s), that represents the set of references where that k -mer (set of k -mers compressed into a node) was present in, it forms the colored (compacted) de Bruijn graph.

1.2.3 Pseudoalignment

Pseudoalignment is a lightweight mapping procedure that aims to assess the “compatibility” between a read and a collection of references without specifically finding the coordinates of each

base of the read in the reference. The term was originally coined in the context of RNA-seq quantification [25] where the transcripts to which a read maps were found by doing an intersection across the colors (transcripts) mapped by a set of selected k -mers of a read. Certain characteristics of this mapping process, however, predate the coining ([43], [44]). Several approaches and algorithms for pseudoalignment have been proposed. They vary from how and which set of k -mers of a read is selected for mapping to how the colors corresponding to the selected k -mers are aggregated to get the final color set. The color set denotes the references to which a read maps [26, 45].

Chapter 2: TreeTerminus — Creating transcript trees using inferential replicate counts

2.1 Introduction

Transcript abundance estimation is among the key target applications of RNA-Seq. The variation in transcript expression plays a key role in development [46–48]; and characterization of diseases and their subtypes [49–51]. One of the approaches towards estimation of transcript abundances is to probabilistically assign a given fragment to the transcripts using maximum likelihood or Bayesian inference [25, 28, 52]. This has to be done since there exists an ambiguity towards finding the true locus of origin for a given sequencing fragment, when it can map equally well to the shared sequences within transcripts. Thus, a certain degree of uncertainty is associated with the transcript abundance estimates, depending on the nature of the fragments. This in turn makes downstream analysis, such as differential testing, difficult for certain transcripts. Uncertainty also exists for gene expression estimates but is less compared to transcripts, since there will be less ambiguity when mapping fragments to genes. The uncertainty of a transcript/gene can be estimated from the inferential replicates generated either through MCMC/Gibbs Sampling; or through bootstrap sampling of the reads [25, 28, 52–54].

The highly uncertain transcripts/genes may become invisible when the inferential replicates

are incorporated into the downstream tasks. To circumvent this problem, some methods have grouped transcripts/genes into distinct inferential units that share a lot of multi-mapping reads [55]. `mmcollapse` [56] proposes to group transcripts by computing correlation between every pair of transcripts on the posterior replicates and the pair that has the most negative correlation is grouped. This process is repeated for multiple iterations until a stopping criterion is reached, where at each iteration the correlation is recomputed for every transcript/group with the group from the previous iteration and a new group is formed. `Terminus` [30] also provides transcript groups as an output but employs difference in inferential relative variance [57] between the group and the mean of its underlying transcripts/subgroups as the statistic for group creation. It first creates a graph on the transcripts, with an edge denoting that the transcripts co-occur in at least one `range-factorized` equivalence class [38] and only the transcripts that are connected by a path on the graph are considered for grouping. `Terminus` first finds groups across each individual sample and uses a consensus approach to output consensus groups across samples. Importantly, both `mmcollapse` and `Terminus` provide a single level of resolution for analysis, not permitting analysis across different levels.

Another limitation with both of the above methods, is that the resolution at which grouping is stopped is governed by a threshold determined heuristically that does not take the downstream analysis into account. When performing downstream tasks such as differential analysis, these methods can either thus over-aggregate transcripts, masking the signal in the process or under-aggregate leading to weaker signals, and an inability to make confident calls. Since the goal of these methods is to provide concrete groups as an output without doing much aggregation, they introduce many filtering constraints on transcripts in order for them to be considered for aggregation which might further lead to missing good candidates for aggregation.

In this work, we introduce `TreeTerminus`, a new method that aims to address some of the above-mentioned shortcomings. `TreeTerminus` expands upon the idea of `Terminus` and groups transcripts in a tree structure where leaves represent individual transcripts and internal nodes represent an aggregated set of leaf transcripts. Across the samples in the experiment, inferential uncertainty decreases on average, ascending the tree topology. `TreeTerminus` can be run on a single sample and we provide two different approaches in order to extend it for the multi-sample settings. The tree provides the flexibility to analyze data at nodes that are at different levels of resolution in the tree and can be tuned depending on the analysis of interest. To obtain fixed groups for downstream analysis, we provide a dynamic programming (DP) approach that can be used to find a cut through the tree that optimizes one of several different objectives. In addition, the DP approach has the ability to optimize for other user-defined objective functions provided they adhere to the required constraints necessary to be efficiently optimized on a tree.

To the best of our knowledge, this is the first time that transcripts have been arranged in a tree-like structure, that too in a data-driven manner. We evaluated `TreeTerminus` on two simulated and two experimental datasets, and observed an improved performance compared to transcripts (leaves) and other methods under several different metrics. `TreeTerminus` has been implemented in Rust and the source code can be downloaded from <https://github.com/COMBINE-lab/TreeTerminus>. We have also created an R Package `beaver` that parses the output of `TreeTerminus`, implements DP algorithms for finding an optimal cut, and provides helper functions to obtain useful statistics for subtrees within the `TreeTerminus`-derived tree structures. The R Package can be downloaded from <https://github.com/NPSDC/beaver>.

2.2 Methods

2.2.1 Overview of TreeTerminus

The transcripts in a `Terminus` group consist of at least two transcripts that have reads multi-mapped to them, which is the source of uncertainty for transcript abundance estimation. `TreeTerminus` outputs a tree for each individual group, encoding the summarized order in which a set of transcripts should be aggregated within a group, such that across the samples, average uncertainty decreases as we ascend the tree created for that group.

`TreeTerminus` has a `group` step, that constructs transcript-trees for a single sample. It starts by creating a single node tree corresponding to each transcript. For any pair of transcripts/subgroups that are aggregated in the `group` step of `Terminus`, a binary tree is created with its children being the individual trees corresponding to the node pairs that are grouped. The process of tree creation continues till the time heap H becomes empty. In addition, some of the constraints that `Terminus` imposed on transcripts/groups before they could be even considered for further grouping have also been relaxed. It is no longer necessary that a node v_i should have $\inf RV(v_i) \geq \delta$ (called filtered or `Filt`) and that for a pair of nodes v_i, v_j , $s(v_i, v_j) \leq \tau$ (called early stop or `ES`). For the N groups that will be obtained from a sample m , a set of N trees $T^m = \{T_{1m}, \dots, T_{Nm}\}$ will be generated, such that for a group g , $\Lambda(g) = \Lambda(T_{gm})$, with Λ representing the set of transcripts covered by a tree or a group and Δ^m represents all the trees belonging to sample m . We next propose and describe two different approaches to obtain trees, representative of all samples in the experiment.

2.2.2 Mean Tree

- This is a single-step procedure where all the samples are processed together and a single tree is produced from all the samples w.r.t a group. The steps in the tree construction follow directly from what we have described above, with some modifications. To find transcripts that have near identical conditional probability weights across all the equivalence classes, the search space is expanded to all samples rather than a single sample. Later, when the graph across transcripts is constructed, an edge between any two nodes implies that they co-occur in the same equivalence class in at least one sample where the edge score $s'(v_i, v_j)$ is updated as, $s'(v_i, v_j) = \frac{\sum_{j=1}^M s_j(v_i, v_j)}{M}$, and M represents the total number of samples in the experiment. The indicator function $h(\cdot)$ that determines whether a transcript is a good candidate for aggregation when considered in isolation is also updated as:

$$h(v_i) = \begin{cases} 1 & \text{if } \max_{j=1}^M \mu_{ij} \geq 1 \text{ and } \max_{j=1}^M sp_{ij} \geq \lambda \\ 0 & \text{otherwise} \end{cases} \quad (2.1)$$

where $\mu_{ij} = \text{mean}(p_{ij})$ and $sp_{ij} = \frac{\max(p_{ij}) - \min(p_{ij})}{\text{mean}(p_{ij})}$, with p_{ij} denoting the posterior replicates for transcript i in sample j and λ is a user-defined parameter that is set to 0.1 that serves as a threshold for spread.

2.2.3 Consensus tree

- Construction of consensus trees is more involved and takes as input the sample group trees obtained by running `group` step on each sample. Given a set of trees for a given transcript

group across samples, we want to find a tree that summarizes the topological structure of the input trees. Such a tree is called consensus tree [58,59]. There exists a wide variety of methods to get consensus trees, depending on the input and downstream applications [59–62]. In this work, we have used the majority rule extended or greedy consensus tree algorithm [60,63] which we describe below.

Given a tree T , let $V(T)$ denote the set of all nodes and $\Lambda(T)$ denotes the set of transcripts covered by the tree T . For any node $u \in V(T)$, $T[u]$ is the subtree rooted at u , $\Lambda(T[u])$ is the cluster associated with it and $C(T)$ represents the set of all clusters associated with tree T . Let S denote the collection of trees $(T_1, T_2, ..T_M)$, with $\Lambda(T_i) = L, \forall i \in \{1, ..., M\}$, aka all trees have the same leaf set. Let X be the set of all clusters that occur in S sorted by the decreasing order of the frequencies with which they occur in the trees. Construct a set Y of clusters as:

Initialize $Y = \emptyset$, then traverse X and for each cluster C encountered in this order, check if C and C' are pairwise compatible for all $C' \in Y$, if yes then $Y = Y \cup \{C\}$. A greedy or majority extended consensus tree of S is a tree T such that $\Lambda(T) = L$ and $C(T) = Y$.

A constraint of a consensus tree algorithm is that it requires all the input trees should span the same leaf set. To obtain a consensus tree T_g for group g in an experiment containing M samples, we want to provide group trees across samples $\Delta_g = \{T_{g1}, ..., T_{gM}\}$ as an input to the Majority Rule Extended Consensus Tree Algorithm, with T_{gi} representing the tree for sample i for the group g and $\Lambda(g) = \Lambda(T_{gi}), \forall i \in \{1, ..., M\}$. Λ represents the transcripts covered by a tree or a group. However, a group covering a transcript set in one sample might not be preserved in other samples. We can have tree/s within a sample that covers a transcript set that overlaps but is not the same as the transcript(leaf) set in a tree from some other sample in the experiment. This

is demonstrated in the example shown in Figure 2.1a. For the five different samples, we have trees that cover different transcript sets, covering overlapping transcripts. First sample contains the tree built on 5 transcripts. The second and fifth samples span the same set of 4 transcripts but are subsets of the transcripts covered by T_{11} . The third and fourth samples each have two trees containing transcripts that are also a subset of $\Lambda(T_{11})$. Thus, we cannot directly apply the consensus tree algorithm on the trees obtained by the `group` step of `TreeTerminus`.

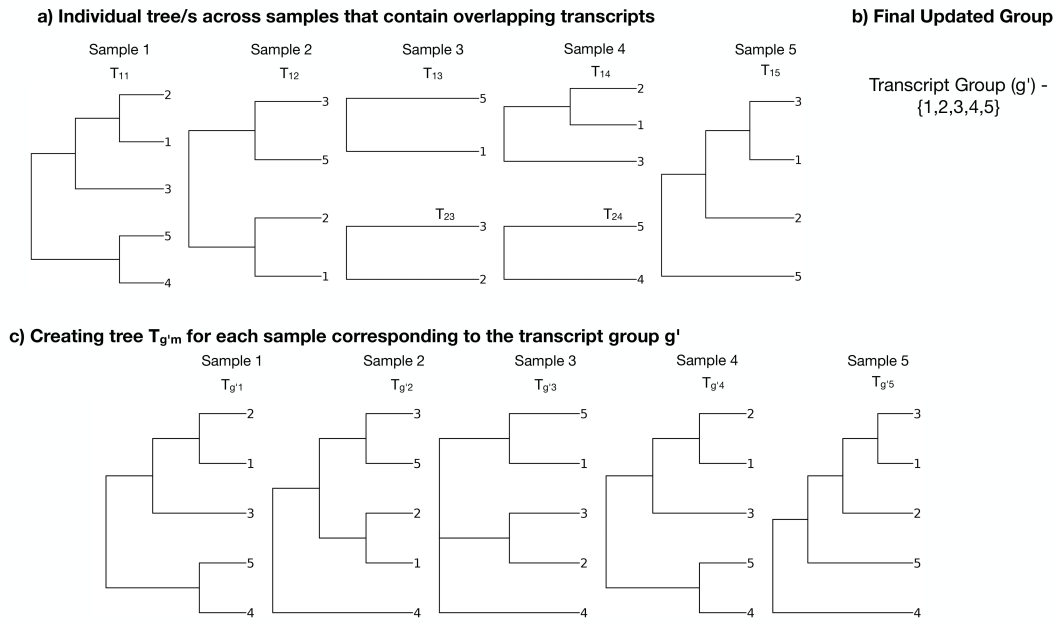


Figure 2.1: Example of a sample group to demonstrate why consensus step cannot be directly applied on the trees obtained from `group` step for samples and the modifications made in order to apply the consensus tree algorithm. **a** Trees across the 5 different samples that contain overlapping transcripts. Each tree is labeled as T_{gm} , where m denotes the sample and g denotes a group in that sample. **b** Updated group that is a superset of all the transcripts in the individual trees across samples. **c** Tree created for each sample w.r.t updated group g' , with $\Lambda(T'_g) = \Lambda(g')$.

To resolve this issue, we thus first create a set of updated groups $G' = \{g'_1, \dots, g'_N\}$, with updated group g'_i representing a union of all transcripts for all trees across samples that contain overlapping transcripts. Further, the transcripts covered by any such tree should not contain any overlapping transcripts with the transcripts of any other updated group g'_j or:

$$\Lambda(g'_i) = \bigcup_{m=1}^M \bigcup_{g=1}^{N_m} \Lambda(T_{gm}) \text{ where } \Lambda(T_{gm}) \subset \Lambda(g'_i) \text{ and } \Lambda(T_{gm}) \cap \Lambda(g'_j) = \emptyset, \forall j \in \{1, \dots, N\}, j \neq i.$$

To create the updated group g'_i , we employ the union-find data structure. We scan through the trees across all samples and apply `Union` operation on the transcript set covered by the tree, which will group these transcripts covered. This ensures that the transcripts belonging to any two trees T_i, T_j across the samples where $\Lambda(T_i) \cap \Lambda(T_j) \neq \emptyset$ will be grouped together. For the toy example in Figure 2.1b, this leads to the creation of the updated group spanning transcript set $\{1, 2, 3, 4, 5\}$. The consensus tree $T_{g'_i}$ will be created for the updated group g'_i , with $\Lambda(T_{g'_i}) = \Lambda(g'_i)$.

We next create trees $T_{g'_i m}$ for every sample $m \in \{1, \dots, M\}$ w.r.t each updated group $g'_i \in G'$, such that tree $T_{g'_i m}$ covers same set of transcripts as the updated group or g'_i or $\Lambda(T_{g'_i m}) = \Lambda(g'_i)$. Tree $T_{g'_i m}$ on the updated group g'_i for sample m is created using the following steps:

1. For each sample m , all the trees T_{gm} from Δ^m which cover transcripts that overlap with the transcripts covered by g'_i are extracted to form the set $\Delta_{g'_i m}$ where

$$\{\Delta_{g'_i m} = \{T_{1m}, \dots, T_{dm}\} \subset \Delta^m \mid \Lambda(T_{gm}) \subset \Lambda(g'_i) \forall g \in \{1, \dots, d\} \ \& \ \Lambda(g'_i) \cap \Lambda(T_j) = \emptyset, \forall T_j \in \Delta^m - \Delta_{g'_i m}\}.$$

2. A transcript set D is constructed that consists of transcripts covered by the updated group

$$g'_i \text{ but not by the trees in } \Delta_{g'_i m}. \text{ Formally } D = \Lambda(g'_i) - \bigcup_{i=1}^d \Lambda(T_{im}), \text{ with } T_{im} \in \Delta_{g'_i m}.$$

3. If the set $\Delta_{g'_i m}$ consists of only one tree T_{gm} and D is empty, it implies that the tree T_{gm} already covers the same transcript set as g'_i and thus we can safely output T_{gm} as $T_{g'_i m}$.

4. Otherwise, an empty tree is constructed and added to it's children are all the trees in $\Delta_{g'_i m}$ along with the transcripts in the set D . This newly created tree forms $T_{g'_i m}$.

For each group $g'_i \in \{g'_1, \dots, g'_N\}$, there now exists a set of M trees $\Delta_{g'_i} = \{T_{g'_i1}, \dots, T_{g'_im}\}$, - one for every sample. PHYLIP's [64, 65] implementation of Majority Rule Extended Consensus Algorithm is then applied on the set $\Delta_{g'_i}$ to get the consensus tree $T_{g'_i}$ for the updated group g'_i . The motivation behind using the majority rule extended consensus algorithm is to provide a chance to preserve a topology present in the set of samples that represent a phenotype not present in the majority of samples, which would have been ignored otherwise.

2.2.4 Creating a unified tree from the output of TreeTerminus

Given the output of TreeTerminus $\Delta = \{T_{g'_1}, \dots, T_{g'_N}\}$, a unified tree T_U is created such that its children consist of all the trees in Δ as shown in Figure 2.2. This is done through our R package beaver, which also provides the option to append transcripts to the children of T_U that are not covered by the trees in Δ but still should be considered for downstream analysis.

2.2.5 Solving objective functions to obtain discrete inferential units

We provide a dynamic programming (DP) approach that can be used to optimize different objective functions on a tree, following certain constraints. The DP solving a given objective function outputs a set of nodes in the tree T or cut $C = (c_1, \dots, c_d)$, where C has the following properties:

- The union of leaf nodes belonging to nodes in the cut should cover all the leaf nodes in the tree T or $\bigcup_{i=1}^d \Lambda(c_i) = \Lambda(T_U)$.
- The intersection of leaf nodes belonging to any two distinct nodes in the cuts should lead to an empty set or $\Lambda(c_i) \cap \Lambda(c_j) = \emptyset, \forall i, j \in \{1 \dots d\}$.

The DP optimizes the objective functions of the following form to obtain the optimal value and a cut C :

$$\operatorname{argmin}_C f(C) \text{ or } \operatorname{argmax}_C f(C)$$

$$\text{where } f(C) = \sum_{c \in C} \text{metric}(c)$$

where $\text{metric}(c)$ represents the metric for node c in the tree T , whose sum we want to optimize over all nodes in the cut. We solve for $f(C)$ using the procedure described in Algorithm 1. For a given metric of interest, it outputs the optimal value for the objective function and a set of nodes (cut), on which summing up the metric provides the optimal value. Algorithm 2 (CompOptVal) returns the optimal value for the objective function at a given node and Algorithm 3 (CompOptCut) returns the cut for a tree/subtree for that objective function.

Algorithm 1 Finding the optimal value of the objective function and the corresponding cut for a given metric

Input: $T, met_arr, type$

▷ T is the tree, met_arr is the array containing the value of the metric for all nodes in the tree, $type$ tells whether to minimize or maximize the objective function

Output: $cut = [c_1, \dots, c_d]$

procedure FINDOPTVALCUT

$root \leftarrow \text{FindRoot}(T)$

▷ get the index corresponding to the root node in the tree

$t_nodes \leftarrow \text{length}(T)$

▷ t_nodes are the total number of nodes in the tree

$cut \leftarrow []$

$opt_arr \leftarrow [0; t_nodes]$

for n in $0..t_nodes$ **do**

$opt_arr[n] \leftarrow \text{CompOptVal}(T, n, met_arr, type)$ ▷ Optimal value at all nodes for the objective function

end for

$\text{CompOptCut}(T, root, met_arr, opt_arr, cut)$.

return cut

end procedure

Algorithm 2 Function for finding optimal value of objective function at given node

Input: $T, node, met_arr, type$

Output: opt_val_node

procedure COMPOPTVAL

if $node$ isLeaf **then**

return $met_arr[node]$

end if

$children \leftarrow$

$\text{findChildren}(T, node)$

$child_opt \leftarrow \sum_{n \in children} \text{CompOpt}$

$\text{Val}(n,$

$met_arr, type)$

if $type = \text{min}$ **then**

return

$\text{min}(met_arr[node], child_opt)$

else

return

$\text{max}(met_arr[node], child_opt)$

end if

end procedure

Algorithm 3 Function for finding optimal cuts

Input: $T, node, opt_arr, met_arr, opt_cut$

Output: $cut = [c_1, \dots, c_d]$

procedure COMPOPTCUT

if $opt_arr[node] = met_arr[node]$

then

$opt_cut.append(node)$

return opt_cut

end if

$children \leftarrow$

$\text{findChildren}(T, node)$

for $child$ in $children$ **do**

return

$\text{CompOptCut}(child, opt_arr,$

$met_arr, opt_cut)$

end for

end procedure

We have solved for the objective functions employing two different metrics to obtain cuts

on the tree T_U which we describe in the section below.

Minimize mean infRV and height - We minimize the sum for the node metric - the sum of mean inferential relative variance and height weighted by γ ; and then multiplied by the number of descendant leaf nodes. This metric is called `irv_height_desc` and is described formally below with the objective function:

$$\operatorname{argmin}_c \sum_{c \in \mathcal{C}} \text{irv_height_desc}(c) \quad (2.2)$$

$$\text{irv_height_desc}(c) = \text{irv_height}(c) * |\Lambda(c)| \quad (2.3)$$

$$\text{irv_height}(c) = \text{mirv}(c) + \gamma \text{height}(c) \quad (2.4)$$

$\text{mirv}(c)$ is the mean inferential relative variance for a node across samples, $\text{height}(c)$ is defined as the number of edges on the longest path from node c to its descendant leaf, $\Lambda(c)$ represents the descendant leaves for a given node c , with $|\Lambda(c)| = 1$ if c is a leaf node. The reason that the number of descendants of a node are multiplied is because the value of the objective function is dependent on the number of nodes in the cut and will be biased towards nodes with higher height as they end up replacing multiple lower-height descendant nodes. To get the optimal value and desired cut for the above objective function we use the Procedure defined in Algorithm

1 as:

$$\text{opt_val}, \text{cut} = \text{FindOptValCut}(T_U, \text{irv_height_desc}, \text{min}) \quad (2.5)$$

Obtaining fixed groups from TreeTerminus for downstream analysis

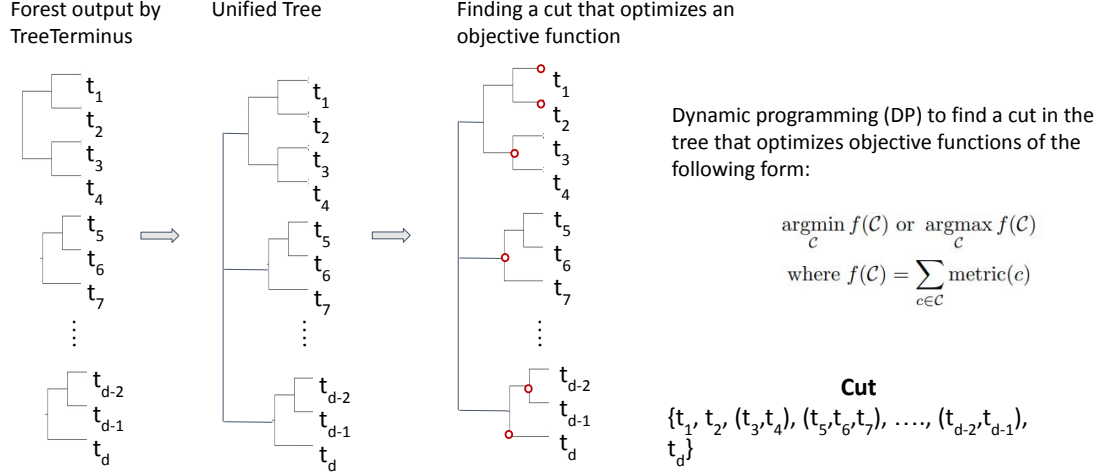


Figure 2.2: Demonstration of creating a unified tree from the forest that is obtained as the output of TreeTerminus. The unified tree can then be used for optimizing an objective function of interest in order to obtain a cut for downstream analysis.

Maximize log fold change - In this objective function, we thus maximize the sum of node metric - absolute log fold change using CPM (transcript per million) multiplied by the number of the descendant leaves for that node.

$$\operatorname{argmax}_{\mathcal{C}} \sum_{c \in \mathcal{C}} \text{lfc_desc}(c) \quad (2.6)$$

$$\text{lfc_desc}(c) = \text{wlf}(c) \cdot |\Lambda(c)| \quad (2.7)$$

$$(2.8)$$

where lfc is the $\log_2$ fold change. Assuming the samples can be grouped into two conditions represented by the sets m_1 and m_2 , lfc is given by:

$$\text{lfc}(c) = \text{median}\{\text{lfc}(c)_k\}, k \in \{1..R\} \quad (2.9)$$

$$\text{lfc}(c)_k = \log_2 \left(\frac{\sum_{j=1}^{m_1} \text{cpm}(c)_{jk}}{|m_1|} + \text{pc} \right) - \log_2 \left(\frac{\sum_{j=1}^{m_2} \text{cpm}(c)_{jk}}{|m_2|} + \text{pc} \right) \quad (2.10)$$

where R represents the total number of inferential replicates, $\text{cpm}(c)_{jk}$ represents k th inferential replicate's CPM for node c of sample j . We again use the procedure defined in Algorithm 1 to get the optimal value and the cut as:

$$\text{opt_val, cut} = \text{FindOptValCut}(T_U, \text{wlfc_desc}, \text{max}) \quad (2.11)$$

2.2.6 Datasets

To demonstrate the benefits of TreeTerminus, we ran it on both simulated and experimental datasets spanning different organisms.

Simulated human datasets Polyester [66] was used to generate simulated RNA-seq data. TPM estimates were extracted from GTEx V8 frontal cortex dataset with the distribution of mean and dispersion values derived from GEUVADIS samples [67]. The process of read generation has been described in detail in [68]. We generated 12 samples with 6 samples in each condition. All transcripts were differentially expressed for 10% of the genes with all having the same fold change (DGE) and 10% genes had a single transcript differentially expressed (DTE). We created two variations of this simulation by varying the range of fold change - in the first variation we keep the same fold change as in [68] and in the second variation, the maximum range of fold change is lowered from 6 to 3. The first variation is referred to as BrSimNorm and the second is referred to as BrSimLow.

Mouse muscle dataset This dataset is taken from the skeletal muscle study GSE100505 [69]. In this paper, we have used 12 samples with 6 samples belonging to Atria and 6 to Tibialis Anterior, with the accession numbers provided in Table A1. All the samples belong to organism

mus musculus. This dataset is referred to as `MouseMuscle`.

Chimpanzee brain dataset The final dataset that has been analyzed in our study is the RNA-Seq data from [70], SynapseID - `syn7067053` collected from 5 Chimpanzees (*Pan Troglodyte*). We refer to this dataset as `ChimpBrain`. For each specimen, samples from 16 different tissues representing the hippocampus, amygdala, cerebellar cortex, mediodorsal nucleus of thalamus, striatum and 11 areas of neocortex were sequenced. The samples belonging to the medial dorsal nucleus were removed before running `TreeTerminus`. The `lfc` change was computed by taking 5 cerebellum samples as the first group and the remaining 68 samples belonging to other tissues as the second group. Batch effects were observed w.r.t specimen label and corrected using `sva` [71].

2.2.7 Construction of anti-correlation tree

All transcripts that had 0 counts across the samples and 0 counts across the inferential replicates for any sample were removed. For each sample k , Pearson correlation r_{ij}^k was computed between each pair of transcripts across the inferential replicates. r_{ij}^k is multiplied by -1 to give $r_{ij}^{k'}$ so that any pair of transcripts that had the highest negative correlation now have the largest positive correlation. To convert (anti)correlation into distance, $r_{ij}^{k'}$ is transformed as $d_{ij}^k = (1 - r_{ij}^{k'})/2$ or $d_{ij}^k = \frac{1+r_{ij}^k}{2}$. Unweighted Pair Group Method with Arithmetic Mean (UPGMA) [72] was used to create the anti-correlation tree with the implementation derived from the R package `phangorn` [73]. The mean for d_{ij}^k across samples is computed and fed as input to UPGMA in order to get the final tree. The AC tree could also have been created for every sample and then provided as input to the consensus tree algorithm to get the final tree, however

consensus tree algorithms do not scale well on the number of leaves. Since for most cases the number of leaves(transcripts) on trees would be in the order $10^4 - 10^5$, consensus tree algorithms would have taken a lot of time to converge.

2.2.8 Quantification pipeline

For the Chimpanzee dataset, only bam files were available as the raw data which were converted into fastq using `bamToFastq` in `bedtools` [74]. The quality control analysis for the `ChimpBrain` and `MouseMuscle` dataset was done using `fastqc` [75] and `multiqc` [76]. For creating the salmon indexes, gencode versions v26, vM25 and Pan_tro 3.0 were used for human, mouse and chimpanzee datasets. `Salmon` was used for quantification and generating 100 Gibbs replicates for each sample with a thinning factor of 100. All the pipelines used for analysis in this paper were created using `Snakemake` [77].

2.2.9 Mapping to gene families

The gene family labels for a gene were extracted using R package `biomaRt` [78] using `Ensembl` version 101. We want to extract the total number of unique gene families to which an inner node maps. A gene can map to more than one gene family and when an inner node maps to multiple genes; with atleast one gene mapping to more than one family, finding the unique number of gene families for that node is not trivial. Simply, taking a union of gene families across genes for an inner node is undesirable since this might lead to a situation where the number of reported gene families is larger than the number of mapped genes for some nodes. Between different trees, the interpretation of the distribution of the number of gene families to

which an inner node maps will also be biased towards the tree that has more nodes containing genes mapping to multiple gene families. Thus, in order to find the number of unique gene families associated with a node, we formulate this as the minimal hitting set problem [79]. We want to find the minimum number of gene families for a node in the tree, whose intersection with the gene families associated with every gene for that node leads to a non-empty set. We use the implementation provided by the `Python` library `PySAT` [80] to solve the minimal hitting set problem.

2.2.10 Differential expression analysis

Differential expression analysis is carried out on both the simulated datasets using `Swish` [57]. It is performed individually for the cuts obtained on the `Cons` tree by optimizing the objective functions defined in Section 2.2.5. The inner nodes belonging to a cut are directly used as the inferential units for the analysis, and their underlying descendant transcripts are not considered. The performance is evaluated using True Positive and False Discovery rates computed at different nominal FDR thresholds (0.01, 0.05, 0.1). The performance is also evaluated when the inferential units consist of genes, transcripts(`Txps`), and Terminus groups(`Terminus`). The true differential status of a transcript group is determined by computing `lfc` on its aggregated transcript counts w.r.t each condition in the `sim.counts.mat` matrix obtained from `Polyester` [66] and checking if $|\text{lfc}|$ is beyond a threshold. It is important to keep in mind that these evaluation metrics are not directly comparable across the methods as the base reference units are different.

2.3 Results

2.3.1 Overview of TreeTerminus

We briefly describe TreeTerminus pipeline as shown in Figure 2.3. Using the abundance estimates and inferential replicates for an RNA-Seq experiment that are generated by Salmon, TreeTerminus outputs a forest of trees, where leaves represent individual transcripts and internal nodes represent an aggregation of a transcript set. The trees are constructed such that, on average, the inferential uncertainty decreases as we ascend the tree branches. To construct trees for a single sample, first, a graph is created on the set of transcripts, with an edge denoting that they co-occur in at least one range-factorized equivalence class for that sample. The edge weight denotes the reduction in inferential uncertainty (Eq 1.1 in Section 2.2) that will be observed if the two transcripts/transcript groups were to be merged. The edges are then greedily collapsed from the graph creating a forest of trees, each representing a transcript group. The group consists of transcripts, such that any between any two transcripts there exists a path between them in the initial transcript graph at iteration I_0 . TreeTerminus provides two modes to output a forest of transcript trees across the set of samples for an RNA-Seq experiment. The `Mean` tree uses the mean reduction in inferential variance across samples as the edge weight for aggregating nodes whereas the `Cons` tree applies a Majority rule extended consensus algorithm on the individual trees obtained across samples for a given transcript group.

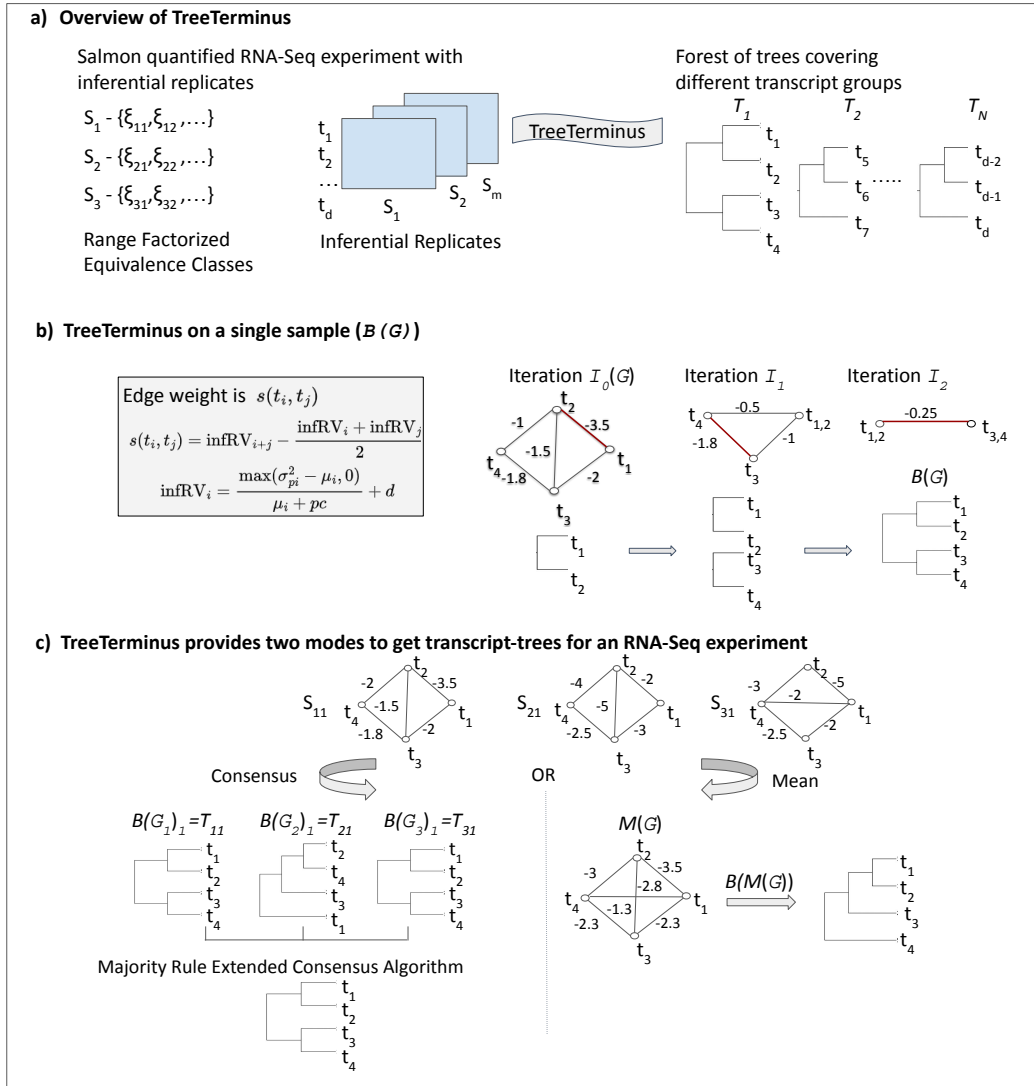


Figure 2.3: a) Schematic overview of TreeTerminus pipeline. Taking an RNA-Seq experiment as an input, that has been quantified with Salmon and for which inferential replicates have been generated, TreeTerminus outputs a forest of trees. b) Toy example demonstrating how from the transcript-graph (G), tree $B(G)$ shall be generated by TreeTerminus for a single sample. A forest will be the output when the graph consists of disconnected components or when some of the necessary conditions that are required for aggregating transcripts are not met. The colour red indicates the edge with the lowest weight which is chosen for aggregation at each iteration. c) When there are multiple samples - TreeTerminus provides two modes to generate the forest. In the Consensus mode, trees corresponding to each transcript group are first extracted and then passed as input into a majority rule extended consensus algorithm. In the toy example, there is only 1 transcript group with the forest consisting of a single tree. For the Mean mode, the graph $M(G)$ is constructed, where the edge weight is given by the mean of $s(t_i, t_j)$ across all samples. The forest is then obtained by applying the procedure B on $M(G)$ as demonstrated in part b of the figure.

2.3.2 Experimental Setup

Datasets We ran `TreeTerminus` on two simulated and two experimental bulk RNA-Seq datasets [69, 70]. TPM estimates extracted from GTEx V8 frontal cortex dataset were used as an input to generate the simulated datasets, containing 12 samples in 2 conditions with 6 samples in each. Two simulated data variations were created and referred to as `BrSimNorm` and `BrSimLow` respectively. The first experimental dataset is derived from two different mouse muscle tissues, consisting of 6 samples in each and we refer it to as `MouseMuscle`. The second experimental dataset is obtained for the brain tissues of Chimpanzee, consisting of a total of 73 samples. The two groups for our analysis consist of 5 and 68 tissues respectively. The datasets are described in detail in Section 2.2.6. The resource usage and time performance comparing `Terminus` to `TreeTerminus` are provided in Table (2.1, A2).

Table 2.1: Peak memory usage and running time for the different steps of `Terminus` and `TreeTerminus` (both `Mean` and `Cons` modes) on `BrSimNorm` dataset.

Method	Terminus		TreeTerminus (Mean)	TreeTerminus (Cons)	
	Group	Consensus	Group	Group	Consensus
Peak Memory (MB)	234	2721	3340	435	458
Time (h:m:s)	0:23:40	0:02:29	0:03:08	0:20:14	0:17:26

Creation of a baseline anti-correlation tree For comparison with `TreeTerminus`, we are unaware of any other existing methods that create transcript-trees. Thus, to create a baseline tree, we create a tree using anti-correlation (negative correlation) between transcripts computed on the inferential replicates, which we also refer to as the AC tree. The motivation for using anti-correlation comes from `mmcollapse` [56] which was also used as a baseline method for comparison in `Terminus` [30]. Since `mmcollapse` does not create trees, we use UPGMA

[72]. UPGMA applies a hierarchical clustering procedure on the leaf set to generate a tree, which based on the current input, will group an anti-correlated set of transcripts, leading to a reduced uncertainty. We thus believe the UPGMA tree is a fair baseline for comparison with the `TreeTerminus` trees, where uncertainty decreases as we ascend the tree. The exact details of constructing the AC tree are provided in Section 2.2.7.

Different tree methods that have been compared in this study In this study, we have used both the variations of `TreeTerminus`, with the unified tree obtained after running the procedure in Section 2.2.4 on the consensus and mean trees referred to as `Cons` and `Mean`. We evaluate their performance on several parameters. To see if there are any benefits of removing the constraints imposed in `Terminus`, we also compare with the unified trees obtained after running the consensus algorithm on the sample trees that were constructed with the variations of those constraints (Section 2.2.1). In the first variation, the consensus tree approach is applied on trees obtained using both constraints i.e `Filt` and `ES` which is referred to by `ConsFiltES`. In the second variation, we only apply the constraint `Filt` and the corresponding consensus tree that is created is referred to by `ConsFilt`. The Anti-Correlation (AC) tree serves as the baseline tree. We create trees for all the above-mentioned methods for both the simulated and experimental datasets. We also compare with `Terminus` groups (`Terminus`) and transcripts (`Txps`) whenever possible. The total number of transcripts covered by different methods across the different datasets is provided in Table A3. `Mean` tree covers the most transcripts followed closely by `Cons` tree while `ConsFiltES` tree and `Terminus` cover the least number of transcripts. To facilitate the comparison between the methods, we ensure that all trees cover the same transcripts. A transcript set is created by taking the union of the transcripts covered by the trees in the set $\{\text{Mean}, \text{Cons}, \text{ConsFilt}, \text{ConsFiltES}, \text{AC}\}$, and groups in `Terminus`, with `Txps`

referring to that transcript set. To the children of the root node of the above trees, we append any transcripts that were present in `Txps` but missing in the transcripts covered by the individual trees. Similarly, nodes in `Terminus` consist of all nodes output by `Terminus` along with any transcripts present in `Txps` but not covered by `Terminus` groups.

2.3.3 TreeTerminus nodes have the lowest mean inferential variance

The violin plot in Figure 2.4 compares the distribution of $\log_2$ MIRV (mean inferential relative variance) for the inner nodes across trees stratified by their height for the `BrSimNorm` dataset. For comparison, $\log_2$ MIRV for genes and transcripts has been plotted along with the trees at each height. The transcripts have the highest range of variation in the mean inferential relative variance (MIRV) and as expected have the largest values. Genes have the lowest MIRV with the majority of genes having the lowest possible value of 0.01 that can be obtained using the default thresholds(Equation 1.2). Among the trees, the nodes of `Cons` and `Mean` trees have the lowest MIRV across different heights followed by `ConsFilt` and Anti-Correlation trees (AC). The nodes belonging to `ConsFiltES` tree have the highest MIRV. As expected, going up the tree, MIRV decreases across the methods getting closer to gene MIRV levels. These trends are summarized by the median values of MIRV in Table A4. The above pattern is also repeated in the `BrSimLow` dataset(Table A5, Figure A1). A similar trend is observed for the `MouseMuscle` dataset (Figure A2, Table A6). The only change is observed on the nodes at height greater than or equal to 5 for the `ConsFiltES` tree which shows the most downward shift in MIRV distribution. It is important to note that that across all the datasets, `ConsFiltES` has the lowest the number of nodes. For the `ChimpBrain` dataset, a very similar distribution is observed for the trees

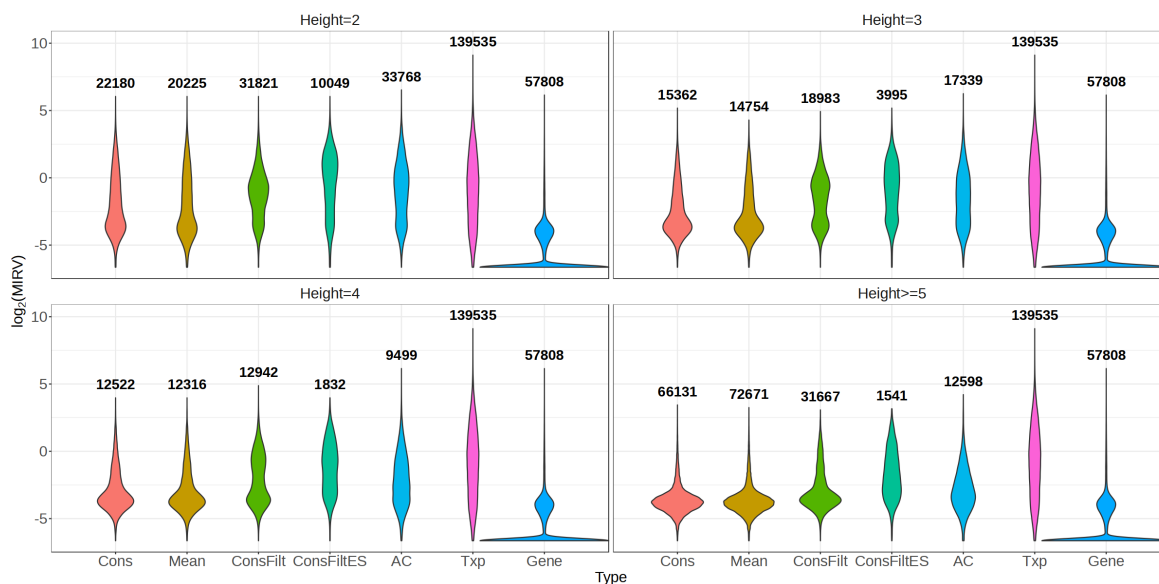


Figure 2.4: Distribution of $\log_2$ MIRV(mean inferential relative variance) across the inner nodes stratified by their height for different trees for the BrSimNorm Dataset, with the total number of inner nodes belonging to a method at a given height written on top of the violin plot. Also plotted for comparison at each height is the distribution of lg of MIRV for the transcripts and genes. across all the methods and a modest downward shift in the upper tail is observed as the height increases. The MIRV though is very low for the height 2 inner nodes to begin with(Table A7, Figure A3). However, a considerable downward shift in the distribution of MIRV is still observed for the inner nodes compared to the transcripts.

2.3.4 TreeTerminus nodes map to relatively fewer genes and gene families

We next look at the distribution of the number of uniquely mapped genes at the inner nodes located at different heights across the trees. The mapped genes associated with a node are found by mapping the descendant leaves (transcripts) for that node to the genes. Figure 2.5, A4, A5 and A6 plots this distribution for the different datasets. The proportion of nodes that map to more than one gene is considerably higher for the AC tree compared to others, with the number of nodes at height two that map to two genes, tens to hundreds of times larger for the anti-correlation (AC) trees. The increase in the relative number of multi-gene mapping nodes is also the highest for the

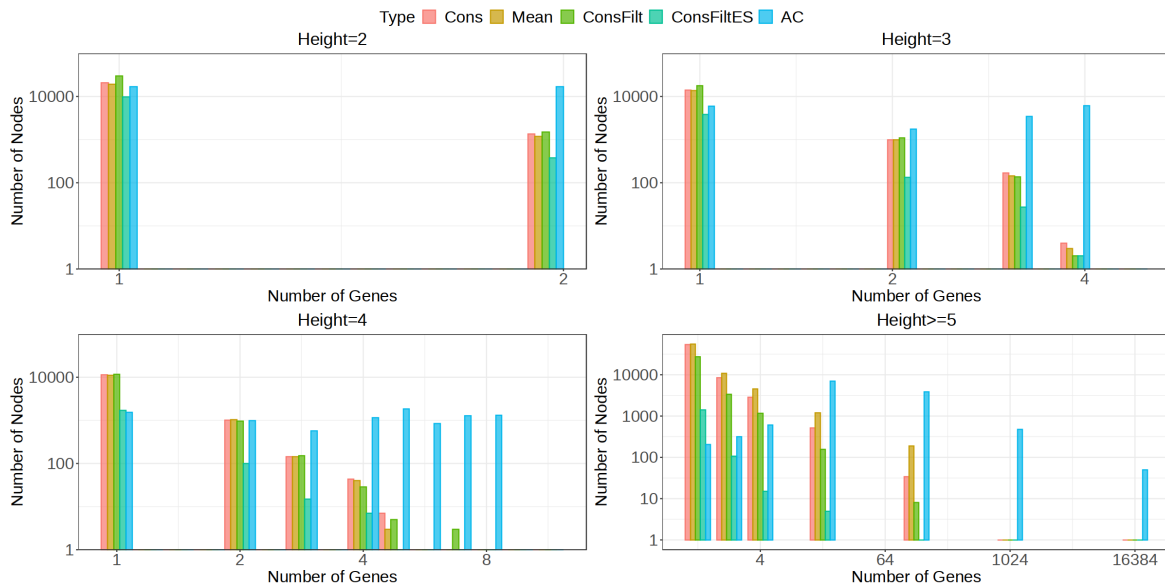


Figure 2.5: Comparison of different methods with respect to the number of genes to which an inner node in the tree maps for the `BrSimNorm` dataset, stratified by their height. The x-axis represents the number of unique genes that transcripts belonging to the inner nodes map to and y-axis represents the frequency of such mappings at a given height for a tree. For all the inner nodes located at a height greater than or equal to 5, the number of unique genes was binned using the set $\{1, 2, 4, 16, 128, 1024, 16384\}$, with the bin representing a number of unique genes less than or equal to the bin but larger than the bin left to it.

AC tree and it has the highest magnitude of the number of unique genes to which an inner node can map. With the exception of AC tree, the proportion of nodes that map to only one unique gene dominate nodes mapping to multiple genes across different heights in the other trees. The AC tree has the highest number of inner nodes that map to more than one hundred genes across the datasets (Table A8). No such nodes exist in the other trees obtained across the datasets, barring the `MouseMuscle` dataset, for which there exists a reasonable number of such inner nodes, with some even mapping to more than a thousand genes. When we looked at the genes mapping to these nodes, we found that a large proportion of such genes were either predicted or pseudogenes, which was not necessarily the case for most of such nodes in the AC tree.

We also explored the distribution of the number of gene families to which inner nodes belonging to different trees map in Figure 2.6. The procedure for finding the gene families that

map to a given inner node is described in 2.2.9. The number of gene families has been binned into bin numbers $\{1, 2, 4, 10, 100, 500, 1000, 12000\}$ so that each bin represents the number of unique gene families less than or equal to the bin but larger than the bin left to it. For the `MouseMuscle` dataset, the number of nodes mapping to more than one family is the largest for the AC tree and the difference in the number of nodes increases between AC and the remaining trees as the magnitude of the number of gene families they map to is increased on the x-axis. While the AC tree has more than 5000 nodes that map to $(10, 100]$ gene families, this number is only 124 for the Mean tree and much less for the other trees. There are more than 500 nodes that map to more than 100 gene families for the AC tree while hardly any such node exists for the other trees. A similar trend is observed for the `ChimpBrain` dataset, while for the AC tree there are more than 2500 nodes that map to more than 10 gene families, such nodes don't exist for the other trees. The nodes belonging to AC tree thus map to many more gene families. While the nodes belonging to the other trees map to a fewer gene families, we have not explored them in the current work.

2.3.5 Cuts obtained from TreeTerminus trees are among the top performers when optimizing for the different objective functions

In order to do any downstream analysis and interpret its results, we require discrete inferential units. The discrete inferential units can be obtained by finding a cut on the tree that solves for an objective function which optimizes a node metric. The inferential unit that represents a transcript group might represent a stronger signal w.r.t a node metric that the end user is interested in optimizing as compared to aggregation for that metric on the individual transcripts contained in

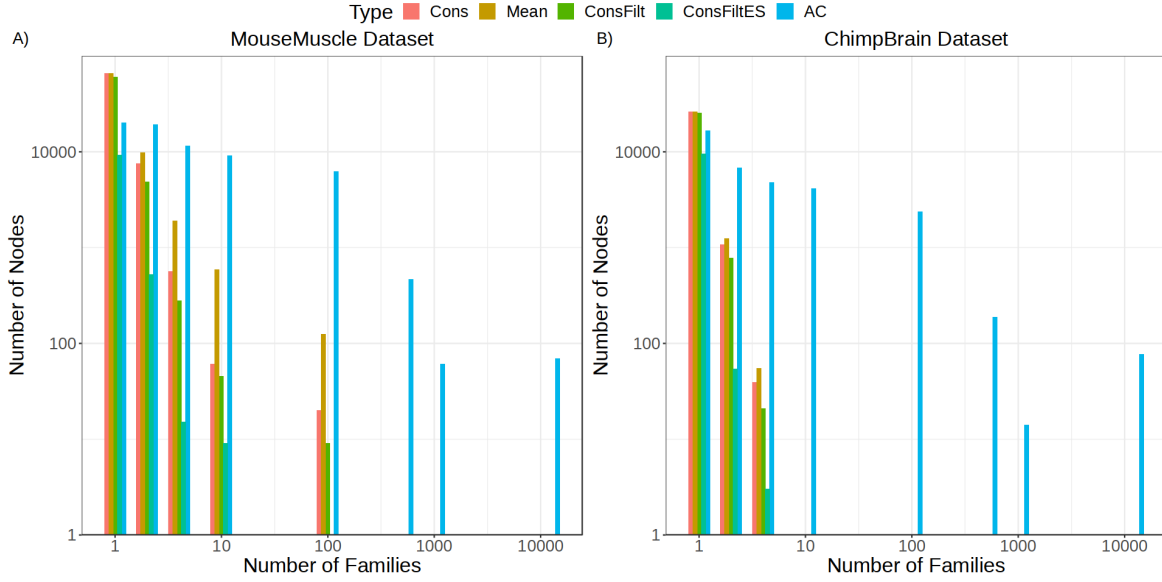


Figure 2.6: Comparison of different tree methods with respect to the number of gene families to which an inner node maps. The number of gene families that map to an inner node has been binned into the set $\{1, 2, 4, 10, 100, 500, 1000, 12000\}$, with the bin representing the number of unique gene families less than or equal to the bin but larger than the bin left to it. The x-axis represents the bin and the y-axis represents the total frequency of inner nodes mapping to the gene families in that bin for a given tree. This has been plotted for A) `MouseMuscle` Dataset and B) `ChimpBrain` Dataset.

that group. The type of objective function that we optimize is described in detail in Section 2.2.5

and requires a metric as input for every node in the tree. Specifically, in this manuscript, we have used two metrics to demonstrate the benefits of our proposed tree structure. Figure 2.7, A7 plots the optimal values and the cut sizes obtained by solving for the objective function that minimizes the sum of metric `irv_height_desc` (Section 2.2.5) on the different trees for each dataset. Also plotted for comparison are these values when the cut consists of only the transcripts(`Txp`). We want to find nodes that have a low mean inferential relative variance and at the same time are at a level close to the leaf in order to provide the finest resolution for downstream analysis.

The cuts for different trees have been obtained by using γ values from the set $\{0.05, 0.1, 0.5, 1, 5\}$.

The highest(worst) optimal value is seen for the cut consisting of transcripts, whereas the cuts obtained for `Cons`, `ConsFilt`, `Mean` trees consistently have the lowest(best) optimal values.

The relative differences in the optimal value between them is smaller compared to differences in optimal values for the cuts obtained from the other trees. As γ increases, the size of the cut increases as expected along with the increase in the optimal value. Further, the differences in the optimal values and cut sizes between the methods also decreases, with these values individually becoming comparable at $\gamma = 5$ across the trees. We also looked at the distribution of the values of the metric in the cuts for the different trees and the transcripts in Figure A8. A large proportion of nodes for the `Cons`, `Mean`, `ConsFilt` trees show a very low value for the metric compared to the other methods, showing that the net value of the objective function is not dominated by just a few outlier nodes.

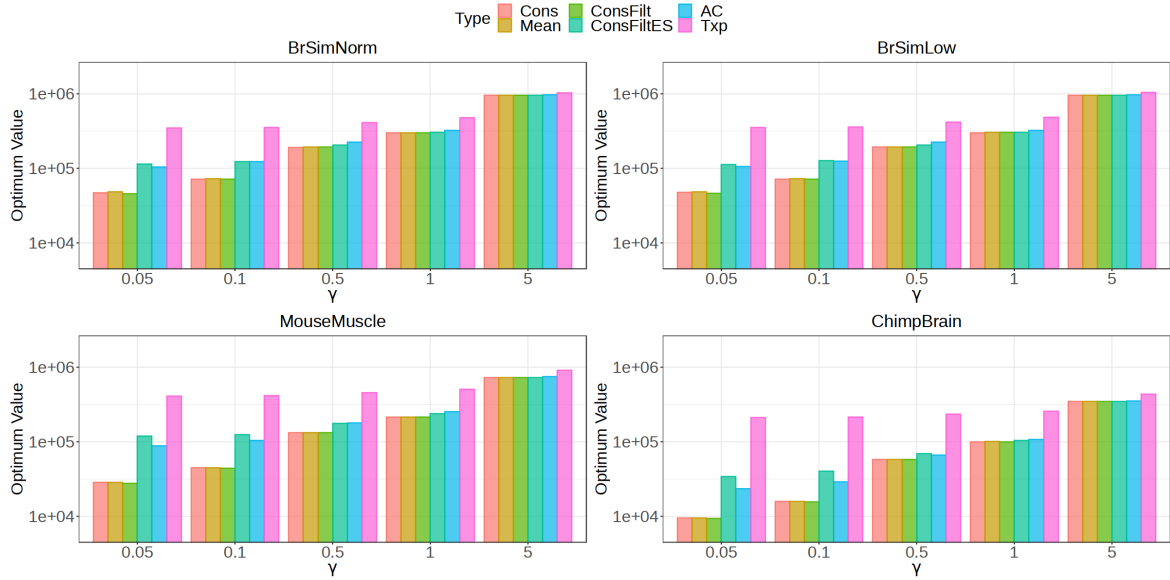


Figure 2.7: Barplot representing the values for the objective function on the cut obtained by minimizing the sum of metric - the sum of the metric (`irv_height_desc`) on the different datasets. For each dataset, optimal values of the objective function are compared across the methods for a range of γ values. Also plotted for comparison is the optimal value obtained by using transcripts(`Txp`) as the cut.

For the second objective function, we find nodes that have a high log fold change. We compared the optimal values and the corresponding cut sizes, obtained by solving for the objective

function that maximizes the sum of the metric `lfc_desc`, which finds nodes (see Section 2.2.5) for the different trees on each dataset in Figure 2.8. Also compared are the values that are obtained when the cut consists of - Terminus (`Term`) groups and the transcripts(`Txp`). The cuts obtained from the `Cons` tree have the highest (best) optimal value followed closely by the `Mean` tree. These are then followed by the cuts obtained for the `ConsFilt` and `AC` tree. The cuts obtained for `ConsFiltES` tree, `Terminus` and `Txps` have the lowest (worst) optimal values. An opposite trend is observed for the cut sizes, with `Txps` having the largest size followed by cuts from `Terminus`, and `ConsFiltES` tree.

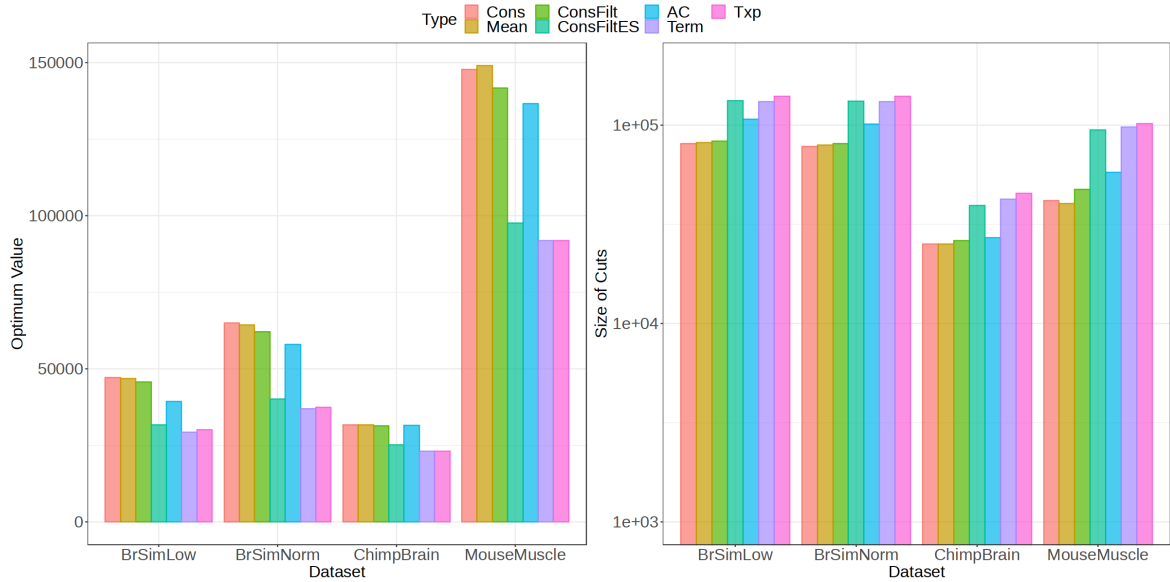


Figure 2.8: Comparing the performance on the cuts across different trees obtained by solving the objective function that maximizes the sum of metric CPM log fold change multiplied by the number of descendant leaves (`lfc_desc`) for the nodes in a cut on the different datasets. The performance is also compared when the transcripts and `Terminus` groups are taken as the cut. A) Value of the Objective Function. B) Size of the cut.

2.3.6 An optimal performance is observed for differential expression analysis on the cuts obtained from TreeTerminus trees

We also evaluate the performance of doing differential expression analysis on the base inferential units consisting of cuts obtained from the `Cons` tree along with genes, transcripts and `Terminus` groups for the `BrSimLow` and `BrSimNorm` datasets (Table 2.2, A9). For the `BrSimLow` dataset, we observe that the genes have the lowest FDR and a high TPR, while the `lfc_desc` cut has the highest TPR. Similarly, the cuts obtained for `irv_height_desc` at the various γ values have comparable FDR and higher TPR compared to `Txps` and `Terminus`. For the `BrSimNorm` dataset, barring the 0.01 FDR threshold, a similar trend is observed across the cuts compared to `Terminus` and `Txps`. It is important to note that since the reference units are different, the metrics are not directly comparable between the methods. We thus next compared the number of transcripts belonging to the significant nodes in the `lfc_desc` cut that are also true positives but are not covered by either `Txps` or transcripts belonging to `Terminus` for the 0.1 FDR threshold. We find more than 2200 such transcripts while only 278 true positive transcripts exist in `Terminus` that are not covered by `lfc_desc` cut. These $2K$ transcripts map to around 828 inner nodes in the cut which in total map to more than 9500 transcripts. However, this cut allows us to recover signal w.r.t these true positive transcripts at a higher level in the tree which would have been lost by looking at just the transcripts.

Table 2.2: True Positive Rate and False Discovery Rate for the different methods at nominal FDR cutoffs 0.01, 0.05, 0.10 for the BrSimLow Dataset. The cuts are obtained by optimizing the metric `irv_height_desc` at different γ values and `lfc_desc` on the Cons tree. The performance is also computed when the inferential units consist of genes, transcripts and terminus groups.

Method	FDR			TPR		
	0.01	0.05	0.10	0.01	0.05	0.10
<code>irv_height_desc</code> ($\gamma = 0.05$)	0.006	0.033	0.070	0.283	0.398	0.455
<code>irv_height_desc</code> ($\gamma = 0.10$)	0.005	0.034	0.071	0.273	0.385	0.445
<code>irv_height_desc</code> ($\gamma = 0.50$)	0.006	0.034	0.074	0.244	0.373	0.432
<code>irv_height_desc</code> ($\gamma = 1$)	0.007	0.040	0.076	0.235	0.369	0.426
<code>irv_height_desc</code> ($\gamma = 5$)	0.007	0.040	0.083	0.220	0.356	0.418
<code>irv_height_desc</code> ($\gamma = 10$)	0.001	0.024	0.065	0.124	0.301	0.390
<code>lfc_desc</code>	0.006	0.037	0.081	0.314	0.437	0.495
Genes	0.002	0.025	0.061	0.256	0.413	0.491
Txps	0.008	0.039	0.080	0.208	0.345	0.407
Terminus	0.008	0.038	0.079	0.223	0.352	0.415

2.4 Discussion

TreeTerminus organizes the transcripts in a tree structure, with the leaves representing the transcripts. The tree accounts for experiment-wide inferential uncertainty and provides the flexibility to analyze data at nodes that are at different levels of resolution, that are best supported by data for the downstream analysis of interest. The cut given by the dynamic programming (DP) implementation on the objective functions used in this paper provides one possible way to get a set of nodes that can be used for downstream tasks. The DP is generalized and the base metric inside the objective function can be easily replaced by a user-defined function. Furthermore, the end user can also define a completely different approach to find a cut from the tree.

We have provided two different approaches for getting the transcript trees as output from

an RNA-Seq experiment, `Mean` and `Cons`. They both cover a similar number of transcripts and have comparable performance across the different datasets for the various analysis that have been explored in this paper. They have superior performance compared to the consensus trees that were obtained when the constraints were kept on the various evaluation metrics and cover the largest number of transcripts. The sub-par performance for the other methods, especially `ConsFilterES` and `Terminus` perhaps indicate that these methods under-aggregate. While no other method exists that arrange the transcripts in a tree-like structure, for comparison we also provide the AC tree as a baseline, which is constructed using UPGMA - a well-known method for tree construction in phylogenetics. However, construction of the AC tree is both highly memory and time intensive, with the tree construction process not taking into account the equivalence class information. As a result, we observe that the nodes belonging to AC tree map to a large number of distinct genes and gene families, not offering much biological interpretation. The nodes belonging to trees constructed from `TreeTerminus` map to a smaller number of distinct genes and gene families, even though annotation of the underlying organism was not a part of the input.

There do exist some areas where the underlying tree construction methods can be improved further. Taking the mean of difference in inferential relative variance on all samples for constructing `Mean` tree might prevent transcripts from aggregation that had high uncertainty but were expressed only in the samples belonging to a population that was not in the majority in the experiment. This can be replaced by a weighted reduction in inferential relative variance, where the mean is computed over the samples in which the transcripts are expressed. A limitation with the `Cons` tree is the consensus tree algorithm that requires all the underlying input trees to span the same leaf set, causing us to modify the individual sample group trees that were provided as input.

An alternative to consensus algorithms can be supertree methods such as STELAR [81] and FastRFS [82], which don't require that all the input trees should cover the same leaf set. However, STELAR is very slow for trees that span large leaf sets and FastRFS provides an unrooted tree as an output, which makes using them as a direct replacement not trivial. Modifications to these approaches can be explored in the future.

Differential expression or differential abundance analysis is an important downstream application of RNA-Seq and other sequencing-based datasets. Various differential testing methods that leverage a tree structure [83–86] have been proposed, reporting an increased power. In microbiome analysis, investigators have tried to associate nodes, located at different levels on the tree built through phylogenetic analysis, with a response variable of interest [87], with some using replicability as a metric to determine optimal levels of aggregation [88]. We believe that the `tree(unified)` provided by `TreeTerminus` can be used with these methods for improved differential analysis in RNA-Seq. While we do perform differential testing on the cuts, they are not necessarily optimized for that task. Further, `TreeTerminus` in the future can also be extended to tagged-end scRNA-seq protocols where the trees will be constructed on genes rather than transcripts, owing to read mapping ambiguity at the gene level itself since the exons of one gene can overlap with exons or introns of other genes [89], as these are 3' - biased, often single end reads.

Chapter 3: Tree-based differential testing using inferential uncertainty for RNA-seq

3.1 Introduction

RNA-seq has become the *de facto* technology for measuring the expression profiles of different genomic features. Finding the features that differ in expression between the biological samples under different conditions, such as normal versus tumor samples, is called differential analysis and is a fundamental task downstream of quantification. The starting point of differential analysis is to accurately estimate the abundance of the features to be tested, with genes and transcripts being the most common features of interest for most RNA-seq analysis. A gene can express multiple isoforms or transcripts due to alternative splicing, where the transcripts can comprise an overlapping set of exons and thus share sequences. Commonly used RNA-seq quantification methods such as RSEM [52], Salmon [28] and kallisto [25] probabilistically assign each observed fragment to the transcripts using maximum likelihood or Bayesian inference methods. A probabilistic model is required because a substantial fraction of the sequenced reads can multi-map (align similarly or equally well to multiple reference transcripts) as they arise from the regions and sequences shared between the transcripts. Thus, there is ambiguity concerning the true locus of origin of such reads, leading to uncertainty when trying to quantify

transcript abundances. Uncertainty can also exist for gene abundance estimates when the reads belong to shared sequences within the genes (for example, homologous genes); however, in general, gene-level estimates will be more precise compared to transcripts. One approach that estimates the uncertainty associated with the measurement of the abundance estimates is to generate additional samples called inferential replicates through different sampling strategies. These include bootstrap sampling or posterior sampling using MCMC/Gibbs sampling. Such capabilities are provided by several existing quantification methods [25, 28, 52, 53].

The robustness and accuracy of any downstream analysis, such as differential testing, is directly impacted by the quality of the abundance estimates. The methods designed for differential transcript expression testing that utilize inferential replicates report a more robust performance than the methods that do not include them [57, 90–93]. However, if inferential replicates are incorporated, then we might observe a reduced power for the transcripts that exhibit high uncertainty since we might not be confident of the observed differential signal. In such cases, we might be able to discover more existing differential signals by aggregating transcripts that share reads together into a transcript group. This idea was first proposed in *mmcollapse* [56], also described by Roberts and Watson [55] and further expanded in *Terminus* [30], where these groups that contain multiple transcripts, rather than single transcripts, form the feature set for differential analysis.

Expanding upon the above motivation, we recently published *TreeTerminus* [29], which outputs a forest of transcript trees for the samples in an RNA-seq experiment. The leaves represent the transcripts, and the inner nodes represent the aggregated group of the constituent transcripts rooted at each node. The tree encodes different layers of resolution for the interpretation of the abundance of transcriptional groups, with uncertainty generally decreasing as the tree is

ascended. This tree can be utilized to find differentially expressed nodes between conditions of interest, where the nodes can consist of both leaves (transcripts) and inner nodes (transcript groups). It is possible that if the differential signal exists at a transcript, then it might also propagate at the inner nodes. A finer resolution, i.e., being as close as possible to the leaves, is, however, preferable for many analyses. The inner nodes should be only part of the output when they constitute the subsets of transcripts whose signal gets masked, primarily due to high uncertainty. Otherwise, the transcripts and not their parent nodes should be selected. Thus, a tree-based differential testing method should be able to find such nodes in a data-driven manner.

While several tree-based differential testing methods have been proposed, they differ in the error rates they control, the underlying objective functions they seek to optimize, and the data modalities they intend to target. One class of methods focuses on testing for the global null hypothesis, meaning that for a given inner node, the null hypothesis for all the descendant leaves is true [94]. These methods use a top-down strategy where they descend an inner node only if the null hypothesis at that node is rejected and continue testing until the null hypothesis cannot be rejected or a leaf node is reached [86, 95–98]. While some methods [86, 95] control the family-wise error rate (FWER) on the rejected set, others [96, 97] control the false discovery rate (FDR). More recently, [98] introduced and controlled a new error rate called selective false discovery rate, which is the expectation of the proportion of falsely discovered nodes (FDP) across each level of a tree. Applying these methods can lead to accepting the null hypothesis at a node in the tree, which can consist of children that contain the differential signal but exhibit a contrasting direction of sign change between the conditions of interest (e.g. two sibling nodes for an inner node that have similar abundance and strong fold change but with opposite signs, can reduce or nullify the strength of the signal at the parent node.). This is very likely to happen when we do

the hypothesis testing or compute the p-value at each node of the tree individually. We might even end up not rejecting the null hypothesis at the root node, even if there is strong evidence of the signal existing at a node situated at a lower height in the tree.

BOUTH [84] proposes an alternative to the above approaches by defining a new hypothesis test called the modified null hypothesis and controlling an error rate called the false selection rate. It employs a bottom-up procedure and examines whether for a node the null hypothesis is true for all the descendant nodes not previously detected (detected implies that the null hypothesis is rejected) at the lower level. It identifies and outputs “driver” nodes such that none of its ancestor nodes is detected. If the driver nodes were to be analyzed, then the features might provide a coarse-grained level of analysis, especially if a clearer signal already exists at a lower height in the tree. On the other hand, it is not clear which descendant nodes corresponding to a driver node should be analyzed to provide a finer resolution of analysis.

Another category of methods aims to leverage the tree structure to enhance power and control the FDR at the leaf level [83,85,99]. `StructFDR` [99]) and `zazou` [85] initially convert the p-values into z-scores and then propose smoothing on the leaves using a tree-based correlation structure. `treeclimbR` [83] proposes multiple candidate sets, performs multiplicity correction on each set, and selects a final candidate set based on a set of criteria. It rejects the null hypothesis for all the leaves belonging to the inner nodes in the selected candidate set. `TEAM` [100] also proposes another approach for controlling FDR on an aggregated tree but in a different context than the methods described so far. The leaves in the tree are called bins, which consist of cells from both conditions. The tree structure aggregates bins and not the features as we go higher up the tree. The null hypothesis tested at each bin is if the PDF of features is different between the two populations in that bin. If the null hypothesis at a bin can’t be rejected at a lower level, then

such bins are aggregated and the hypothesis is tested at the aggregated bin. Importantly, if the aggregated bin is rejected at a higher level in the tree, then the method rejects all the underlying leaf bins. Thus, all these methods cannot be directly applied to our use case. There is uncertainty in measuring the abundance estimate of the transcripts, thus we do not know which transcript corresponding to the node may be differentially expressed. As a result, passing the information from an inner node to all the descendant leaves might lead to an exaggeration of the false positives or missing the branches where there exists a signal at a node located at a higher height in the tree, to control for the false positives on the leaf set.

There exists another class of methods that fit regression models to compositional data for the covariates using a tree-guided regularization in a maximum likelihood (`trac`- [87] or Bayesian setting (`tascCODA`- [101])). For `trac`, the parsimony of feature selection (where more nodes at greater heights result in a lesser number of total nodes) depends on the weight assigned to regularization during model fitting. For both `trac` and `tascCODA`, when an inner node is selected, its learned coefficients are passed equally to its underlying leaves. In addition to some of the limitations mentioned above that also exist for these methods, both these methods do not produce p-values. This can complicate comparisons with other methods. `trac` requires cross-validation to determine appropriate hyperparameters for the model, thus requiring a substantial number of samples, which may not be feasible for many RNA-seq experiments. `tascCODA` also does not scale computationally for datasets with a large number of features, such as RNA-seq.

To overcome some of the above limitations, we introduce our differential testing method, `mehenDi`, designed to output a set of nodes called the selected nodes from the tree. The selected nodes are differentially expressed and can comprise both transcripts and inner nodes, with no two nodes having an ancestor/descendant relationship. It utilizes the tree/s obtained from `Tree`

Terminus, and applies the Swishmethod for hypothesis testing, accounting for inferential uncertainty (though it is conceptually compatible with other differential testing methods that incorporate inferential uncertainty). The selected nodes are determined in a data-driven fashion by traversing the tree in a top-down manner using a set of rules to maximize the signal that can be extracted from the data while controlling for the uncertainty associated with measuring the abundance estimates. The selected inner node represents a subset of transcripts whose signals are masked at lower levels due to inferential uncertainty, thus we do not report the differential status of the constituent transcripts. We applied our method to experimental and simulated datasets, comparing its performance with the other tree-based differential testing methods and uncertainty-aware differential transcript/gene estimation methods. Our method demonstrates improved sensitivity compared to alternative approaches, particularly when the features consist only of transcripts or transcript groups. It identifies inner nodes exhibiting a strong signal for differential expression between the conditions, which might be overlooked when examining their underlying transcripts alone. The differential signal can often be restricted to only a subset of the annotated isoforms of a gene. Thus, our approach also mitigates the potential over-aggregation of signals if reporting is done at the gene level.

3.2 Methods

3.2.1 Overview

There can exist substantial uncertainty in estimating the abundance of the transcripts due to the presence of reads that can map equally well to shared regions of sequence between transcripts. This quantification uncertainty, in turn, can impact the quality and robustness of differential

analysis using transcripts as features. `mehenDi` provides a different approach to tackle this problem for RNA-seq data. For a given set of samples belonging to an RNA-seq experiment, `TreeTerminus` [29] arranges transcripts in a hierarchical tree structure/s based on quantification uncertainty (Figure 3.1A), with the leaves representing the transcripts and an internal node representing an aggregation of the transcripts underlying the node. The uncertainty is computed using the Gibbs/Bootstrap samples (called inferential replicates) which are generated for the RNA-seq samples using Salmon. `mehenDi` has been designed to perform differential testing directly over the tree/s produced by `TreeTerminus`. The uncertainty across the samples generally decreases on ascending the tree/s.

3.2.2 Problem Statement

For the given tree $\mathcal{T}$, constructed on the set of P transcripts with all nodes including the leaves labelled as $\mathcal{N} = \{1, \dots, N\}$ and $N \geq P$, we want to develop a method that outputs a set of **selected** nodes $\mathcal{C} \subset \mathcal{N}$ which are differentially expressed between the conditions of interest. We want to leverage the tree to select nodes that may consist of both transcripts and inner nodes, with the nodes determined in a data-dependent manner to maximize the signal from the data while controlling for uncertainty. At the same time, we do not want to over-aggregate, especially if the features with signal exist at a lower level in the tree, since a finer resolution is always preferable. Further, any two nodes in the selected set should not share an ancestor/descendant relationship, i.e., for any two nodes n_i, n_j in $\mathcal{C}$, $\Lambda(n_i) \cap \Lambda(n_j) = \emptyset$, where $\Lambda(n_i)$ represents the set of transcripts that are the descendants of the node n_i in the tree $\mathcal{T}$. This ensures that a set of reads is not counted twice in the selected feature set. Our method `mehenDi` is designed taking

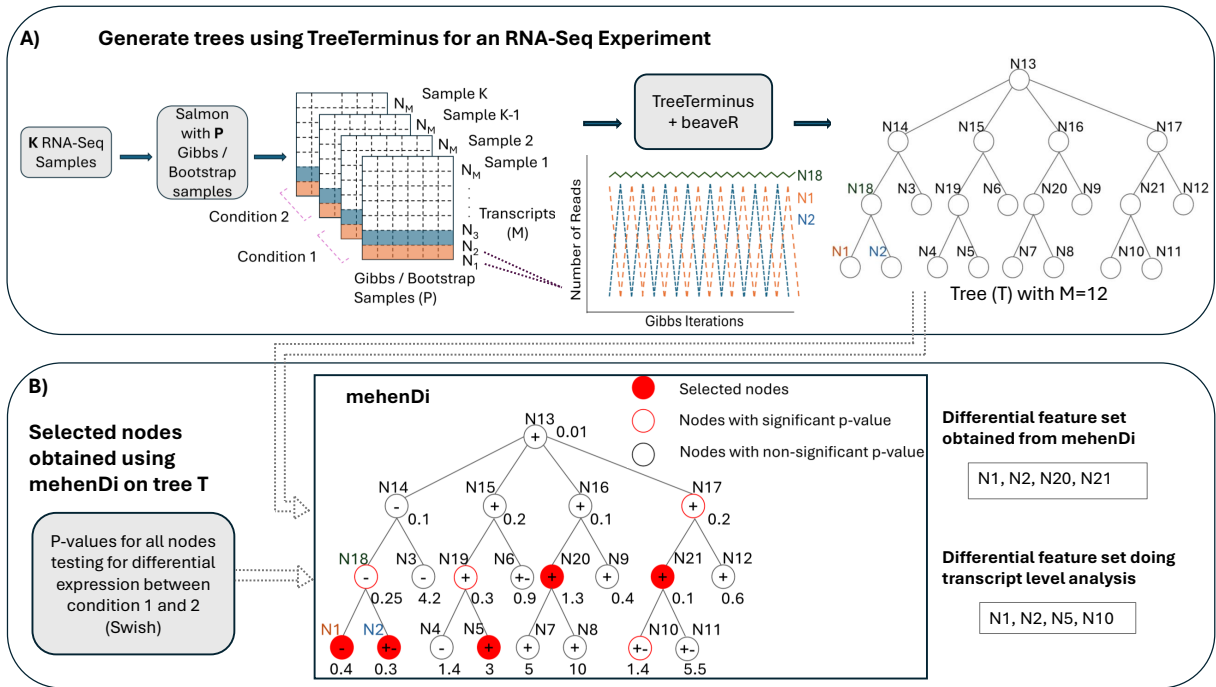


Figure 3.1: Schematic overview of mehenDi. A) Given a set of RNA-seq samples, TreeTerminus creates a tree that encodes the uncertainty structure by using the Gibbs/Bootstrap samples generated by Salmon to compute uncertainty. B) Given a tree, mehenDi outputs a set of selected nodes. The selected are differentially expressed between the conditions of interest and can consist of transcripts and inner nodes, with n . Using an example, with a tree having 12 leaves to demonstrate mehenDi. A highlighted node implies that it is selected by mehenDi, with a red outlined node indicating that the node has a significant p-value. The +, - on a node indicates the direction of sign change between the conditions, whereas +- indicates we are unsure about the direction. The values plotted along the nodes represent their mean inferential relative variance across the samples (meanInfRV). The meanInfRV threshold in this example is 0.4. The nodes N1 and N2 are chosen instead of node N18 since the meanInfRV of one of its children, node N2, is below the threshold of 0.4. The node N5 is chosen instead of the node N19 since the children have opposite sign changes. Node N20 is the only significant node in its branch and meets all the required criteria. Node N21 is picked since it has at least one non-significant child node.

the above principles into account. The inputs to mehenDi are the tree $\mathcal{T}$, and the p-values for differential testing between the conditions, mean inferential relative variance (meanInfRV), and the direction of sign-change between conditions for all the nodes N belonging to $\mathcal{T}$.

It is important to note that when mehenDi reports an inner node as selected, we do not imply anything about the differential status of its underlying descendant transcripts. The inner node should be interpreted as a standalone feature, the same way a gene is when doing

gene-level analysis, where we are not always trying to comment on the differential status of its transcripts. Not all transcripts of a gene might be differentially expressed. A selected inner node might suggest that the signal existed in at least one underlying transcript, but was masked due to uncertainty.

3.2.3 mehenDi description

We will now outline our method, `mehenDi`. We first compute the p-values, `meanInfRV` and direction of sign change for all the nodes in the tree. `Swish`, which employs inferential replicates, is used to compute the p-values for the tree nodes. The counts for an inner node are obtained by aggregating the counts of the underlying transcripts. We recommend computing the p-values separately for the inner nodes and leaves. For the transcript-group corresponding to a given node, the direction of sign change is examined for each inferential replicate. This is achieved by computing the log fold changes between the biological samples corresponding to that inferential replicate, with each biological sample being associated with a fixed label of interest. If a direction emerges in a certain proportion of inferential replicates beyond a threshold, the node is marked as confident, with that direction determining its sign change. Otherwise, the nodes are labelled as non-confident and are not assigned a direction of sign change. This is controlled by the parameter `minP`, which is set to 0.70 by default. Tree nodes with p-values below a specified threshold (`pthresh`) are identified and labelled as significant. The set of selected nodes, denoted as $\mathcal{C}$, will be a subset of the significant nodes. The p-value threshold that is used for determining the significance of a node is computed by applying a multiple-testing procedure such as Benjamini-Hochberg (BH) [102] on the leaf nodes that controls the FDR on

the leaves at rate α .

`mehenDi` implements a top-down procedure starting from the root node, which first determines the branches which contain at least one significant node. Proceeding with this approach, it traverses each selected branch and, upon encountering a significant inner node, checks for the following criteria:

1. It has at least one non-significant node among its children.
2. All the descendant nodes that are marked confident have the same direction of sign change.

This also means that all or a subset of descendant nodes can be non-confident.

3. All the child nodes have a meanInfRV above a certain threshold (*mIrvThresh*).

If all three criteria are met, that node is selected, and traversal along this branch terminates. Otherwise, this branch is traversed (recursively), checking the above criteria on each newly encountered significant node until either the node is selected or a leaf is reached. If a leaf is significant, it is added to the list of selected nodes. A small, illustrative example demonstrating the above algorithm is provided in Figure 3.1B.

3.2.3.1 Explanation of the different criteria

The main intuition behind `mehenDi` is to aggregate transcripts in a data-driven manner to recover the signal that could have been lost due to uncertainty. The first criterion ensures a finer resolution of analysis since, if all children of a significant node are also significant, then the signal already exists at a lower level and aggregation is unnecessary. Owing to uncertainty, the direction of the sign change of a node can be ambiguous, as it might not be consistent across the inferential

replicates. We thus assign a particular direction of sign change (positive/up, negative/down) to the nodes between the conditions of interest, only if we are confident. The direction and confidence of sign change of the descendant nodes should also have an impact in determining if a node is selected by `mehenDi` upstream. The second criterion helps to find a node having a consistent direction of the sign change across its underlying descendant nodes if they can be determined with certainty. This aids in having a clearer and more consistent biological interpretation of the node. The magnitude of effect size can also become smaller at a node if the underlying transcripts have an opposite direction of sign change and we should descend the tree to select nodes. The third criterion also checks for over-aggregation, since if we are certain about the abundance estimates for the nodes, we can confidently assess their differential signal and these nodes should not be aggregated. The default value of the parameter *mIrvThresh* is set to 0.40.

3.2.3.2 Computing the p-values

`mehenDi` can take p-values produced by any differential analysis tool as input. However, in this study, we have used `Swish` [57] for computing the p-values, since it takes inferential uncertainty into account. The p-values for the leaves and inner nodes are computed separately. The reasoning behind this choice is that the leaves have higher uncertainty compared to the inner nodes, especially the ones that appear higher in the tree (the height of a node refers to the maximum distance between the node and its underlying leaves). This leads to a relatively lower shrinkage of the test-statistic towards 0 for the inner nodes and increases the width of the distribution of the test-statistic used by `Swish` for computing the p-values for the inner nodes compared to those of the leaves. As a result, the p-values of the inner nodes will be smaller, on

average, compared to the leaves when the p-values are computed together.

3.2.4 Experimental Setup

3.2.4.1 Pipeline to generate trees

For the `ChimpBrain` dataset, only BAM files were available as raw data, which were converted into FASTQ using the `bamToFastq` command from `bedtools` [74]. The quality control analysis for `ChimpBrain` and `MouseMuscle` datasets was done using `fastqc` [75] and `multiqc` [76]. Salmon indexes were built on gencode versions v26, vM25, and `Pan_tro` 3.0 to analyze human, mouse and chimpanzee datasets [103, 104] respectively. Salmon was used to quantify and generate Gibbs replicates for the RNA-Seq samples. For each sample, 100 Gibbs replicates were generated using a thinning factor of 100. The trees from `TreeTerminus` were created using the `Consensus` mode. All the pipelines used for analysis in this paper were constructed using `Snakemake` [105].

3.2.4.2 Differential expression analysis

We compare the performance of carrying out the differential expression analysis using the different feature sets and tree-based methods on the analyzed datasets. The compared features comprise the selected transcripts (`Txps`), genes (`Genes`), `Terminus` groups (`Terminus`), and the best candidate, driver and selected nodes obtained by running the different tree-based methods on the unified tree $\mathcal{T}$ from `TreeTerminus`. The base set for `Terminus` comprises the discrete transcript groups and any remaining transcripts that belong to the transcript set of interest but are not covered by `Terminus` groups. For the tree-based methods, we compare

`mehenDi`, `treeclimbR`, and `BOUTH`. Median-ratio scaling [106] was used to normalize the counts. When scaling the counts for the nodes in the tree, we first compute the size factor only using the leaf nodes and then divide the bias-corrected scaled counts of the inner nodes by the size factor. This procedure is described in detail in Supplementary Section B.1.1. `Swish` is then used for generating p-values for the normalized counts. While `treeclimbR` and `mehenDi` require p-values for all the nodes in the tree, `BOUTH` requires the p-values only for the leaf nodes. The p-values were generated separately for the leaf and inner nodes using `Swish`. For `treeclimbR`, the parameter α is set to the nominal threshold, which aims to control the FDR at leaf nodes to extract the best candidates. We consider two feature sets for `treeclimbR` in our evaluation. `treeclimbR(N)` refers to the best candidate set obtained after running `treeclimbR`, and `treeclimbR(L)` consists of the descendant transcripts for the nodes in `treeclimbR(N)`. We also consider `treeclimbR(L)` in our analysis because `treeclimbR` is designed to control FDR at the leaf level, which, in this case, corresponds to the transcripts.

For `BOUTH`, we set the parameter *FAR* to the nominal FDR, which controls for a false assignment rate, and we then extract the driver nodes from its output. A driver node provides the highest level of resolution with none of their ancestors being associated for differential analysis. Many of the descendant nodes of a driver node at different resolutions can be labelled as detected for differential analysis, which can complicate the analysis when trying to decide which resolution node to select, thus a driver node is chosen for our analysis. For `mehenDi`, the nominal FDR threshold is used to select the p-value threshold (on the leaves) to obtain the selected nodes. For the other features (`Txps`, `Genes`, `Terminus`), `Swish` is applied directly to their base sets individually, and the selected set consists of features that have their adjusted p-values less than the given nominal FDR threshold.

3.2.4.3 Evaluation on the simulated datasets

Since the ground truth is known for the simulated datasets, we compute the True Positive Rate (TPR) and False Discovery Rate (FDR) on the selected features obtained from the different methods at the different nominal thresholds (0.01, 0.05, 0.1). When computing these metrics for the features that consist of transcript groups or the inner nodes in the tree, we only consider the status of the differential expression for that node or the transcript group in our evaluation. We do not evaluate the differential status at the underlying descendant transcripts or other inner nodes corresponding to the group or inner node. To obtain the ground truth for differential expression at an inner node, we first create the aggregated counts matrix for the tree $\mathcal{T}$, `sim.tree.counts.mat` using `sim.counts.mat`. The `sim.counts.mat` consists of the true transcript counts corresponding to sample groups, on which we are evaluating the differential expression. This is the same matrix that was provided as an input to `Polyester` to simulate reads. The count of an inner node in `sim.tree.counts.mat` is computed by aggregating the counts of all the descendant transcripts in `sim.counts.mat`. The inner node is differentially expressed if the absolute log fold change (LFC) for that node computed using `sim.tree.counts.mat` is larger than a threshold. The threshold is set to the LFC obtained at the tree's root node in the `sim.tree.counts.mat`.

3.3 Results

3.3.1 Running mehenDi on null simulation

We recommend computing p-values for leaves and inner nodes separately (see Methods) before running `mehenDi`. Through this analysis, we want to demonstrate why we recommend computing the p-values in the specified manner. We also aim to evaluate the false positive rate for `mehenDi` specifically comparing it to transcript-level differential analysis.

The simulation framework involves simulating reads from the human transcriptome for 12 samples belonging to two groups (which represent the conditions of interest), with 6 samples in each group. In the null simulation experiment, the fold change between the groups is fixed at 1 for all transcripts. We create a total of 10 null simulation experiments. The framework has been described in detail in the Methods section. Thus, by design, the simulation has no true signal for differential expression for the transcripts and any node deemed as significant by a method would be a false positive.

3.3.1.1 Inner nodes have smaller p-values if the hypotheses testing is carried out on all the nodes of the tree simultaneously

For each null simulation, we ran `Swish` on all the tree nodes simultaneously, and then the computed p-values across all the null simulations were concatenated into one vector. As expected, the distribution of the p-values across all the nodes is uniform (Figure 3.2A i). However, if we look at the distribution of the p-values at the leaves, there is a shift in the distribution towards the right for the leaf p-values and the left for the inner node p-values (Figure 3.2A ii,iii). Ideally,

irrespective of the subset of features that are picked, the distribution should be uniform. This happens because the width of the distribution of the test-statistic which is used by `Swish` is wider for the inner nodes compared to the leaves (Figure 3.2A iv). Thus, if the p-values are generated together for all nodes, we might inflate false positives for the inner nodes and false negatives for the leaves. We thus compute the p-values for the leaves and inner nodes separately and, as expected, both of these show a uniform distribution (Supplementary figure B1).

3.3.1.2 The False Positive Rate (FPR) obtained by `mehenDi` on the tree is comparable to running `Swish` on only the transcripts

We next looked at the False Positive Rate (FPR) reported by `mehenDi` on the null simulations and compared it with the FPR obtained by running `Swish` only on the transcripts. We varied the p-value threshold from $1e - 9$ to $1e - 1$, increasing the threshold by the power of 10 and then computed the FPR across the 10 simulations, shown in Figure 3.2B. The empirical FPR that is obtained across both settings is comparable to each other. This experiment demonstrates that `mehenDi` should not exaggerate the false positive rate compared to `Swish` on transcripts.

3.3.2 Comparing the methods on simulated datasets with signal

We next benchmark and analyze the simulated datasets with differential signals i.e. `BrSimNorm` and `BrSimLow` datasets (see Methods), which again consist of 12 samples belonging to two groups. Specifically, transcripts in `BrSimLow` show a relatively weaker signal for differential expression on average compared to `BrSimNorm`.

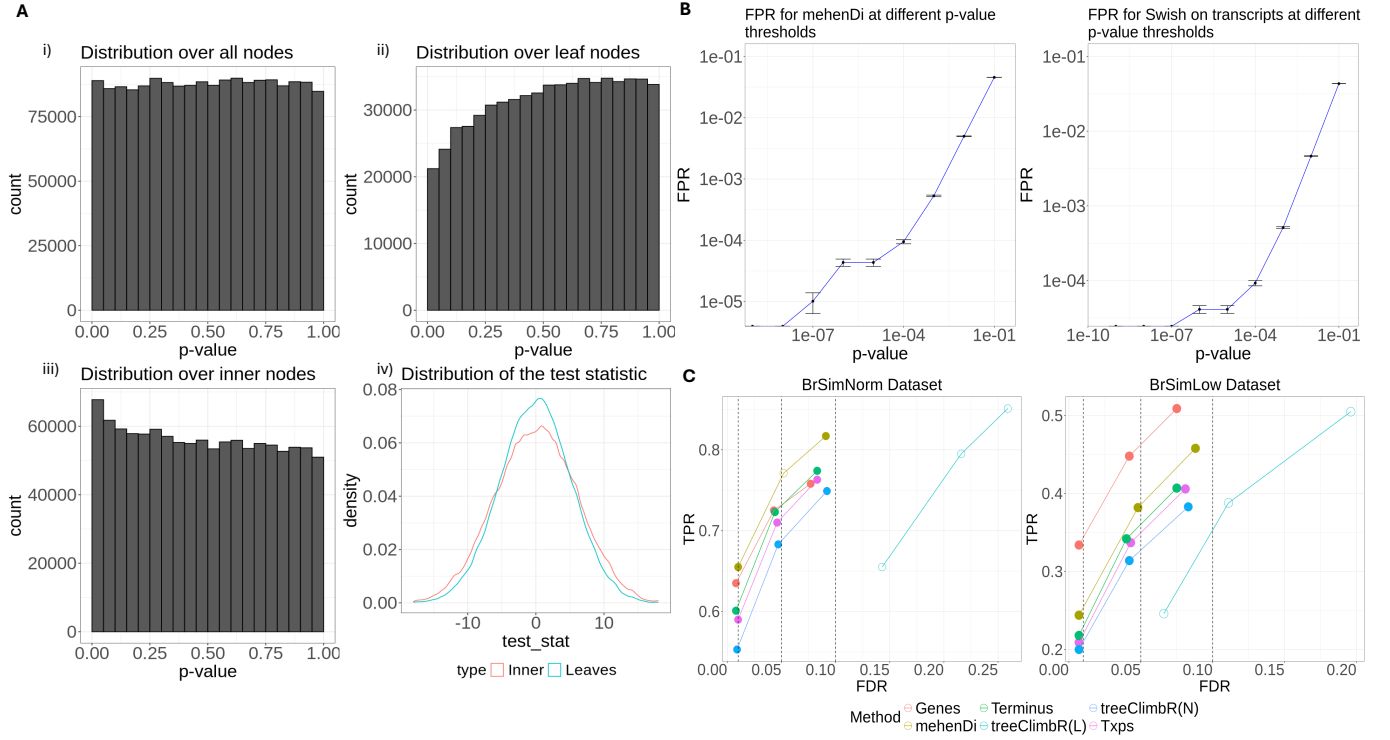


Figure 3.2: A) Distribution of p-values and test-statistic for Swish for the null brain simulation. i) Distribution of the p-values on the entire node set over the tree when all hypotheses are tested together using Swish. ii) Distribution of the p-values over the leaf nodes. iii) Distribution of the p-values over the inner nodes. iv) Distribution of the test-statistic on the inner and leaf nodes, which is used by Swish to generate the p-values. B) False Positive Rate (FPR) obtained for the null simulations by varying the p-value threshold. The bar along the y-axis represents the variation observed across the 10 simulations. C) True Positive Rates and Empirical False Discovery Rates at the different nominal FDR thresholds across the different methods and entities for the BrSimNorm and BrSimLow Dataset. Both the metrics have been rounded to 3 decimal places.

3.3.2.1 Features obtained from mehenDi show good sensitivity while controlling for FDR

We compare the features obtained after differential testing using different methods. The features consist of genes (Genes), transcripts (Txps, treeclimbR(L)) and transcript-groups (Terminus, mehenDi, treeclimbR(N), BOUTH). Txps refers to the transcripts obtained by running vanilla Swish only on the transcripts. When treeclimbR selects an inner node

as a candidate node, this implies that all of its descendant transcripts are also differentially expressed. Thus, in our analysis, we compare two feature sets corresponding to `treeclimbR`: `treeclimbR(N)`, which consists of only the best candidate nodes, and `treeclimbR(L)`, which consists of the underlying transcripts belonging to the nodes. This enables us to compare the performance of `treeclimbR` at both the node and transcript levels. On all the simulated and experimental datasets, `BOUTH` outputs the root node as the driver. Since the root node represents the aggregation of all the transcripts, interpreting the results from `BOUTH` does not provide any information. We first look at the number of true positive transcripts that are covered by the different methods on the simulated datasets (Supplementary figure [B2](#), [B3](#)) (excluding `BOUTH`). We observe that `mehenDi` yields the largest number of true positives, followed by `treeclimbR`.

We next compute TPR and FDR at different nominal FDR thresholds on the feature sets obtained from the different methods (Figure [3.2C](#)). For the `BrSimNorm` dataset, `treeclimbR(L)` and `mehenDi` have the largest TPR, while for the `BrSimLow` dataset, `Genes` has the highest TPR, followed by `treeclimbR(L)` and `mehenDi`. `treeclimbR(L)` has an inflated FDR on both datasets (since all transcripts belonging to a candidate node might not be true positives), whereas `treeclimbR(N)`, while controlling for FDR, does not seem to lead to that much overall improvement in sensitivity compared to `Txps`. `mehenDi` seems to have a good sensitivity while controlling for the FDR at the same time, with FDR only going slightly beyond the nominal FDR threshold of 0.05 for the `BrSimNorm` dataset. We have excluded the results of `BOUTH` in the figure since it outputs the root node at all the different nominal FDR values we have tried in our experiments. By design, the root node is not significant in the simulation and thus the TPR and FDR for `BOUTH` are 0 and 1 respectively.

We also individually varied the parameters minP and mIrvThresh (see Methods) for `mehenDi` and observed how the performance changed for the `BrSimNorm` dataset, shown in Supplementary figure B4. We first kept the parameter mIrvThresh constant to its default value 0.4 and varied minP from 0.6 to 0.9, incrementing it at an interval of 0.05. We however don't observe much change in the performance, with a very slight increase in TPR and FDR as minP is increased. We next kept the parameter minP constant to its default value 0.7 and varied mIrvThresh from 0 to 1, incrementing it at an interval of 0.1. The FDR and TPR both decrease when mIrvThresh is increased from 0 to 1, with the FDR for 0.01 controlled after mIrvThresh 0.3, for 0.05 after mIrvThresh 0.6 and for 0.1 after mIrvThresh 0.2. At the lower values of mIrvThresh , `mehenDi` becomes less conservative and prefers nodes with larger heights, which increases the FDR. Higher height nodes are also preferred for larger values of minP , since the nodes downstream are more likely not to be assigned a confident sign change. Further, more nodes located at different branches in the tree are also observed for lower values of mIrvThresh and higher values of minP , compared to selected nodes obtained at the default parameters.

It is important to note that since the underlying features across the methods are different, these metrics may not be directly comparable. However, we obtain some information about their respective performances. We next investigate the increased sensitivity shown by `mehenDi`, by doing a pairwise comparison with the other methods on the `BrSimNorm` dataset.

3.3.2.2 Features output by mehenDi covers more true positive transcripts compared to Terminus

Next, we perform a pairwise comparison between the features output by `mehenDi` to `Terminus` obtained at 0.01 nominal FDR. Specifically, we examine the unique true positive transcripts that have been covered by the output features in comparison to one another. We find that features output by `Terminus` candidate nodes correspond to only 31 unique true positive transcripts, which map to 26 `Terminus` groups. For `mehenDi`, transcripts descending from its selected nodes yield a total of 681 unique true positive transcripts and correspond to 473 nodes. While the general expectation might be that most of these additional selected nodes in `mehenDi` would be as a result of more aggregation, we find 27 nodes for which the signal exists at a lower level in `mehenDi` that get lost in `Terminus` due to overaggregation. We provide some examples of such cases in Figure 3.3A, Supplementary figure B5 and B6. Similarly, there are 103 such groups (inner nodes in the tree) in `mehenDi` that contain at least one underlying descendant transcript, which was differentially expressed. The differential signal is lost for these transcripts in `Terminus`, either because the transcripts were not grouped at all or not aggregated enough to produce the signal. In Figure 3.3B, Supplementary figure B7 and B8, we show some examples of such cases.

3.3.2.3 mehenDi finds more unique branches in the tree compared to treeclimbR

We next perform a pairwise comparison between `treeclimbR` and `mehenDi` at the 0.01 nominal FDR on the `BrSimNorm` dataset. We first inspect the relative height of the nodes in

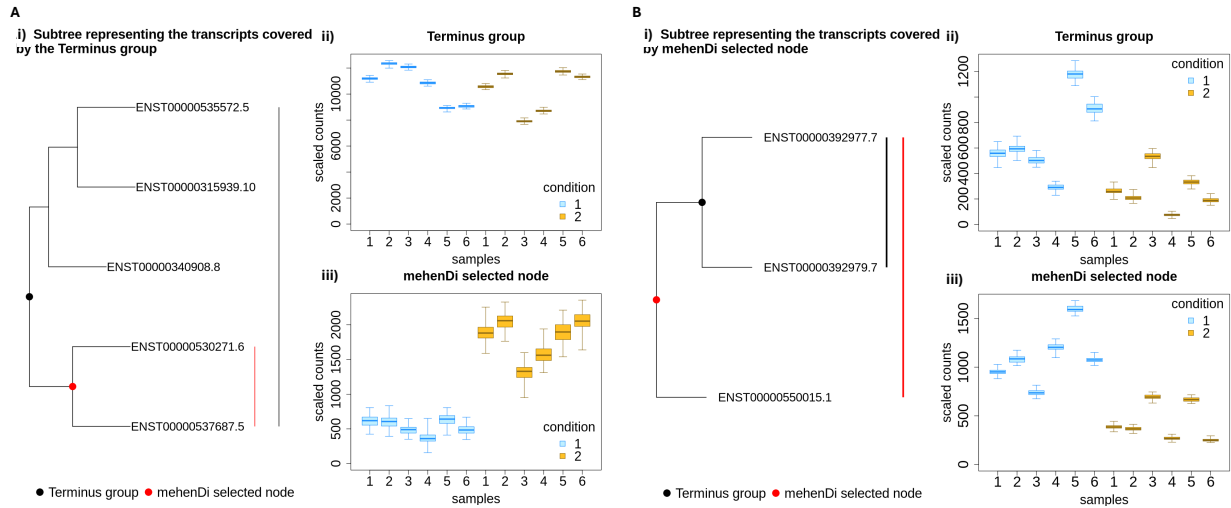


Figure 3.3: A) Example of a mehenDi node that is overaggregated in Terminus for the BrSimNorm dataset. i) Subtree representing the transcripts covered by the Terminus group. ii) Inferential replicates for the Terminus group. iii) Inferential replicates for the selected node output by mehenDi. B) Example of a mehenDi node that is not aggregated enough in Terminus for the BrSimNorm dataset. i) Subtree representing the transcripts covered by the mehenDi group. ii) Inferential replicates for the Terminus group. iii) Inferential replicates for the selected node output by mehenDi.

Supplementary table B1. We observe more higher-level nodes for treeclimbR compared to mehenDi, with the maximum height of the output node being 19 and 14 in treeclimbR and mehenDi, respectively. It is not trivial to compare these two methods when the nodes output by them are located at different heights and when the differential signal can exist at multiple levels within the same branch in the tree. We thus try to identify the branches unique to a method using the approach below. For example, imagine that we want to find the nodes unique to mehenDi. To do so, we first extract the nodes that are output by both of these methods (treeclimbR, mehenDi). From the remaining selected nodes (excluding common nodes) that are output by mehenDi, we remove all nodes i for which there exists a node j that is an output of treeclimbR and is either an ancestor or descendant of node i in the tree. The selected nodes that are left represent the unique branches of mehenDi. We repeat a similar process for

`treeclimbR`, to find the unique branches of `treeclimbR`. We find 199 such nodes unique to `mehenDi` that map to 258 true positive transcripts, while for `treeclimbR` we find only 2 such nodes that map to 4 true positive transcripts. Thus `mehenDi` finds more unique branches in the tree which in turn can enable us to recover the signal from more true positive transcripts.

3.3.2.4 FDR is not controlled across all the methods on the unique nodes

Since each method can output unique nodes, we next assess the error rate only on the unique branches/nodes by comparing the methods pairwise. The unique nodes are extracted using the approach described above for the different nominal FDR thresholds. Specifically, we extract the nodes for pairwise comparison between `mehenDi` versus `treeclimbR`, and `treeclimbR` versus `Txps`. We provide the observed FDR and the total number of unique nodes for these comparisons in Supplementary table [B2](#), [B3](#). The multiple hypothesis correction methods aim to control the average false discovery proportion of the overall set of rejected hypotheses. However, the error guarantee might not necessarily hold on a specific subset, which in this case are the unique nodes corresponding to each method. We observe an inflated observed FDR for the unique features corresponding to `Txps` and `mehenDi`, although it decreases for lower nominal FDR thresholds. The observed FDR is also inflated for `treeclimbR` on pairwise comparison with `Txps`, containing significantly fewer unique nodes. We next try to see if we can reduce the FDR by filtering the unique set based on the threshold of absolute log fold change between the conditions of interest. We plot this in Figure [3.4](#), Supplementary figure [B9](#) and [B10](#). We see that the observed FDR starts decreasing as the effect size increases and then again starts to increase. The FDR again starts increasing because, after a certain threshold of effect size, only a

few nodes are left after filtering. As a result, the false positive rate increases more dramatically due to a small denominator, leading to a heavier penalty for each incorrect prediction. We thus recommend using a conservative nominal FDR threshold with an appropriate effect size when the novel features unique to any method are analyzed.

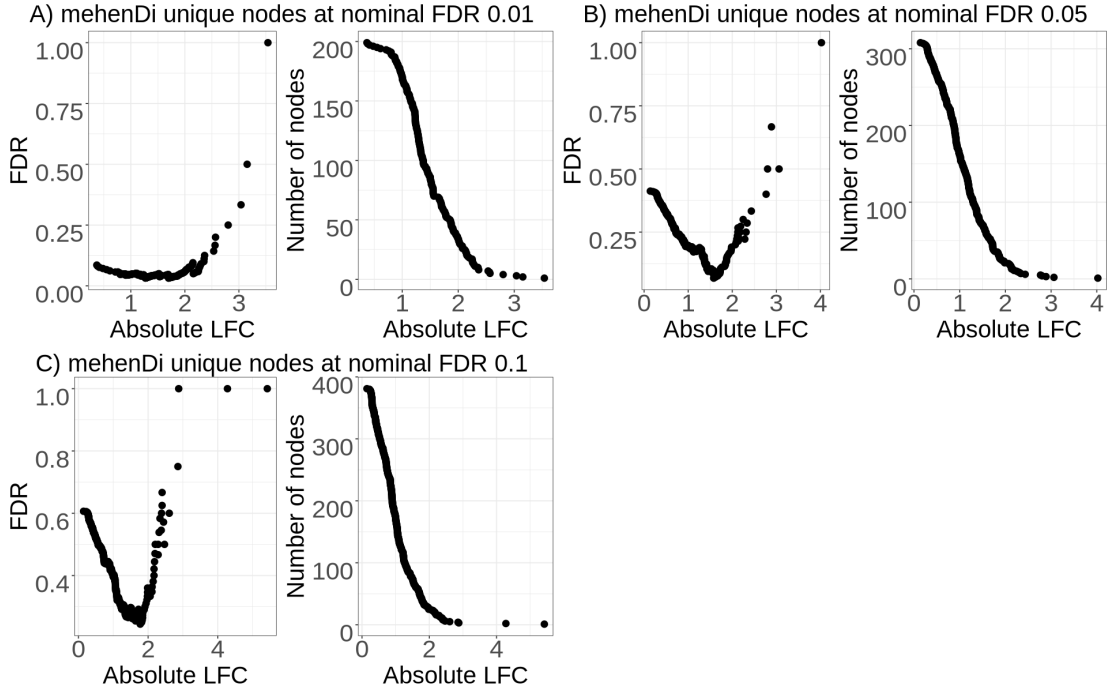


Figure 3.4: Examination of error metrics for the unique nodes obtained for mehenDi when doing mehenDi vs treeclimbR analysis at the different nominal FDR thresholds for the BrSimNorm dataset. We vary the magnitude of log fold change (LFC) and plot the empirical FDR and the total number of nodes that are left after filtering the unique nodes based on LFC.

3.3.3 Analyzing the MouseMuscle dataset

We next analyzed the MouseMuscle dataset which consists of 6 samples each from EDL and MAST muscle tissues (see Methods). The first two dimensions of the PCA using the top 1000 variable genes for the dataset are shown in Supplementary figure B11.

3.3.3.1 Increase in variation in the parameter values compared to the default setting leads to larger distances between the selected nodes

We first varied the parameters of `mehenDi` individually and computed the distance of the selected nodes obtained at those parameters from the tree with the selected nodes obtained at the default parameters (Supplementary Figure B12). For both *minP* and *mIrvThresh*, when the parameter values are closer to the threshold, the distance is lower and increases as the parameters are varied in both directions. The distance increases more sharply when *mIrvThresh* is varied compared to when *minP* is varied. As seen in the `BrSimNorm` dataset, we also observed nodes belonging to distinct branches of the tree and located at higher heights, at low values of *mIrvThresh* and high values of *minP*. The method for computing the distances between the nodes has been described in Supplementary section B.1.2

3.3.3.2 `mehenDi` can find nodes for genes for which the differential signal could not be detected at the transcript-level

We next compared the nodes obtained by `mehenDi` at the default parameter values with the different features and the nodes obtained across the other methods. The total number of differentially expressed genes (DEGs), differentially expressed transcripts (DETs) at the different nominal FDRs and the total number of transcripts that are covered by the nodes/groups output by the different methods at the 0.01 nominal FDR are shown in Supplementary table B4, B5 and Supplementary figure B13. We observe that `mehenDi`-reported nodes cover more transcripts compared to `treeclimbR`-reported nodes. The distribution of the node heights output by

`mehenDi` is shown in Supplementary table B6, whereas the distribution of the number of unique genes that the transcripts output by `mehenDi` map to is provided in Supplementary table B7 for the different nominal FDR thresholds. We observe that the majority of the output nodes consist of transcripts. Further, most of the transcripts covered by these nodes map to only one gene. We find many genes that are called significant when differential testing is carried out on the genes, but the underlying transcripts are not called as significant when the testing is performed on the transcripts (Supplementary table B8). Similarly, when differential testing is carried out on the transcripts, some of the significant transcripts map to non-significant genes (Supplementary table B9). Some of the selected nodes output by `mehenDi` neither map to a significant gene nor contain a single underlying transcript that is significant (Supplementary table B10). We also find many selected nodes in `mehenDi` that map to significant genes, but the genes do not contain a single transcript that is called significant during transcript-level differential analysis (Table 3.1). We provide some examples for these genes in (Figure 3.5, Supplementary figure B14, B15, B16, B17 and B18) obtained for 0.01 nominal FDR. For example, we can see that for the gene *Prrgl*, the transcripts highlighted in red (which underlie the selected node) constitute a very similar set of exons but vary in length. This is reflected in the tree structure in Figure 3.5A ii, where the transcripts ENSMUST000000114024.8 and ENSMUST000000177904.7 are aggregated first, followed by transcript ENSMUST000000114025.7. Thus, in Figure 3.5A iii we see that the transcript ENSMUST000000177904.7 has a high inferential relative variance (`infRV`). This also affects the quality of the signal, which gets stronger at the selected node level constituting the aggregated transcripts (Figure 3.5A iv).

Table 3.1: Number of *mehenDi* nodes that map to a significant gene but the gene doesn't consist of any significant transcripts for the MouseMuscle and the ChimpBrain dataset respectively.

Nominal FDR	Number of <i>mehenDi</i> Nodes	
	MouseMuscle	ChimpBrain
0.01	309	407
0.05	428	194
0.10	465	213

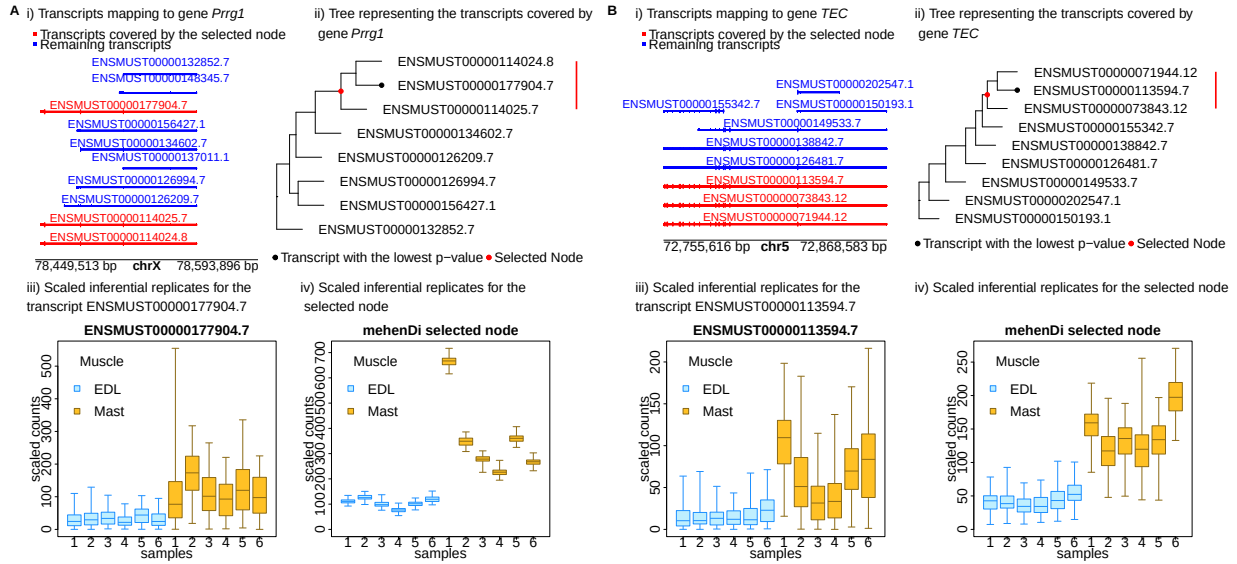


Figure 3.5: Examining the transcript profile for the gene *Prrg1*(A) and *Tec*(B) in the MouseMuscle dataset. i) Transcripts in a pileup style. ii) Tree representing the transcripts covered by the gene, with the red node representing the transcripts covered by the *mehenDi* selected node. iii) Inferential replicates for the transcript which had the lowest p-value among all the transcripts in the tree. iv) Inferential replicates for the *mehenDi* selected node.

3.3.4 Analyzing the ChimpBrain dataset

We also assessed the performance of *mehenDi* on the ChimpBrain dataset, which contains 5 samples each from both cerebellum and medial dorsal nucleus of thalamus (see Methods). The number of genes and transcripts that are differentially expressed between the conditions at the different FDR thresholds is provided in Supplementary table B11 and B12 with the total number of features

across the different methods shown in Supplementary figure B19. We again observe that the `mehenDi`-reported nodes cover the largest number of transcripts. The distribution of the node heights output by `mehenDi` is shown in Supplementary table B13, whereas the distribution of the number of unique genes that the transcripts output by `mehenDi` map to is provided in Supplementary table B14 for the different nominal FDR thresholds. We observe that the majority of the output nodes consist of transcripts. Further, most of the transcripts covered by these nodes map to only one gene.

3.3.4.1 `mehenDi` can find nodes for genes for which the differential signal could not be detected at the transcript-level

As observed for the `MouseMuscle` dataset, we have many examples for both cases where significantly called transcripts do not necessarily map to significantly called genes and vice versa for significant called genes (Supplementary table B15, B16). Similarly, we find some of the nodes reported by `mehenDi` map to non-significant genes and also don't contain a single transcript that is called significant (Supplementary table B17).

On the other hand, there also exist many other `mehenDi` nodes that map to those significant genes whose underlying transcripts were not called significant (Table 3.1). We report some of these cases in Figure 3.6, Supplementary figure B20, B21, B22, B23, B24 and B25 for the 0.01 nominal FDR. For example, for gene *EYA4* in Figure 3.6B, we observe that the transcripts ENSPTRT00000034383 and ENSPTRT00000104666 differ by only 1 exon, and the selected group that aggregates both these transcripts has lower uncertainty and shows an increased signal, compared to the transcript ENSPTRT00000104666 which has high uncertainty.

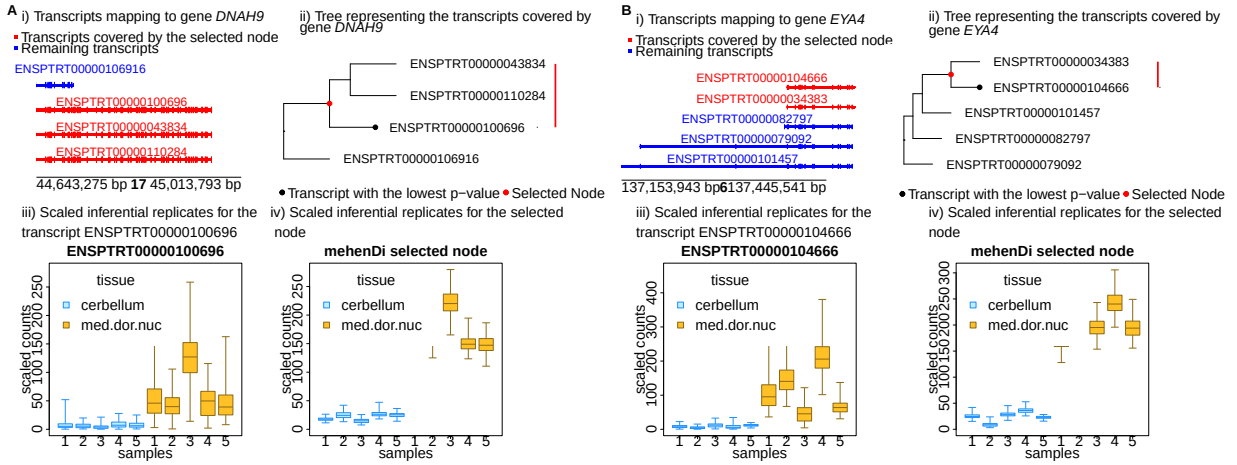


Figure 3.6: Examining the transcript profile for the gene *DNAH9* and *EYA4* in the ChimpBrain dataset. i) Transcripts in a pileup style. ii) Tree representing the transcripts covered by the gene, with the red node representing the transcripts covered by the *mehenDi* selected node. iii) Inferential replicates for the transcript ENSMUST00000104666, which had the lowest p-value among all the transcripts in the tree. iv) Inferential replicates for the *mehenDi* selected node.

3.4 Discussion

The high level of uncertainty that is observed in the abundances of some transcripts during RNA-seq quantification poses a challenge for downstream analyses, particularly for tasks such as differential testing. Various differential testing methods have been proposed that incorporate uncertainty into testing, and these methods have been applied either to transcripts or genes. This approach can occasionally result in overlooking the signal for the features that are difficult to quantify. Our method *mehenDi*, makes use of the tree structure computed by *TreeTerminus* [29] to produce the selected nodes that are significant between conditions. *mehenDi* provides a data-driven procedure for finding signal at a resolution between the gene and transcript level for RNA-seq differential analysis, where a selected inner node captures the signal for a set of underlying transcripts that have high uncertainty. While many tree-based differential testing procedures have been proposed, our method, *mehenDi* is among the few tailored toward RNA-seq data, which requires specific considerations. Our method requires p-values for all the nodes of the tree

and thus is flexible in the choice of the differential testing method used, though we have used `Swish` [57] in this study.

We assessed and evaluated `mehenDi` on both the simulated and experimental datasets. In the variations of the simulated datasets without true signal, we observed that the FPR obtained by `mehenDi` was comparable to that generated by doing testing only on the transcripts. Similarly, on both variations of simulated data that included true signals, we observe an increased sensitivity for `mehenDi` while controlling the FDR at or very near the nominal level. The enhanced sensitivity is not only evident quantitatively through the metrics we report but is also reflected in the discovery of more transcript branches on the tree when conducting pairwise comparisons with other methods. Interestingly, `treeclimbR` [83], which has been designed to control FDR at the leaf level, shows a very high sensitivity but also outputs a large number of false positives. Similarly, on the experimental datasets, we observe many significant genes that do not contain a single underlying transcript that is deemed significant when doing transcript-level differential analysis, but which contribute to nodes reported by `mehenDi`. This is a reflection on the ability of `mehenDi` to find signals that would have been lost when doing transcript-level analysis. Shared exons are commonly observed among these transcripts that belong to the selected nodes, increasing our confidence in the utility of the approach. We also observed a clear reduction in uncertainty for the aggregated transcript groups compared to the individual transcripts. This can provide biologists the opportunity to narrow down to a set of transcripts for their analysis that belong to the transcript groups, which would have been lost when doing either transcript or gene-level analysis.

There exist several areas for improvement of our method and also opportunities for developing new and related methods. The difference in the p-value distribution observed for the inner and

leaf nodes in the null simulation when the p-values for all nodes are calculated together suggests the need for a differential testing method that takes the hierarchy of features into account. It seems that, at least in the different simulations that we have created, we are controlling the FDR. However, since we are using the same dataset to create the tree and do differential testing, this technically leads to double-dipping [107], which can lead to an exaggeration of false positives. This could be one of the reasons why we observe an inflated FDR for `treeclimbR`.

To mitigate double-dipping, a Poisson/negative binomial sampling of the original data into independent partitions of the same dimensions was introduced in an approach called `count splitting` [108, 109]. The first partition can be used for latent-variable estimation and the second for inference. Incorporating `count splitting` into `mehenDi` will be a promising future direction. The challenge, however, will be how to do this without negatively affecting the power, since `mehenDi` already appears empirically well-calibrated.

There might also be another source of false positives. When doing a multiple-hypothesis correction on a set of hypotheses, the error rate is controlled over the entire set and not on a subset. Using our filtering strategy, we are outputting a subset of hypotheses that have p-value below a certain threshold. `Focused BH` [110] provides a way of selecting a p-value threshold by running the filter for the different p-value thresholds and finding the largest threshold at which the computed conservative estimate of the false discovery proportion for the set is less than the desired error rate. The limitation of this approach is that, apart from being conservative, it also puts constraints on the properties of the filters. Further, `Focused BH` would be computationally slow in its native form, specifically if filters have to be computed over hundreds or thousands of p-value thresholds and each filter takes non-trivial time to run. Adapting this approach for `mehenDi` can also be explored in the future.

Finally, it would be intriguing to explore the development of a `TreeTerminus` and `mehenDi` like approach in the context of single-cell RNA-seq analysis, where there are quite distinct, but related, opportunities for data-driven aggregation to increase the power and specificity of differential analysis. While differential expression analysis has been the focus of this paper, an important future direction would also be to explore the usage of `TreeTerminus` trees and `mehenDi` for differential transcript usage (DTU) analysis.

Chapter 4: `Alevin-fry-atac` enables rapid and memory frugal mapping of single-cell ATAC-seq data using virtual colors for accurate genomic pseudoalignment

4.1 Introduction

ATAC-seq [111] is a widely-used assay that enables the profiling of open-chromatin regions within the genome. It has a wide variety of applications, such as providing insight into gene regulation through the identification of promoters and enhancers [112], studying cell differentiation and development [113, 114], identification of transcription factors [115] to name a few. As with RNA-seq, it is also now possible to profile chromatin at the level of single cells [116, 117]. However, the efficient mapping and processing of single-cell ATAC-seq is a computationally-challenging task.

Currently, three main pipelines/methods are used for this task. One involves aligning reads to the genome using methods such as `BWA-MEM` [118], or `Bowtie2` [19], followed by demultiplexing into cells and then finally using existing general-purpose programs like `SAMtools` [119] and `Picard` [120] to sort and filter the alignments. This pipeline can be quite slow when working with large datasets and can be sped up to some extent by (e.g. replacing one of the above alignment tools with `minimap2` [121] as is used in workflows such as `MAESTRO` [122]).

The second approach, `Cell Ranger ATAC` by 10x genomics, provides end-to-end analysis, from read mapping through downstream tasks like cell clustering using called peaks and counts. It is substantially faster than the traditional pipelines due to optimized components that minimize redundant computation and file I/O. However, there is still much potential for further improvement. Additionally, `Cell Ranger ATAC` is released under a non-free license, that permits any use or modification with only 10x genomics' own proprietary and patented technologies. The third, and the fastest method is `Chromap` [123], which uses a minimizer-based [124] index to map reads by generating candidate anchors from minimizer hits and aligning candidate positions to the reference. It further speed up mapping by caching the frequently occurring reads

Some approaches, like `scATAK` and `snATAK` [125], use lightweight mapping methods such as pseudoalignment to single-cell and single-nucleus ATAC-seq data. However, they still rely on traditional read alignment tools to align reads to the genome to produce the accurate genomic mappings necessary to extract peaks, whose sequences are treated as experiment-specific targets for pseudoalignment [25]. While lightweight mapping approaches have proven very successful in providing fast read mapping in transcriptomics [25, 28] and metagenomics [26, 45, 126–128] without much loss in accuracy compared to alignment-based approaches, no method has yet employed lightweight mapping directly to the genome in the context of single-cell ATAC-seq.

One may expect that such approaches might also help in improving the read mapping efficiency for genomic mapping as well. However, the large reference sequences frequently consisting of repetitive regions in the genome make it difficult to applying pseudoalignment. A k -mer can map to many locations within the same reference which may lead to an incorrect determination of the mapping position, even when the specific reference (e.g. chromosome

of origin) is determined correctly. These issues can lead both to reporting spurious mapping positions as well as missing the most likely approximate mapping location of a read and can also lead to exacerbated multimapping effects.

Recently, KMCP [129] described an approach that uses a modified COBs [130] index to tackle the metagenomic profiling problem. The COBs data structure is an (approximate) presence/absence index based on the matching of some threshold fraction of k -mers between a query and the reference. To reduce the potential for spurious matching based off of genomically distant k -mers, and to ensure that the evidence supporting the existence and abundance of a taxon isn't due to many reads assigned to only one narrow range of the underlying genome (akin to the strategy employed in KrakenUniq [131]), KMC divides the metagenomic references into a fixed number of chunks, which themselves (rather than the genomes) become the set of distinct targets represented in the COBs index. Taking the success of such an approach as motivation, we propose a modified pseudoalignment approach for mapping reads to the genome, which partitions the references into “virtual colors” of fixed maximum extent. To determine the set of references (and their approximate positions) from which a read originates, the intersection across the virtual colors to which the k -mers read map is performed instead of taking the intersection over the original colors (i.e. references/chromosomes). This also necessitates careful handling of the boundaries between virtual colors (to ensure reads spanning virtual colors are not lost) as well as ensuring that duplicate mappings induced by virtual color overlap are properly identified with only a single position in the original reference space.

4.1.1 Contributions

- We present `alevin-fry-atac`, a novel tool designed for the efficient processing and mapping of single-cell ATAC-seq data. It extends the memory frugal `piscem` [33] index, incorporating a modified pseudoalignment algorithm, based upon the concept of virtual colors, that is tailored for accurate genomic mapping without sacrificing efficiency.
- To further enhance mapping speed, we optimized and improved the streaming querying procedure of `SSHash` [132] (a key component of the `piscem` index) and additionally developed a caching mechanism to speed up search of k -mers not amenable to streaming query optimizations. Both of these enhancements leverage insights about the structure of the compacted de Bruijn graph, which is a core component of our index.
- We have enhanced the processing functionalities of `alevin-fry` by incorporating a new, data-adaptive, parallelized, external-memory sorting procedure, which outputs a cell-barcode corrected, deduplicated, and coordinated-sorted BED file. The BED file contains all necessary information to be directly used with popular peak callers such as `MACS2` [133, 134] and other downstream tools for analyzing single-cell ATAC-seq data.
- We evaluated `alevin-fry-atac` on both simulated and experimental datasets and compared it with the other methods for mapping single-cell ATAC-seq data. We observed a high concordance between the peaks and clusters generated on the mapped fragment file produced by `Cell Ranger ATAC` and `Chromap` and those generated on the mapped fragment file produced by `alevin-fry-atac`. `Alevin-fry-atac` uses 300% less memory compared to `Chromap` while taking 280% less time at 32 threads for preprocessing and

mapping single-cell ATAC-seq data.

- We have integrated `Alevin-fry-atac` into the `alevin-fry` ecosystem and `simpleaf` [135], a framework originally designed to simplify single-cell RNA-seq analysis using `alevin-fry`. This integration creates a unified, open-source framework that simplifies and streamlines multi-modal analysis.

4.2 Material and Methods

4.2.1 Preliminaries

The `piscem` [33] index is a modular index that enables efficient retrieval of a set of references (colors), as well as all of the associated positions and orientations for a given k -mer x . Like the `pufferfish` [136] index that preceded it, the `piscem` index is constructed over the compacted colored reference de Bruijn graph, permitting the same set of operations. However, the `piscem` index is substantially smaller than that of `pufferfish`. For a given set of reference sequences, `piscem` uses `cuttlefish` [137] to efficiently construct the compacted colored reference de Bruijn graph and the associated tiling of the reference by unitigs. It then uses `SSHash` [132] to create a k -mer to unitig mapping $\mathcal{K} \rightarrow \mathcal{U}$, which maps each k -mer to a unitig, relative offset, orientation triplet (u_k, p_k, o_k) . `piscem` builds a dense inverted index to represent the unitig-to-reference tiling $\mathcal{U} \rightarrow \mathcal{R}$, which allows recalling, for each unitig the reference sequences, and relative positions and orientations of the unitig’s occurrences in the original reference. The k -mer to unitig $\mathcal{K} \rightarrow \mathcal{U}$ and unitig-to-reference $\mathcal{U} \rightarrow \mathcal{R}$ can be combined to correctly and efficiently determine the mapping position (and orientation) of a k -mer in the

reference, which might be essential for genome-centric mapping tasks. The `piscem` index can be constructed on different types of reference sequences (genome, transcriptome, metagenome, etc.) and the associated software can currently support the processing and mapping of single-cell RNA-seq and bulk RNA-seq. With the introduction of `alevin-fry-atac`, single-cell ATAC-seq is now also supported. A complete manuscript describing the `piscem` index, the key details of the software implementation, and its applications to various indexing tasks is currently under preparation.

The pseudoalignment strategy that has been employed in the current manuscript is the “hybrid” method described in `Themisto` [45] or the “threshold-union” method described in `Fulgor` [26]. Let $\mathcal{R} = \{R_1, R_2, \dots, R_N\}$ denote the set of reference sequences on which we want to map a query sequence Q . Let $K(Q)$ denote the set of k -mers for which a mapping $\mathcal{K} \rightarrow \mathcal{U}$ exists in the index. Given the threshold parameter $\tau \in (0, 1]$, the output of pseudoalignment for a read is a subset $R_C \subset \mathcal{R}$ and the relative read mapping position within a reference, such that each reference in the output R_C appears as a mapped hit (i.e. k -mer match) across at least $\tau * |K(Q)|$ k -mers. The mapping position of the leftmost k -mer hit of Q to the reference is assigned as the mapping position of the read to reference.

4.2.2 Mapping and Pseudoalignment using virtual colors

The accuracy of compatibility information generated by pseudoalignment may be similar to alignment-based approaches when the reference sequences are comparably short, which is the case for transcriptomics and metagenomics (and when the sequenced sample is devoid of non-indexed references that are sequence-similar to the indexed references [138]). However, when

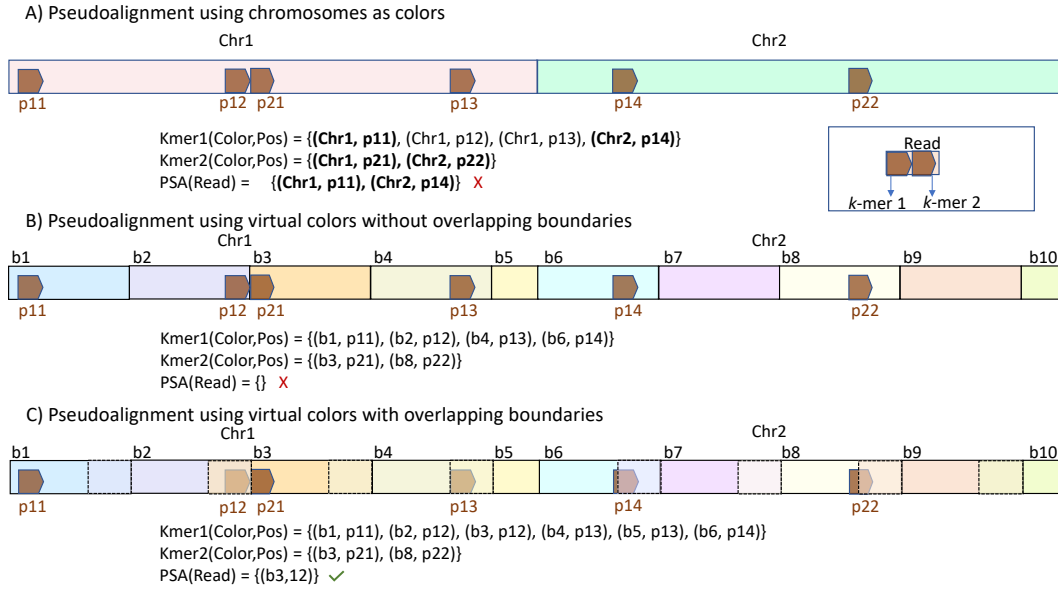


Figure 4.1: Toy example demonstrating how traditional pseudoalignment on genomic references could lead to spurious mapping or completely missing the correct mapping. In the example, we have two k -mers, which map to multiple references, with k -mer 1 mapping to multiple locations within chromosome 1. The individual mapped positions within a reference are denoted by p_{ij} , with i denoting the k -mer id and j denoting the order of the positions globally for that k -mer. PSA(Read) denotes the reference ids and the relative positions on the reference along which Read maps using pseudoalignment. The true mapping for the read in the example is (Chr1, p12). A) Using traditional pseudoalignment both Chr1 and Chr2 are selected as references from which the read originates. The position to which a read maps in pseudoalignment is governed by where the left-most k -mer of a read maps. Thus, for reference Chr2, p14 is picked as a position for the hit. If a k -mer maps to multiple positions within a reference, then by design the position with the smallest coordinate is picked, which for Chr1 is p11. We thus have two false positive mappings (Chr1, p11), (Chr2, p14) and a false negative since (Chr1, p12) is not reported. B) When the references are projected onto a set of virtual colors whose boundary spaces do not overlap, we miss the mapping if the read spans across the boundaries (b2, b3), creating an $\emptyset$ set when taking the intersection across the colors. C) When the boundaries of virtual colors overlap, each k -mer is assigned two colors if it falls within the overlapping region. This ensures that the read gets mapped after pseudoalignment.

directly applying pseudoalignment for mapping to a genome, where the reference sequences are large, comprising of chromosomes that often contain repetitive sequences, we observe a marked drop-off in mapping accuracy using various traditional pseudoalignment approaches. This can happen due to multiple matching locations for a k -mer within a single reference that can lead to false positive and false negative mapping positions during read mapping, as demonstrated in

Figure 4.1 A.

Construction of the virtual colors — We propose modifying the pseudoalignment approach by projecting the original reference sequences (i.e. the original colors) onto a set of “virtual colors”, that themselves do not exist as individual elements in the index, but which will represent the targets to which queries can map. The virtual colors, conceptually, consist of a set of overlapping bins, each of fixed maximal extent, on the reference sequences. This overlap is needed since the matching k -mers of a read can span the boundary of multiple disjoint virtual colors (and therefore become lost) (Figure 4.1 B, C). The virtual colors are designed not to extend over the individual reference sequences. Thus a virtual color b_i belonging to reference R_i will not overlap with any virtual color b_j belonging to reference j , $\forall i \neq j$. The construction of the virtual colors is controlled by two parameters namely ℓ_{vcol} and ov_length . The parameter ov_length controls how many bases a virtual color will overlap with the virtual color on its left. It should be at least equal to the read length to ensure that the entire read can fit within it. The parameter ℓ_{vcol} controls the total number of bases that will be spanned by a virtual color with most virtual colors on a reference spanning length of $(ov_length + \ell_{\text{vcol}})$ bases. The leftmost virtual color on a reference will cover a total of ℓ_{vcol} bases. The rightmost virtual color on the reference will cover $ov_length + rl$ bases such that $(B[R_i] - 1) \cdot \ell_{\text{vcol}} + rl = L[R_i]$, where $B[R_i]$ denotes the total number of virtual colors contained within the reference sequence R_i and $L[R_i]$ denotes the length of the reference sequence. $B[R_i]$ can be computed as $\text{ceil}\left(\frac{L[R_i]}{\ell_{\text{vcol}}}\right)$. The virtual colors are created dynamically, independent of the index, not creating any overhead.

Assigning virtual color to a k -mer — Before starting the mapping process, we create an

array CB of length $|\mathcal{R}|$, such that:

$$CB[R_i] = \begin{cases} 0, & \text{if } R_i = 0 \\ CB[R_i - 1] + B[R_i - 1] & \text{otherwise} \end{cases}$$

where $CB[R_i]$ stores the cumulative number of virtual colors up to reference R_i . To assign a virtual color to a k -mer x for which there exists a mapping in the index, we extract the reference id, R_i , and p_{ij}^x — the relative position of the j -th occurrence of the k -mer x within reference i in the index. Corresponding to each occurrence of k -mer x on reference R_i . The color id for the k -mer x is computed as $V(x) = CB[R_i - 1] + \frac{p_{ij}^x}{\ell_{\text{vcol}}}$. Further, we check if the k -mer x lies in the overlapping region between the bins. If that is the case, then it is assigned an additional color id $V(x) + 1$. This ensures a read gets mapped if it entirely (or partly) lies within the overlapping regions.

Obtaining the final color set — Since a k -mer can be assigned two virtual colors corresponding to the same matching position, it might happen that a read after pseudoalignment also gets assigned two virtual colors for that same reference position. This is not a problem for mapping single-end reads, as removing mappings with duplicate color ids and positions is straightforward. However, individual mate mappings must be merged for paired-end reads to get the final set of mapped references and the corresponding starting positions. To do this, we first remove duplicate mappings for the individual mates. Two mappings a, b are deemed duplicate $\leftrightarrow$: $(\text{ori}(a) = \text{ori}(b))$, $(\text{ref}(a) = \text{ref}(b))$, and $(\text{pos}(a) = \text{pos}(b))$, where $\text{ori}(\cdot)$, $\text{ref}(\cdot)$, and $\text{pos}(\cdot)$ denote the orientation, reference id (original color) and position for a mapped hit, respectively.

By design, a mapped hit will always have *at most one other duplicate* under the above

criteria. These hits are deduplicated by keeping the hit with the smaller virtual color id. After removing the duplicate mapping hits for each mate, hits a and b are merged, with a being the mapping for mate 1 and b being the mapping for mate 2, if they meet the following criteria :

$$\begin{aligned} \text{ref}(a) &= \text{ref}(b) \\ |\text{vid}(a) - \text{vid}(b)| &\leq 1 \\ \text{ori}(a) &\neq \text{ori}(b) \\ -\text{dovetail}_{\max} \leq \text{pos}(a) - \text{pos}(b) &\leq \text{ins}_{\max}, \text{ if } \text{ori}(a) = r \\ -\text{dovetail}_{\max} \leq \text{pos}(b) - \text{pos}(a) &\leq \text{ins}_{\max}, \text{ if } \text{ori}(a) = f \end{aligned}$$

Here $\text{vid}(\cdot)$ represents the virtual color id and orientation f, r represents the forward and reverse orientation respectively. We use the values $\text{dovetail}_{\max} = 20$ and $\text{ins}_{\max} = 1000$ in this work.

The mapped hits supported by different numbers of k -mers can arise when the threshold $\tau < 1$. In such cases, we want to pick the hits for which we have the highest confidence, provided by the number of k -mer hits. Thus, we keep track of all valid mappings for the read pair, and let τ' denote the largest fraction of k -mer hits supporting any reported mapping. We subsequently filter out all merged mappings having $< \tau'$ fraction of valid mapped k -mers.

To map the paired-end reads, we first try to merge the individual reads and create a unified fragment. If we are successfully able to merge the reads, then the merged fragment is mapped to the genome instead of the individual reads. We have described this in detail in *Mapping and*

Pseudoalignment subsection under Supplementary Methods.

4.2.3 Speeding up k -mer query

A key component of efficiently mapping reads using pseudoalignment is the ability to efficiently retrieve the unitig (and relative offset and orientation within the unitig) to which the k -mers in a read map. While the current `pisces` index enables fast extraction of this information, we further optimize the querying procedure to enable even faster retrieval of the unitig and reference information associated with k -mers.

Streaming queries

Consecutive k -mers of read are likely to map to consecutive positions on the reference(s). Consequently, many sequences of adjacent k -mers may also originate from the same unitig. Consider, for example, the error (and variation) free case. If the first k -mer of a read that is truly drawn from the genome maps to some position p in unitig u , then the subsequent $\ell_u - p$ k -mers will map to the following positions in the unitig. When a k -mer is verified to exist in a unitig, we need not search for it anywhere else in the index, since a k -mer may appear at most once in the de Bruijn graph. Hence, by optimistically checking for subsequent query k -mers that may reside in the same unitig (which, itself, is likely in CPU cache), we can often avoid having to refer to the index entirely until the unitig of origin shifts.

Specifically, when querying a k -mer, we maintain a state variable. This variable stores the information about the unitig to which the previously mapped k -mer of the read matched, including its id, orientation, relative offset, and distance from the unitig's end. The state variable is empty either in two cases: the first k -mer of a read is being queried, or the distance of the previous k -mer

from the end of unitig is 0 (i.e. the last k -mer on the unitig w.r.t the orientation of mapping).

In both of these cases, the current k -mer is directly queried to the index. Otherwise, the next nucleotide is extracted from the unitig using the previous k -mer's offset and orientation. This nucleotide is matched with the last nucleotide of the current k -mer. If they match, then we have the relevant mapping information for this k -mer. Otherwise, a query to the index is made. The fewer queries that are made to the main index, the faster the mapping process, the opportunity for which is provided by consecutive k -mers in the read. The approach is similar to the streaming iterator in `SSHash` [132]. However, the implementation in `SSHashtracks` the minimizer of the current k -mer and queries the corresponding minimizer buckets of the index whenever the minimizer changes, regardless of whether or not a straightforward extension would succeed. In our implementation, by exploiting the structure of the unitigs and the guarantees provided by the de Bruijn graph, we avoid even this lookup, which we find can lead to a substantial (30-40%) improvement in average lookup speed when querying each k -mer in a read.

Caching terminal k -mers

We aim to exploit the observation that chromatin profile enrichment is confined to specific genomic regions. The mapping information for k -mers can be efficiently extracted as described above if the k -mers lie within the same unitig. However, a general query in the index is always necessary when querying a k -mer that would reside on a different unitig. Whenever the k -mer preceding such a k -mer was the first or last in a unitig, we refer to such k -mers as *fk*-mers and implement a caching mechanism to enable efficient retrieval of their mapping information. Specifically, we suspect that mapped reads will repeatedly traverse specific unitig paths in the underlying de Bruijn graph, and so that by inserting the terminal (first or last) k -mers in these unitigs in a small and fast cache, we can often avoid lookups in the main index even when moving

between unitigs.

The cache is implemented as a shared, concurrent, hash map (we use the `Boost unordered_concurrent` maintained across all mapping threads. Hash map keys are fk -mers, and the values are the results of querying the index for those k -mers. The map has a fixed size (default 5,000,000 entries) and follows a write-once policy, meaning, once a key-value pair is added, it is not replaced.

For each k -mer, we first determine if it an fk -mer and if so, we check the cache. If the k -mer is present in the cache, the corresponding unitig information is retrieved directly. Otherwise, a full query is performed on the index, and if the k -mer is found and the map size permits, the k -mer is added to the cache.

To identify fk -mers, we leverage state information. Specifically, a k -mer is labeled as an fk -mer if the distance of the previous k -mer from the unitig's end is less than the distance between the offsets of the previous and the current k -mers. This policy allows us to avoid even attempting to consult the cache when we expect the next k -mer to occur within the same unitig, and to only consult and modify the cache for unitig-terminal k -mers (i.e. fk -mers).

We note that `Chromap` also employs a cache [123], where the cache is created on a chain of frequently occurring minimizers within a read, storing their corresponding mapping positions from the index. While the idea of using a cache to speed up mapping to a data-dependent subset of the genome is inspired by `Chromap`, the nature of our cache, the elements that are cached, and the manner in which it is implemented, are completely different.

4.2.4 `Alevin-fry-atac` pipeline

We now describe the `alevin-fry-atac` pipeline for preprocessing single-cell ATAC-seq data. The first step is to map all the reads using the modified pseudoalignment algorithm described above. The default output of this phase is a RAD (reduced alignment data) format file. The RAD format is a chunk-based, binary file optimized for machine parsing that encodes the relevant information necessary for subsequent data processing. Further details of this format are described in [34], though it worth noting that the single-cell ATAC-seq variant of such files carries some distinct information from the single-cell RNA-seq variant (e.g. it includes positions for each mapping, but doesn't include UMI information).

Cell barcode correction — After mapping, the next step is cell barcode correction. While various approaches to barcode detection and correction are possible (`alevin-fry` implements several distinct modes), for `alevin-fry-atac` we currently only support correcting to an unfiltered permit list. The permit list file contains a list of experiment-independent barcodes that are a superset of the barcodes that should be observed in any given sample. We first scan through the RAD file and count the number of times a barcode corresponding to the mapped record appears (exactly). If a cell or barcode has at least `min-reads` records (default 10) mapping to it, and is also in the whitelist, it is marked as present. For all the other mapped records corresponding to cells not currently marked as present, a nearest neighbor search is performed on the barcode against the list of present cells. If a barcode has a unique (i.e. only one) present neighbor at edit distance 1, that barcode is corrected to that of the neighbor. All other records are ignored.

Sorting and Deduplication — In this step, a coordinate-sorted BED file is produced. First, a

collection of temporary files are created, each corresponding to contiguous ranges of the underlying genome. The input RAD file is parsed and all records are routed to their appropriate temporary file based on their starting mapping location. The temporary files are then individually sorted (in parallel) and the sorted files are merged (in genomic coordinate order) to produce a BED file. During the sorting process, the duplicate entries for a record (i.e., entries with the same reference, start position, end position, and barcode) are collapsed, and the corresponding number of duplicate entries is stored. For each mapped fragment, the BED file contains the reference name, start position, end position, cell barcode and the total number of duplicates (including the fragment itself). The BED file can then be passed to a peak caller such as MACS2 2 [134], the output of which, along with the original BED file containing the counts, forms the input for downstream analysis.

4.2.5 Dataset and the analysis pipeline

We evaluated and compared the performance of the different methods on three simulated and three experimental datasets. `Mason` [139] was used to generate the simulations for read lengths 50, 100, 150 bases. One million paired-end fragments for each read length were created with a probability mismatch 0.25, according to the Illumina model in the simulator. The experimental datasets used in this study include the `Human 10K PBMC`, `Mouse 8K Cortex` and `Human 3K Brain` respectively, some of which have been utilized for analysis and benchmarking in previous studies. The datasets have been downloaded from 10X Genomics website, for which the URLs are provided in Table C1. The details regarding the datasets, analysis pipeline, experimental setup, evaluation, benchmarking, and the software that have been used are provided in detail in

Analysis pipeline subsection under Supplementary Methods.

4.3 Results

4.3.1 Mapping accuracy on the simulated data

Evaluating the effect of different parameters for alevin-fry-atac— As described in the methods section, `alevin-fry-atac` includes several parameters: the k -mer size k , pseudo alignment threshold τ and ℓ_{vcol} , which controls the number of bases a virtual color will span (with most virtual colors spanning $\ell_{\text{vcol}} + \text{ov_length}$ bases). The mapping accuracy is closely tied to the value of these parameters. To evaluate this, we generated simulated datasets with varying read lengths (50, 100, 150 base pairs) and on each dataset, tested the accuracy under different combinations of these parameters. For each evaluation, two parameters were held constant while the third was varied. The following parameter values were tested:

$$\begin{aligned}\ell_{\text{vcol}} &= \{1000, 10000, 100000, 1000000, \text{chr}\} \\ k &= \{23, 25, 27, 29\}, \\ \tau &= \{0.6, 0.7, 0.8, 1\}.\end{aligned}$$

chr implies mapping is done using chromosome sequence as colors.

The mapping accuracy for each parameter combination is detailed in Supplementary Results and visualized in Figures C1 - C12. In summary, we observe that virtual colors, regardless of the ℓ_{vcol} value, *substantially* enhance the mapping accuracy compared to using chromosomes as colors for pseudoalignment. It has the largest impact on accuracy of all the parameters, where

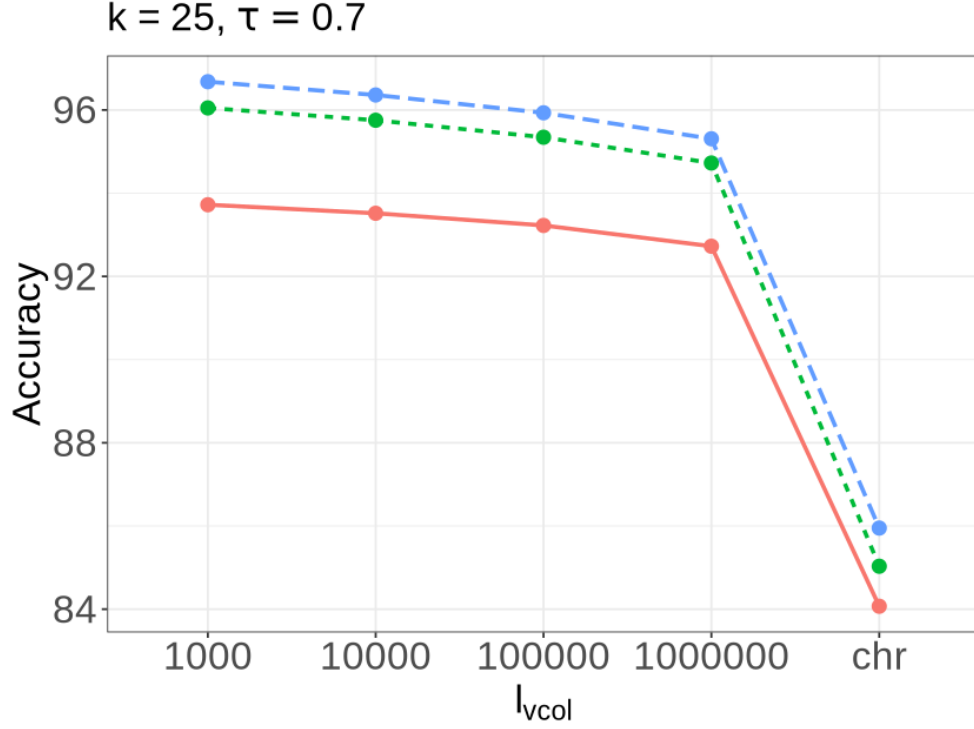


Figure 4.2: Evaluating accuracy across different read lengths for `alevin-fry-atac` by varying A) ℓ_{vcol} for $k = 25, \tau = 0.7$. The x-axis label *chr* implies mapping is done using chromosome sequence as colors (removing the impact of virtual colors on mapping procedure). the impact is consistent across different read lengths and parameter combinations. Specifically, the accuracy drops from 93.7% to 84% when the ℓ_{vcol} value is changed from 1000 to *chr*, with $k = 25, \tau = 0.7$ for reads of length 50 (Figure 4.2).

For reads of length 50, the highest accuracy is observed at $k = 25$, while for reads of length 100 and 150, accuracy improves at larger k values. Additionally, accuracy generally decreases as τ increases from 0.6 to 1.

Comparing the mapping accuracy with the other methods — We next compare the mapping accuracy of `alevin-fry-atac` with Bowtie2 [19] (which does full read alignment) and Chromapon the simulated dataset which is ummarized in Table 4.1. Similar to `alevin-fry-atac`, the other methods also demonstrate higher accuracy for longer read lengths. Among the methods, Bowtie2 has the highest accuracy, followed closely by Chromap, while `alevin-`

`fry` shows slightly lower accuracy across all the evaluated read lengths. Specifically, for reads of length 50, the mapping accuracy of `alevin-fry` is $\sim 2\%$ lower than `Bowtie2` and around $\sim 1\%$ lower for reads of length 150.

Table 4.1: Mapping accuracy on the simulated data for the different methods

Method	Read Length		
	50	100	150
<code>alevin-fry-atac</code> ($k = 25, \tau = 0.7, \ell_{\text{vcol}} = 1,000$)	93.72	96.05	96.68
Chromap ($w=7, k=17$)	95.48	97.11	97.61
Bowtie2	95.71	97.16	97.65

4.3.2 Experimental datasets

Mapping Rate — We also looked at the impact of varying the different parameters of `alevin-fry-atac` on the overall mapping rate for the different experimental datasets, namely Human 10K PBMC, Mouse 8K Cortex and Human 3K Brain (Figure C13). As with the simulated dataset, we also find a decrease in mapping rate as ℓ_{vcol} increases for different k and τ values, with the fall being the sharpest when chromosomes are used as colors (i.e. when virtual colors are disabled). Further, since we have relatively shorter read lengths in these datasets, higher mapping rate is observed at lower values of k . The highest mapping rate is observed for $k = 25$, followed closely by $k = 23$ for the human datasets, while for the mouse dataset the highest mapping rate is achieved at $k = 23$ followed by $k = 25$, with the lowest values observed at $k = 31$ across all datasets. We observe an overall higher mapping rate for `alevin-fry-atac` using $k = 25, \tau = 0.7, \ell_{\text{vcol}} = 1,000$ (98.27%, 97.73% and 96.99) for the Human 10K PBMC, Mouse

8K Cortex and Human 3K Brain datasets respectively while for Chromap(without using the `--preset atac` argument) is (95.31%, 95.56% and 94.50%).

Comparing peaks — After obtaining the mapped fragment file—which has been barcode-corrected, deduplicated, and coordinate-sorted—a key step in single-cell ATAC-seq analysis is peak identification. Peaks represent genomic regions with significant enrichment of sequencing reads. To identify peaks, we used MACS2 on the mapped fragment files generated by `alevin-fry-atac`, `Chromap` and `Cell Ranger ATAC` across different datasets. The total number of bases covered by the peaks is provided in (Table C2), which also includes the bases covered by peaks identified using the custom `Cell Ranger ATAC` peak caller, which could only be run on the `Cell Ranger ATAC` fragment file. Additionally, we did not process the Human 3K Brain dataset with `Cell Ranger ATAC`, as it is a multi-omic dataset required analysis with a different software, namely `Cell Ranger Arc`.

We observed that the peaks produced by the `Cell Ranger ATAC` output cover the largest number of bases across all datasets. Notably, peaks identified using `Cell Ranger ATAC` custom peak caller cover more bases than those identified by MACS2 [134]. On human datasets, peaks produced by `alevin-fry-atac` cover more bases compared to `Chromap`, while on mouse datasets, `Chromap` covers more bases. The peak overlap between the methods exceeds 90% across all datasets, and in most cases, it is greater than 95% (Table 4.2). The overlap between `Chromap` and `Cell Ranger ATAC` is slightly higher than the overlap between `alevin-fry-atac` and `Cell Ranger ATAC`— though this difference reduces substantially when using the peak range of `Cell Ranger ATAC` as the denominator.

Comparing the clusters — We performed clustering of cells for the different datasets using `Signac Signac` [140] and conducted a pairwise comparison between the clusters, both qualitatively

Table 4.2: Percentage overlap between the peaks pairwise for the different methods across the datasets. Depending on which method is used as a denominator, percentage overlap can change. The values in the parenthesis denote the overlap when the second method was used as the denominator. N/A entries denote that Cell Ranger ATAC was not run on that dataset.

Method / Dataset	alevin-fry-atac vs Chromap	alevin-fry-atac vs Cell Ranger ATAC
Human 10K PBMC	96.59 (97.97)	96.84 (97.02)
Mouse 8K Cortex	97.99 (98.31)	97.56 (94.35)
Human 3K Brain	92.23 (97.80)	N/A

and quantitatively. Across datasets, we observed a large overlap between clusters (Figures C14 and C15). On the Human 10K PBMC dataset, there was less crossover (i.e. more concordance) between the clusters obtained for Cell Ranger ATAC and alevin-fry-atac compared to other pairwise comparisons. Similarly, the UMAP plots (Figures C16 to C18) revealed very similar cluster projections.

Quantitatively, we evaluated the cluster labels (Table 4.3) using Adjusted Rand Index (ARI) and Normalized Mutual Information (NMI). High concordance was observed across all datasets. On the Human 10K PBMC dataset, ARI and NMI for the comparison between alevin-fry-atac and Cell Ranger ATAC were higher than the other pairwise comparisons. On the Mouse 8K Cortex dataset, ARI and NMI were highest for comparison between alevin-fry-atac and Chromap. Similarly, for Human 3K Brain dataset, NMI > 0.9 is observed between alevin-fry-atac and Chromap. Although all methods showed high concordance with close values, clusters produced using alevin-fry-atac peaks consistently demonstrated equal or higher concordance with clusters from Cell Ranger ATAC compared to those from Chromap.

Table 4.3: Evaluating the concordance between the clusters obtained for the different methods pairwise across the datasets. The two entries per cell separated by comma denote Adjusted Rand Index (ARI) and Normalized Mutual Information (NMI). N/A entries denote that Cell Ranger ATAC was not run on that dataset.

Method / Dataset	alevin-fry-atac vs Chromap	alevin-fry-atac vs Cell Ranger ATAC
Human 10K PBMC	0.87, 0.91	0.93, 0.94
Mouse 8K Cortex	0.95, 0.96	0.94, 0.94
Human 3K Brain	0.89, 0.91	N/A

4.3.3 Memory and Running Time

Finally, we benchmarked the memory usage and runtime of alevin-fry-atac and Chromap for mapping and processing by evaluating their performance on Human 10K PBMC and Mouse 8K Cortex datasets using varying numbers of threads (Figure 4.3, Figure C19). Across both datasets, Chromap was faster than alevin-fry-atac when using 4 or 8 threads. However, starting at 12 threads, alevin-fry-atac began to outperform Chromap, becoming substantially faster at 32 threads. Notably, Chromap showed poor scalability beyond 12 threads, whereas alevin-fry-atac demonstrated significantly improved scalability.

In terms of memory usage, alevin-fry-atac consistently consumed substantially less memory than Chromap. Alevin-fry-atac used approximately 6.4 and 5.6 Gbs for the Human 10K PBMC and Mouse 8K Cortex datasets (Figure 4.3 B, Figure C19 B), compared 19.65, 18.8 Gbs used by Chromap. Additionally, Chromap’s mapping-only mode (which only does read mapping and no other processing) and referred to as (Chromap(Map Only)) used 39.4, 35 Gbs for the two datasets, respectively (Figure C20, Figure C21). The benchmarking results are summarized in Table 4.4 for 32 threads, which also includes the results for Cell Ranger ATAC. However, it is important to note that the reported metrics for Cell Ranger ATAC also include the time spent on peak calling and other downstream analyses, as the tool does

not allow for executing each step individually.

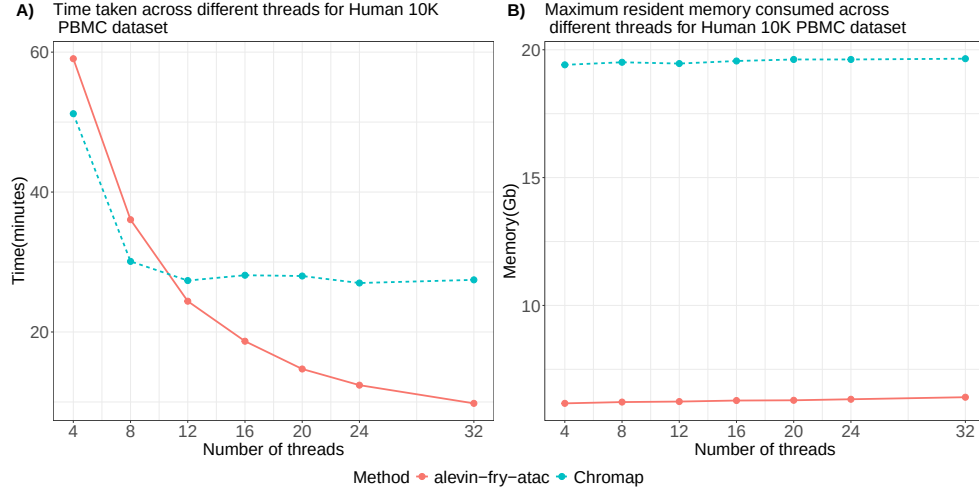


Figure 4.3: Comparing the time taken and maximum resident memory consumed by alevin-fry-atac and Chromap for the Human 10K PBMC dataset across the different threads.

Table 4.4: Time and maximum resident memory consumed by the different methods for mapping scATAC-seq reads. For both Chromap and alevin-fry-atac results are reported using 32 threads¹.

Method	Memory (Gb)	Time (minutes)
Human 10K PBMC		
Cell Ranger ATAC *	14.9	222
alevin-fry-atac	6.3	9.80
Chromap(Whitelist)	20.6	27.45
Mouse 8K Cortex		
Cell Ranger ATAC *	8.6	190
alevin-fry-atac	5.5	7.91
Chromap(Whitelist)	19.7	21.85

¹For Cell Ranger ATAC it is not possible to run and thus benchmark each step individually, thus the time and memory consumption also include other tasks such as peak calling and clustering.

4.4 Discussion

We have introduced `alevin-fry-atac`, which provides a computationally efficient, memory-frugal, scalable and accurate framework for processing and mapping single-cell ATAC-seq data. It uses only $\frac{1}{3}$ of the memory of `Chromap` and less than $\frac{1}{2}$ of the memory of `Cell Ranger ATAC` when processing single-cell ATAC-seq data (and `alevin-fry-atac` utilizes only $\frac{1}{6}$ of the memory of `Chromap` when the latter is not provided with a permit list during mapping). Since `Chromap` only maps the reads corresponding to the barcodes present in the permit list file, this could be one reason why we observe an increased time usage in the mapping-only mode. Internally, `alevin-fry-atac` uses the `piscem` index, which is small and versatile and enables mapping reads from other technologies such as bulk RNA-seq, single-cell RNA-seq and now single-cell ATAC-seq.

The enhanced streaming query and the introduction of cache help in significantly improving the speed and scalability of `alevin-fry-atac`. When using 12 threads, `alevin-fry-atac` outperforms `Chromap` in terms of speed and achieves even greater gains at the higher thread counts. At 32 threads, `alevin-fry-atac` processes and maps single-cell ATAC-seq data in 280% less time than `Chromap`. The introduction of virtual colors, and modifications to the pseudoalignment procedure enable accurate mapping of reads at an accuracy, achieving precision close to alignment-based methods. This level of accuracy would not have been attainable without the use of virtual colors. Despite the slightly lower mapping accuracy compared to alignment-based approaches, this does not seem to impact downstream analyses. Across experimental datasets, we observe a very high concordance between the peaks and clusters produced by `alevin-fry-atac` and those produced by the other tested methods. The `alevin-fry` and `simpleaf`

ecosystem already supports the processing of single-cell RNA-seq. With the added support for single-cell ATAC-seq makes, it becomes only fully open-source software that directly supports both modalities, simplifying and facilitating multi-modal analysis. We thus believe that `alevin-fry-atac` is a viable and strong alternative to the other methods, combining efficiency, scalability, and accuracy in single-cell ATAC-seq data analysis.

While the main focus in this manuscript has been on mapping single-cell ATAC-seq data, it should be trivial to extend and apply pseudoalignment using virtual colors to other genomic-centred mapping-based tasks and modalities, such as DNase-seq, Chip-seq and Hi-C.

Finally, for our pseudoalignment strategy, we have queried all k -mers and utilized the “threshold-union” approach. However, other lightweight mapping strategies such as selective alignment [138] and pseudoalignment with structural constraints [34] can also be explored to see how they impact speed and mapping accuracy. Finally, all pipelines, including `alevin-fry-atac`, currently discard reads that map to more than one location. Such reads constitute 4 – 10% of the total reads in the datasets analyzed in this manuscript, and might have a non-trivial impact on downstream analysis [141]. One interesting future direction would be to develop methods to resolve the origin of such reads probabilistically using methods such as expectation-maximization [142].

Chapter 5: Conclusion and Future Directions

In this dissertation, I have focused on developing novel methods that address two main challenges: incorporating uncertainty in RNA-seq data analysis and creating an efficient method for processing and mapping single-cell ATAC-seq data. In chapter 2, we introduced `TreeTerminus`, which, to the best of my knowledge currently, is the only framework that constructs transcript trees based on uncertainty. The analysis presented highlights the advantages of using a tree-based structure over the discrete grouping approaches. While I presented a generalized dynamic programming module for identifying optimal cuts within the tree based on specific objective functions, this framework offers significant potential for future extensions and novel applications in transcriptomic research. In chapter 3, I present `mehenDi`, a method built on top of `TreeTerminus` that utilizes the tree-structure for differential analysis. The results illustrate that signals often overlooked at the transcript level can be preserved and highlighted through the internal nodes of the tree, offering deeper biological insights and improving the robustness of differential expression analysis. In chapter 4, I focused on a step upstream of the traditional bioinformatics analysis pipeline by introducing `alevin-fry-atac`. This method extends pseudoalignment techniques for efficient genome-wide mapping, enabling rapid and accurate processing of single-cell ATAC-seq data. The improvements in efficiency and memory usage make it a valuable tool for large-scale single-cell studies, with the potential to unlock new insights

into chromatin accessibility dynamics at the single-cell level.

Future Work

The methods developed in this thesis lay the groundwork for several promising extensions and future research opportunities. Thus far, we have applied `TreeTerminus` and `mehenDi` primarily to short-read RNA-seq data. However, the increasing popularity and abundance of long-read sequencing technologies [143, 144] present new opportunities for advancement. These developments hold significant implications for transcriptomics, particularly as recent research focuses on novel transcript discovery and quantification [145–148].

Despite the advantage of capturing longer fragments, long-read sequencing still faces challenges such as multimapping of reads across transcripts. Additionally, the lower throughput of long-read platforms can result in fewer uniquely mapped reads for transcripts with ambiguous mappings. Consequently, both `TreeTerminus` and `mehenDi` could prove valuable for analyzing long-read sequencing data as well. While `oarfish` [146] offers the capability to output bootstrap samples for inferential replicates, it does not currently leverage equivalence classes within its quantification model. This omission is likely due to the comparatively lower fragment counts in long-read datasets relative to short-read sequencing. Nonetheless, creating equivalence classes for long-read data within `oarfish` should be straightforward, even if they are not directly integrated into the core model.

There is also the potential to extend `TreeTerminus` to single-cell RNA-seq (scRNA-seq) data. For datasets generated using the widely adopted 10X Genomics protocol, tree construction would be based on genes rather than transcripts, given the protocol’s focus on capturing 3’ or 5’

ends of transcripts. However, it remains feasible to construct transcript-level trees for datasets produced by protocols that capture full-length transcripts, such as SMART-seq or long-read sequencing. Single-cell datasets often involve tens or hundreds of thousands of cells, and treating each cell as an individual sample presents a scalability challenge for extending `TreeTerminus`. The current `Mean` mode requires loading inferential replicates for each sample into memory, which makes processing large datasets computationally untenable. To address this, developing better data representation schemes and minimizing disk-read operations would be essential for supporting efficient single-cell analysis. For the `Consensus` mode, scalability improvements would require a more efficient reimplementaiton of the consensus method, with approaches like those described in [59] offering a promising starting point. Specifically, current implementation in PHYLIP has a runtime $O(N^3K)$, where [59] provides $O(NK)$ implementaiton, with N being the total number of leaves and K being the number of samples. Despite these optimizations, there may still be an upper limit to the number of samples that can be effectively processed. One potential solution is to merge samples, a strategy that could be particularly useful for single-cell datasets where data sparsity may hinder the accurate capture of uncertainty profiles. Grouping cells into MetaCells [149] would be an intriguing future direction to address this challenge.

There are some other areas that can also be explored for algorithmic improvement of `TreeTerminus`. Currently, when a tree is constructed per sample or during `Mean` mode across a set of samples, we are restricting the construction to a binary tree, even if two or more edges have the same score. Another area of exploration would be to consider the class/condition information of a sample before tree-construction.

Similarly, `mehenDi` as `TreeTerminus` has direct applications for long-read RNA-seq data. We have already addressed some of the current limitations for `mehenDi` in the Discussion

section for chapter 4. These include the potential of double-dipping, since the tree construction and differential testing are carried out on the same data; and the lack of rigorous theoretical false discovery rate (FDR) control. We also have suggested some approaches that could be applied and improved upon to tackle these issues. To further enhance analysis and improve interpretability, developing dedicated visualization tools for transcript trees and differential testing would be highly beneficial to the community. Automating the construction of plots for visualizing `mehenDi` nodes on the tree proved to be nontrivial due to the complexity of tree structures and the arrangement of the corresponding transcripts for a gene. Future work could focus on designing software and integrating it with `mehenDi`, facilitating a more intuitive exploration of transcript-level differential expression results.

Finally, for `alevin-fry-atac` the immediate next steps include expanding support single-cell ATAC-seq data from other platforms, such as `dscATAC-Seq` [150], `sci-ATAC-seq3` [117, 151], `nano-well ICell based ATAC-Seq` [152], `SNARE-Seq` [153] in addition to existing support for 10X Genomics. Furthermore, we plan to extend support to protocols with varying specifications. Currently, each file is processed in its entirety, without an option to specify coordinate-based regions or arbitrary geometric selections within the nucleotide sequence. Enabling targeted processing based on predefined coordinates or more complex spatial patterns would enhance flexibility and efficiency for analyzing protocols with different geometries. While our modified pseudoalignment approach with virtual colors has been successfully applied to single-cell ATAC-seq data, the next goal is to extend it to other genomic mapping-based tasks and modalities, such as `DNase-seq`, `ChIP-seq`, and `Hi-C`. Additionally, the introduced caching scheme and modifications to the querying procedure have significantly improved mapping speed. This caching strategy can be further optimized by incorporating a negative-hit cache, which would be

a straightforward but impactful area to explore. Finally, both Cuttlefish and SSHash, which form the foundation of our indexing strategy, are actively evolving. Keeping up with their ongoing improvements will be crucial for further enhancing `alevin-fry-atac` and maintaining its state-of-the-art performance.

Appendix A: Supplementary Material for "TreeTerminus"

A.1 Supplementary material for Chapter 2

A.1.1 Tables

Table A1: Accession IDs for the tissue samples for the `MouseMuscle` dataset that have been used for analysis in this paper

Accession ID	TissueName
SRR5758624	Atria
SRR5758625	Atria
SRR5758626	Atria
SRR5758627	Atria
SRR5758628	Atria
SRR5758629	Atria
SRR5758707	TA
SRR5758706	TA
SRR5758705	TA
SRR5758704	TA
SRR5758703	TA
SRR5758702	TA

Table A2: Peak memory usage and running time for the different steps of Terminus and TreeTerminus (both Mean and Cons modes) on MouseMuscle dataset.

Method	Terminus		TreeTerminus (Mean)	TreeTerminus (Cons)	
	Group	Consensus	Group	Group	Consensus
Peak Memory (MB)	163	1950	2343	337	278
Time (h:m:s)	0:12:58	0:01:40	0:02:18	0:11:34	0:21:49

Table A3: Total number of transcripts covered by different methods across the datasets.

Method	BrSimNorm	BrSimLow	MouseMuscle	ChimpBrain
Mean	135138	135208	98091	39856
Cons	134377	134371	97360	39491
ConsFilt	121784	121657	89429	37858
AC	73205	73493	68062	35114
ConsFiltES	27156	26891	17939	15773
Term	13612	13730	7792	5428

Table A4: Median of mean inferential variance (MIRV) of the inner nodes for different trees stratified by their height for the BrSimNorm dataset. All nodes with height larger than 5 have been labelled as 5.

Tree	2	3	4	5
Mean	0.17	0.11	0.09	0.07
Cons	0.19	0.12	0.10	0.07
AC	0.52	0.32	0.21	0.12
ConsFilt	0.44	0.35	0.19	0.10
ConsFiltES	0.89	0.56	0.42	0.26

Table A5: Median of mean inferential variance (MIRV) of the inner nodes for different trees stratified by their height for the `BrSimLow` dataset. All nodes with height larger than 5 have been labelled as 5.

Tree	2	3	4	5
Mean	0.17	0.11	0.09	0.07
Cons	0.19	0.12	0.10	0.07
AC	0.52	0.33	0.21	0.13
ConsFilt	0.44	0.35	0.19	0.10
ConsFiltES	0.90	0.57	0.42	0.30

Table A6: Median of mean inferential variance (MIRV) of the inner nodes for different trees stratified by their height for the `MouseMuscle` dataset. All nodes with height larger than 5 have been labelled as 5.

Tree	2	3	4	5
Mean	0.12	0.09	0.08	0.07
Cons	0.12	0.09	0.08	0.07
AC	0.47	0.35	0.24	0.19
ConsFilt	0.24	0.14	0.10	0.08
ConsFiltES	0.68	0.54	0.29	0.02

Table A7: Median of mean inferential variance (MIRV) of the inner nodes for different trees stratified by their height for the `ChimpBrain` dataset. All nodes with height larger than 5 have been labelled as 5

Tree	2	3	4	5
Mean	0.07	0.07	0.07	0.07
Cons	0.08	0.07	0.07	0.07
AC	0.09	0.07	0.06	0.06
ConsFilt	0.09	0.07	0.07	0.07
ConsFiltES	0.37	0.17	0.12	0.10

Table A8: Total number of inner nodes mapping to more than 100 genes for different datasets.

Tree	BrSimNorm	BrSimLow	MouseMuscle	ChimpBrain
Mean	0	0	113	0
Cons	0	0	44	0
AC	717	719	696	304
ConsFilt	0	0	37	0
ConsFiltES	0	0	36	0

Table A9: True Positive Rate and False Discovery Rate for the different methods at nominal FDR cutoffs 0.01, 0.05, 0.10 for the BrSimNorm Dataset. The cuts are obtained by optimizing the metric `irv_height_desc` at different γ values and `lfc_desc` on the Cons tree. The performance is also computed when the inferential units consist of genes, transcripts and terminus groups.

Method	FDR			TPR		
	0.01	0.05	0.10	0.01	0.05	0.10
<code>irv_height_desc</code> ($\gamma = 0.05$)	0.011	0.042	0.084	0.629	0.731	0.775
<code>irv_height_desc</code> ($\gamma = 0.10$)	0.010	0.037	0.075	0.624	0.728	0.772
<code>irv_height_desc</code> ($\gamma = 1$)	0.002	0.035	0.077	0.392	0.718	0.778
<code>irv_height_desc</code> ($\gamma = 5$)	0.002	0.037	0.080	0.360	0.709	0.771
<code>irv_height_desc</code> ($\gamma = 10$)	0.010	0.044	0.081	0.593	0.712	0.766
<code>lfc_desc</code>	0.001	0.038	0.080	0.501	0.744	0.796
Genes	0.009	0.031	0.058	0.627	0.706	0.739
Txps	0.010	0.037	0.074	0.579	0.695	0.751
Terminus	0.009	0.038	0.076	0.593	0.712	0.766

A.1.2 Figures

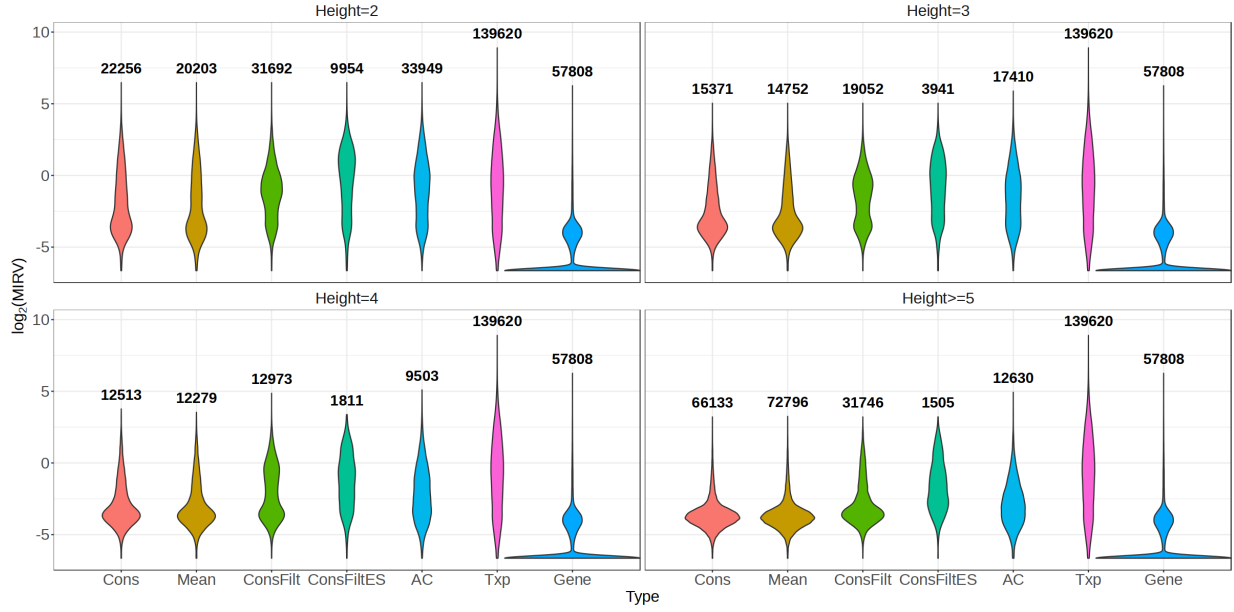


Figure A1: Distribution of $\log_2$ MIRV (mean inferential variance) across samples for the inner nodes stratified by their height for different trees for the BrSimLow Dataset, with the total number of inner nodes belonging to a method at a given height written on top of the violin plot. Also plotted for comparison at each height is the distribution of MIRV for the transcripts and genes.

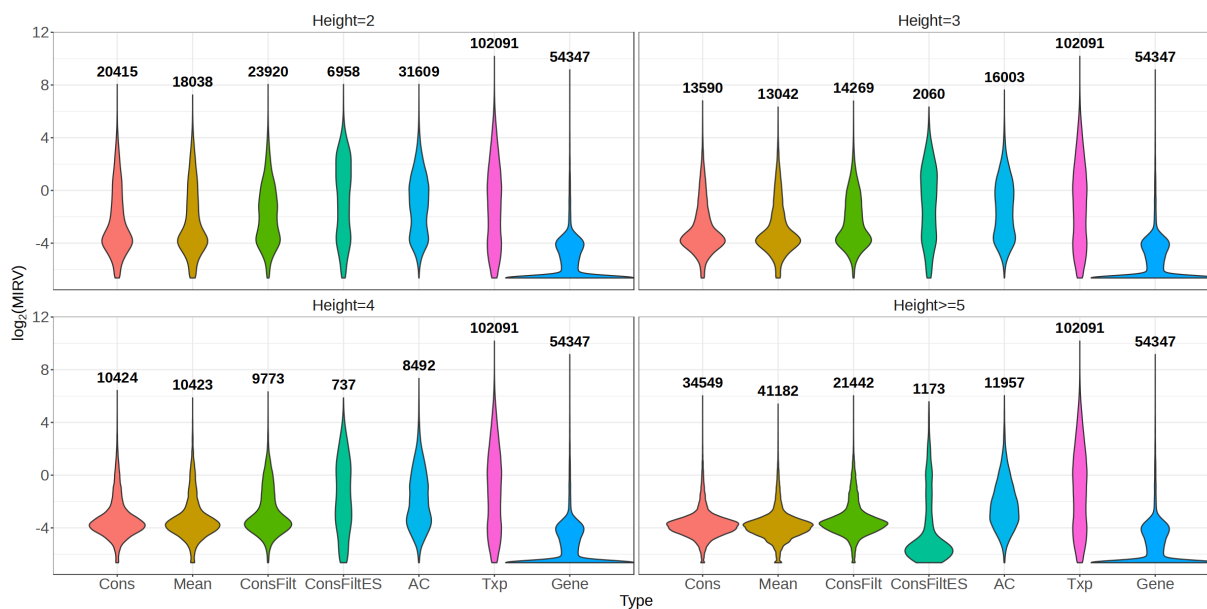


Figure A2: Distribution of $\log_2$ MIRV (mean inferential variance) across the inner nodes stratified by their height for different trees for the MouseMuscle Dataset, with the total number of inner nodes belonging to a method at a given height written on top of the violin plot. Also plotted for comparison at each height is the distribution of $\lg$ of MIRV for the transcripts and genes.

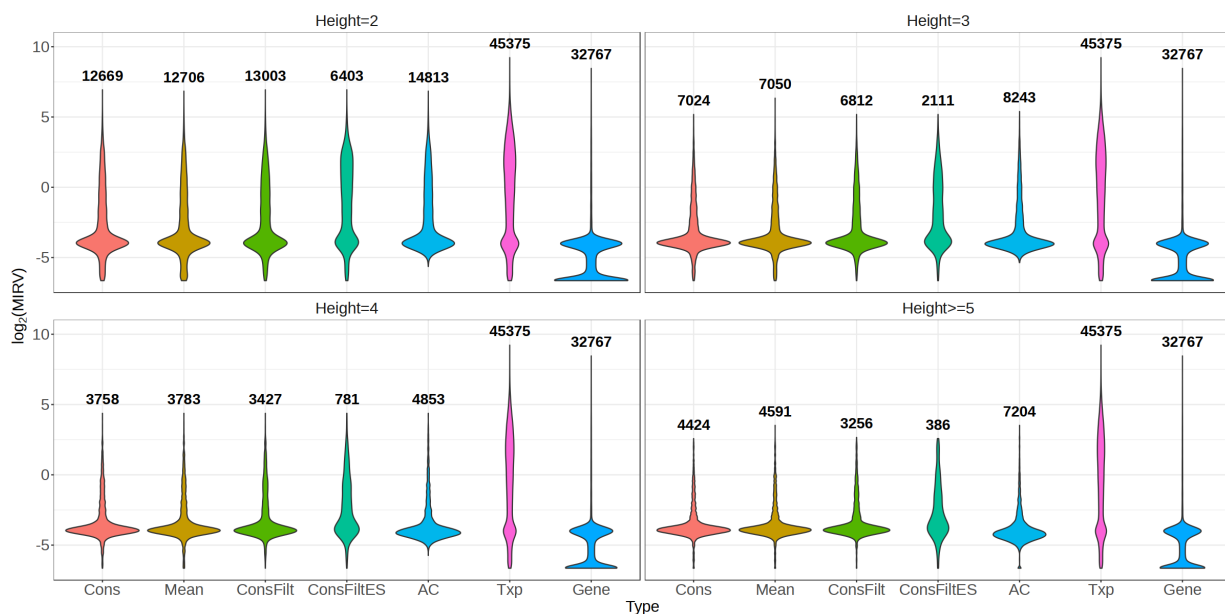


Figure A3: Distribution of $\log_2$ MIRV (mean inferential variance) across samples for the inner nodes stratified by their height for different trees for the ChimpBrain Dataset, with the total number of inner nodes belonging to a method at a given height written on top of the violin plot. Also plotted for comparison at each height is the distribution of MIRV for the transcripts and genes.

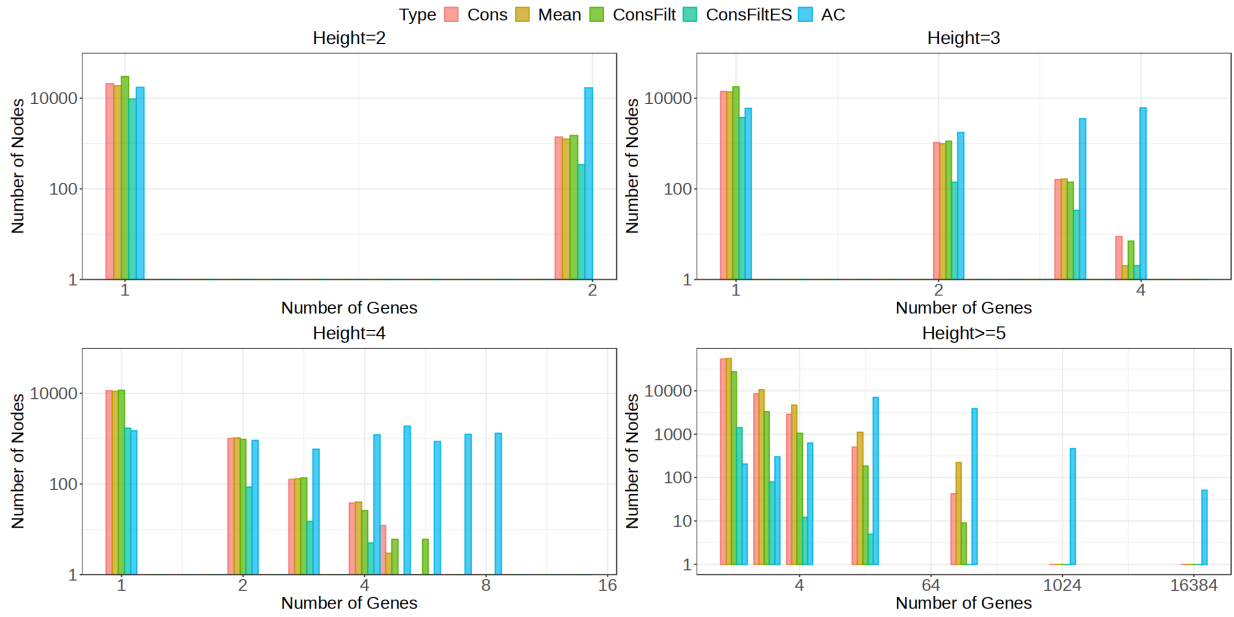


Figure A4: Comparison of different tree methods with respect to the number of genes to which an inner node in the tree maps for the BrSimLow dataset stratified by their height. The x-axis represents the number of unique genes that transcripts belonging to the inner nodes map to and y-axis represents the frequency of such mappings at a given height for a tree. For all the inner nodes located at a height greater than or equal to 5, number of unique genes were binned using the set $\{1, 2, 4, 16, 128, 1024, 16384\}$, with the bin representing number of unique genes less than or equal to the bin but larger than the bin left to it.

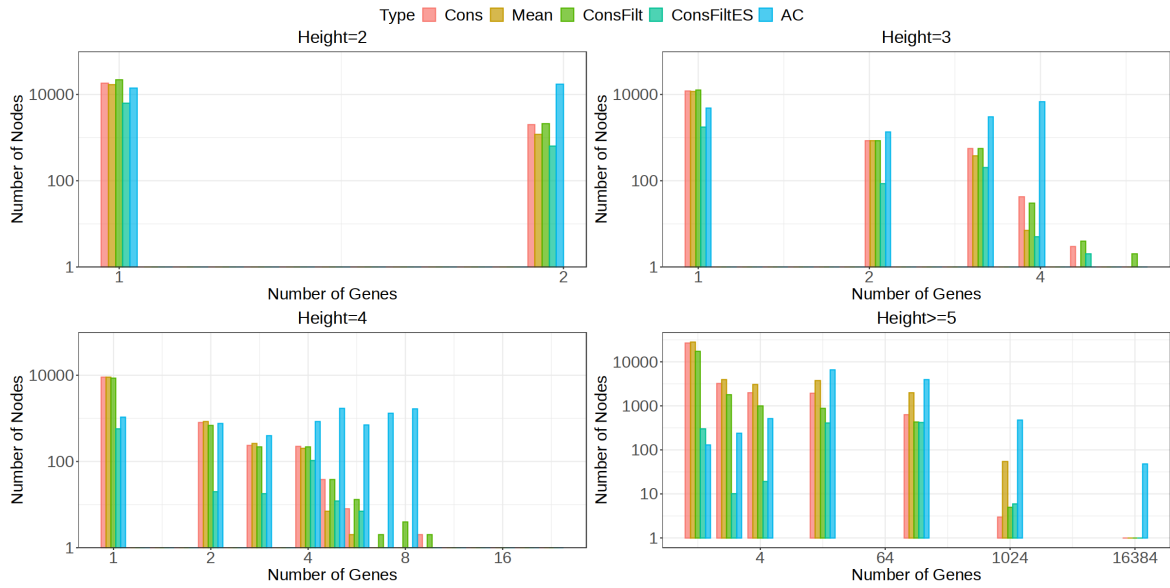
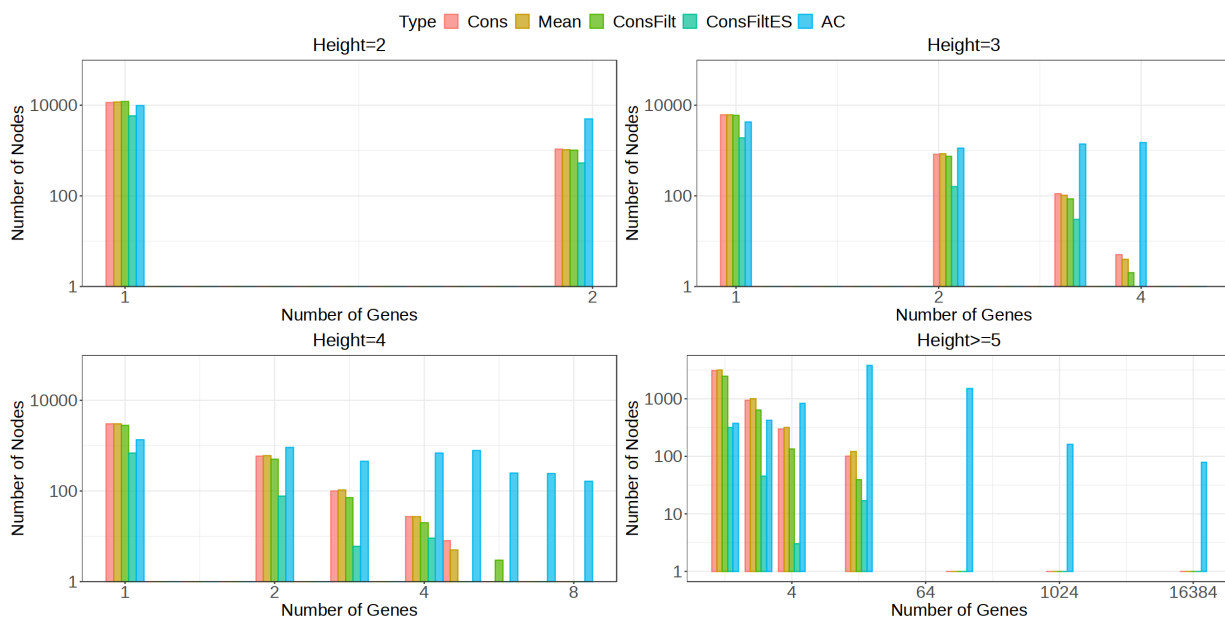


Figure A5: Comparison of different tree methods with respect to the number of genes an to which an inner node in the tree maps for the `MouseMuscle` dataset stratified by their height. The x-axis represents the number of unique genes that transcripts belonging to the inner nodes map to and y-axis represents the frequency of such mappings at a given height for a tree. For all the inner nodes located at a height greater than or equal to 5, number of unique genes were binned using the set $\{1, 2, 4, 16, 128, 1024, 16384\}$, with the bin representing number of unique genes less than or equal to the bin but larger than the bin left to it.



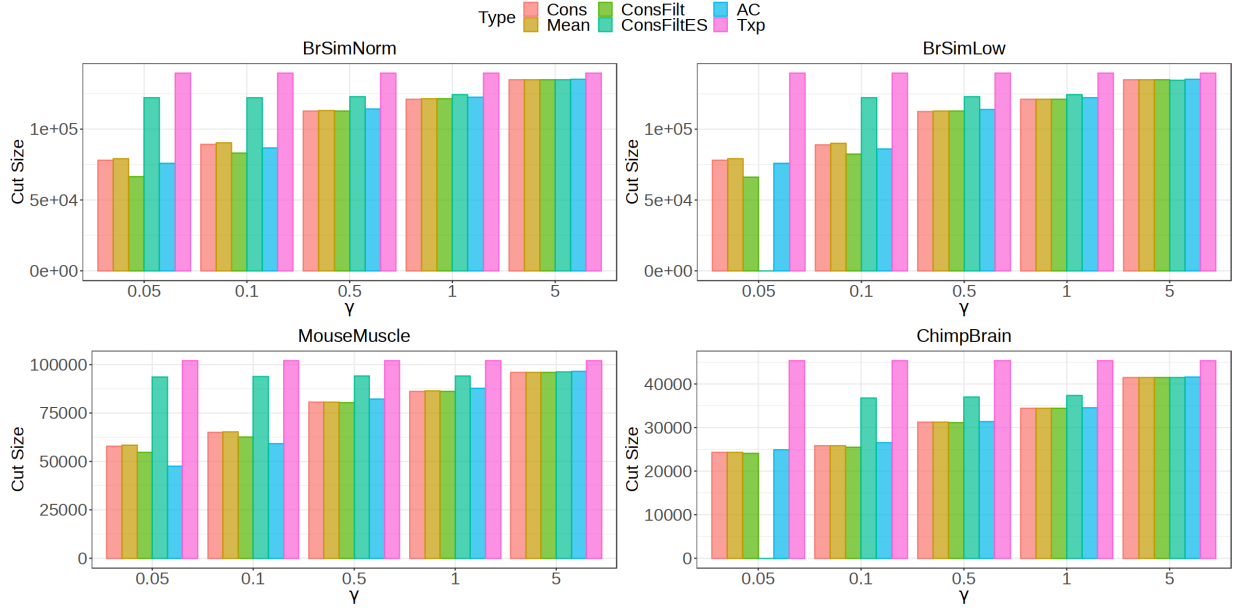


Figure A7: Distribution of the size of cuts on the different datasets obtained after solving for the objective function that minimizes the sum of metric - weighted sum of mean inferential variance and height multiplied by the number of descendant leaves (irv_height_desc) for the nodes in a cut. For each method, the distribution is plotted for a range of γ values. Also plotted for comparison are the total number of transcripts/leaves Txp.

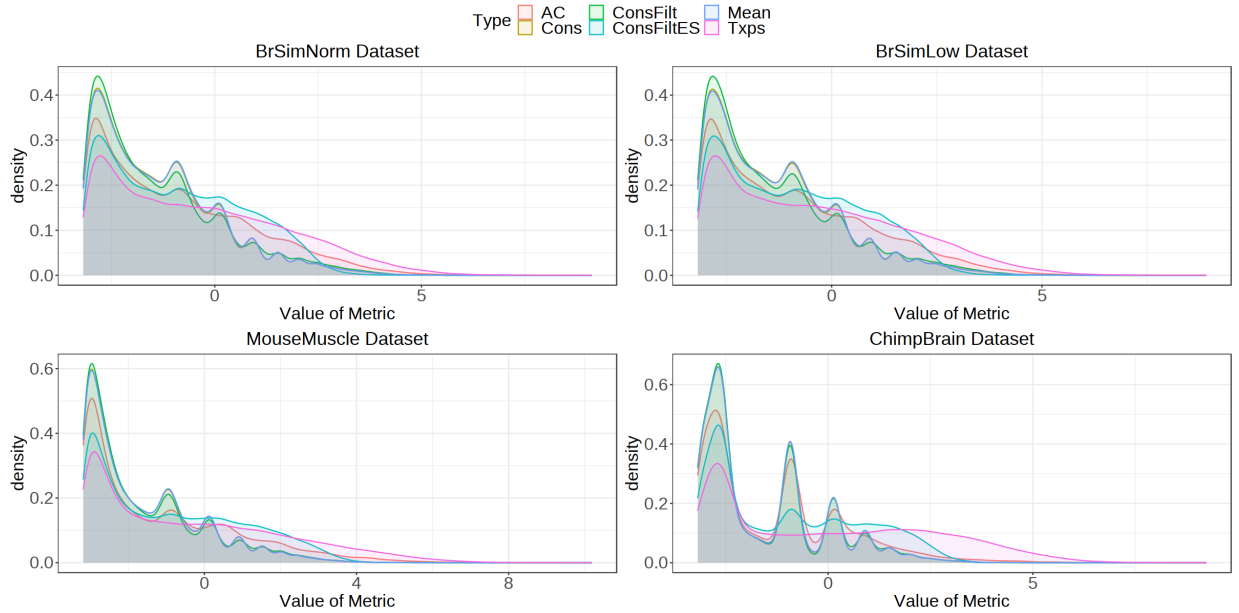


Figure A8: Distribution of the $\lg$ of the metric - weighted sum of mean inferential variance and height multiplied by the number of descendant leaves (irv_height_desc) for the nodes in the cut obtained after minimizing for the objective function using irv_height_desc as the underlying metric across trees on the different datasets. The metric has been computed using $\gamma = 0.1$. Also plotted for comparison is the distribution of irv_height_desc for transcripts.

Appendix B: Supplementary Material for ”Tree-based differential testing using inferential uncertainty for RNA-Seq”

B.1 Methods

B.1.1 Median-ratio scaled counts for the nodes in the tree

Using the formulation from [57], let Y^0 denote the counts matrix obtained for Salmon for the transcript set containing M samples $1, \dots, m$ and P transcripts $1, \dots, p$, with Y_{ji}^0 representing the counts for transcript j in sample i . Let the matrix $Y^{\mathcal{T}0}$ denote the counts obtained for all the nodes in the tree $\mathcal{T}$ that has P leaf nodes, where for an internal node n , $Y_{ni}^{\mathcal{T}0} = \sum_{d=1}^{|\Lambda(n)|} Y_{t_d i}^0, \forall t_d \in \Lambda(n)$, where $\Lambda(n)$ denote the indexes of the descendant transcripts of node n . The counts $Y_{ji}^{\mathcal{T}}$ are divided by a bias length term b_{ji} , accounting for the length w.r.t other transcripts:

$$b_{ji} = \frac{l_{ji}}{(\prod_{i=1}^m l_{ji})^{\frac{1}{m}}}$$

Then we divide the counts by b_{ji} as

$$Y_{ji}^{\mathcal{T}*} = \frac{Y_{ji}^{\mathcal{T}0}}{b_{ji}}$$

The counts are then scaled to the geometric mean of sequencing depth as

$$Y_{ji}^{\mathcal{T}^{**}} = \frac{Y_{ji}^{\mathcal{T}^*}}{\sum_{j=1}^P Y_{ji}^{\mathcal{T}^*}} \times \left(\prod_{i=1}^m \sum_{j=1}^P Y_{ji}^{\mathcal{T}^0} \right)^{\frac{1}{m}}$$

For each sample i , a median-ratio size factor is computed as

$$s_i = \text{median}_j^P \frac{Y_{ji}^{\mathcal{T}^{**}}}{\left(\prod_{k=1}^m Y_{jk}^{\mathcal{T}^{**}} \right)^{\frac{1}{m}}}$$

We compute the size factor over only the leaf nodes. The final normalized counts are then computed as:

$$Y_{ji}^{\mathcal{T}} = \frac{Y_{ji}^{\mathcal{T}^{**}}}{s_i}$$

The lengths of the inner nodes are computed using the strategy employed by `summarizeToGene` function in the R package `tximport` [154]. The length of an inner node n for sample i is computed as:

$$l_{ni}^{\mathcal{T}} = \frac{\sum_{d=1}^{|\Lambda(n)|} l_{t_{di}} tpm_{t_{di}}}{\sum_{d=1}^{|\Lambda(n)|} tpm_{t_{di}}}, \forall t_d \in \Lambda(n)$$

Here tpm refers to transcripts per million estimates that are provided by Salmon.

B.1.2 Distance between nodes

The distance between the set of nodes N_d and N_p is computed as :

$$\mathcal{D}(N_d, N_p) = \frac{dist(N_d, N_p) + dist(N_p, N_d)}{2},$$

$$dist(N_a, N_b) = \frac{\sum_{i=1}^{|N_a|} d(N_{ai})}{\|N_a\|},$$

$$d(N_{ai}) = \begin{cases} 0 & \text{if } N_{ai} \in N_b \\ Path_length(N_{ai}, N_{bk}) & \text{if } N_{bk} \in N_b \text{ and } N_{bk} \text{ is either an ancestor} \\ & \text{or descendant of } N_{ai} \\ Path_length(N_{ai}, \text{root}) + 1 & \text{if } N_{ai} \notin N_b \text{ and no ancestor or} \\ & \text{descendant of } N_{ai} \text{ exists in } N_b \end{cases}$$

where N_d, N_p denote the node set output by `mehenDi` at the default parameters and parameter set p respectively. $Path_length(N_{ai}, N_{bk})$ denote the length of the path between the nodes N_{ai} and N_{bk} . We are computing the average distance per node between the two sets. If the same node is present in both sets, the distance between them would be 0. Similarly, if for a node belonging to one set, there exists a node in the other set which is an ancestor/descendant for it, then the distance is computed by calculating the length of the path between them on the tree. On the other hand, if there is no ancestor/descendant for a node in the other set, then the distance is the length of the path from the root to that node with 1 added. 1 is added since this would be the lowest height node in the other set. The nodes that don't have an ancestor or descendant in the other set, can be the largest contributing factor to the distance metric and can create asymmetry

for the overall distance metric aka $dist(N_a, N_b) \neq dist(N_b, N_a)$, as they do not directly have a counterpart in the other set. This can skew the metric, depending on the set w.r.t which distance is computed, especially if that set consists of nodes that represent unique branches in the tree. To balance this, our final distance metric $\mathcal{D}(N_d, N_p)$ is the average of $dist(N_d, N_p), dist(N_p, N_d)$.

B.2 Tables

Table B1: Distribution of the height of the selected nodes output by `mehenDi` and candidate nodes by `treeclimbR` for the `BrSimNorm` dataset.

Height of nodes	mehenDi	treeclimbR
1	4013	2941
2	554	326
3	256	164
4	125	116
5	62	80
6-10	57	152
>10	5	39
Max height	14	19

Table B2: Proportion of unique nodes that are observed for `mehenDi`, `treeclimbR` when doing a pairwise comparison for the `BrSimNorm` dataset at different nominal FDR. The number inside the parentheses represents the total number of nodes.

Nominal FDR	0.01	0.05	0.10
mehenDi	0.085 (199)	0.412 (308)	0.606 (381)
treeclimbR (N)	0 (2)	0 (4)	0.25(4)

Table B3: Proportion of unique nodes that are observed for `Txps`, `treeclimbR` when doing a pairwise comparison for the `BrSimNorm` dataset at different nominal FDR. The number inside the parentheses represents the total number of nodes.

Nominal FDR	0.01	0.05	0.10
<code>Txps</code>	0.103 (145)	0.416 (255)	0.590 (339)
<code>treeclimbR</code> (N)	0 (15)	0.24 (25)	0.481 (27)

Table B4: Number of differentially expressed genes obtained at the different nominal thresholds for the `MouseMuscle` dataset between EDL and MAST.

Nominal FDR	0.01	0.05	0.10
Number of DEGs	2535	4494	5814

Table B5: Number of differentially expressed transcripts obtained at the different nominal FDR thresholds for the `MouseMuscle` dataset between EDL and MAST.

Nominal FDR	0.01	0.05	0.10
Number of DTEs	3149	6849	9592

Table B6: Distribution of the height of the `mehenDi` nodes for the `MouseMuscle` dataset.

Nominal FDR	0.01	0.05	0.10
1	2522	5406	7500
2	651	1285	1698
3	295	555	696
4	134	252	308
≥ 5	109	195	256

Table B7: Distribution of the total number of unique genes, the transcripts covered by `mehenDi` nodes map to for the `MouseMuscle` dataset.

Nominal FDR	0.01	0.05	0.10
1	3672	7625	10369
2	33	56	73
3	3	8	9
4	0	4	4
5	1	0	0
7	0	0	1
9	0	0	2

Table B8: Number of differentially expressed genes (DEGs) that don't contain a single underlying transcript that is called significant at the different nominal FDR thresholds for the `MouseMuscle` dataset.

Nominal FDR	0.01	0.05	0.10
Number of DEGs	685	839	907

Table B9: Number of differentially expressed transcripts (DETs) that don't map to a single significant gene at the different nominal FDR thresholds for the `MouseMuscle` dataset.

Nominal FDR	0.01	0.05	0.10
Number of DETs	835	1736	2214

Table B10: Number of `mehenDi` nodes that neither map to a significant gene nor contain a single significant transcript for the `MouseMuscle` dataset.

Nominal FDR	0.01	0.05	0.10
Number of <code>mehenDi</code> nodes	231	382	390

Table B11: Number of differentially expressed genes (DEGs) at the different nominal FDR thresholds for the `ChimpBrain` dataset.

Nominal FDR	0.01	0.05	0.10
Number of DEGs	6090	8256	9354

Table B12: Number of differentially expressed transcripts (DETs) at the different nominal FDR thresholds for the ChimpBrain dataset.

Nominal FDR	0.01	0.05	0.10
Number of DETs	3539	10133	11886

Table B13: Distribution of the height of the mehenDi nodes for the ChimpBrain dataset.

Nominal FDR	0.01	0.05	0.10
1	2721	7979	9272
2	705	1387	1731
3	342	554	643
4	126	196	229
≥ 5	107	131	142

Table B14: Distribution of the total number of unique genes, the transcripts covered by mehenDi nodes map to for the ChimpBrain dataset.

Nominal FDR	0.01	0.05	0.10
1	3943	58	11842
2	10106	140	1
3	11842	174	1

Table B15: Number of differentially expressed genes (DEGs) that don't contain a single underlying transcript that is called significant at the different nominal FDR thresholds for the ChimpBrain dataset.

Nominal FDR	0.01	0.05	0.10
Number of DEGs	2984	787	716

Table B16: Number of differentially expressed transcripts (DETs) that don't map to significant genes for the ChimpBrain dataset.

Nominal FDR	0.01	0.05	0.10
Number of DETs	130	835	854

Table B17: Number of `mehenDi` nodes that neither map to a significant gene nor contain a single significant transcript for the `ChimpBrain` dataset.

Nominal FDR	0.01	0.05	0.10
Number of <code>mehenDi</code> nodes	31	59	76

B.3 Figures

Figure B1: Distribution of the p-values for the leaf and inner nodes on the null simulations, when the hypothesis testing is carried out separately.

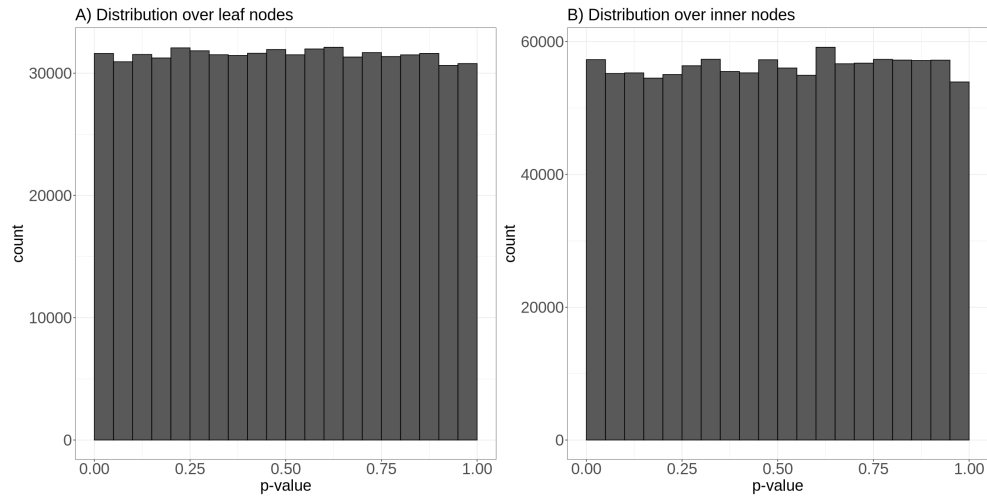


Figure B2: UpSet plot covering the number of true positive transcripts that are covered by the nodes that are output by the different methods for the `BrSimNorm` dataset nominal FDR.

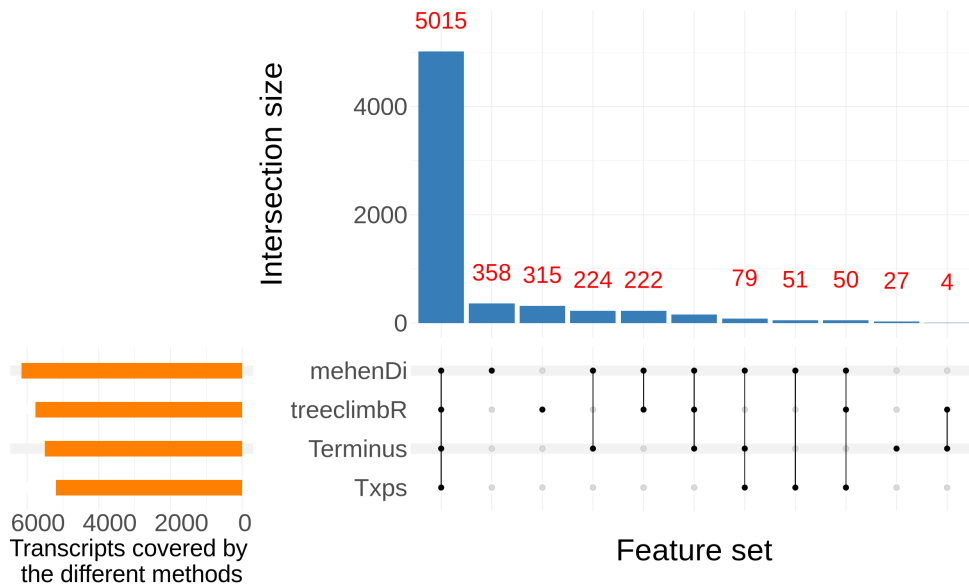


Figure B3: UpSet plot covering the number of true positive transcripts that are covered by the nodes that are output by the different methods for the BrSimLow dataset at the 0.01 nominal FDR.

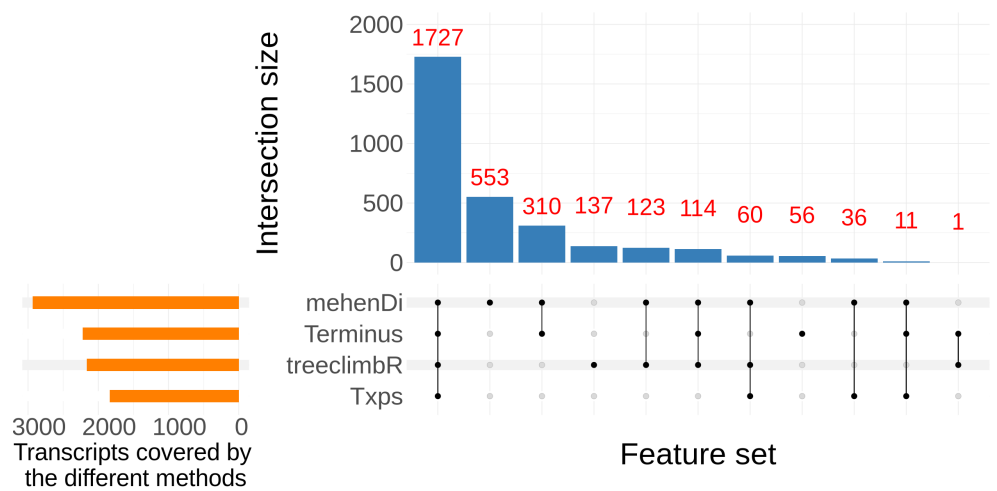


Figure B4: True Positive Rates and Empirical False Discovery Rates at the different nominal FDR thresholds by individually varying the parameters $minP$ and $mIrvThresh$ for the BrSimNorm dataset. Both the metrics have been rounded to 3 decimal places

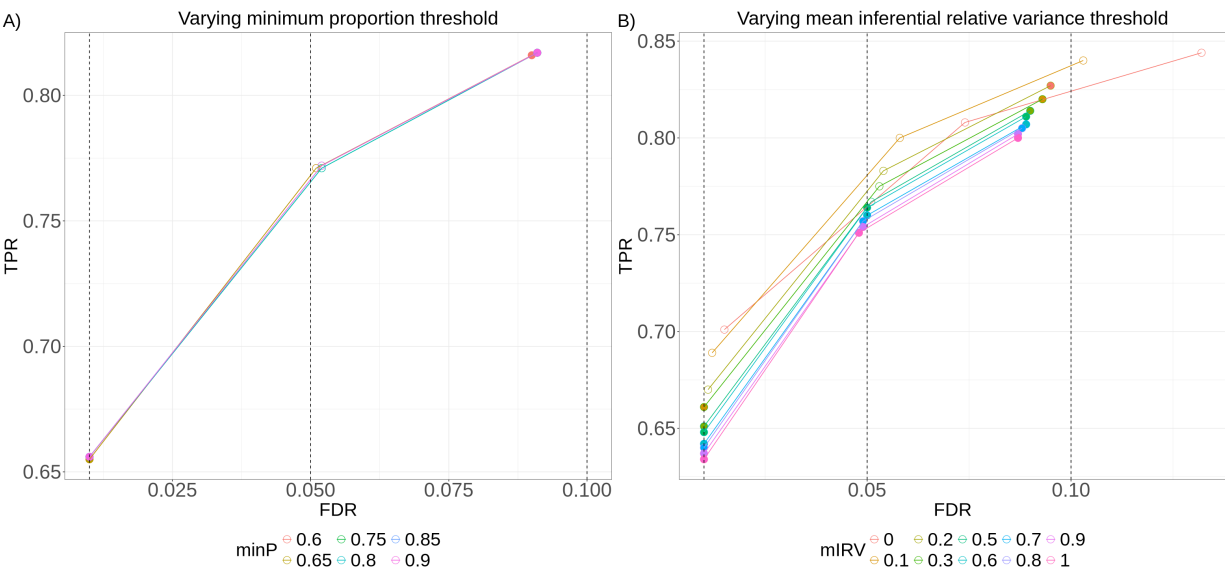


Figure B5: Example of a mehenDi node that is overaggregated in Terminus for the BrSimNorm dataset. A) Subtree representing the transcripts covered by the Terminus group. B) Inferential replicates for the Terminus group. C) Inferential replicates for the selected node output by mehenDi.

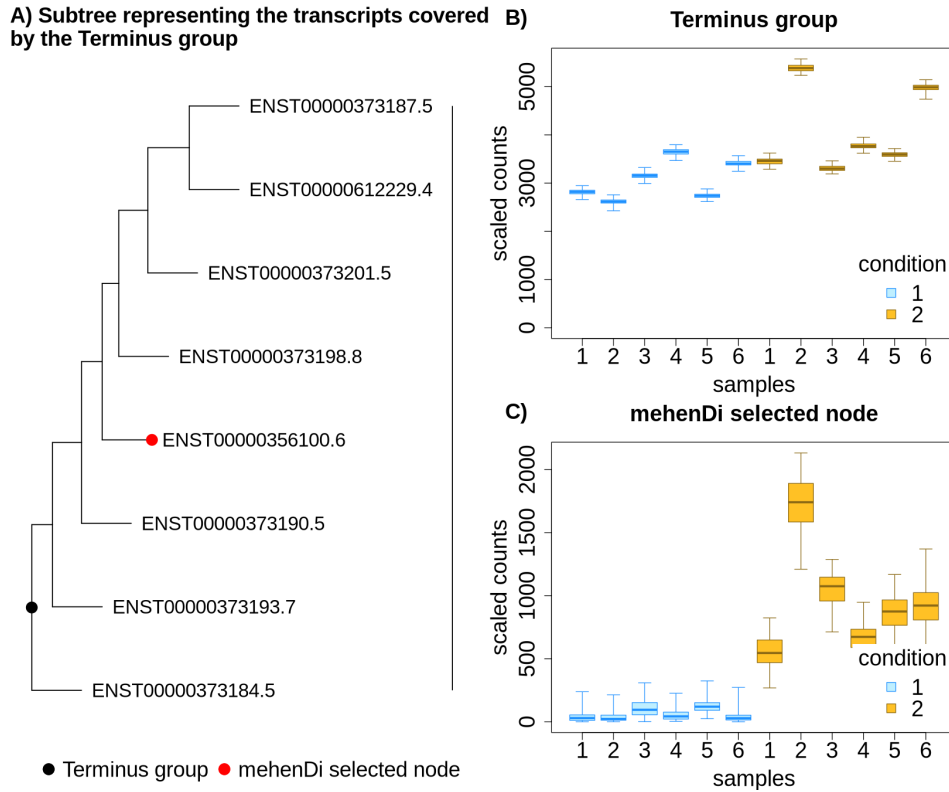


Figure B6: Example of a `mehenDi` node that is overaggregated in `Terminus` for the `BrSimNorm` dataset. A) Subtree representing the transcripts covered by the `Terminus` group. B) Inferential replicates for the `Terminus` group. C) Inferential replicates for the selected node output by `mehenDi`.

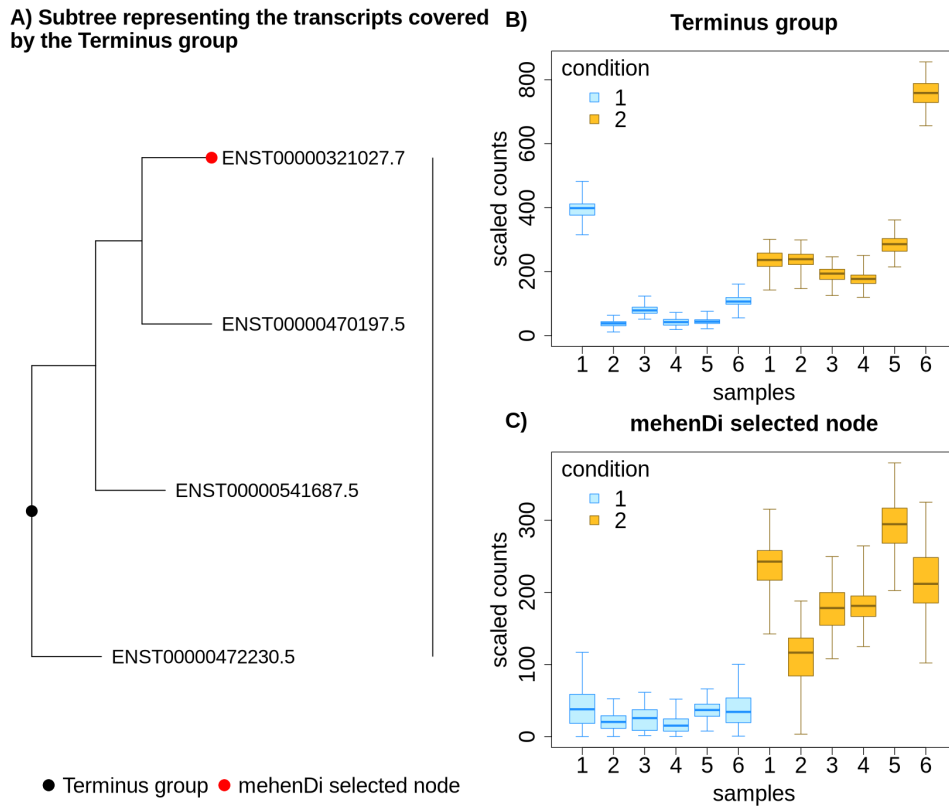


Figure B7: Example of a `mehenDi` node that is not aggregated enough in `Terminus` for the `BrSimNorm` dataset. A) Subtree representing the transcripts covered by the `mehenDi` group. B) Inferential replicates for the `Terminus` group. C) Inferential replicates for the selected node output by `mehenDi`.

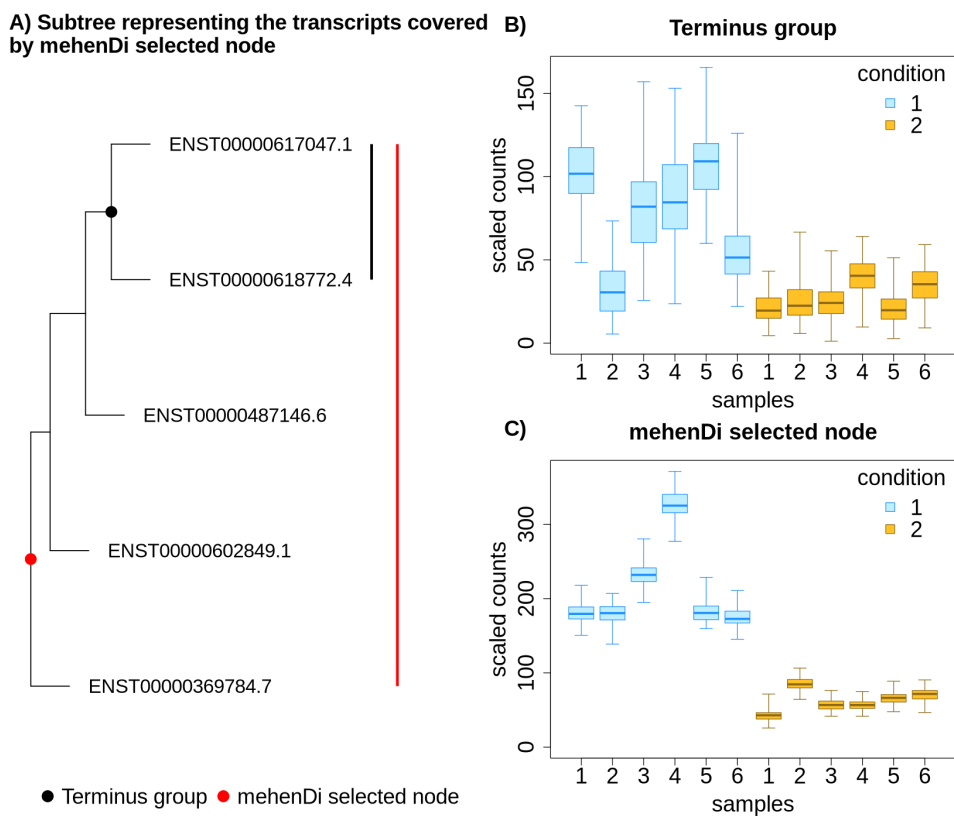


Figure B8: Example of a `mehenDi` node that is not aggregated enough in `Terminus` for the `BrSimNorm` dataset. A) Subtree representing the transcripts covered by the `mehenDi` group. B) Inferential replicates for the `Terminus` group. C) Inferential replicates for the selected node output by `mehenDi`.

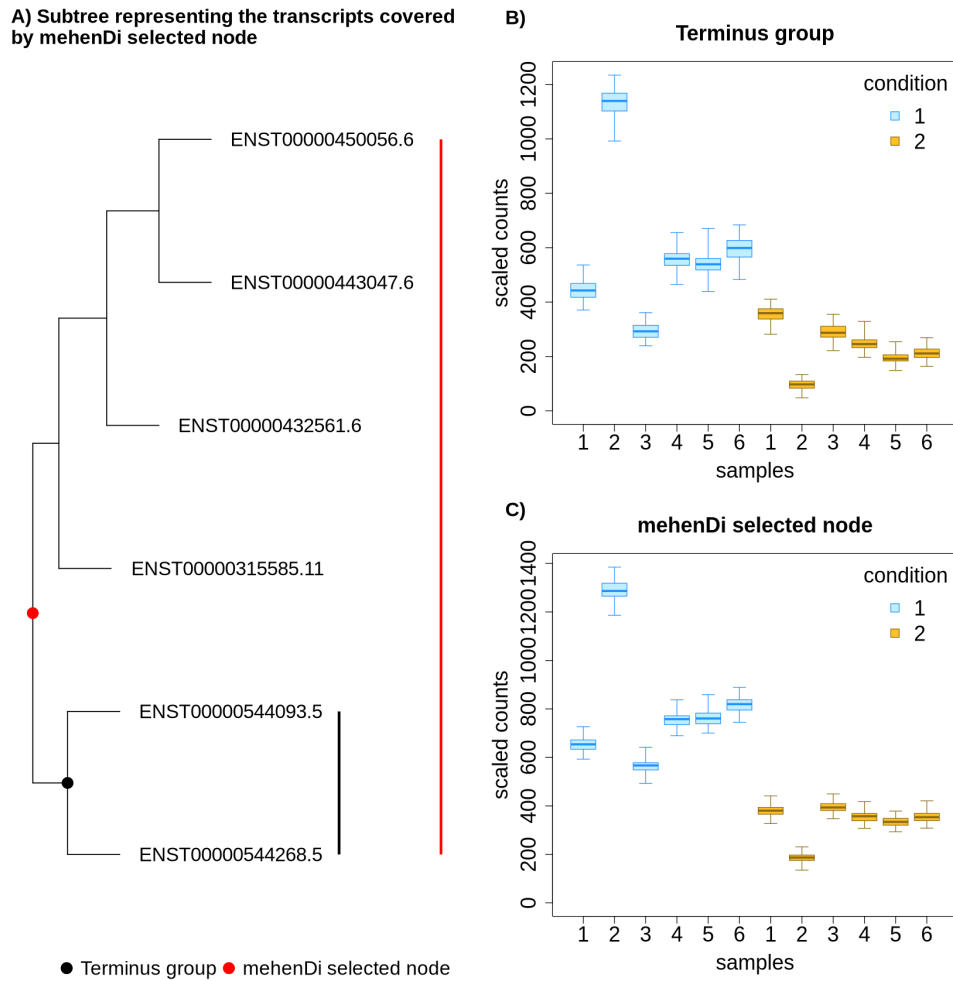


Figure B9: Examination of error metrics for the unique nodes obtained for `treeclimbR` when doing `treeclimbR` vs `Txps` analysis at the different nominal FDR thresholds. We vary the magnitude of log fold change (LFC) and plot the empirical FDR and the total number of nodes that are left after filtering the unique nodes based on LFC.

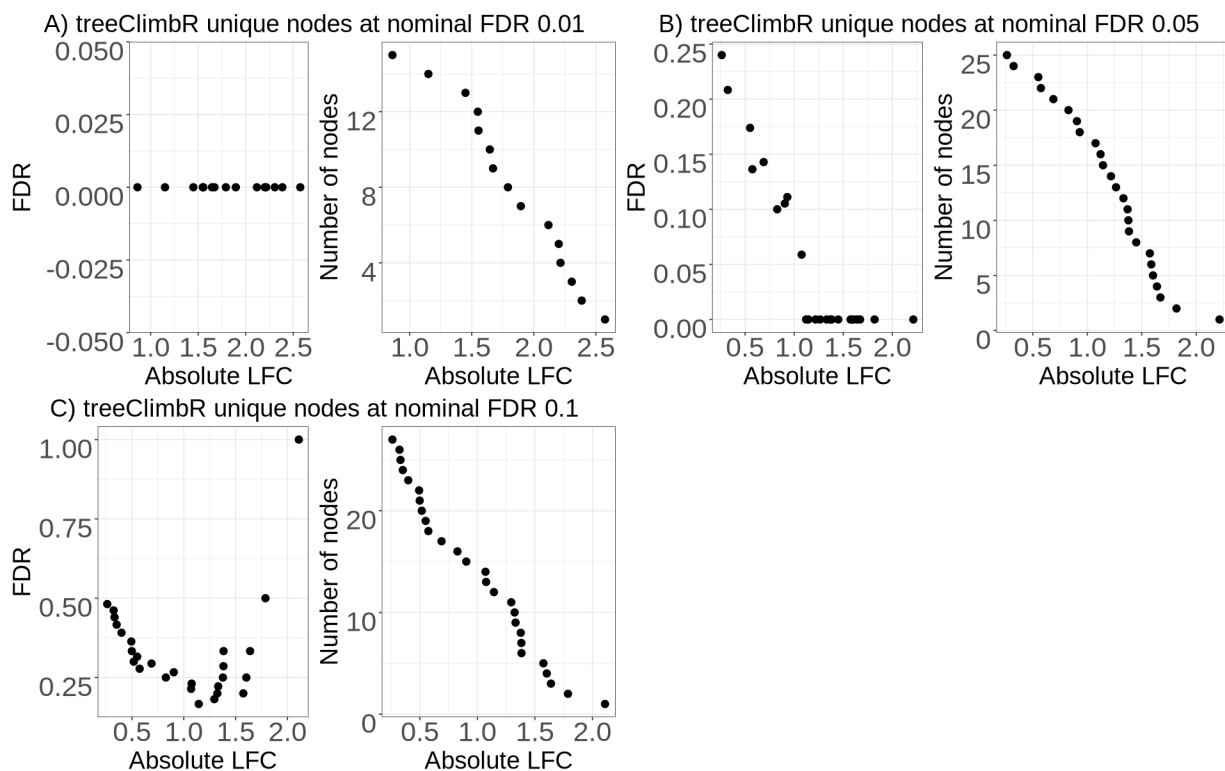


Figure B10: Examination of error metrics for the unique nodes obtained for Txps when doing treeclimbR vs Txps analysis at the different nominal FDR thresholds. We vary the magnitude of log fold change (LFC) and plot the empirical FDR and the total number of nodes that are left after filtering the unique nodes based on LFC.

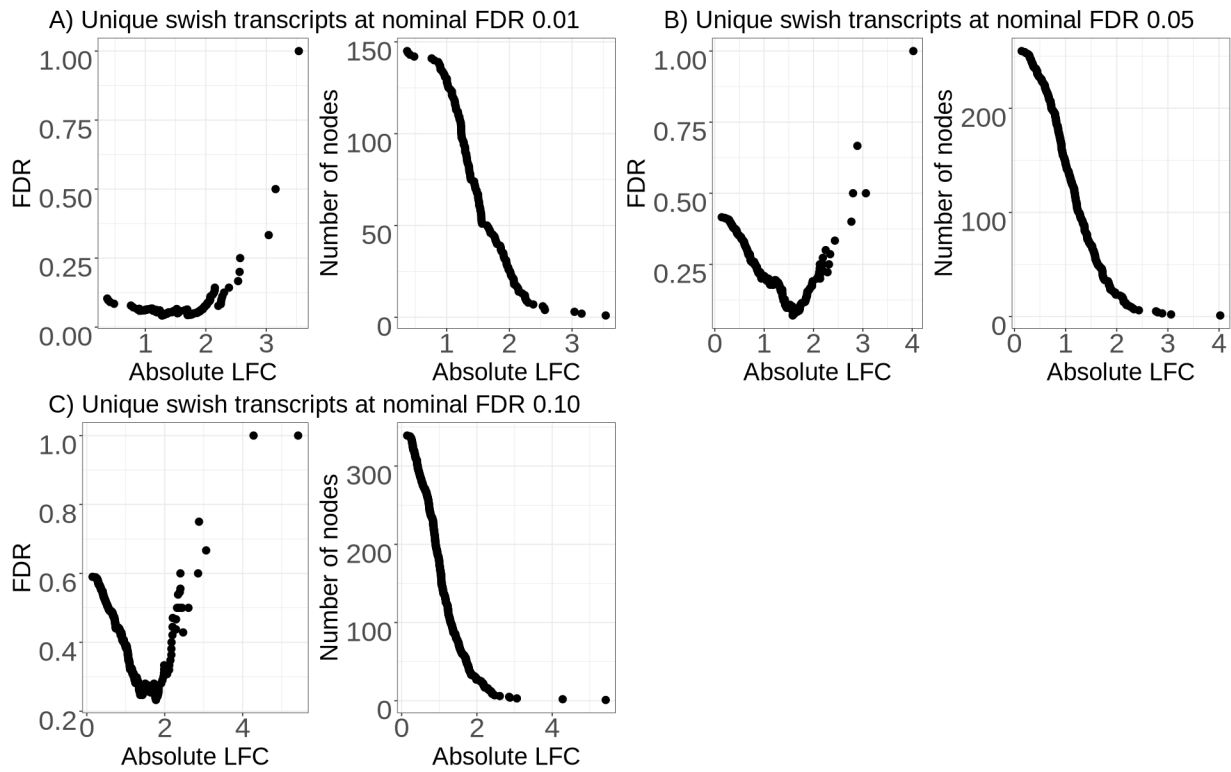


Figure B11: The first two dimensions of the PCA using the top 1000 variable features for the `MouseMuscle` dataset

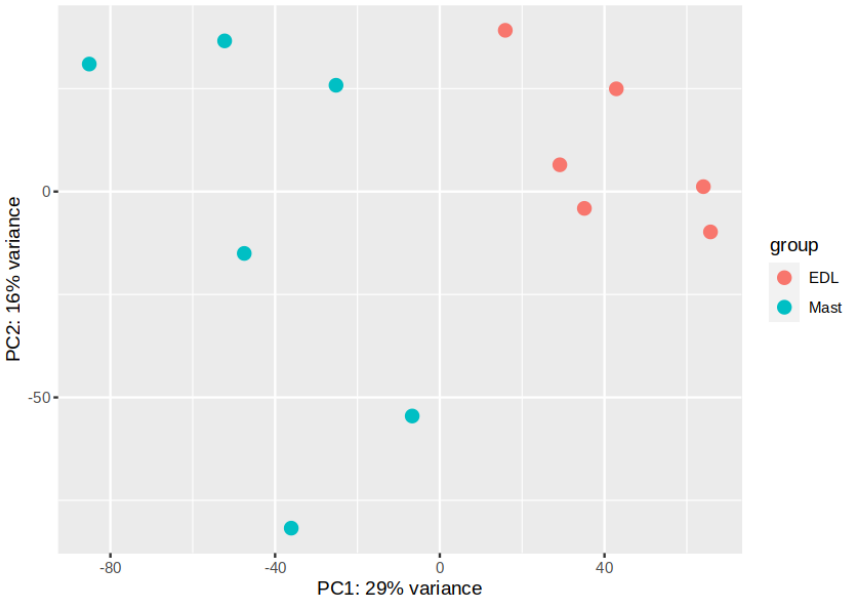


Figure B12: The average distance between the nodes obtained for mehenDi using default parameters and varying $minP$ and $mIrvThresh$ individually for the MouseMuscle dataset.

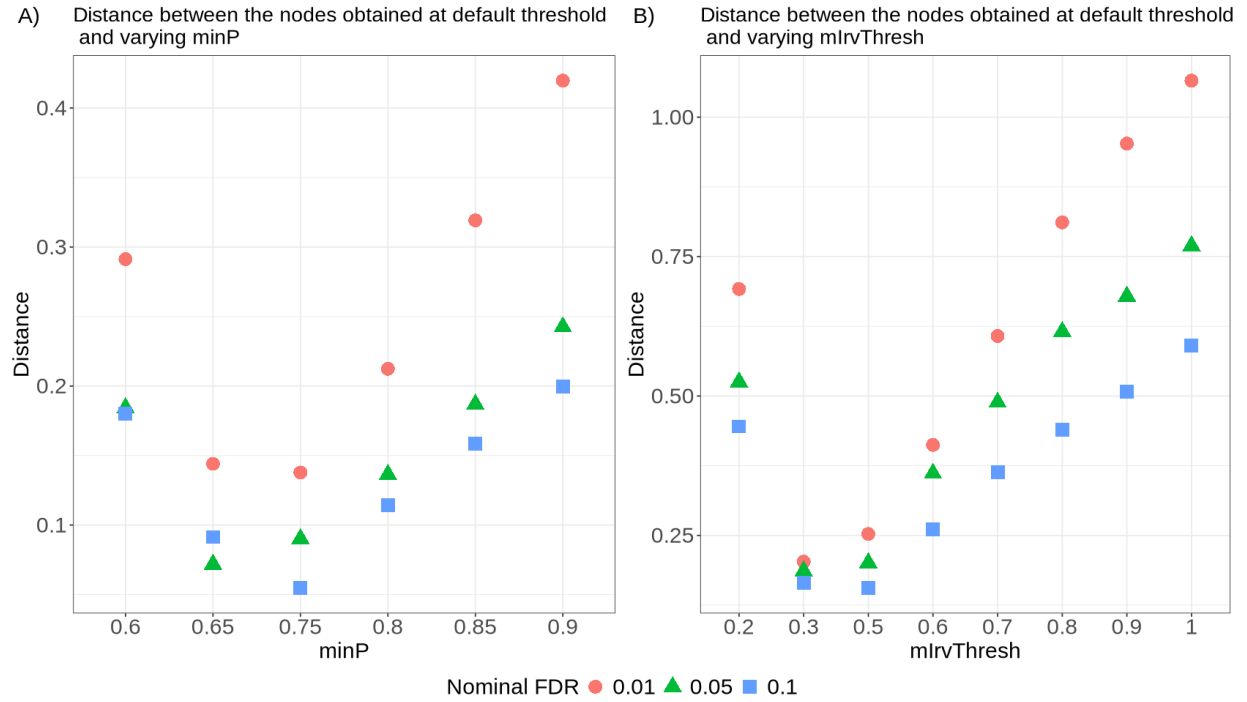


Figure B13: UpSet plot covering the number of transcripts that are covered by the nodes that are output by the different methods for the MouseMuscle dataset.

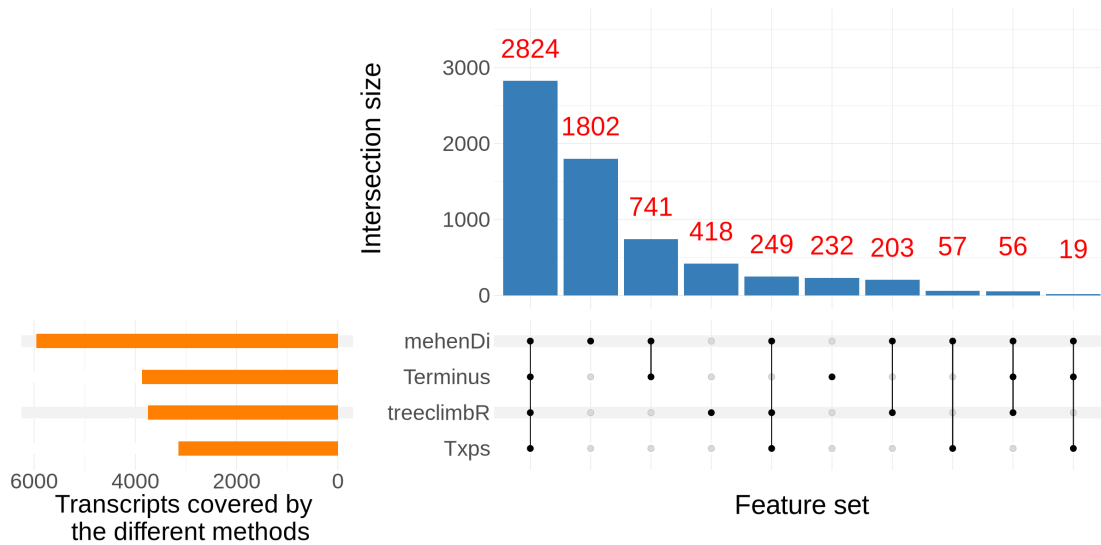


Figure B14: Examining the transcript profile for the gene *Hmcn2* in the `MouseMuscle` dataset. A) Transcripts in a pileup style. B) Tree representing the transcripts covered by the gene *Hmcn2*, with the red node representing the transcripts covered by the `mehenDi` selected node. C) Inferential replicates for the transcript ENSMUST00000138821.7, which had the lowest p-value among all the transcripts in the tree. D) Inferential replicates for the `mehenDi` selected node.

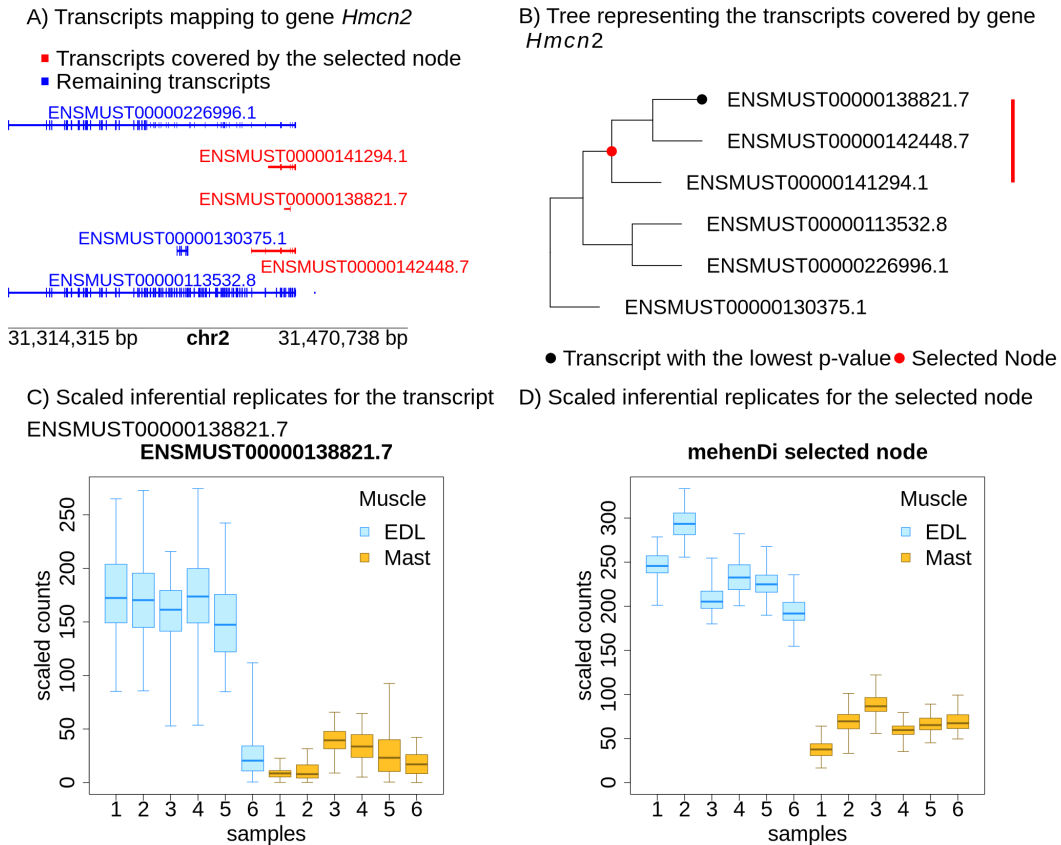


Figure B15: Examining the transcript profile for the gene *Emid1* in the `MouseMuscle` dataset. A) Transcripts in a pileup style. B) Tree representing the transcripts covered by the gene *Emid1*, with the red node representing the transcripts covered by the `mehenDi` selected node. C) Inferential replicates for the transcript ENSMUST00000151906.7, which had the lowest p-value among all the transcripts in the tree. D) Inferential replicates for the `mehenDi` selected node.

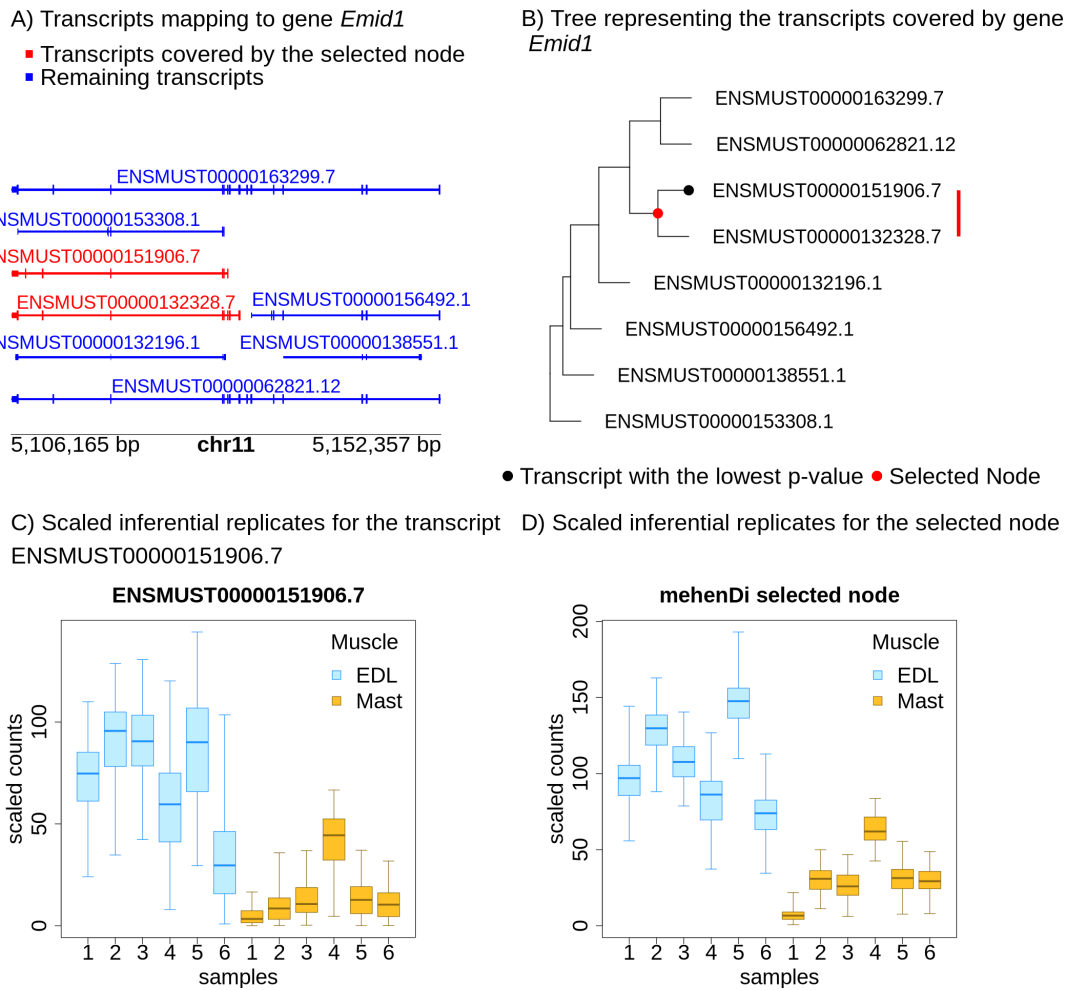


Figure B16: Examining the transcript profile for the gene *Prss55* in the `MouseMuscle` dataset. A) Transcripts in a pileup style. B) Tree representing the transcripts covered by the gene *Prss55*, with the red node representing the transcripts covered by the `mehenDi` selected node. C) Inferential replicates for the transcript ENSMUST00000171503.7, which had the lowest p-value among all the transcripts in the tree. D) Inferential replicates for the `mehenDi` selected node.

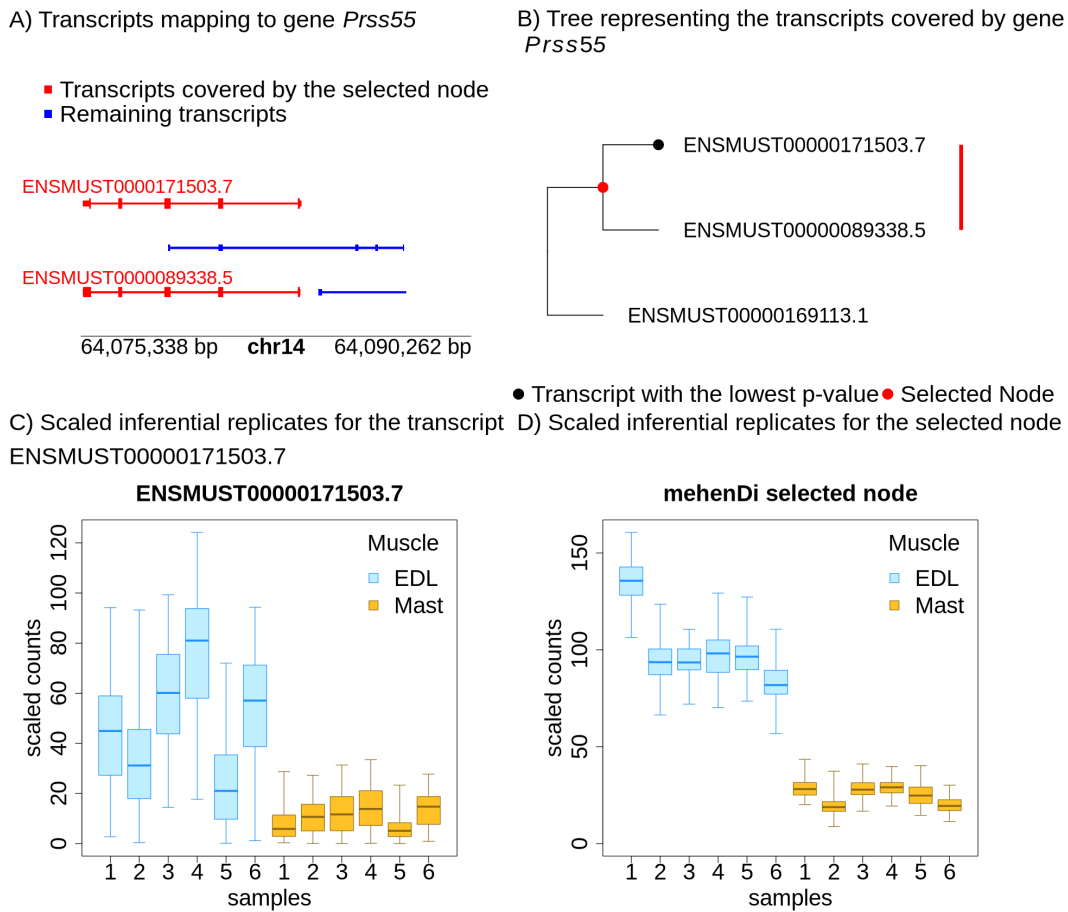


Figure B17: Examining the transcript profile for the gene *Stard10* in the `MouseMuscle` dataset. A) Transcripts in a pileup style. B) Tree representing the transcripts covered by the gene *Stard10*, with the red node representing the transcripts covered by the `mehenDi` selected node. C) Inferential replicates for the transcript ENSMUST000000032927.13, which had the lowest p-value among all the transcripts in the tree. D) Inferential replicates for the `mehenDi` selected node.

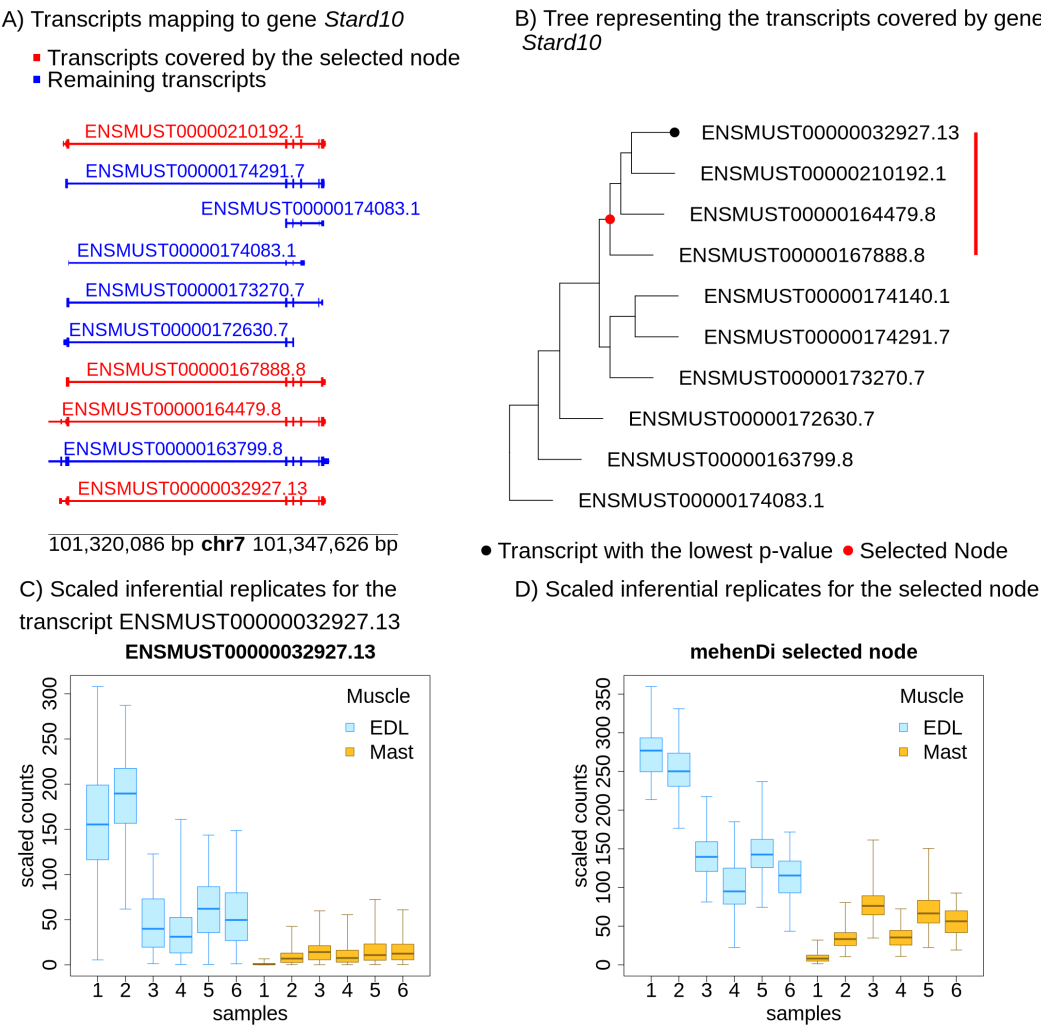


Figure B18: Examining the transcript profile for the gene *Syk* in the *MouseMuscle* dataset. A) Transcripts in a pileup style. B) Tree representing the transcripts covered by the gene *SYK*, with the red node representing the transcripts covered by the *mehenDi* selected node. C) Inferential replicates for the transcript ENSMUST00000055087.6, which had the lowest p-value among all the transcripts in the tree. D) Inferential replicates for the *mehenDi* selected node.

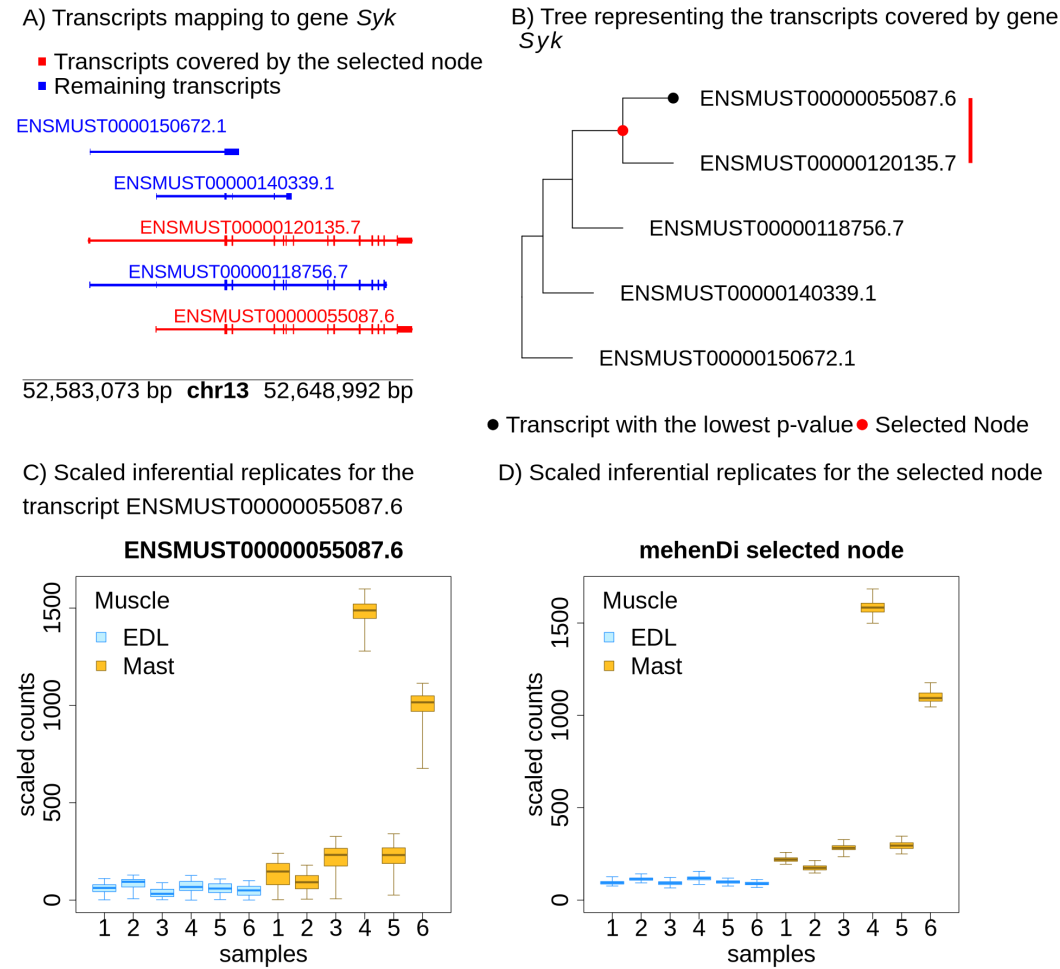


Figure B19: UpSet plot covering the number of transcripts that are covered by the nodes that are output by the different methods for the ChimpBrain dataset.

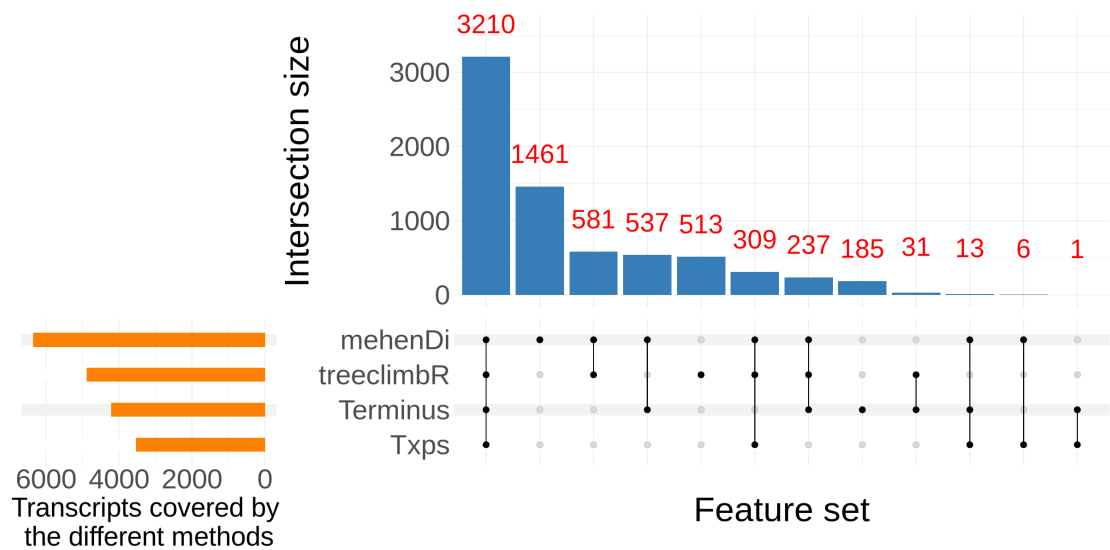


Figure B20: Examining the transcript profile for the gene *CABIN1* in the ChimpBrain dataset. A) Transcripts in a pileup style. B) Tree representing the transcripts covered by the gene *CABIN1*, with the red node representing the transcripts covered by the *mehenDi* selected node. C) Inferential replicates for the transcript ENSMUST00000103768, which had the lowest p-value among all the transcripts in the tree. D) Inferential replicates for the *mehenDi* selected node.

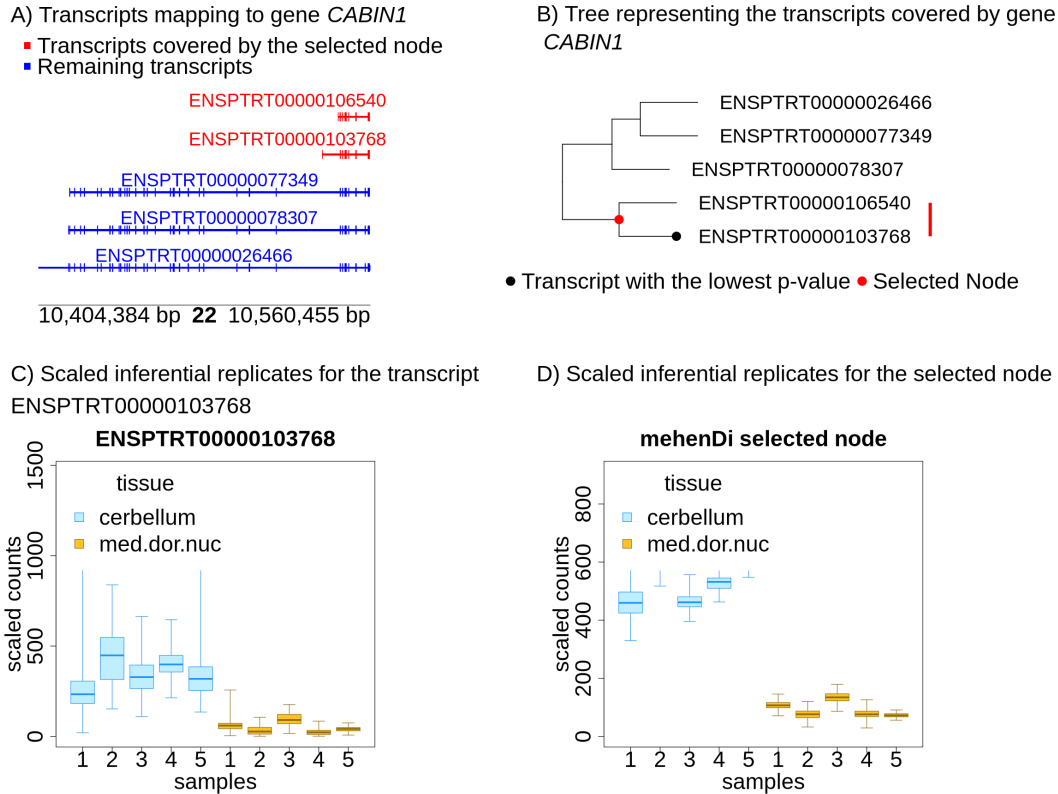


Figure B21: Examining the transcript profile for the gene *CATSPERG* in the ChimpBrain dataset. A) Transcripts in a pileup style. B) Tree representing the transcripts covered by the gene *CATSPERG*, with the red node representing the transcripts covered by the *mehenDi* selected node. C) Inferential replicates for the transcript ENSMUST00000020240, which had the lowest p-value among all the transcripts in the tree. D) Inferential replicates for the *mehenDi* selected node.

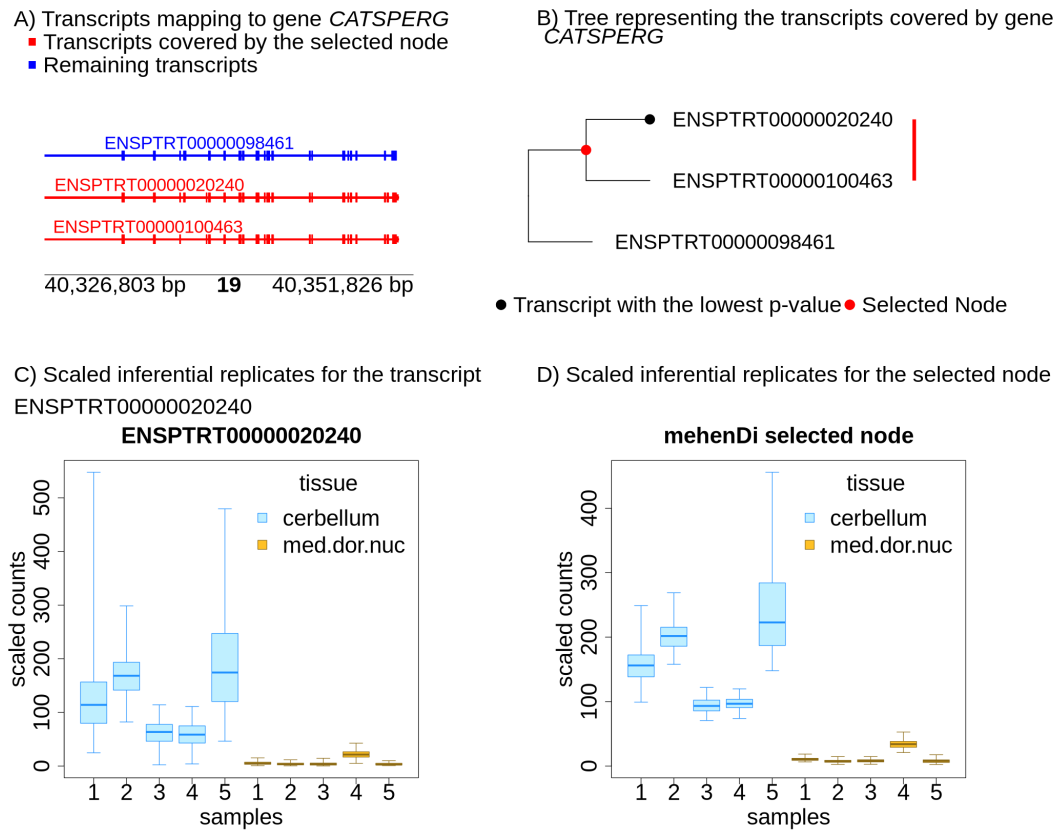


Figure B22: Examining the transcript profile for the gene *EYA1* in the ChimpBrain dataset. A) Transcripts in a pileup style. B) Tree representing the transcripts covered by the gene *EYA1*, with the red node representing the transcripts covered by the *mehenDi* selected node. C) Inferential replicates for the transcript ENSMUST00000097751, which had the lowest p-value among all the transcripts in the tree. D) Inferential replicates for the *mehenDi* selected node.

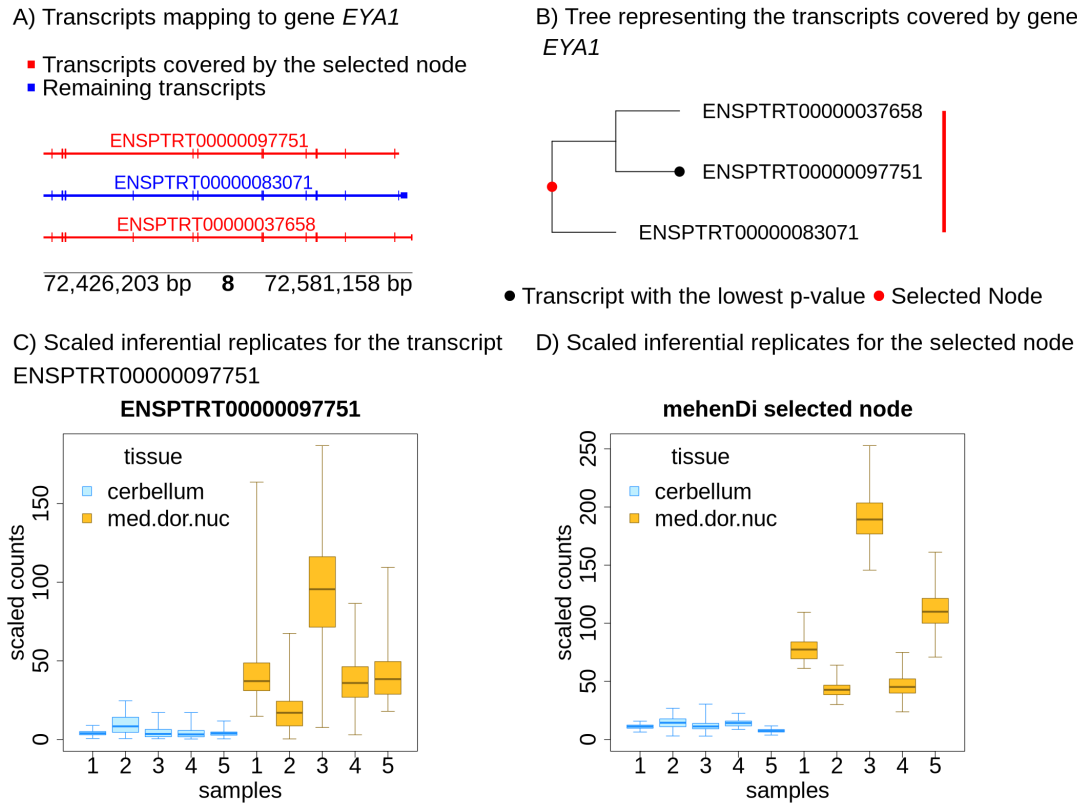


Figure B23: Examining the transcript profile for the gene *MCF2* in the ChimpBrain dataset. A) Transcripts in a pileup style. B) Tree representing the transcripts covered by the gene *MCF2*, with the red node representing the transcripts covered by the *mehenDi* selected node. C) Inferential replicates for the transcript ENSMUST00000105901, which had the lowest p-value among all the transcripts in the tree. D) Inferential replicates for the *mehenDi* selected node.

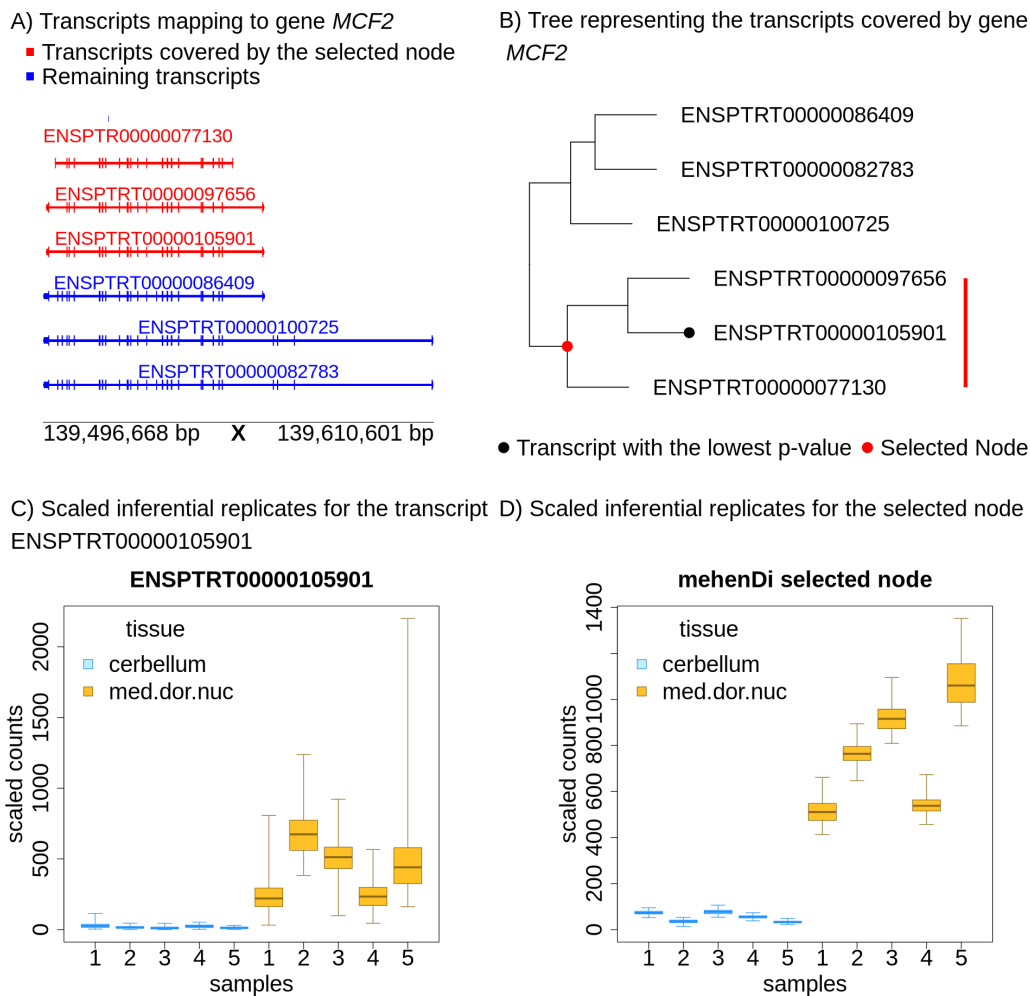


Figure B24: Examining the transcript profile for the gene *MYO5C* in the ChimpBrain dataset. A) Transcripts in a pileup style. B) Tree representing the transcripts covered by the gene *MYO5C*, with the red node representing the transcripts covered by the *mehenDi* selected node. C) Inferential replicates for the transcript ENSMUST00000013074, which had the lowest p-value among all the transcripts in the tree. D) Inferential replicates for the *mehenDi* selected node.

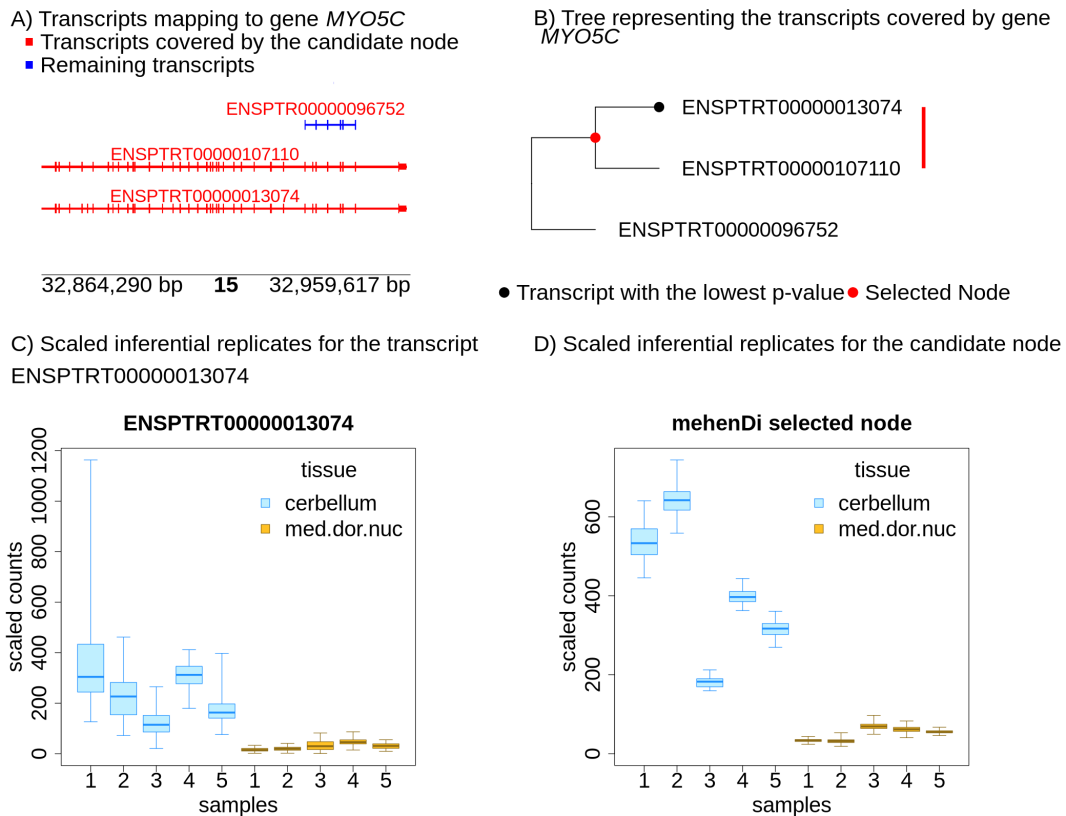
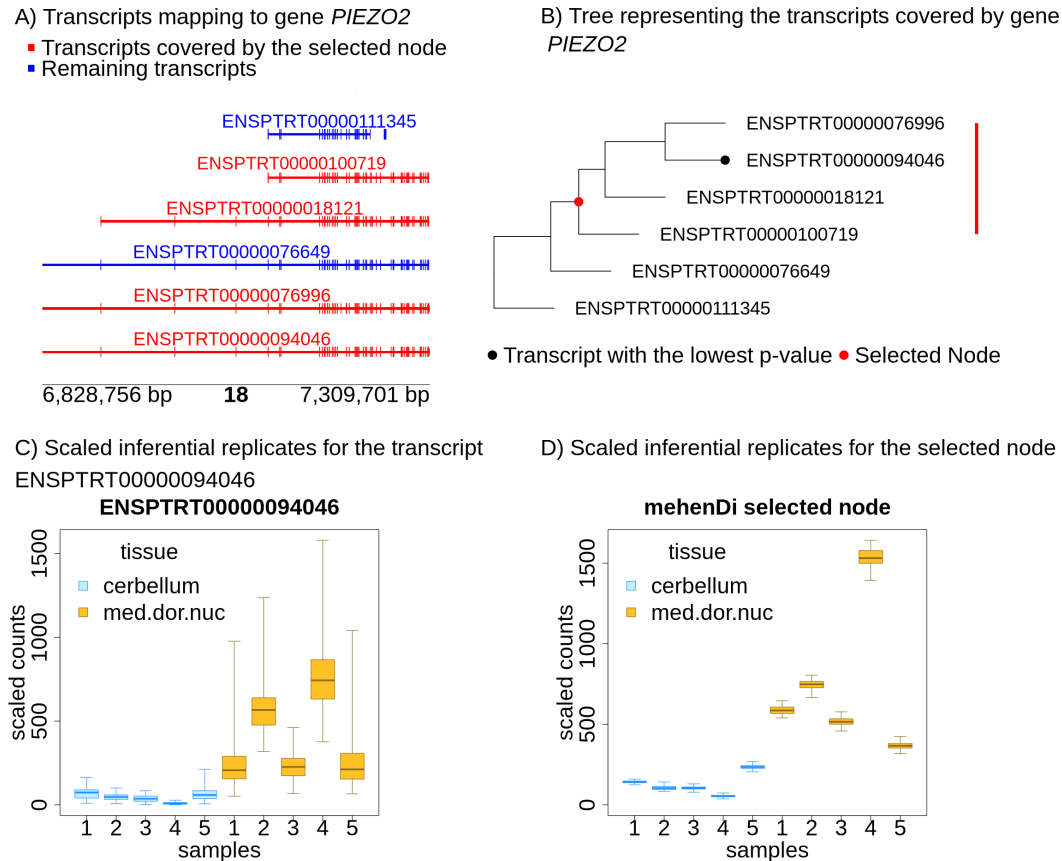


Figure B25: Examining the transcript profile for the gene *PIEZO2* in the ChimpBrain dataset. A) Transcripts in a pileup style. B) Tree representing the transcripts covered by the gene *PIEZO2*, with the red node representing the transcripts covered by the *mehenDi* selected node. C) Inferential replicates for the transcript ENSMUST00000094046, which had the lowest p-value among all the transcripts in the tree. D) Inferential replicates for the *mehenDi* selected node.



Appendix C: Supplementary Material for "Alevin-fry-atac enables rapid and memory frugal mapping of single-cell ATAC-seq data using virtual colors for accurate genomic pseudoalignment"

C.1 Methods

C.1.1 Mapping and Pseudoalignment

Merging paired-end reads before mapping — Before mapping the individual ends of a paired-end read, we try to see if we can merge the reads to create a single fragment before mapping. This idea has been suggested and adopted in several existing read-mapping tools (e.g. STAR [23] attempts to merge overlapping paired-end reads before mapping, and Chromapuses this approach for adaptor trimming [123]). We first try to merge the reads in dovetail orientation and, if that is not possible, then attempt to find an overlap merge. For a dovetail merge, the prefix of one mate should match with the suffix of the reverse complement of the other mate. For an overlap merge, the suffix of one mate should intersect with the prefix of the reverse complement of the other mate. For the merge to be successful, there should be at least an overlap of a certain number of bases between the mates (we use a default of 30). If the mates can be merged, then the merged fragment and not the individual mates are mapped to the reference.

C.1.2 Analysis pipeline

The analysis pipelines used to carry out the experiments in this manuscript were created using Snakemake [105]. Below, we describe how the individual experiments were carried out and evaluated.

Simulated data — Chromap was run without any `--preset` arguments. For Bowtie2, reads were aligned by setting the maximum fragment length (X) to 2000. These settings were taken from Chromap’s experiment repository on GitHub. Alevin-fry-atac always maps all the records irrespective of the whitelist file. The accuracy of the mappings was evaluated using a Python script that was originally used for evaluation of mapping accuracy in the stobemers [155] paper.

Experimental datasets — Chromap was run with `--preset atac` argument, also taking a whitelist file as an input. Human 3K Brain is a multiomic dataset containing single-cell RNA-seq and single-cell ATAC-seq data. In this paper, we have only utilized single-cell ATAC-seq data for it. We ran Cell Ranger ATAC only on the Human 10K PBMC and Mouse 8K Cortex datasets however, since Cell Ranger ATAC could not process the single-cell ATAC-seq data. It required running a different software Cell Ranger Arc, in order to process it, which we did not run (and could not easily compare). The downstream pipelines for analyzing ATAC-seq data such as Signac [140] require as input a tabix-indexed fragment file as produced in the Cell Ranger ATAC pipeline. Thus, a tabix index was created for the fragment files produced by both alevin-fry-atac and Chromap. We then use MACS2 [134] as a peak caller on the fragment files produced by all the methods. The peaks and the fragment files are then passed to Signac for quality control and clustering.

Benchmarking — We benchmark both genomic read mapping and the end-to-end pipeline for processing and mapping single-cell ATAC-seq data for the different methods using `/usr/bin/time` command. For `alevin-fry-atac`, this process is divided into three steps : mapping reads to the genome, correction of barcodes from a whitelist file, and deduplication with sorting. Each steps was benchmarked individually, and their execution times were summed to calculate the total runtime for `alevin-fry-atac`. Chromap was benchmarked with and without using the `--preset atac` arguments. When this argument is used, then Chromap also performs barcode correction and sorting automatically, in addition to mapping reads to genome. Without the argument, Chromap only maps reads to the genome. The methods were benchmarked across multiple thread counts, with each process run three times per thread count, and the average across the three runs was computed to obtain the final metrics (total time taken and maximum memory used). The execution time metrics for genomic read mapping process for Chromap (run using `--preset atac` argument), were obtained by subtracting the time taken for sorting, deduplication and writing from the total time taken for program execution. Chromap dumps the time for non-mapping related tasks to `stdout`. We ran Cell Ranger ATAC only once with 32 cores and 100 Gb of allowed RAM, since it was not possible to benchmark the individual steps. Cell Ranger ATAC has previously been shown to be substantially slower than Chromap.

Software versions — The following software versions were used for the analyses in this manuscript: Cell Ranger ATAC version 2.1.0, Chromap version 0.2.7 – r493, Bowtie2 version 2.5.4, mason version 2.0.9, MACS2 version 2.2.9.1, R version 4.3.2, Signac version 1.13.0, tabix version 1.16, and Snakemake version 8.18.0. Mapping for `alevin-fry-atac` was done using `piscem` version 0.11.0 and subsequent processing was done using the `alevin-fry-atac` branch of the `alevin-fry` repository.

C.2 Results

C.2.1 Mapping accuracy for `alevin-fry-atac`

We first varied the parameter ℓ_{vcol} (which controls the number of bases a virtual color will span, with most virtual colors spanning $\ell_{\text{vcol}} + \text{ov_length}$ bases), for a given k -mer size k and pseudoalignment threshold τ and observed its impact on mapping accuracy on the simulated data for the different read lengths (figs. C1 to C4). We find that accuracy decreases continuously as ℓ_{vcol} increases, irrespective of the values of the other parameters. Importantly, the sharpest decrease is observed when the chromosomes themselves are used as the colors for pseudoalignment (removing the effect of virtual colors on the procedure entirely). The accuracy drops to 84 – 86% from being 92 – 96% (depending on the read length) when using virtual colors. In general, we observe a drop in accuracy of $\sim 10\%$ or more when disabling virtual colors. Longer reads generally yield higher accuracy across most settings, except in the specific parameter configuration where $\tau = 1$ and virtual colors are disabled, where reads with length 100 have the highest accuracy.

We next varied the threshold τ while keeping ℓ_{vcol} and k fixed (figs. C5 to C8). The accuracy decreases as the threshold decreases from 0.6 to 1. The decline seems to be sharpest from 0.8 to 1, with the magnitude of decrease in accuracy in general being much higher for larger read lengths. For $\tau = 1$, at all the ℓ_{vcol} values, the accuracy of reads with length 150 is smaller than reads with length 100. However, when the chromosomes are used as colors, the accuracy for reads with length 150 increases when τ increases from 0.6 – 0.8 and falls again at 1.

Finally, we varied k , while keeping ℓ_{vcol}, τ fixed (figs. C9 to C12). We observe that for

reads of length 50, accuracy increases from $k = 23$ to $k = 25$ and starts decreasing across most virtual color extends and thresholds. However, when chromosomes are used as colors, accuracy increases with an increase in k . For the reads with larger lengths, a larger k is preferred as the accuracy increases uniformly across parameters.

C.3 Tables

Table C1: URLs for the datasets used in the manuscript for analysis.

Dataset	URL
Human 10K PBMC	10K
Mouse 8K Cortex	8k
Human 3K Brain	3K

Table C2: The total number of bases (MB) covered by the different methods across the different datasets. N/A entries denotes that Cell Ranger ATAC was not run on that dataset. Cell Ranger ATAC (MACS2) and Cell Ranger ATAC (Custom) denote that peaks were generated on the Cell Ranger ATAC fragments using MACS2 and the inbuilt peak caller of Cell Ranger ATAC respectively.

Method/ Dataset	alevin-fry-atac	Chromap	Cell Ranger ATAC (MACS2)	Cell Ra
Human 10K PBMC	98.33	97.42	98.60	
Mouse 8K Cortex	107.86	108.32	112.18	
Human 3K Brain	115.14	111.80	N/A	

C.4 Figures

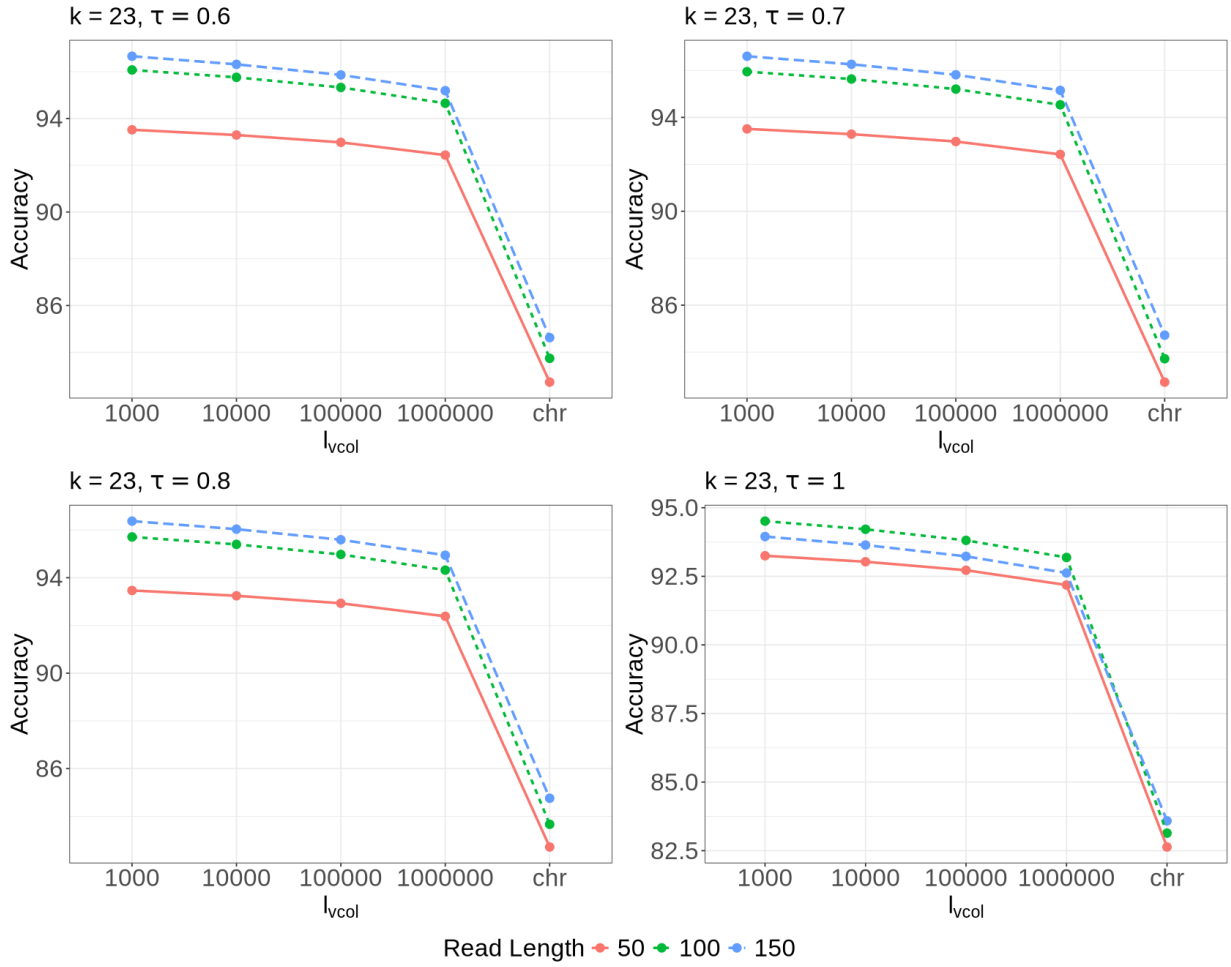


Figure C1: Evaluating accuracy across different read lengths for `alevin-fry-atac` by varying l_{vcol} at $k = 23$, with panels representing different τ values. The label *chr* on the x-axis implies mapping is done using chromosome sequence as colors (removing the impact of virtual colors on mapping procedure).

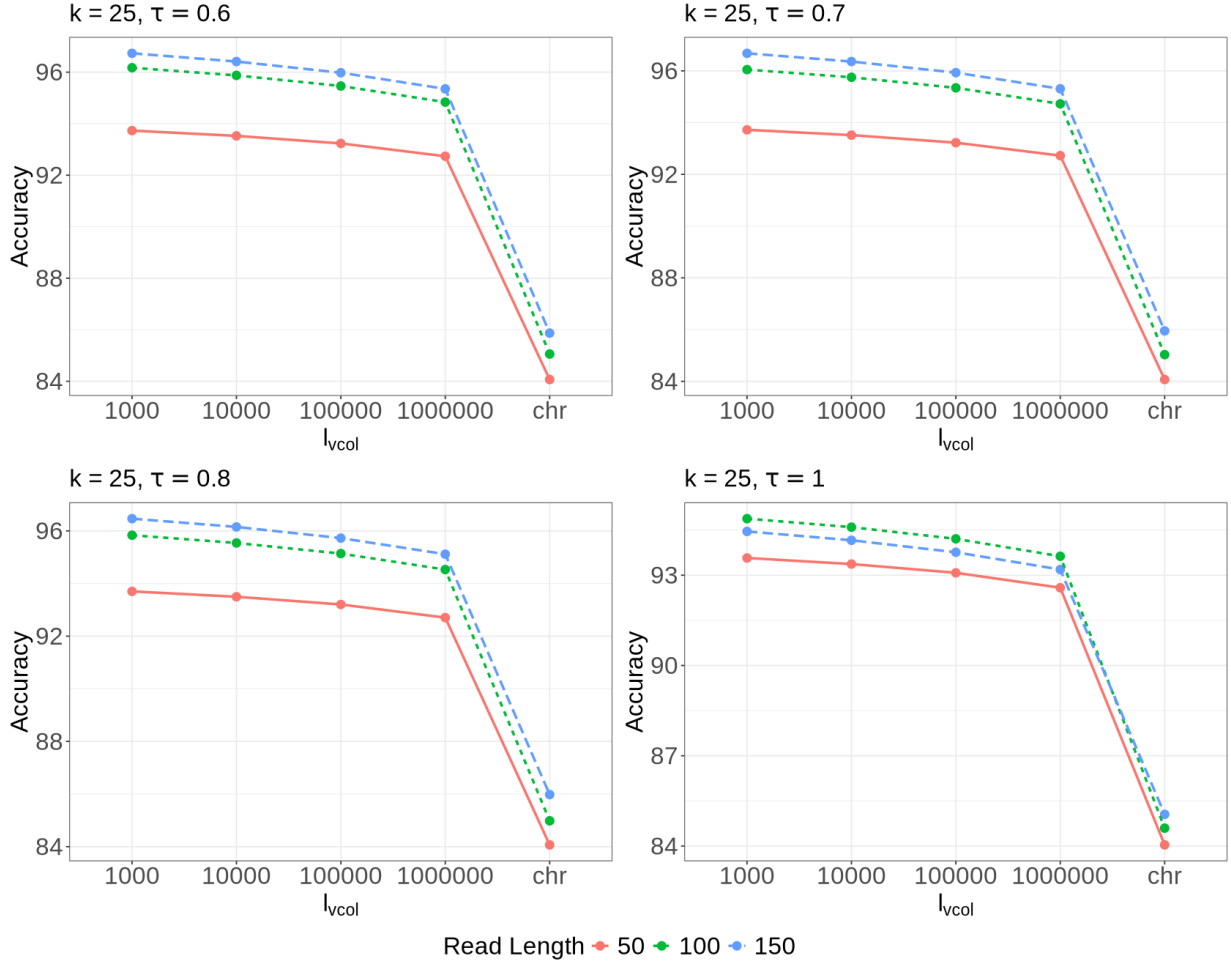


Figure C2: Evaluating accuracy across different read lengths for *alevin-fry-atac* by varying l_{vcol} at $k = 25$, with panels representing different τ values. The label *chr* on the x-axis implies mapping is done using chromosome sequence as colors (removing the impact of virtual colors on mapping procedure).

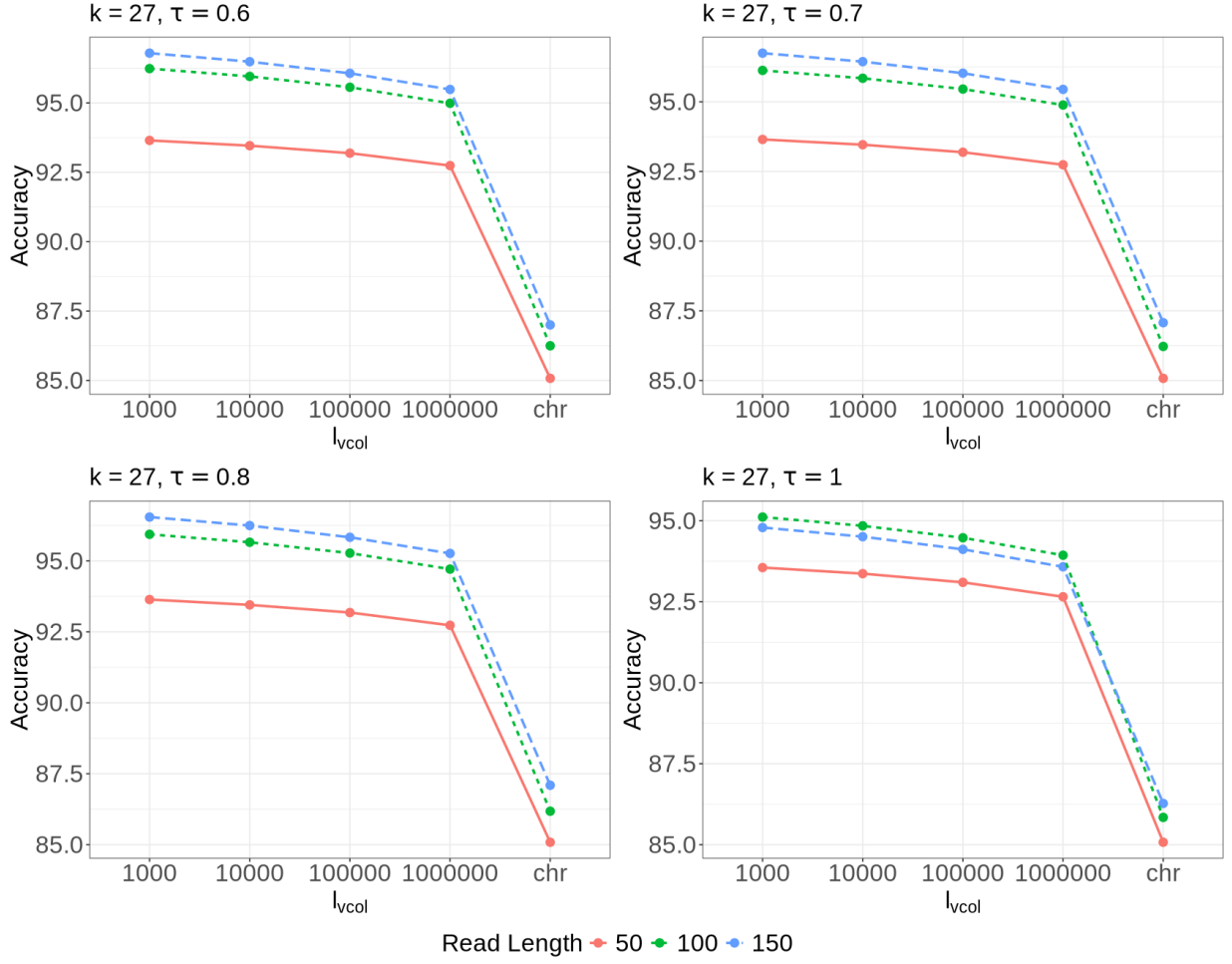


Figure C3: Evaluating accuracy across different read lengths for *alevin-fry-atac* by varying l_{vcol} for $k = 27$ with panels representing different τ values. The label *chr* on the x-axis implies mapping is done using chromosome sequence as colors (removing the impact of virtual colors on mapping procedure).

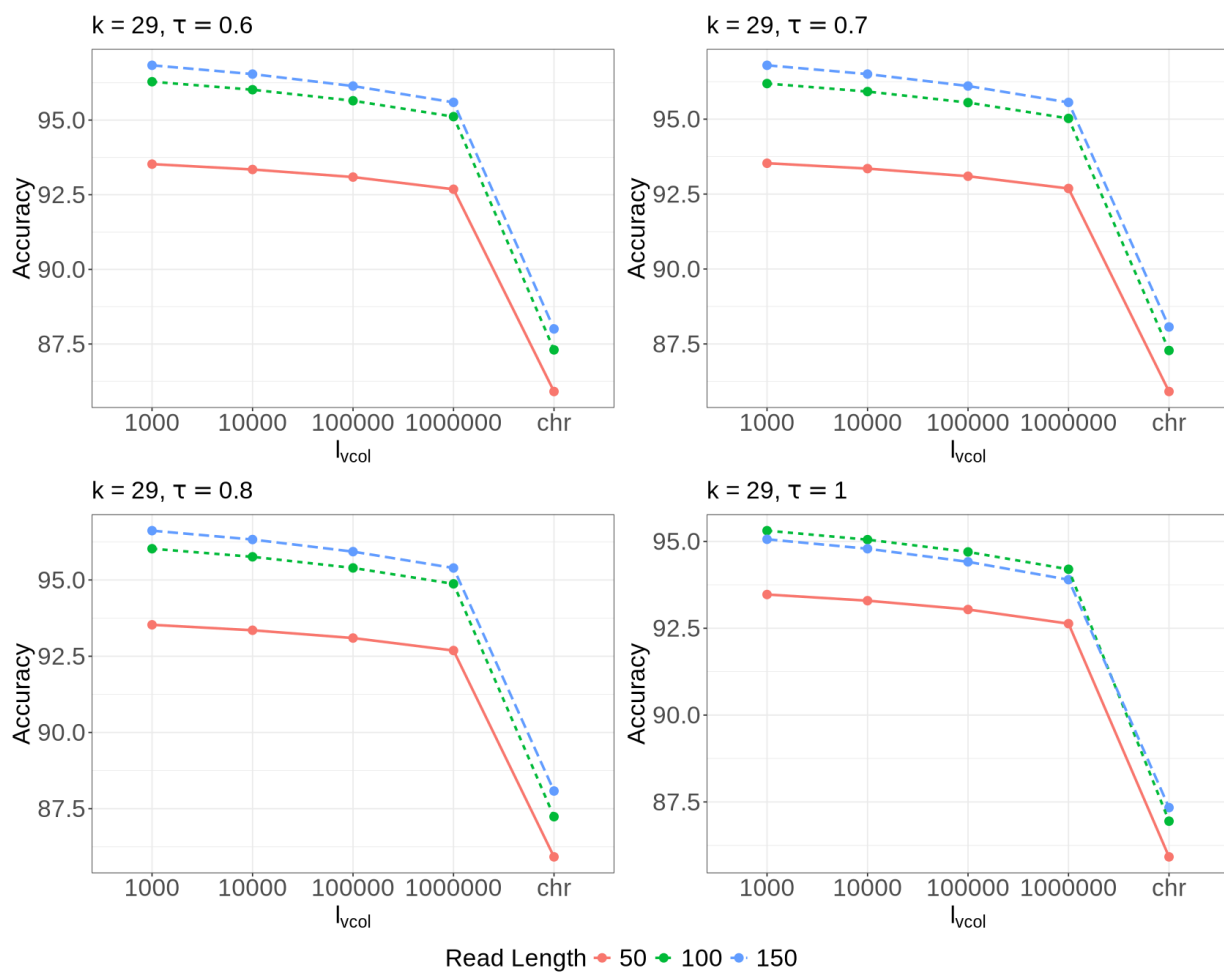


Figure C4: Evaluating accuracy across different read lengths for `alevin-fry-atac` by varying l_{vcol} for $k = 29$ with panels representing different τ values. The label *chr* on the x-axis implies mapping is done using chromosome sequence as colors (removing the impact of virtual colors on mapping procedure).

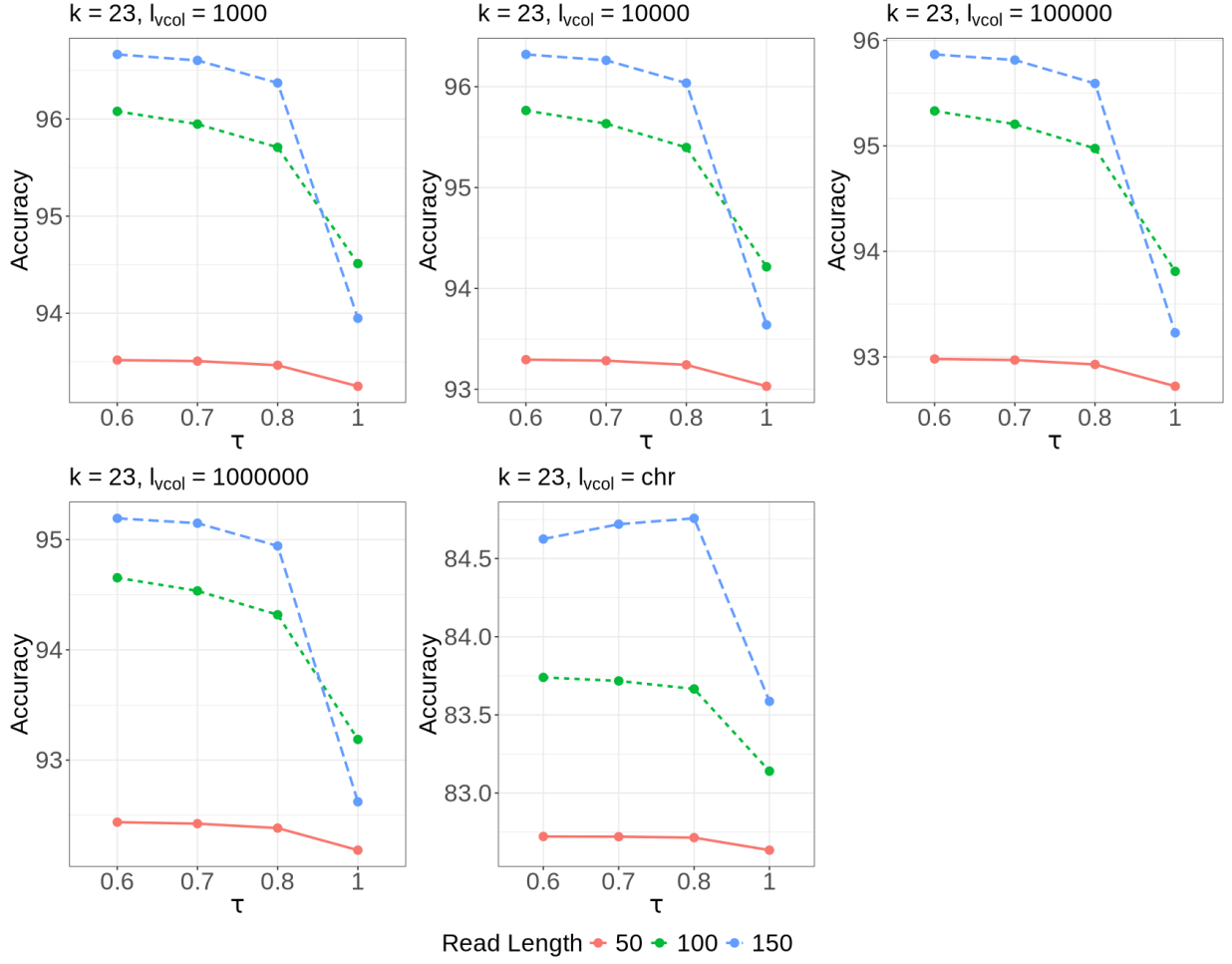


Figure C5: Evaluating accuracy across different read lengths for `alevin-fry-atac` by varying the pseudoalignment thresholds for $k = 23$ with panels representing different ℓ_{vcol} values. The panel with $\ell_{\text{vcol}} = \text{chr}$ implies mapping is done using chromosome sequence as colors (removing the impact of virtual colors on mapping procedure).

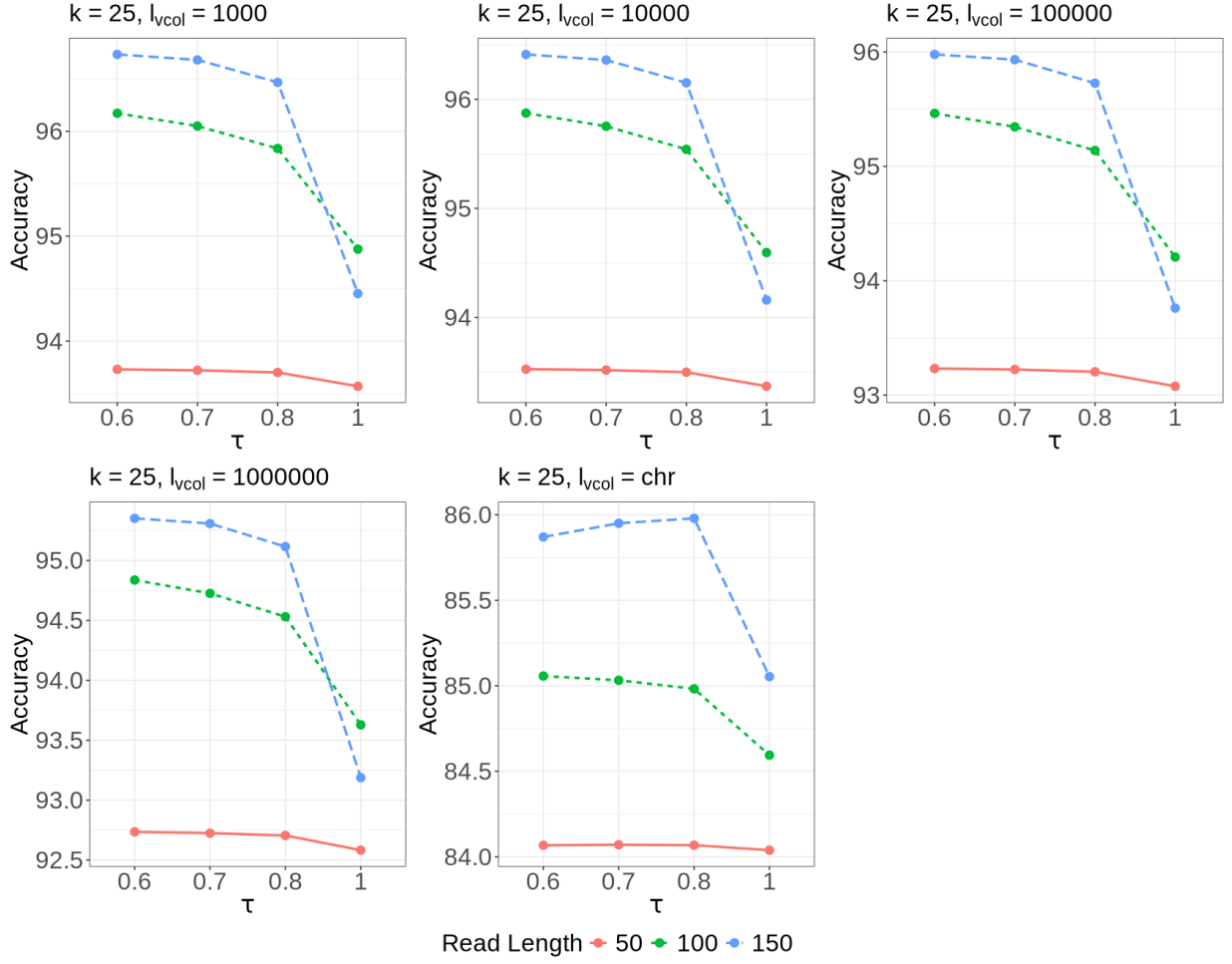


Figure C6: Evaluating accuracy across different read lengths for `alevin-fry-atac` by varying the pseudoalignment thresholds for $k = 25$ with panels representing different l_{vcol} values. The panel with $l_{\text{vcol}} = \text{chr}$ implies mapping is done using chromosome sequence as colors (removing the impact of virtual colors on mapping procedure).

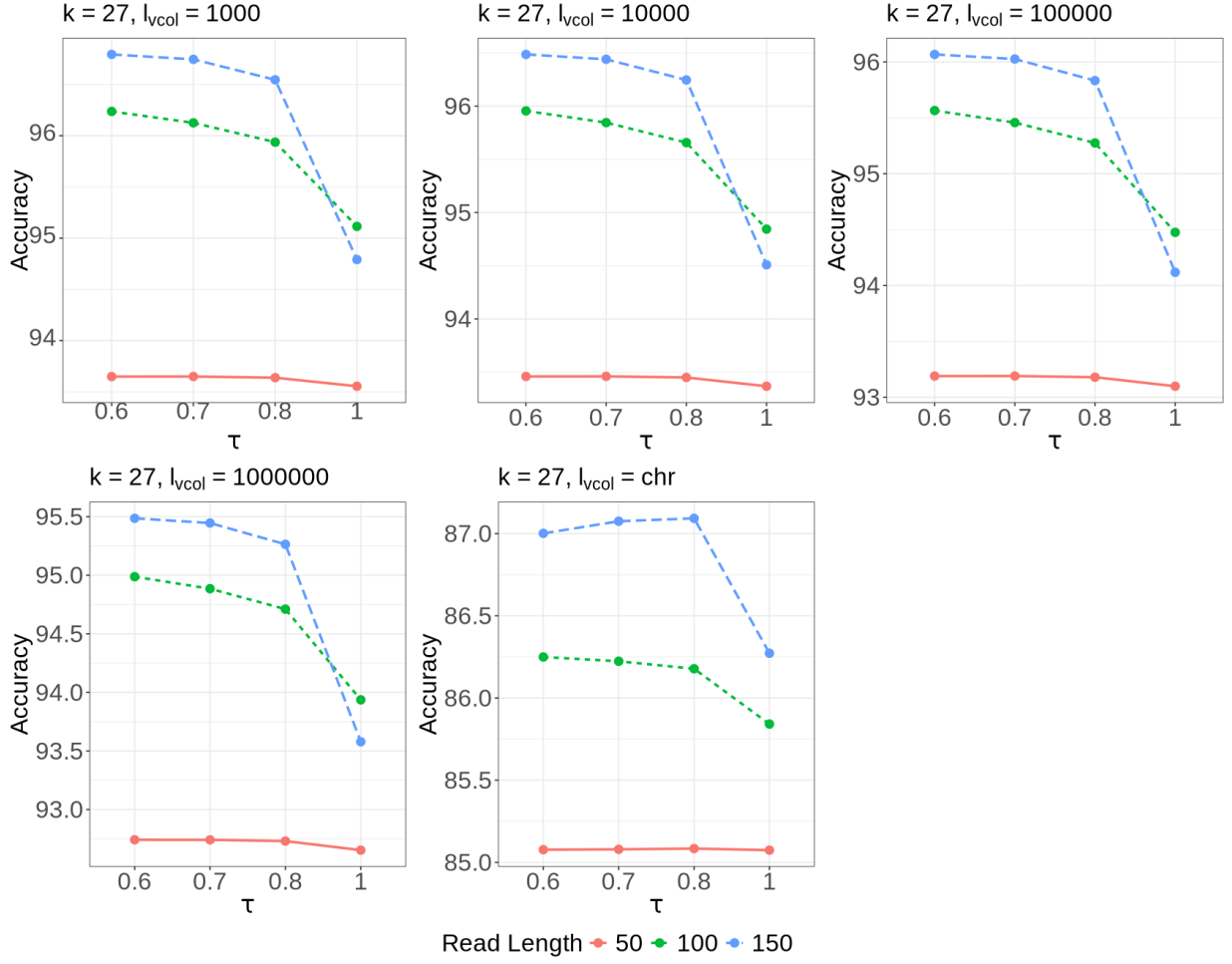


Figure C7: Evaluating accuracy across different read lengths for *alevin-fry-atac* by varying the pseudoalignment thresholds for $k = 27$ with panels representing different ℓ_{vcol} values. The panel with $\ell_{\text{vcol}} = \text{chr}$ implies mapping is done using chromosome sequence as colors (removing the impact of virtual colors on mapping procedure).

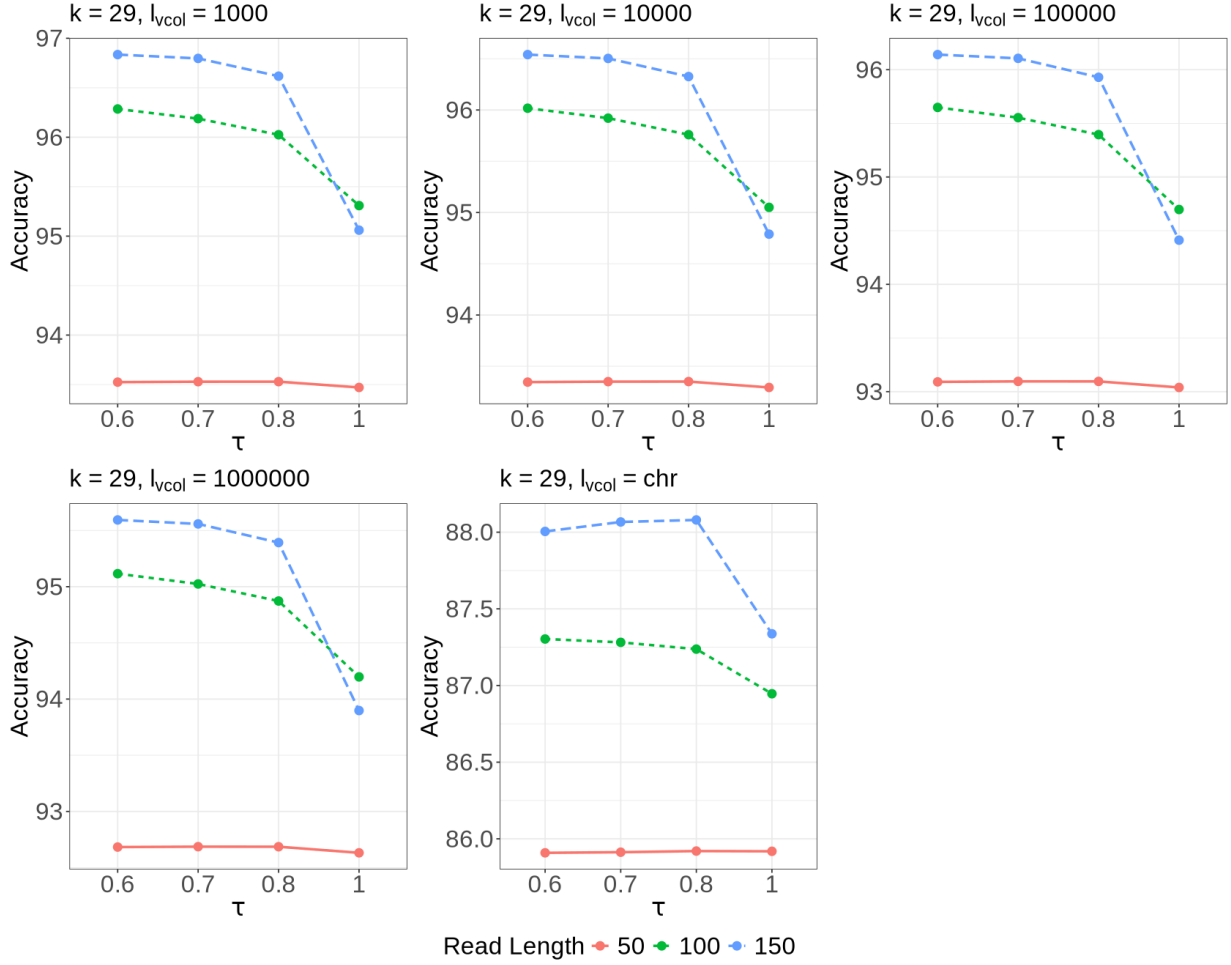


Figure C8: Evaluating accuracy across different read lengths for `alevin-fry-atac` by varying the pseudoalignment thresholds for $k = 29$ with panels representing different ℓ_{vcol} values. The panel with bin length *chr* implies mapping is done using chromosome sequence as colors (removing the impact of virtual colors on mapping procedure).

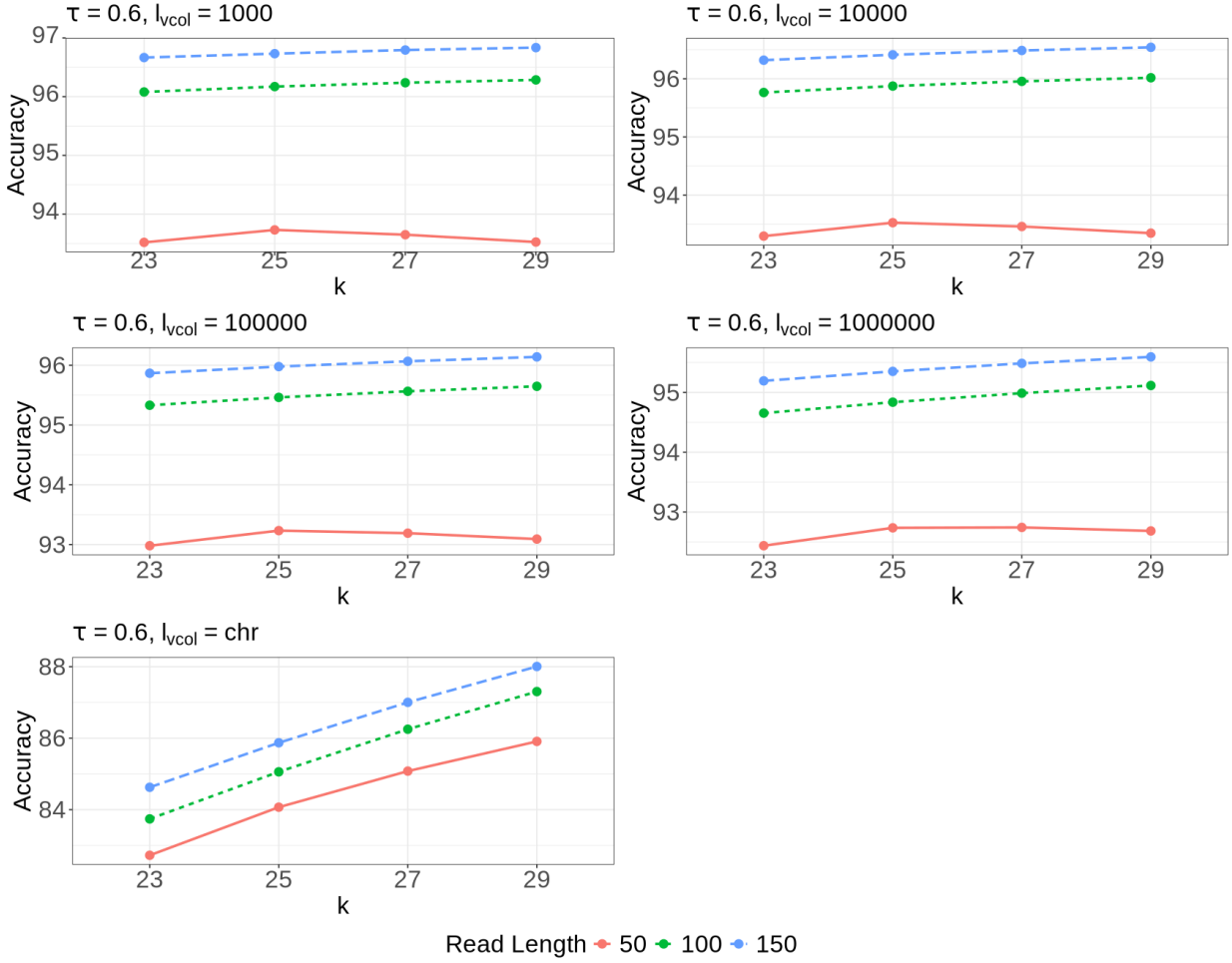


Figure C9: Evaluating accuracy across different read lengths for *alevin-fry-atac* by varying the k -mer size for $\tau = 0.6$ with panels representing different l_{vcol} values. The panel with $l_{vcol} = chr$ implies mapping is done using chromosome sequence as colors (removing the impact of virtual colors on mapping procedure).

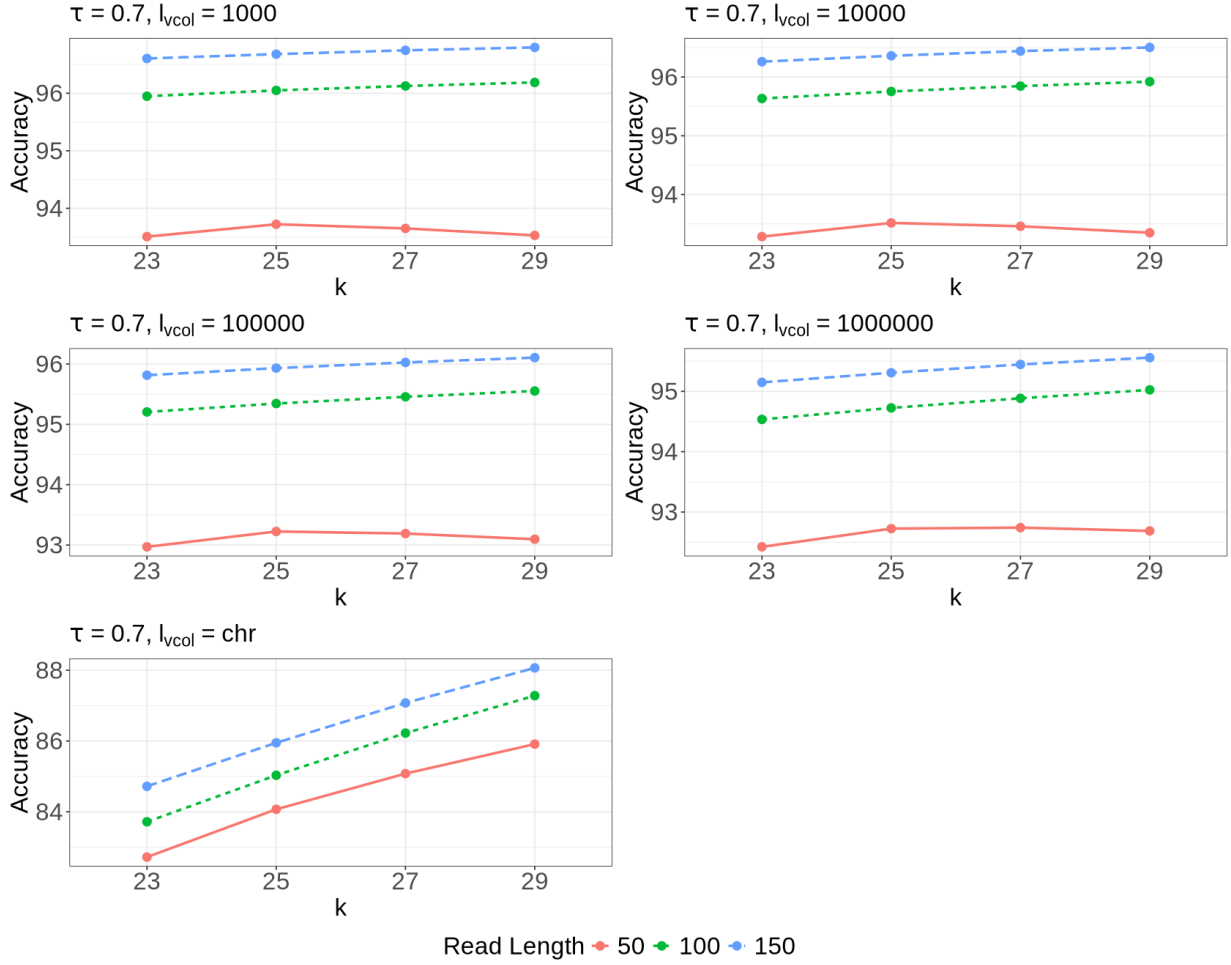


Figure C10: Evaluating accuracy across different read lengths for `alevin-fry-atac` by varying the k -mer size for $\tau = 0.7$ with panels representing different ℓ_{vcol} values. The panel with $\ell_{\text{vcol}} = \text{chr}$ implies mapping is done using chromosome sequence as colors (removing the impact of virtual colors on mapping procedure).

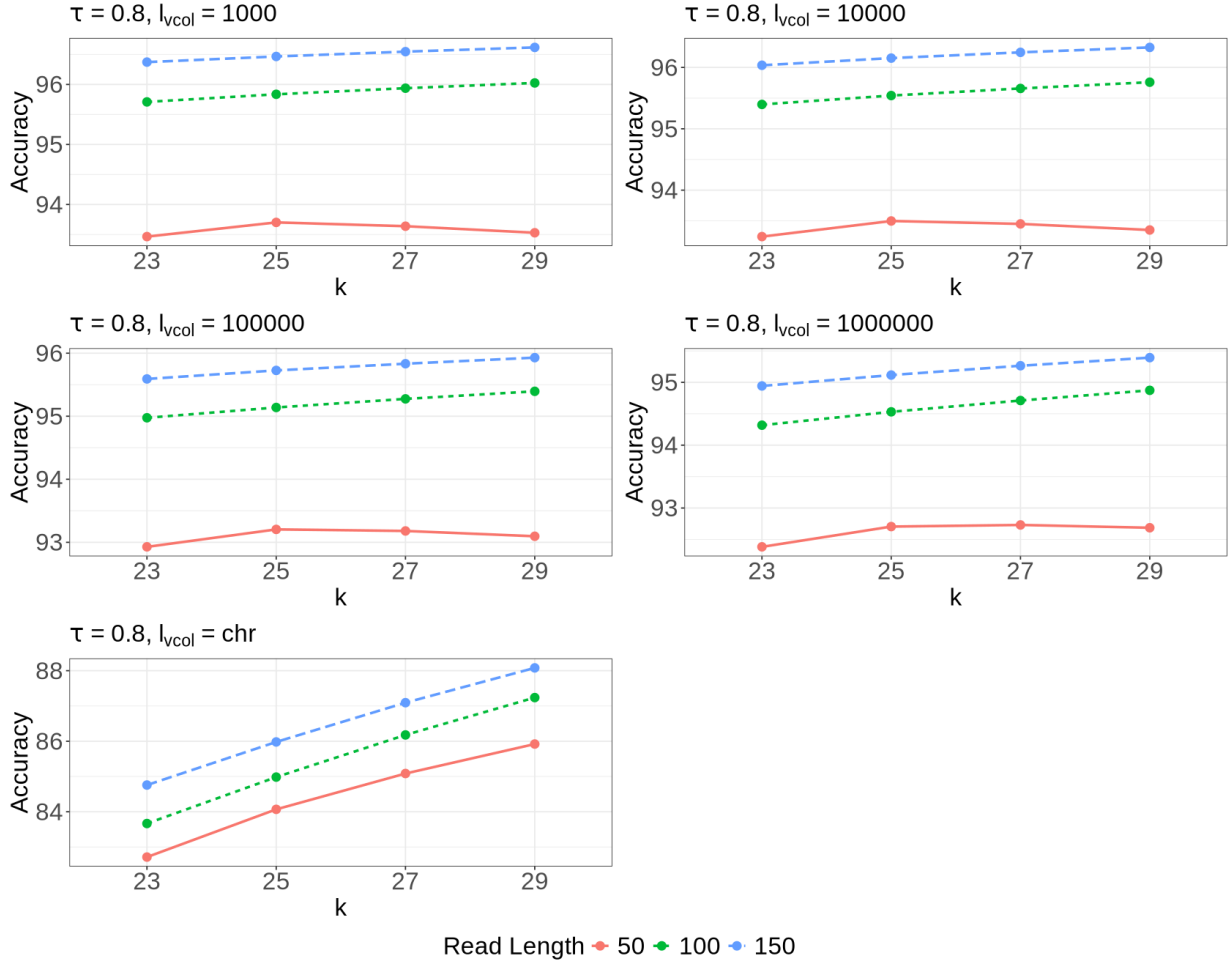


Figure C11: Evaluating accuracy across different read lengths for *alevin-fry-atac* by varying the k -mer size for $\tau = 0.8$ with panels representing different ℓ_{vcol} values. The panel with $\ell_{\text{vcol}} = \text{chr}$ implies mapping is done using chromosome sequence as colors (removing the impact of virtual colors on mapping procedure).

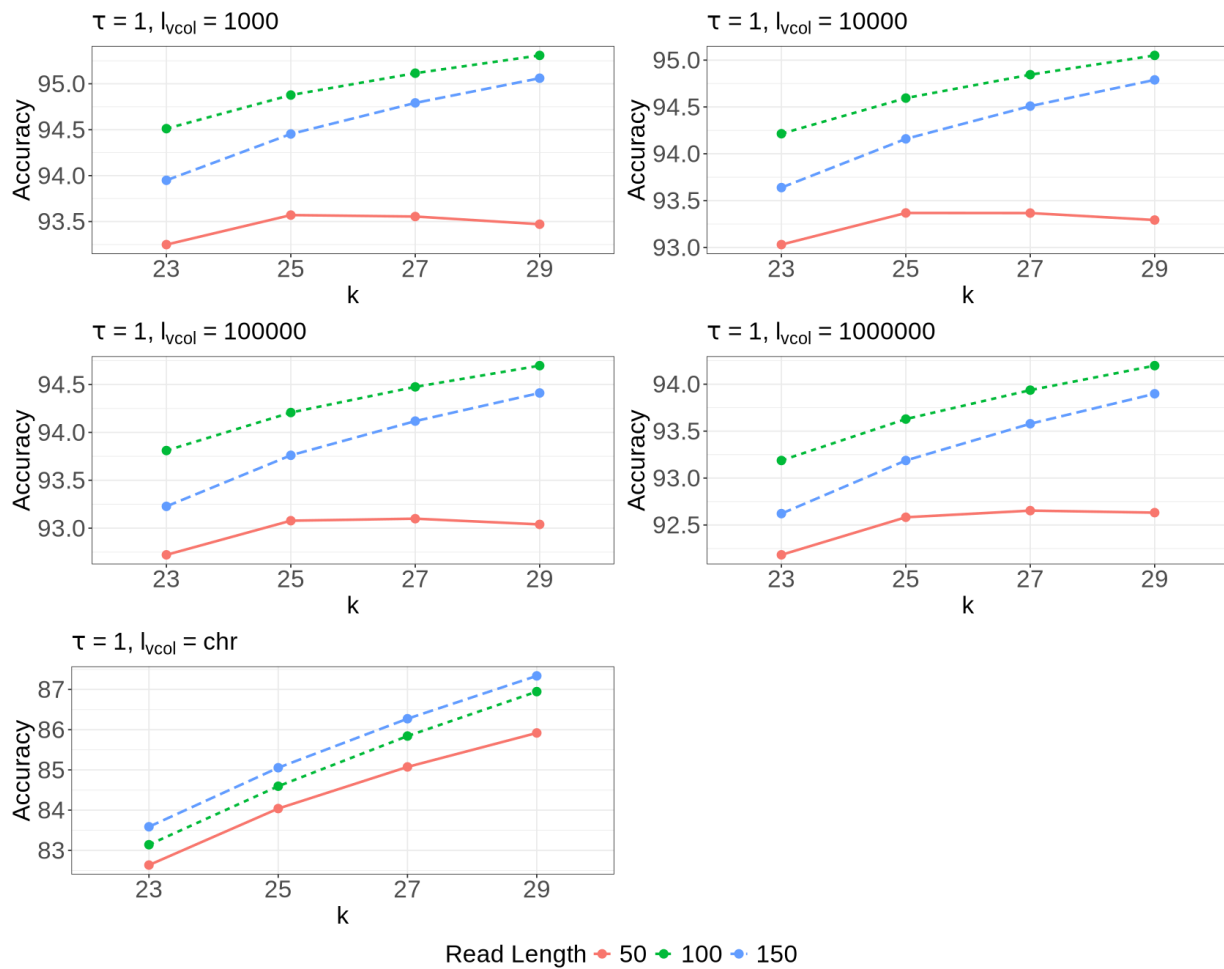


Figure C12: Evaluating accuracy across different read lengths for `alevin-fry-atac` by varying the k -mer size for $\tau = 1$ with panels representing different ℓ_{vcol} values. The panel with $\ell_{\text{vcol}} = \text{chr}$ implies mapping is done using chromosome sequence as colors (removing the impact of virtual colors on mapping procedure).

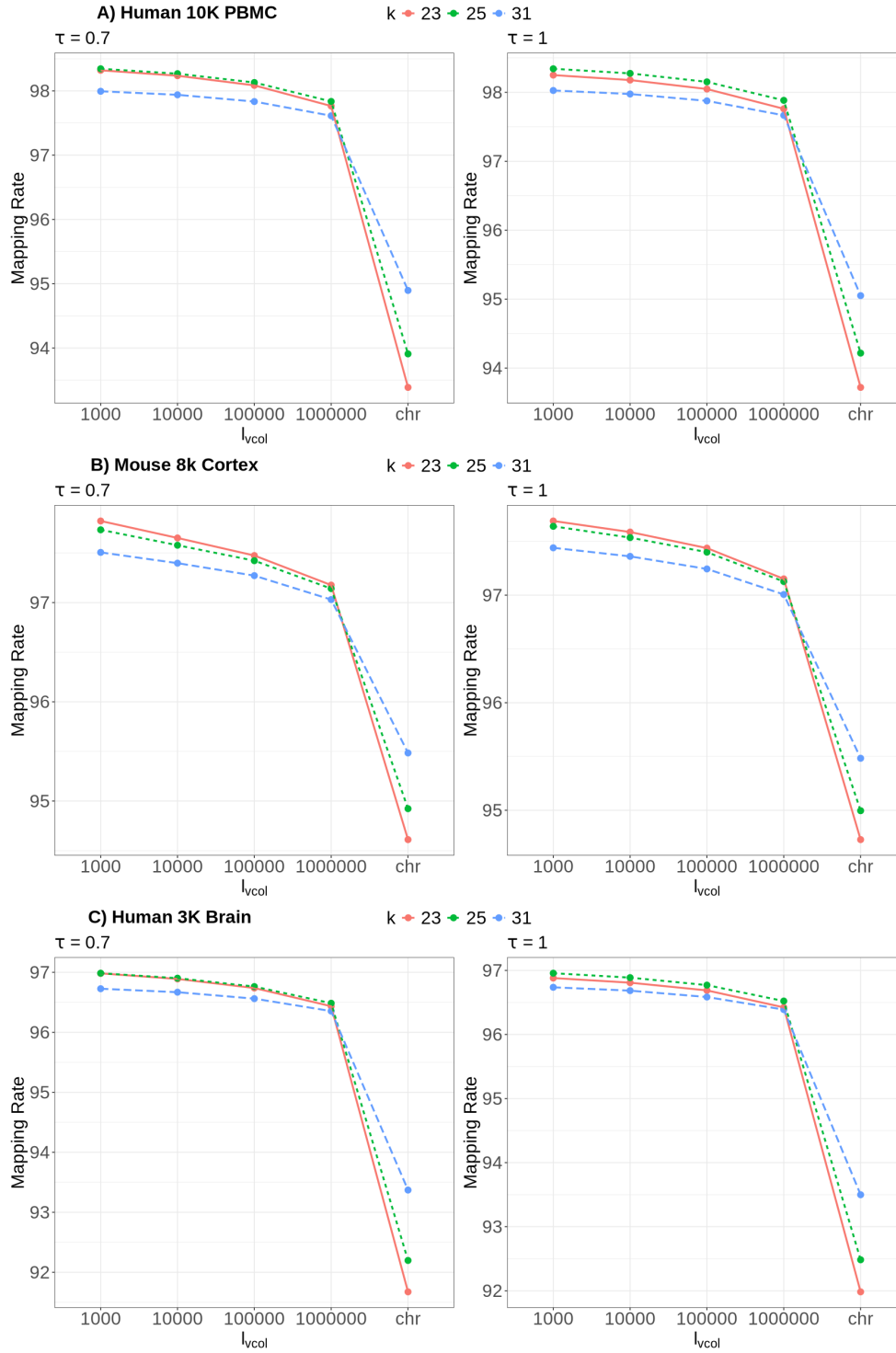


Figure C13: Evaluating the mapping rate across different k -mer size for alevin-fry-atac by varying l_{vcol} for A) $\tau = 0.7$, B) $\tau = 1$ on the Human 10K PBMC, Mouse 8K Cortex and Human 3K Brain datasets respectively. The x-axis labels *chr* implies mapping is done using chromosome sequence as colors (removing the impact of virtual colors on mapping procedure).

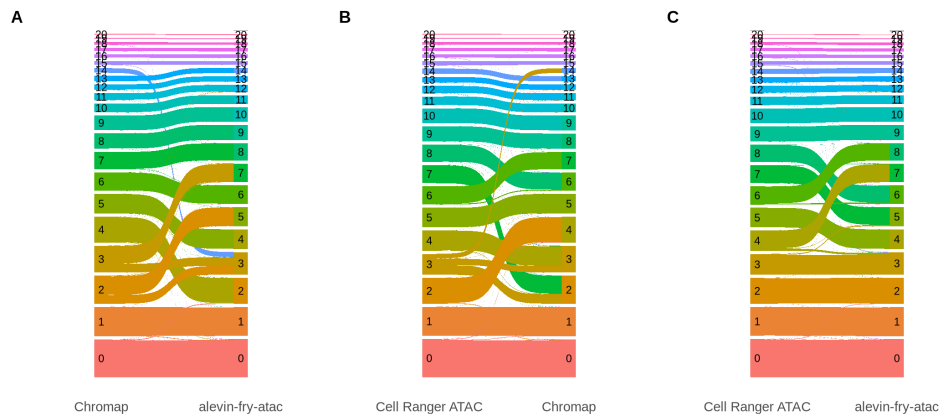


Figure C14: Pairwise Sankey plot on the clusters obtained across the different methods for Human 10K PBMC dataset.

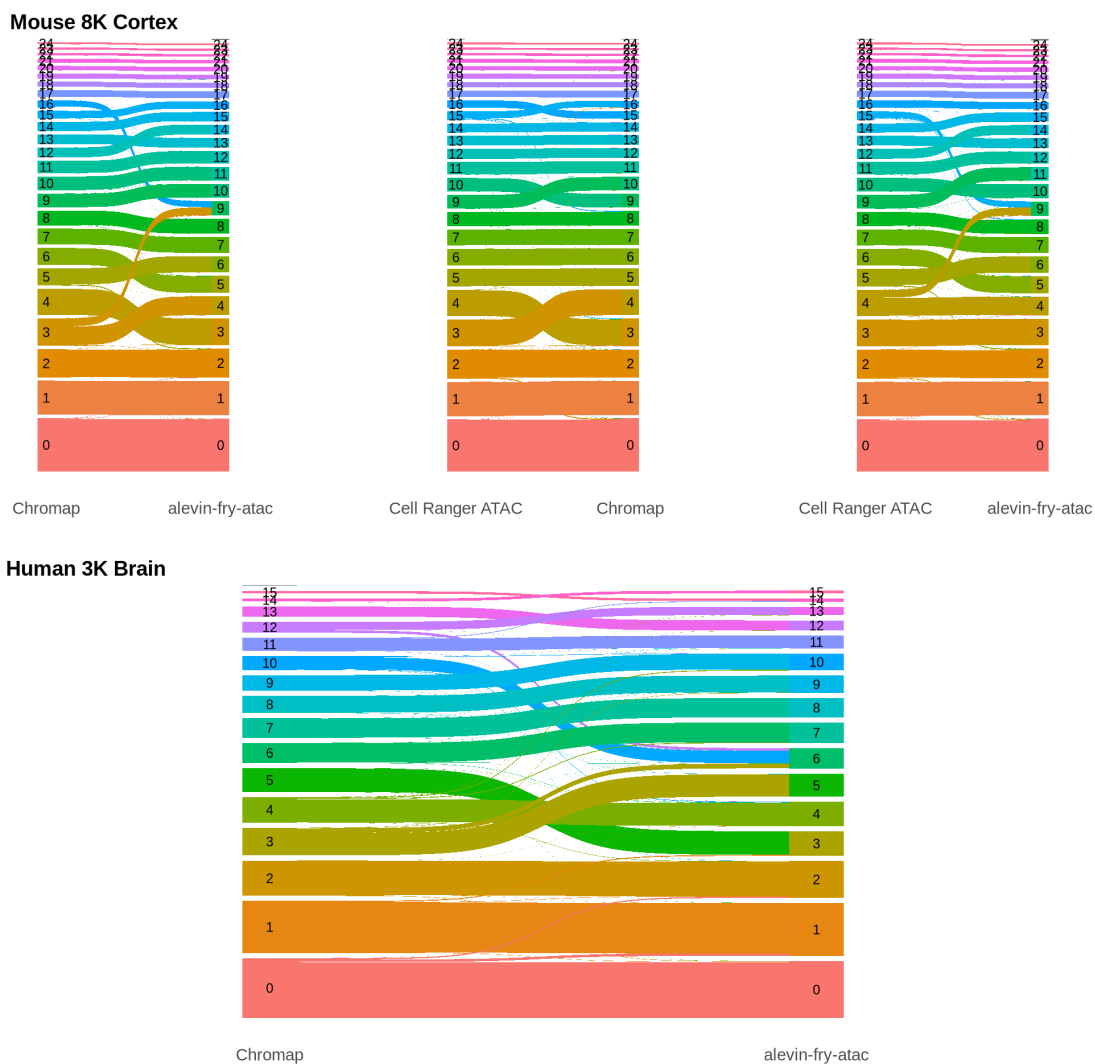


Figure C15: Pairwise Sankey plot on the clusters obtained across the different methods for Mouse 8K Cortex and Human 3K Brain dataset.

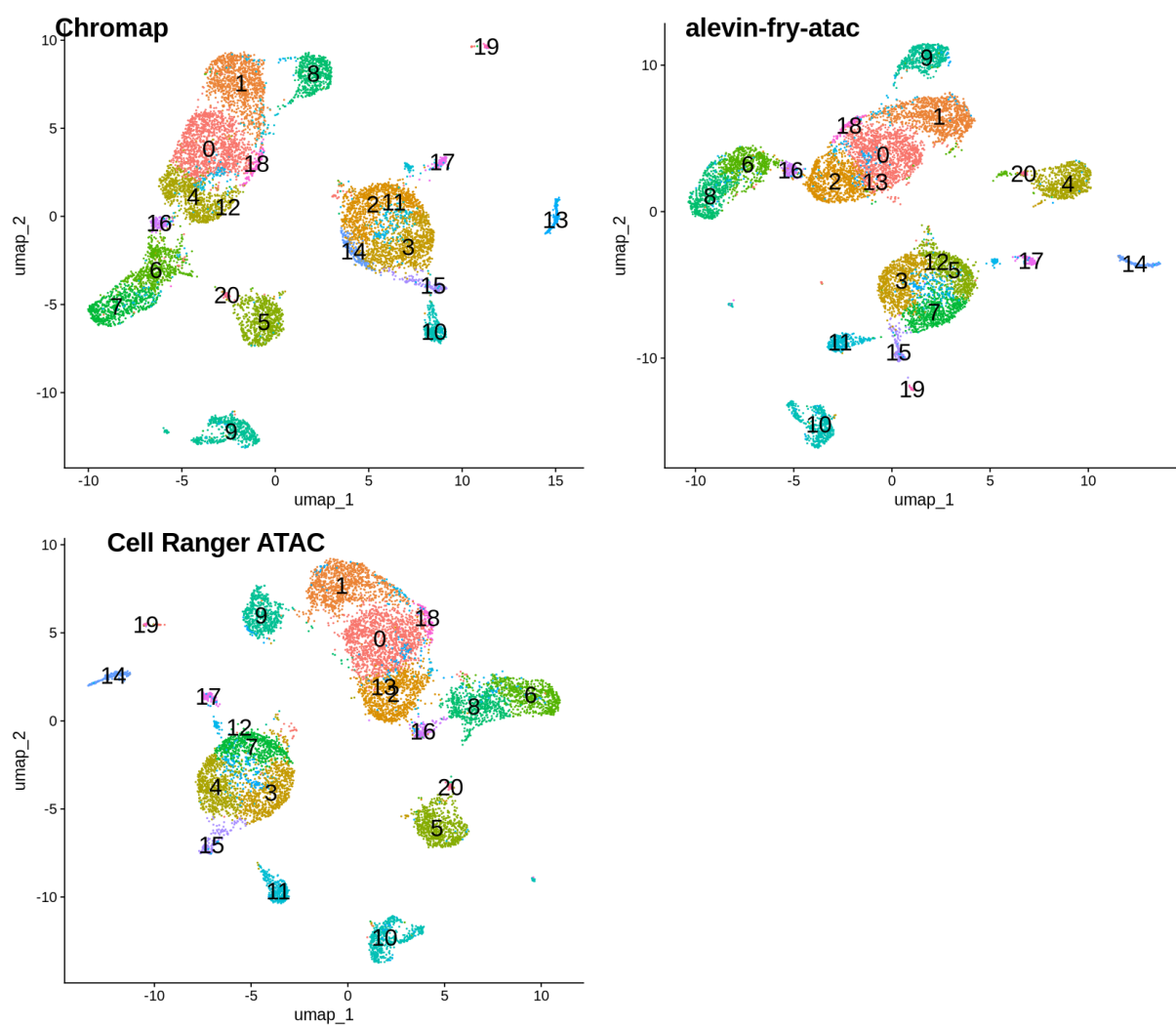


Figure C16: UMAP projections with the cluster labels obtained for the different methods on the Human 10K PBMC dataset.

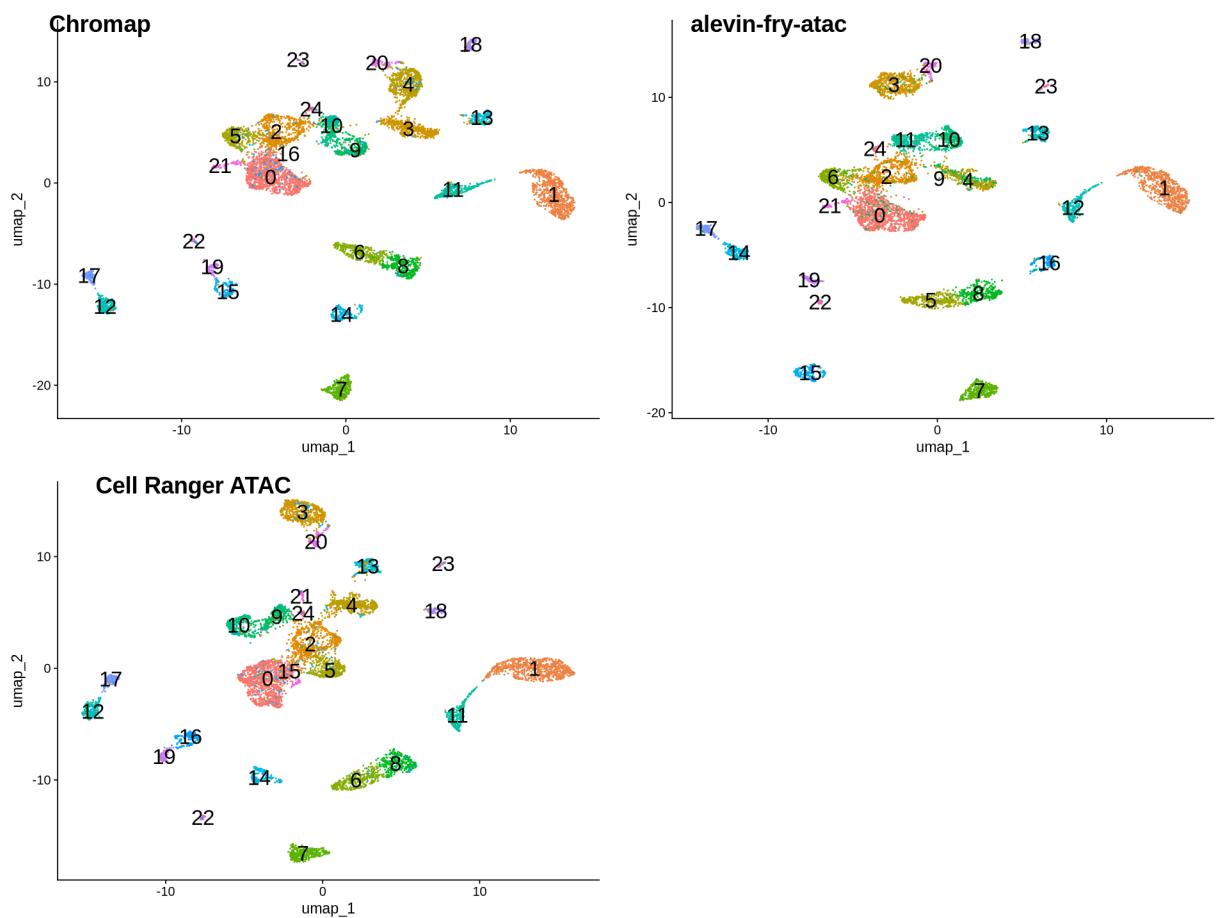


Figure C17: UMAP projections with the cluster labels obtained for the different methods on the Mouse 8K Cortex dataset.

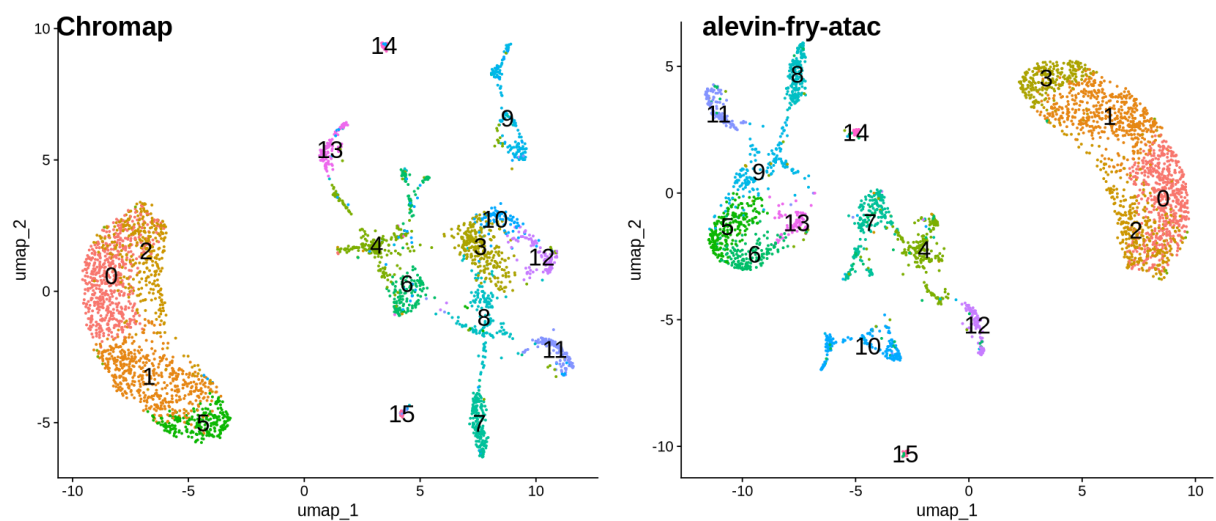


Figure C18: UMAP projections with the cluster labels obtained for the different methods on the Human 3K Brain dataset.

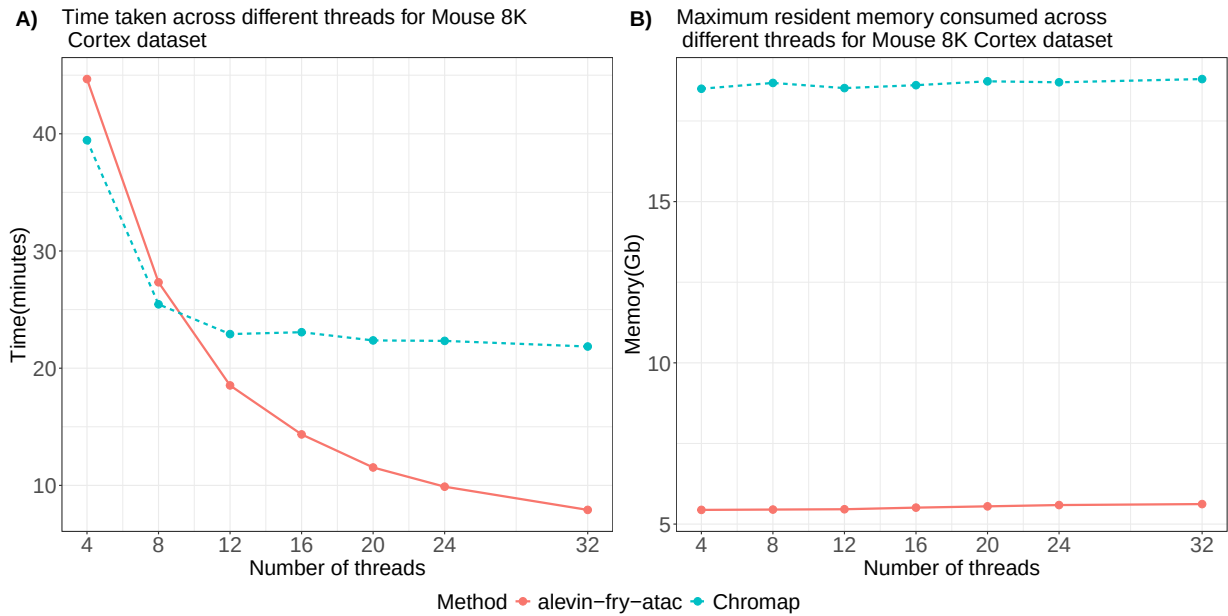


Figure C19: Comparing the time taken and maximum resident memory consumed by alevin-fry-atac and Chromap for the Mouse 8K Cortex dataset across the different threads.

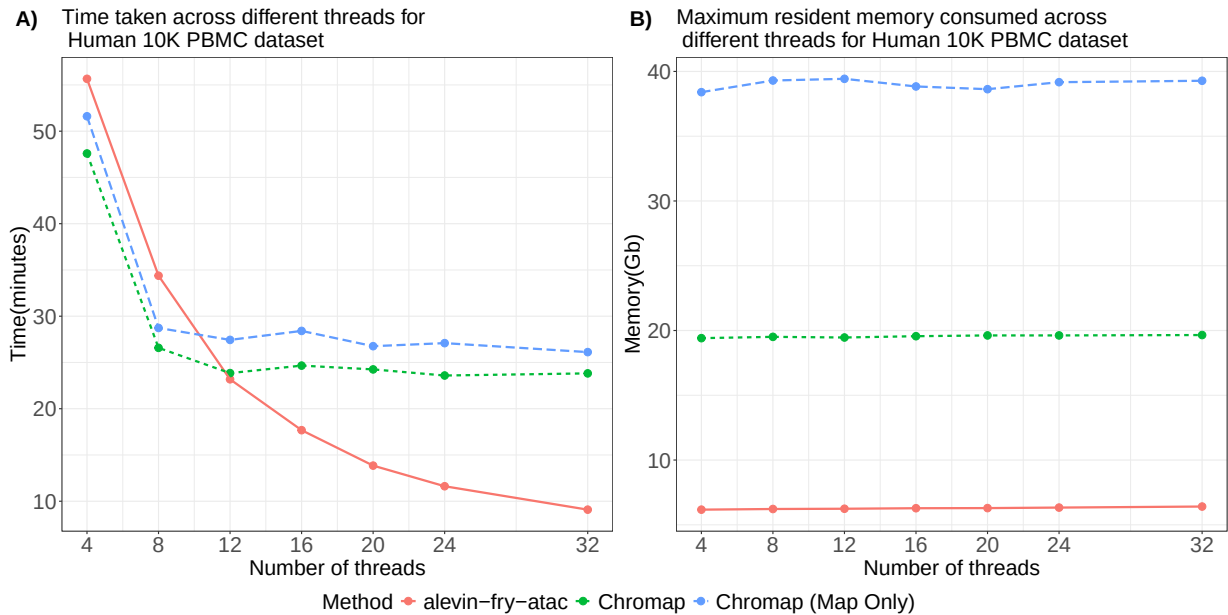


Figure C20: Comparing the time taken and maximum resident memory consumed by alevin-fry-atac and Chromap for the Human 10K PBMC dataset across the different threads to map the reads. Chromap(Map Only) refers to when Chromap is run without the --preset arguments.

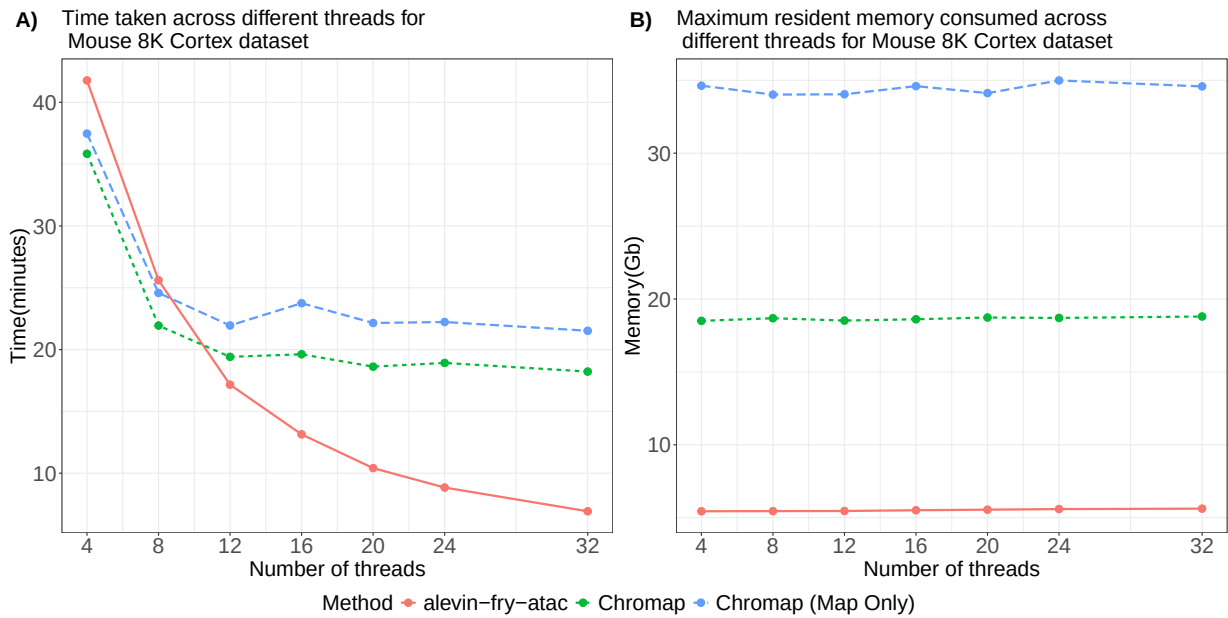


Figure C21: Comparing the time taken and maximum resident memory consumed by alevin-fry-atac and Chromap for the Mouse 8K Cortex dataset across the different threads to map the reads. Chromap(Map Only) refers to when Chromap is run without the `--preset` arguments.

Bibliography

- [1] Fiorella C Grandi, Hailey Modi, Lucas Kampman, and M Ryan Corces. Chromatin accessibility profiling by atac-seq. *Nature protocols*, 17(6):1518–1552, 2022.
- [2] George M Church. Genomes for all. *Scientific American*, 294(1):46–55, 2006.
- [3] Elaine R Mardis. Chip-seq: welcome to the new frontier. *Nature methods*, 4(8):613–614, 2007.
- [4] Jason D Buenrostro, Beijing Wu, Howard Y Chang, and William J Greenleaf. Atac-seq: a method for assaying chromatin accessibility genome-wide. *Current protocols in molecular biology*, 109(1):21–29, 2015.
- [5] Karl V Voelkerding, Shale A Dames, and Jacob D Durtschi. Next-generation sequencing: from basic research to diagnostics. *Clinical chemistry*, 55(4):641–658, 2009.
- [6] Erwin L Van Dijk, Hélène Auger, Yan Jaszczyszyn, and Claude Thermes. Ten years of next-generation sequencing technology. *Trends in genetics*, 30(9):418–426, 2014.

- [7] Sara Goodwin, John D McPherson, and W Richard McCombie. Coming of age: ten years of next-generation sequencing technologies. *Nature reviews genetics*, 17(6):333–351, 2016.
- [8] Aleksandra A Kolodziejczyk, Jong Kyoung Kim, Valentine Svensson, John C Marioni, and Sarah A Teichmann. The technology and biology of single-cell rna sequencing. *Molecular cell*, 58(4):610–620, 2015.
- [9] Jeongwoo Lee, Do Young Hyeon, and Daehee Hwang. Single-cell multiomics: technologies and data analysis methods. *Experimental & Molecular Medicine*, 52(9):1428–1442, 2020.
- [10] Elaine R Mardis. The impact of next-generation sequencing technology on genetics. *Trends in genetics*, 24(3):133–141, 2008.
- [11] Daniel C Koboldt, Karyn Meltz Steinberg, David E Larson, Richard K Wilson, and Elaine R Mardis. The next-generation sequencing revolution and its impact on genomics. *Cell*, 155(1):27–38, 2013.
- [12] Muneer Ahmad Malla, Anamika Dubey, Ashwani Kumar, Shweta Yadav, Abeer Hashem, and Elsayed Fathi Abd.Allah. Exploring the human microbiome: the potential future role of next-generation sequencing in disease diagnosis and treatment. *Frontiers in Immunology*, 9:2868, 2019.
- [13] Margaret Morash, Hannah Mitchell, Himisha Beltran, Olivier Elemento, and Jyotishman Pathak. The role of next-generation sequencing in precision medicine: a review of outcomes in oncology. *Journal of personalized medicine*, 8(3):30, 2018.

- [14] Sergey Nurk, Sergey Koren, Arang Rhie, Mikko Rautiainen, Andrey V Bzikadze, Alla Mikheenko, Mitchell R Vollger, Nicolas Altemose, Lev Uralsky, Ariel Gershman, et al. The complete sequence of a human genome. *Science*, 376(6588):44–53, 2022.
- [15] Charlotte Soneson and Mauro Delorenzi. A comparison of methods for differential expression analysis of rna-seq data. *BMC bioinformatics*, 14:1–18, 2013.
- [16] Temple F Smith, Michael S Waterman, et al. Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197, 1981.
- [17] Saul B Needleman and Christian D Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, 48(3):443–453, 1970.
- [18] Mohammed Alser, Jeremy Rotman, Dhrithi Deshpande, Kodi Taraszka, Huwenbo Shi, Pelin Icer Baykal, Harry Taegyun Yang, Victor Xue, Sergey Knyazev, Benjamin D Singer, et al. Technology dictates algorithms: recent developments in read alignment. *Genome biology*, 22(1):249, 2021.
- [19] Ben Langmead and Steven L Salzberg. Fast gapped-read alignment with bowtie 2. *Nature methods*, 9(4):357–359, 2012.
- [20] Ben Langmead, Cole Trapnell, Mihai Pop, and Steven L Salzberg. Ultrafast and memory-efficient alignment of short dna sequences to the human genome. *Genome biology*, 10:1–10, 2009.

- [21] Paolo Ferragina and Giovanni Manzini. Opportunistic data structures with applications. In *Proceedings 41st annual symposium on foundations of computer science*, pages 390–398. IEEE, 2000.
- [22] Daehwan Kim, Joseph M Paggi, Chanhee Park, Christopher Bennett, and Steven L Salzberg. Graph-based genome alignment and genotyping with hisat2 and hisat-genotype. *Nature biotechnology*, 37(8):907–915, 2019.
- [23] Alexander Dobin, Carrie A Davis, Felix Schlesinger, Jorg Drenkow, Chris Zaleski, Sonali Jha, Philippe Batut, Mark Chaisson, and Thomas R Gingeras. STAR: ultrafast universal RNA-seq aligner. *Bioinformatics*, 29(1):15–21, 2013.
- [24] Rob Patro, Stephen M Mount, and Carl Kingsford. Sailfish enables alignment-free isoform quantification from rna-seq reads using lightweight algorithms. *Nature biotechnology*, 32(5):462–464, 2014.
- [25] Nicolas L Bray, Harold Pimentel, Páll Melsted, and Lior Pachter. Near-optimal probabilistic rna-seq quantification. *Nature biotechnology*, 34(5):525–527, 2016.
- [26] Jason Fan, Jamshed Khan, Noor Pratap Singh, Giulio Ermanno Pibiri, and Rob Patro. Fulgor: a fast and compact k-mer index for large-scale matching and color queries. *Algorithms for Molecular Biology*, 19(1):3, 2024.
- [27] Avi Srivastava, Hirak Sarkar, Nitish Gupta, and Rob Patro. Rapmap: a rapid, sensitive and accurate tool for mapping rna-seq reads to transcriptomes. *Bioinformatics*, 32(12):i192–i200, 2016.

- [28] Rob Patro, Geet Duggal, Michael I Love, Rafael A Irizarry, and Carl Kingsford. Salmon provides fast and bias-aware quantification of transcript expression. *Nature methods*, 14(4):417–419, 2017.
- [29] Noor Pratap Singh, Michael I Love, and Rob Patro. Treeterminus—creating transcript trees using inferential replicate counts. *iScience*, 26(6), 2023.
- [30] Hirak Sarkar, Avi Srivastava, Héctor Corrada Bravo, Michael I Love, and Rob Patro. Terminus enables the discovery of data-driven, robust transcript groups from rna-seq data. *Bioinformatics*, 36(Supplement_1):i102–i110, 2020.
- [31] Noor Pratap Singh, Euphy Y Wu, Jason Fan, Michael I Love, and Rob Patro. Tree-based differential testing using inferential uncertainty for rna-seq. *bioRxiv*, 2023.
- [32] Noor Pratap Singh, Jamshed Khan, and Rob Patro. Alevin-fry-atac enables rapid and memory frugal mapping of single-cell atac-seq data using virtual colors for accurate genomic pseudoalignment. *bioRxiv*, pages 2024–11, 2024.
- [33] Jason Fan, Jamshed Khan, Giulio Ermanno Pibiri, and Rob Patro. Keeping k-mers in check—Building fast, small, and composable indices based on the De Bruijn graph. In *Biological Data Science Meeting, Cold Spring Harbor Laboratory*, 2022.
- [34] Dongze He, Mohsen Zakeri, Hirak Sarkar, Charlotte Soneson, Avi Srivastava, and Rob Patro. Alevin-fry unlocks rapid, accurate and memory-frugal quantification of single-cell rna-seq data. *Nature Methods*, 19(3):316–322, 2022.

- [35] Avi Srivastava, Laraib Malik, Tom Smith, Ian Sudbery, and Rob Patro. Alevin efficiently estimates accurate gene abundances from dscrna-seq data. *Genome biology*, 20:1–16, 2019.
- [36] Zhong Wang, Mark Gerstein, and Michael Snyder. Rna-seq: a revolutionary tool for transcriptomics. *Nature reviews genetics*, 10(1):57–63, 2009.
- [37] Yongjun Chu and David R Corey. Rna sequencing: platform selection, experimental design, and data interpretation. *Nucleic acid therapeutics*, 22(4):271–274, 2012.
- [38] Mohsen Zakeri, Avi Srivastava, Fatemeh Almodaresi, and Rob Patro. Improved data-driven likelihood factorizations for transcript abundance estimation. *Bioinformatics*, 33(14):i142–i151, 2017.
- [39] Zamin Iqbal, Mario Caccamo, Isaac Turner, Paul Flicek, and Gil McVean. De novo assembly and genotyping of variants using colored de bruijn graphs. *Nature genetics*, 44(2):226–232, 2012.
- [40] Pavel A Pevzner, Haixu Tang, and Michael S Waterman. An eulerian path approach to dna fragment assembly. *Proceedings of the national academy of sciences*, 98(17):9748–9753, 2001.
- [41] Antoine Limasset, Bastien Cazaux, Eric Rivals, and Pierre Peterlongo. Read mapping on de bruijn graphs. *BMC bioinformatics*, 17:1–12, 2016.
- [42] Rayan Chikhi, Antoine Limasset, and Paul Medvedev. Compacting de bruijn graphs from sequencing data quickly and in low memory. *Bioinformatics*, 32(12):i201–i208, 2016.

- [43] Derrick E Wood and Steven L Salzberg. Kraken: ultrafast metagenomic sequence classification using exact alignments. *Genome biology*, 15:1–12, 2014.
- [44] Ilya Y Zhbannikov, Samuel S Hunter, Matthew L Settles, and James A Foster. SlopMap: a software application tool for quick and flexible identification of similar sequences using exact k-mer matching. *Journal of data mining in genomics & proteomics*, 4(3), 2013.
- [45] Jarno N Alanko, Jaakko Vuohtoniemi, Tommi Mäklin, and Simon J Puglisi. Themisto: a scalable colored k-mer index for sensitive pseudoalignment against hundreds of thousands of bacterial genomes. *bioRxiv*, pages 2023–02, 2023.
- [46] Francisco E Baralle and Jimena Giudice. Alternative splicing as a regulator of development and tissue identity. *Nature reviews Molecular cell biology*, 18(7):437–451, 2017.
- [47] Chun-Hao Su and Woan-Yuh Tarn. Alternative splicing in neurogenesis and brain development. *Frontiers in molecular biosciences*, 5:12, 2018.
- [48] Pavel V Mazin, Philipp Khaitovich, Margarida Cardoso-Moreira, and Henrik Kaessmann. Alternative splicing during mammalian organ development. *Nature genetics*, 53(6):925–934, 2021.
- [49] Jamal Tazi, Nadia Bakkour, and Stefan Stamm. Alternative splicing and disease. *Biochimica et Biophysica Acta (BBA)-Molecular Basis of Disease*, 1792(1):14–26, 2009.
- [50] Guey-Shin Wang and Thomas A Cooper. Splicing in disease: disruption of the splicing code and the decoding machinery. *Nature Reviews Genetics*, 8(10):749–761, 2007.

- [51] Marina M Scotti and Maurice S Swanson. Rna mis-splicing in disease. *Nature Reviews Genetics*, 17(1):19–32, 2016.
- [52] Bo Li and Colin N Dewey. Rsem: accurate transcript quantification from rna-seq data with or without a reference genome. *BMC bioinformatics*, 12(1):1–16, 2011.
- [53] Ernest Turro, Shu-Yi Su, Ângela Gonçalves, Lachlan JM Coin, Sylvia Richardson, and Alex Lewin. Haplotype and isoform specific expression estimation using multi-mapping rna-seq reads. *Genome biology*, 12(2):1–15, 2011.
- [54] Peter Glaus, Antti Honkela, and Magnus Rattray. Identifying differentially expressed transcripts from rna-seq data with biological variation. *Bioinformatics*, 28(13):1721–1728, 2012.
- [55] Christelle Robert and Mick Watson. Errors in rna-seq quantification affect genes of relevance to human disease. *Genome biology*, 16(1):1–16, 2015.
- [56] Ernest Turro, William J Astle, and Simon Tavaré. Flexible analysis of rna-seq data using mixed effects models. *Bioinformatics*, 30(2):180–188, 2014.
- [57] Anqi Zhu, Avi Srivastava, Joseph G Ibrahim, Rob Patro, and Michael I Love. Nonparametric expression analysis using inferential replicate counts. *Nucleic Acids Research*, 47(18):e105–e105, 2019.
- [58] Edward N Adams III. Consensus techniques and the comparison of taxonomic trees. *Systematic Biology*, 21(4):390–397, 1972.

- [59] Jesper Jansson, Chuanqi Shen, and Wing-Kin Sung. Improved algorithms for constructing consensus trees. *Journal of the ACM (JACM)*, 63(3):1–24, 2016.
- [60] David Bryant. A classification of consensus methods for phylogenetics. *DIMACS series in discrete mathematics and theoretical computer science*, 61:163–184, 2003.
- [61] Timothy Margush and Fred R McMorris. Consensusn-trees. *Bulletin of Mathematical Biology*, 43(2):239–244, 1981.
- [62] Kåre Bremer. Combinable component consensus. *Cladistics*, 6(4):369–372, 1990.
- [63] J Felsenstein. Phylip version 3.6. *Software package, Department of Genome Sciences, University of Washington, Seattle, USA*, 2005.
- [64] Joseph Felsenstein. Phylip (phylogeny inference package) version 3.6. distributed by the author. <http://www.evolution.gs.washington.edu/phylip.html>, 2004.
- [65] Joseph Felsenstein. *PHYLP (phylogeny inference package), version 3.5 c*. Joseph Felsenstein., 1993.
- [66] Alyssa C Frazee, Andrew E Jaffe, Ben Langmead, and Jeffrey T Leek. Polyester: simulating rna-seq datasets with differential transcript expression. *Bioinformatics*, 31(17):2778–2784, 2015.
- [67] Tuuli Lappalainen, Michael Sammeth, Marc R Friedländer, Peter Ac’t Hoen, Jean Monlong, Manuel A Rivas, Mar Gonzalez-Porta, Natalja Kurbatova, Thasso Griebel, Pedro G Ferreira, et al. Transcriptome and genome sequencing uncovers functional variation in humans. *Nature*, 501(7468):506–511, 2013.

- [68] Michael I Love, Charlotte Soneson, and Rob Patro. Swimming downstream: statistical analysis of differential transcript usage following salmon quantification. *F1000Research*, 7, 2018.
- [69] Erin E Terry, Xiping Zhang, Christy Hoffmann, Laura D Hughes, Scott A Lewis, Jiajia Li, Matthew J Wallace, Lance A Riley, Collin M Douglas, Miguel A Gutierrez-Monreal, et al. Transcriptional profiling reveals extraordinary diversity among skeletal muscle tissues. *Elife*, 7:e34613, 2018.
- [70] André MM Sousa, Ying Zhu, Mary Ann Raghanti, Robert R Kitchen, Marco Onorati, Andrew TN Tebbenkamp, Bernardo Stutz, Kyle A Meyer, Mingfeng Li, Yuka Imamura Kawasawa, et al. Molecular and cellular reorganization of neural circuits in the human lineage. *Science*, 358(6366):1027–1032, 2017.
- [71] Jeffrey T Leek. Svaseq: removing batch effects and other unwanted noise from sequencing data. *Nucleic acids research*, 42(21):e161–e161, 2014.
- [72] R. R. Sokal and C. D. Michener. A statistical method for evaluating systematic relationships. *University of Kansas Science Bulletin*, 38:1409–1438, 1958.
- [73] K.P. Schliep. phangorn: phylogenetic analysis in r. *Bioinformatics*, 27(4):592–593, 2011.
- [74] Aaron R Quinlan and Ira M Hall. Bedtools: a flexible suite of utilities for comparing genomic features. *Bioinformatics*, 26(6):841–842, 2010.
- [75] Simon Andrews, Felix Krueger, Anne Segonds-Pichon, Laura Biggins, Christel Krueger, and Steven Wingett. FastQC. Babraham Institute, January 2012.

- [76] Philip Ewels, Måns Magnusson, Sverker Lundin, and Max Käller. Multiqc: summarize analysis results for multiple tools and samples in a single report. *Bioinformatics*, 32(19):3047–3048, 2016.
- [77] Johannes Köster and Sven Rahmann. Snakemake—a scalable bioinformatics workflow engine. *Bioinformatics*, 28(19):2520–2522, 2012.
- [78] Damian Smedley, Syed Haider, Steffen Durinck, Luca Pandini, Paolo Provero, James Allen, Olivier Arnaiz, Mohammad Hamza Awedh, Richard Baldock, Giulia Barbiera, et al. The biomart community portal: an innovative alternative to large, centralized data repositories. *Nucleic acids research*, 43(W1):W589–W598, 2015.
- [79] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.
- [80] Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. PySAT: A Python toolkit for prototyping with SAT oracles. In *SAT*, pages 428–437, 2018.
- [81] Mazharul Islam, Kowshika Sarker, Trisha Das, Rezwana Reaz, and Md Shamsuzzoha Bayzid. Stelar: A statistically consistent coalescent-based species tree estimation method by maximizing triplet consistency. *BMC genomics*, 21(1):1–13, 2020.
- [82] Pranjal Vachaspati and Tandy Warnow. Fastrfs: fast and accurate robinson-foulds supertrees using constrained exact optimization. *Bioinformatics*, 33(5):631–639, 2017.
- [83] Ruizhu Huang, Charlotte Soneson, Pierre-Luc Germain, Thomas SB Schmidt, Christian Von Mering, and Mark D Robinson. treeclimbr pinpoints the data-dependent resolution of hierarchical hypotheses. *Genome biology*, 22(1):1–21, 2021.

- [84] Yunxiao Li, Yi-Juan Hu, and Glen A Satten. A bottom-up approach to testing hypotheses that have a branching tree dependence structure, with error rate control. *Journal of the American Statistical Association*, 117(538):664–677, 2022.
- [85] Antoine Bichat, Christophe Ambroise, and Mahendra Mariadassou. Hierarchical correction of p-values via an ultrametric tree running ornstein-uhlenbeck process. *Computational Statistics*, 37(3):995–1013, 2022.
- [86] Nicolai Meinshausen. Hierarchical testing of variable importance. *Biometrika*, 95(2):265–278, 2008.
- [87] Jacob Bien, Xiaohan Yan, Léo Simpson, and Christian L Müller. Tree-aggregated predictive modeling of microbiome data. *Scientific Reports*, 11(1):1–13, 2021.
- [88] David S Clausen and Amy D Willis. Evaluating replicability in microbiome data. *Biostatistics*, 2021.
- [89] Charlotte Sonesson, Avi Srivastava, Rob Patro, and Michael B Stadler. Preprocessing choices affect rna velocity results for droplet scrna-seq data. *PLoS computational biology*, 17(1):e1008585, 2021.
- [90] S. Al Seesi, Y.T. Tiagueu, A. Zelikovsky, and I.I. Mandoiu. Bootstrap-based differential gene expression analysis for rna-seq data with and without replicates. *BMC Genomics*, 15(Suppl 8):S2, 2014.
- [91] I. Mandric, Y.T. Tiagueu, T. Shcheglova, S. Al Seesi, A. Zelikovsky, and I.I. Mandoiu. Fast bootstrapping-based estimation of confidence intervals of expression levels and differential expression from rna-seq data. *Bioinformatics*, 33(20):3302–3304, Oct. 2017.

- [92] Harold Pimentel, Nicolas L Bray, Suzette Puente, Páll Melsted, and Lior Pachter. Differential analysis of rna-seq incorporating quantification uncertainty. *Nature methods*, 14(7):687–690, 2017.
- [93] Pedro L Baldoni, Yunshun Chen, Soroor Hediye-zadeh, Yang Liao, Xueyi Dong, Matthew E Ritchie, Wei Shi, and Gordon K Smyth. Dividing out quantification uncertainty allows efficient assessment of differential transcript expression. *bioRxiv*, pages 2023–04, 2023.
- [94] Jeffrey C Miecznikowski and Jiefei Wang. Error control in tree structured hypothesis testing. *Wiley Interdisciplinary Reviews: Computational Statistics*, 15(4):e1603, 2023.
- [95] Jelle J Goeman and Aldo Solari. The sequential rejection principle of familywise error control. *The Annals of Statistics*, pages 3782–3810, 2010.
- [96] Daniel Yekutieli. Hierarchical false discovery rate–controlling methodology. *Journal of the American Statistical Association*, 103(481):309–316, 2008.
- [97] Gavin Lynch and Wenge Guo. On procedures controlling the *fdr* for testing hierarchically ordered hypotheses. *arXiv preprint arXiv:1612.04467*, 2016.
- [98] Marina Bogomolov, Christine B Peterson, Yoav Benjamini, and Chiara Sabatti. Hypotheses on a tree: new error rates and testing strategies. *Biometrika*, 108(3):575–590, 2021.
- [99] Jian Xiao, Hongyuan Cao, and Jun Chen. False discovery rate control incorporating phylogenetic tree increases detection power in microbiome-wide multiple testing. *Bioinformatics*, 33(18):2873–2881, 2017.

- [100] John Pura, Xuechan Li, Cliburn Chan, and Jichun Xie. Team: A multiple testing algorithm on the aggregation tree for flow cytometry analysis. *arXiv preprint arXiv:1906.07757*, 2019.
- [101] Johannes Ostner, Salomé Carcy, and Christian L Müller. tascoda: Bayesian tree-aggregated analysis of compositional amplicon and single-cell data. *Frontiers in genetics*, 12:766405, 2021.
- [102] Yoav Benjamini and Yosef Hochberg. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal statistical society: series B (Methodological)*, 57(1):289–300, 1995.
- [103] Adam Frankish, Mark Diekhans, Anne-Maud Ferreira, Rory Johnson, Irwin Jungreis, Jane Loveland, Jonathan M Mudge, Cristina Sisu, James Wright, Joel Armstrong, et al. Gencode reference annotation for the human and mouse genomes. *Nucleic acids research*, 47(D1):D766–D773, 2019.
- [104] Fiona Cunningham, James E Allen, Jamie Allen, Jorge Alvarez-Jarreta, M Ridwan Amode, Irina M Armean, Olanrewaju Austine-Orimoloye, Andrey G Azov, If Barnes, Ruth Bennett, et al. Ensembl 2022. *Nucleic acids research*, 50(D1):D988–D995, 2022.
- [105] Felix Mölder, Kim Philipp Jablonski, Brice Letcher, Michael B Hall, Christopher H Tomkins-Tinch, Vanessa Sochat, Jan Forster, Soohyun Lee, Sven O Twardziok, Alexander Kanitz, et al. Sustainable data analysis with snakemake. *F1000Research*, 10, 2021.
- [106] S Anders. Huber w ((2010)) differential expression analysis for sequence count data. *Genome Biol*, 11(10):R106, 2010.

- [107] Nikolaus Kriegeskorte, W Kyle Simmons, Patrick SF Bellgowan, and Chris I Baker. Circular analysis in systems neuroscience: the dangers of double dipping. *Nature neuroscience*, 12(5):535–540, 2009.
- [108] Anna Neufeld, Lucy L Gao, Joshua Popp, Alexis Battle, and Daniela Witten. Inference after latent variable estimation for single-cell rna sequencing data. *arXiv preprint arXiv:2207.00554*, 2022.
- [109] Anna Neufeld, Joshua Popp, Lucy L Gao, Alexis Battle, and Daniela Witten. Negative binomial count splitting for single-cell rna sequencing data. *arXiv preprint arXiv:2307.12985*, 2023.
- [110] Eugene Katsevich, Chiara Sabatti, and Marina Bogomolov. Filtering the rejection set while preserving false discovery rate control. *Journal of the American Statistical Association*, 118(541):165–176, 2023.
- [111] Jason D Buenrostro, Paul G Giresi, Lisa C Zaba, Howard Y Chang, and William J Greenleaf. Transposition of native chromatin for fast and sensitive epigenomic profiling of open chromatin, DNA-binding proteins and nucleosome position. *Nature methods*, 10(12):1213–1218, 2013.
- [112] Rebekah R Starks, Anilisa Biswas, Ashish Jain, and Geetu Tuteja. Combined analysis of dissimilar promoter accessibility and gene expression profiles identifies tissue-specific genes and actively repressed networks. *Epigenetics & chromatin*, 12:1–16, 2019.
- [113] Sebastian Preissl, Rongxin Fang, Hui Huang, Yuan Zhao, Ramya Raviram, David U Gorkin, Yanxiao Zhang, Brandon C Sos, Veena Afzal, Diane E Dickel, et al. Single-

- nucleus analysis of accessible chromatin in developing mouse forebrain reveals cell-type-specific transcriptional regulation. *Nature neuroscience*, 21(3):432–439, 2018.
- [114] David Lara-Astiaso, Assaf Weiner, Erika Lorenzo-Vivas, Irina Zaretsky, Diego Adhemar Jaitin, Eyal David, Hadas Keren-Shaul, Alexander Mildner, Deborah Winter, Steffen Jung, et al. Chromatin state dynamics during blood formation. *science*, 345(6199):943–949, 2014.
- [115] Alicia N Schep, Jason D Buenrostro, Sarah K Denny, Katja Schwartz, Gavin Sherlock, and William J Greenleaf. Structured nucleosome fingerprints enable high-resolution mapping of chromatin architecture within regulatory regions. *Genome research*, 25(11):1757–1770, 2015.
- [116] Jason D Buenrostro, Beijing Wu, Ulrike M Litzenburger, Dave Ruff, Michael L Gonzales, Michael P Snyder, Howard Y Chang, and William J Greenleaf. Single-cell chromatin accessibility reveals principles of regulatory variation. *Nature*, 523(7561):486–490, 2015.
- [117] Darren A Cusanovich, Riza Daza, Andrew Adey, Hannah A Pliner, Lena Christiansen, Kevin L Gunderson, Frank J Steemers, Cole Trapnell, and Jay Shendure. Multiplex single-cell profiling of chromatin accessibility by combinatorial cellular indexing. *Science*, 348(6237):910–914, 2015.
- [118] Heng Li and Richard Durbin. Fast and accurate short read alignment with Burrows–Wheeler transform. *bioinformatics*, 25(14):1754–1760, 2009.
- [119] Heng Li, Bob Handsaker, Alec Wysoker, Tim Fennell, Jue Ruan, Nils Homer, Gabor Marth, Goncalo Abecasis, Richard Durbin, and 1000 Genome Project Data Processing

- Subgroup. The sequence alignment/map format and SAMtools. *bioinformatics*, 25(16):2078–2079, 2009.
- [120] Picard Toolkit. Broad institute. (*No Title*), 2019.
- [121] Heng Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, 34(18):3094–3100, 2018.
- [122] Chenfei Wang, Dongqing Sun, Xin Huang, Changxin Wan, Ziyi Li, Ya Han, Qian Qin, Jingyu Fan, Xintao Qiu, Yingtian Xie, et al. Integrative analyses of single-cell transcriptome and regulome using maestro. *Genome biology*, 21(1):1–28, 2020.
- [123] Haowen Zhang, Li Song, Xiaotao Wang, Haoyu Cheng, Chenfei Wang, Clifford A Meyer, Tao Liu, Ming Tang, Srinivas Aluru, Feng Yue, et al. Fast alignment and preprocessing of chromatin profiles with chromap. *Nature communications*, 12(1):6566, 2021.
- [124] Michael Roberts, Wayne Hayes, Brian R Hunt, Stephen M Mount, and James A Yorke. Reducing storage requirements for biological sequence comparison. *Bioinformatics*, 20(18):3363–3369, 2004.
- [125] A. Sina Boeshaghi, Fan Gao, and Lior Pachter. Assessing the multimodal tradeoff. *bioRxiv*, 2023.
- [126] Lorian Schaeffer, Harold Pimentel, Nicolas Bray, Páll Melsted, and Lior Pachter. Pseudoalignment for metagenomic read assignment. *Bioinformatics*, 33(14):2082–2088, 2017.

- [127] Giorgos Skoufos, Fatemeh Almodaresi, Mohsen Zakeri, Joseph N Paulson, Rob Patro, Artemis G Hatzigeorgiou, and Ioannis S Vlachos. AGAMEMNON: an Accurate metaGenomics And MEtatranscriptoMics quaNtificationON analysis suite. *Genome biology*, 23(1):39, 2022.
- [128] Tommi Mäklin, Teemu Kallonen, Sophia David, Ben Pascoe, Guillaume Méric, David M Aanensen, Edward J Feil, Samuel K Sheppard, Jukka Corander, and Antti Honkela. High-resolution sweep metagenomics using ultrafast read mapping and inference. *bioRxiv*, page 332544, 2018.
- [129] Wei Shen, Hongyan Xiang, Tianquan Huang, Hui Tang, Mingli Peng, Dachuan Cai, Peng Hu, and Hong Ren. KMCP: accurate metagenomic profiling of both prokaryotic and viral populations by pseudo-mapping. *Bioinformatics*, 39(1):btac845, 2023.
- [130] Timo Bingmann, Phelim Bradley, Florian Gauger, and Zamin Iqbal. *COBS: A Compact Bit-Sliced Signature Index*, page 285–303. Springer International Publishing, 2019.
- [131] F. P. Breitwieser, D. N. Baker, and S. L. Salzberg. KrakenUniq: confident and fast metagenomics classification using unique k-mer counts. *Genome Biology*, 19(1), November 2018.
- [132] Giulio Ermanno Pibiri. Sparse and skew hashing of k-mers. *Bioinformatics*, 38(Supplement_1):i185–i194, 2022.
- [133] Yong Zhang, Tao Liu, Clifford A Meyer, Jérôme Eeckhoute, David S Johnson, Bradley E Bernstein, Chad Nusbaum, Richard M Myers, Myles Brown, Wei Li, et al. Model-based analysis of ChIP-Seq (MACS). *Genome biology*, 9:1–9, 2008.

- [134] John M. Gaspar. Improved peak-calling with MACS2. *bioRxiv*, 2018.
- [135] Dongze He and Rob Patro. simpleaf: a simple, flexible, and scalable framework for single-cell data processing using alevin-fry. *Bioinformatics*, 39(10):btad614, 2023.
- [136] Fatemeh Almodaresi, HIRAK SARKAR, AVI SRIVASTAVA, and Rob Patro. A space and time-efficient index for the compacted colored de Bruijn graph. *Bioinformatics*, 34(13):i169–i177, 2018.
- [137] Jamshed Khan and Rob Patro. Cuttlefish: fast, parallel and low-memory compaction of de bruijn graphs from large-scale genome collections. *Bioinformatics*, 37(Supplement_1):i177–i186, 2021.
- [138] AVI SRIVASTAVA, LARAIB MALIK, HIRAK SARKAR, MOHSEN ZAKERI, FATEMEH ALMODARESI, CHARLOTTE SONESON, MICHAEL I. LOVE, CARL KINGSFORD, and Rob Patro. Alignment and mapping methodology influence transcript abundance estimation. *Genome Biology*, 21(1), September 2020.
- [139] M. Holtgrewe. Mason - A Read Simulator for Second Generation Sequencing Data. *Technical Report FU Berlin*, October 2010.
- [140] Tim Stuart, AVI SRIVASTAVA, SHAISTA MADAD, CALEB A LAREAU, and RAHUL SATIJA. Single-cell chromatin state analysis with signac. *Nature methods*, 18(11):1333–1341, 2021.
- [141] Alexis Morrissey, Jeffrey Shi, Daniela Q James, and Shaun Mahony. Allo: Accurate allocation of multi-mapped reads enables regulatory element analysis at repeats. *bioRxiv*, 2023.

- [142] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the royal statistical society: series B (methodological)*, 39(1):1–22, 1977.
- [143] Glennis A Logsdon, Mitchell R Vollger, and Evan E Eichler. Long-read human genome sequencing and its applications. *Nature Reviews Genetics*, 21(10):597–614, 2020.
- [144] Vivien Marx. Method of the year: long-read sequencing. *Nature Methods*, 20(1):6–11, 2023.
- [145] Ying Chen, Andre Sim, Yuk Kei Wan, Keith Yeo, Joseph Jing Xian Lee, Min Hao Ling, Michael I Love, and Jonathan Göke. Context-aware transcript quantification from long-read rna-seq data with bambu. *Nature methods*, 20(8):1187–1195, 2023.
- [146] Zahra Zare Jousheghani and Rob Patro. Oarfish: Enhanced probabilistic modeling leads to improved accuracy in long read transcriptome quantification. *bioRxiv*, 2024.
- [147] Hyun Joo Ji and Mihaela Pertea. Enhancing transcriptome expression quantification through accurate assignment of long rna sequencing reads with transigner. *bioRxiv*, 2024.
- [148] Yuan Gao, Feng Wang, Robert Wang, Eric Kutschera, Yang Xu, Stephan Xie, Yuanyuan Wang, Kathryn E Kadash-Edmondson, Lan Lin, and Yi Xing. Espresso: robust discovery and quantification of transcript isoforms from error-prone long-read rna-seq data. *Science Advances*, 9(3):eabq5072, 2023.
- [149] Yael Baran, Akhiad Bercovich, Arnau Sebe-Pedros, Yaniv Lubling, Amir Giladi, Elad Chomsky, Zohar Meir, Michael Hoichman, Aviezer Lifshitz, and Amos Tanay. Metacell:

- analysis of single-cell rna-seq data using k-nn graph partitions. *Genome biology*, 20:1–19, 2019.
- [150] Caleb A Lareau, Fabiana M Duarte, Jennifer G Chew, Vinay K Kartha, Zach D Burkett, Andrew S Kohlway, Dmitry Pokholok, Martin J Aryee, Frank J Steemers, Ronald Lebofsky, et al. Droplet-based combinatorial indexing for massive-scale single-cell chromatin accessibility. *Nature biotechnology*, 37(8):916–924, 2019.
- [151] Darren A Cusanovich, James P Reddington, David A Garfield, Riza M Daza, Delasa Aghamirzaie, Raquel Marco-Ferrerres, Hannah A Pliner, Lena Christiansen, Xiaojie Qiu, Frank J Steemers, et al. The cis-regulatory dynamics of embryonic development at single-cell resolution. *Nature*, 555(7697):538–542, 2018.
- [152] Anja Mezger, Sandy Klemm, Ishminder Mann, Kara Brower, Alain Mir, Magnolia Bostick, Andrew Farmer, Polly Fordyce, Sten Linnarsson, and William Greenleaf. High-throughput chromatin accessibility profiling at single-cell resolution. *Nature communications*, 9(1):3647, 2018.
- [153] Song Chen, Blue B Lake, and Kun Zhang. High-throughput sequencing of the transcriptome and chromatin accessibility in the same cell. *Nature biotechnology*, 37(12):1452–1457, 2019.
- [154] Charlotte Soneson, Michael I Love, and Mark D Robinson. Differential analyses for rna-seq: transcript-level estimates improve gene-level inferences. *F1000Research*, 4, 2015.
- [155] Kristoffer Sahlin. Effective sequence similarity detection with strobemers. *Genome research*, 31(11):2080–2094, 2021.