NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCES
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

BSc THESIS

MLscANApp: Creating an Interactive RShiny Interface for scRNA-seq Bioinformatics Data Analysis

Violetta D. Gkika

Supervisor: **Elias Manolakos,** Professor

ATHENS

OCTOBER 2024

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

MLscANApp: Δημιουργία Εφαρμογής RShiny για Ανάλυση scRNA-seq Δεδομένων Βιοπληροφορικής

Βιολέττα Δ. Γκίκα

Επιβλέπων: **Ηλίας Μανωλάκος,** Καθηγητής

ΑΘΗΝΑ

ΟΚΤΩΒΡΙΟΣ 2024

BSc THESIS

MLscANApp: Creating an Interactive RShiny Interface for scRNA-seq Bioinformatics Data Analysis

Violetta D. Gkika

S.N.: 1115201600222

SUPERVISOR: **Elias Manolakos,** Professor

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

MLscANApp: Δημιουργία Εφαρμογής RShiny για Ανάλυση scRNA-seq Δεδομένων Βιοπληροφορικής

Βιολέττα Δ. Γκίκα

Α.Μ.: 1115201600222

ΕΠΙΒΛΕΠΩΝ: **Ηλίας Μανωλάκος,** Καθηγητής

ABSTRACT

Single-cell RNA sequencing (scRNA-seq) has revolutionized the study of gene expression, profiles offering detailed insights into cellular heterogeneity and dynamic biological processes. However, analyzing scRNA-seq datasets presents significant challenges due to their high dimensionality, noisy gene expressions, and the need for sophisticated computational methods to draw meaningful biological conclusions.

MLscAN (Machine Learning for Single-Cell ANalytics) is an R package that provides a comprehensive pipeline for unbiased scRNA-seq data analysis. By using exclusively only unsupervised machine learning methods such as dimensionality reduction, clustering, trajectory inference, and gene regulatory network reconstruction, MLscAN enables researchers to infer cell states and identify cell state transitions and the molecular drivers of these biological progression processes—all without relying on any prior knowledge. Despite its powerful features, MLscAN requires some programming expertise in R which presents a barrier to use for many life scientists.

To overcome this challenge, this thesis introduces MLscANApp (Machine Learning for Single-Cell ANalytics Application), an RShiny application designed to make MLscAN's many capabilities accessible through a user-friendly graphical interface (GUI). MLscANApp allows life scientists to conduct the MLscAN workflow of comprehensive scRNA-seq analyses without the need for coding skills. Users can leverage the full functionality of the MLscAN pipeline, by getting guidance for their analysis at each step and interpreting results through a plethora of interactive and insightful visualizations. This application bridges the gap between complex computational methods and practical research, making advanced scRNA-seq-based analysis accessible to a broader scientific community.

SUBJECT AREA: Machine Learning in Computational Biology

KEYWORDS: Single-cell RNA-sequencing, R, Biological Data Analysis, RShiny Application, Machine Learning, Bioinformatics, Data Visualization

ΠΕΡΙΛΗΨΗ

Η αλληλούχιση RNA μονήρους κυττάρου (scRNA-seq) έχει φέρει επανάσταση στη μελέτη της γονιδιακής έκφρασης, προσφέροντας λεπτομερείς γνώσεις για την κυτταρική ετερογένεια και τις δυναμικές βιολογικές διαδικασίες. Ωστόσο, η ανάλυση των δεδομένων scRNA-seq παρουσιάζει σημαντικές προκλήσεις λόγω της υψηλής διαστατικότητάς τους, του θορύβου και της ανάγκης για προηγμένες υπολογιστικές μεθόδους για την εξαγωγή σημαντικών βιολογικών συμπερασμάτων.

Το MLscAN (Μηχανική Μάθηση για Ανάλυση Μονοκυττάρων) είναι ένα πακέτο R που παρέχει μια ολοκληρωμένη ροή εργασιών για την ανάλυση scRNA-seq δεδομένων χωρίς να χρησιμοποιεί προηγούμενη βιολογική γνώση. Ενσωματώνοντας τεχνικές μη επιβλεπόμενης μηχανικής μάθησης όπως η μείωση διαστάσεων, η ομαδοποίηση, η εξαγωγή τροχιακών δεδομένων και η ανακατασκευή δικτύων ρύθμισης γονιδίων, το MLscAN επιτρέπει στους ερευνητές να συμπεραίνουν καταστάσεις κυττάρων, να εντοπίζουν μεταβάσεις καταστάσεων κυττάρων και τα γονίδια που οδηγούν αυτές τις βιολογικές διαδικασίες—όλα αυτά χωρίς να βασίζεται σε οποιαδήποτε προηγούμενη γνώση. Παρόλα αυτά η χρήση του MLscAN απαιτεί τη γνώση προγραμματισμού σε γλώσσα R κάτι που δυσκολεύει πολλούς επιστήμονες στο χώρο της βιοιατρικής.

Για να ξεπεραστεί αυτή η δυσκολία, η πτυχιακή αυτή εργασία παρουσιάζει το MLscANApp (Εφαρμογή Μηχανικής Μάθησης για Αναλύσεις Μονοκυττάρων), μια εφαρμογή RShiny σχεδιασμένη να καθιστά τις πολλές δυνατότητες του MLscAN προσβάσιμες μέσω ενός φιλικού προς τον χρήστη γραφικού περιβάλλοντος (GUI). Η εφαρμογή MLscANApp επιτρέπει στους ερευνητές να εκτελούν τη ροή εργασίας MLscAN για ολοκληρωμένες αναλύσεις scRNA-seq χωρίς την ανάγκη προγραμματιστικών δεξιοτήτων. Οι χρήστες μπορούν να αξιοποιήσουν πλήρως τη λειτουργικότητα του MLscAN, λαμβάνοντας καθοδήγηση για την ανάλυσή τους σε κάθε βήμα της διαδικασίας και ερμηνεύοντας τα αποτελέσματα μέσω μιας πληθώρας διαδραστικών οπτικοποιήσεων. Αυτή η εφαρμογή γεφυρώνει το χάσμα μεταξύ σύνθετων υπολογιστικών μεθόδων και της πρακτικής έρευνας, καθιστώντας την προηγμένη ανάλυση δεδομένων scRNA-seq προσβάσιμη στην ευρύτερη επιστημονική κοινότητα.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Μηχανική Μάθηση στην Υπολογιστική Βιολογία

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Αλληλούχιση RNA Μεμονωμένων Κυττάρων, R, Ανάλυση Βιολογικών Δεδομένων, Εφαρμογή RShiny, Μηχανική Μάθηση, Βιοπληροφορική, Οπτικοποίηση Δεδομένων

ACKNOWLEDGEMENTS

I would like to express my heartfelt gratitude to everyone who has supported me throughout the completion of this thesis.

Firstly, I extend my deepest thanks to my thesis advisor, Prof. Elias Manolakos, for providing me the opportunity to explore such a fascinating and significant field. His invaluable guidance, expert advice, and unwavering support have been crucial in shaping this work and my future academic and professional aspirations.

To my parents, their constant emotional support and belief in my abilities have been a profound source of strength and motivation. Their sacrifices and encouragement have been instrumental in my journey.

Finally, I want to thank my dear friend and collaborator, Ioannis Mystakidis, for being a source of inspiration and for contributing his knowledge and enthusiasm to this project. His collaborative spirit has made this journey both productive and enjoyable.

CONTENTS

LIST OF FIGURES

1. INTRODUCTION

At the heart of all living organisms lies the cell, the smallest unit of life, yet the source of the greatest biological complexity. Since the formulation of cell theory, the idea that all living beings are composed of cells has reshaped our understanding of life itself. Each cell carries the intricate genetic blueprints that guide its functions, allowing it to adapt, interact, and contribute to the larger biological systems of which it is part. Cells are not just building blocks; they are the architects of development, growth, and adaptation, a dynamic, autonomous unit driving the organism's development and survival [1]. Within each cell, the transcriptome—the complete set of RNA molecules—plays a crucial role, as it reflects which genes are actively being expressed and thus governs the cell's behavior. The transcriptome provides a window into the regulatory mechanisms controlling cellular states, allowing researchers to explore how genes are regulated in different conditions and cell types. Dysregulation of these processes often leads to diseases such as cancer, making transcriptome analysis vital for understanding health and disease [2]. To understand life in its fullest form, we must first unravel the mysteries of the cell and its transcriptome, for it is within these microscopic units that the secrets of health and disease reside.

Advancements in molecular biology have enabled researchers to study the transcriptome in an unprecedented resolution through RNA sequencing (RNA-seq) [3]. Traditionally, bulk RNA-seq averaged gene expression across thousands of cells, masking tissue heterogeneity. In contrast, single-cell RNA sequencing (scRNA-seq) has revolutionized the field by allowing scientists to examine gene expression at the resolution of individual cells [4]. scRNA-seq is a high-throughput technology that enables the isolation and sequencing of RNA extracted from individual cells, providing comprehensive insights into cellular heterogeneity, and gene regulatory mechanisms, and facilitating the identification of rare or novel cell populations within complex biological systems.

The versatility of scRNA-seq has led to its widespread adoption across various fields of biology and medicine. With its ability to analyze individual cells, scRNA-seq has been instrumental in large-scale projects like the Human Cell Atlas and Tabula Muris, which aim to catalog cell types across different tissues in Homo sapiens and Mus musculus, respectively [5] [6]. Beyond these initiatives, understanding cellular processes at the single-cell level has been crucial for advancements in developmental biology and disease research. Cellular heterogeneity, the existence of diverse cell types within tissues, has been shown to play a significant role in biological processes and disease progression [7]. For instance, in cancer and leukemia, scRNA-seq enables the identification of rare subpopulations of cells that drive disease progression and treatment resistance [8]. By uncovering the molecular heterogeneity within tumors, scRNA-seq aids in developing precision medicine approaches tailored to specific cellular signatures. It has also been applied to autoimmune diseases such as lupus, where it helps pinpoint pathogenic cell types, and in drug discovery, where it reveals novel therapeutic targets [9] [10].

Since the initial scRNA-seq study [11], technological advancements and improved equipment have significantly increased the number of cells that can be sequenced in a single experiment—from hundreds to millions. This rapid expansion in sequencing capacity has resulted in the generation of vast amounts of data, necessitating robust computational tools to process and interpret these complex datasets. Consequently, from data generation to analysis and interpretation, a comprehensive bioinformatics workflow is essential to effectively manage this increased volume and complexity [12]. Critical steps in such workflows include quality control, read mapping, gene expression quantification, batch ef-

fect correction, normalization, imputation, dimensionality reduction, feature selection, cell clustering, trajectory inference, differential expression analysis, alternative splicing, allelic expression, and gene regulatory network (GRN) reconstruction. Several software applications and packages have been developed to support these tasks. Among these tools, MLscAN (Machine Learning for single-cell Analytics) stands out as a particularly effective solution for unbiased single-cell data analysis. It provides an end-to-end computational pipeline that allows users to infer cell state landscapes, transition dynamics, and key regulatory genes without requiring any prior knowledge about cell types. However, MLscAN requires researchers to have at least some programming skills in the R language to utilize its capabilities fully.

To address this limitation, this thesis presents MLscANApp, an RShiny application designed for analyzing scRNA-seq data using MLscAN. MLscANApp functions as an interactive bioinformatics tool with a user-friendly graphical interface, covering all steps MLscAN supports, from dimensionality reduction and clustering to trajectories and Gene Regulatory Network reconstruction. It empowers biomedical researchers with no programming skills, to still engage with machine learning for an unbiased downstream analysis of gene expression profile datasets and obtain insightful visualizations of the results.

1.1 Thesis goals

The main goal of this thesis is to design and develop a graphical interface that utilizes the existing MLscAN R package. This interface aims to wrap all the functionalities of MLscAN into a user-friendly application, primarily serving researchers who lack programming skills. By providing an accessible platform, the interface enables biologists to perform complex scRNA-seq analyses without the need for coding knowledge, effectively bridging the gap between advanced bioinformatics tools and practical research requirements.

A crucial part of this thesis was designing and implementing the first fully functional application version. This involved integrating the diverse functionalities of the MLscAN package into the interface, ensuring all the essential stages of the MLscAN computational pipeline, from dimensionality reduction to gene regulatory network reconstruction, were seamlessly incorporated. The development process focused on creating a robust foundation to support future enhancements while offering researchers a comprehensive tool for their single-cell studies.

Another goal was to improve the visualization of results. While MLscAN offers a wide range of plots, this project aimed to make these visualizations interactive to utilize their full potential. By enhancing the ability to explore and manipulate graphical outputs, researchers can gain deeper insights into their data, making the analysis process more informative and intuitive.

Lastly, the thesis aims to demonstrate the utility of this interface by conducting a complete analysis using real scRNA-seq data from previously published works. By presenting a detailed workflow, we showcase the application of the Shiny App in a practical research setting, highlighting its potential to streamline and enhance the process of scRNA-seq data analysis for researchers. This demonstration underscores the app's value as a comprehensive tool for exploring single-cell gene expression patterns, inferring cell states and how they interact.

1.2 Thesis Organization

This thesis is organized into five chapters, each focusing on different aspects of the development and application of MLscANApp, an RShiny application for analyzing single-cell RNA sequencing (scRNA-seq) data.

Chapter 2 provides a comprehensive overview of the existing MLscAN pipeline. It details how the pipeline was developed, its core functionalities, and the features it offers for single-cell data analysis. Additionally, this chapter analyzes the need for the RShiny app, highlighting the limitations of MLscAN that the new application aims to address.

Chapter 3 focuses on the software engineering aspects of the developed Shiny application, MLscANApp. It provides an in-depth description of both the interface and the underlying logic, as well as the overall architecture of the application. This section will be particularly beneficial for developers and those with coding expertise, as it breaks down the design principles, modular components, and implementation details that make MLscANApp a robust and scalable tool for scRNA-seq analysis.

Chapter 4 showcases two potential workflows for scientists who wish to utilize this app for their analyses. It uses real data sets from published papers and provides a step-by-step description of the application's features, demonstrating how researchers can leverage MLscANApp for different types of scRNA-seq studies.

Finally, **Chapter 5** concludes the thesis by summarizing the work, reflecting on its significance, and outlining potential future improvements. This last chapter offers insights into how the tool can continue to evolve to meet the growing needs of the bioinformatics community.

2. BACKGROUND & RELATED WORK

2.1 MLscAN Pipeline Overview

A complete single-cell data analysis pipeline, developed over the years by graduate students in the research group of Prof. Manolakos at the National and Kapodistrian University of Athens [13] [14] [15] [16], the MLscAN R package builds on the foundational research outlined in [17]. This collaborative effort aims to provide an unbiased analysis of gene expression profiles through a flexible pipeline implemented in R that uses unsupervised machine learning methods and integrates into one package critical analytical steps, including dimensionality reduction, cell states inference, identification of the key genes that drive the interaction of state pairs, and their dynamic gene regulatory networks (GRN). The package offers a streamlined workflow that allows researchers to reconstruct the landscape of cell states in an unbiased manner and explore the dynamics and gene regulation in a robust and reproducible manner. The MLscAN pipeline is implemented in such a way that the user can execute the whole pipeline using one key function (`MLscAN` function), altering different parameters to achieve the desired results. In that sense, it requires minimal knowledge of the R language. The MLscAN pipeline encompasses several core phases that guide users through the entire analysis workflow:

MLscAN's pipeline encompasses several core phases that guide users through the entire analysis process:

- **Input Data**: Users begin by inputting a pre-processed gene expression matrix, forming the analysis's foundation.
- **Feature Selection and Dimensionality Reduction**: MLscAN selects the most biologically relevant genes and reduces the dimensionality of the data using Principal Component Analysis (PCA).
- **Clustering and Model Selection**: MLscAN employs Gaussian Mixture Models (GMMs) to cluster cells into distinct states, optimizing the number of clusters using the Bayesian Information Criterion (BIC).
- **Transitions and Trajectories**: MLscAN infers state-to-state transitions and cellular trajectories, carving the dynamic pathways cells follow during biological processes.
- **Micro-State Partitioning**: MLscAN partitions cell transitions into phases, called "micro-states", to provide a detailed understanding of cellular progression.
- **Key-Gene Identification**: MLscAN identifies "key" regulatory genes that drive each inferred state-to-state transition, highlighting the molecular mechanisms behind cellular changes.
- **Gene Regulatory Network (GRN) Inference**: The package can reconstruct GRNs of the key genes for each micro-state, shedding light on their dynamic regulatory interactions controlling gene expression dynamics during all phases of the state transitions.

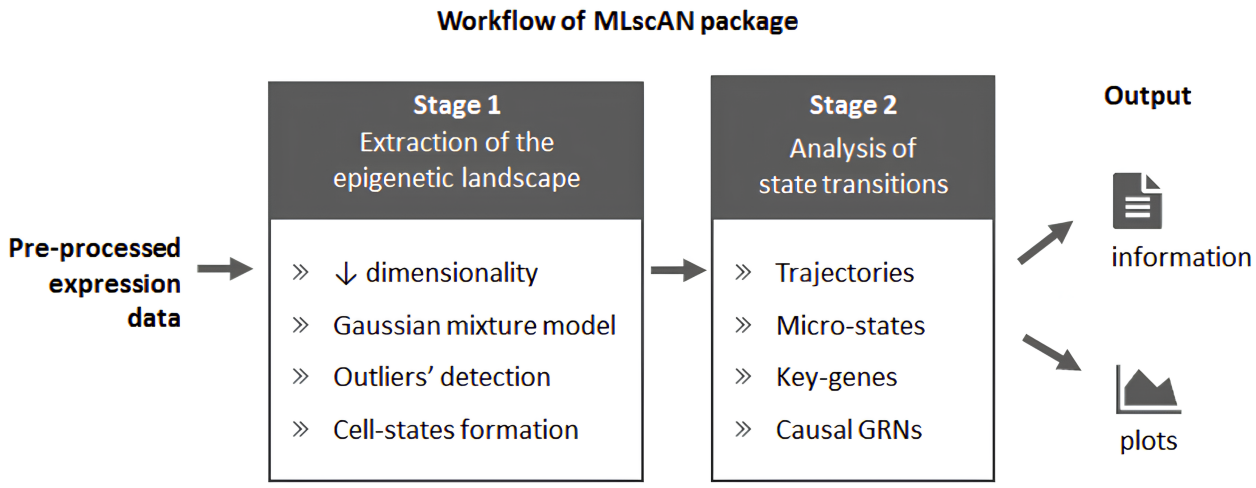

Figure 2.1: MLscANApp pipeline workflow as described by the first developer of the MLscAN pipeline, Efi Malesiou.

All these functionalities are organized into distinct steps:

1. Data set Input
2. Feature Selection and Dimensionality Reduction
3. Clustering and Modeling
4. Transition identification, Trajectory inference, and Micro-States partition
5. Key Genes and their Gene Regulatory Networks (GRNs) reconstruction

One key characteristic of MLscAN is its flexibility, allowing users to intervene at each analysis step by setting different parameters. Additionally, users can incorporate their own script codes for various methods, seamlessly integrating them into the pipeline. For steps such as dimensionality reduction and clustering, MLscAN even permits users to provide their own datasets that have been pre-processed or clustered using custom methods. However, accessing these advanced features requires considerable knowledge of R programming.

2.2 Need for MLscANApp

The need for accessible, GUI-based tools in single-cell analysis is well-recognized by the bioinformatics community. Several existing software applications, including SeuratWizard [18], Azimuth [19], Cerebro [20], iCellR [21], and Scope [22] offer graphical interfaces to facilitate complex analyses. The prevalence of these tools underscores the importance of GUIs in making advanced bioinformatics accessible to a broader range of researchers.

Given this trend and the many capabilities of MLscAN, developing a dedicated GUI for this package was a logical next step that motivated our work. MLscANApp was designed to address the gap between powerful computational tools and their accessibility, building

on the core functionalities of MLscAN but packaging them into a user-friendly application specifically tailored for non-programmers, or simply users who want to get familiar with the package before delving deeper into its capabilities programmatically. This decision aligns with the observed preference in the bioinformatics community for GUI-based tools, reinforcing the notion that researchers should focus mainly on data interpretation rather than developing coding skills.

The primary aim of MLscANApp is to provide life scientists with an intuitive interface to harness the robust analytical capabilities of MLscAN for unbiased scRNA-seq data analysis without the need to write or understand code. Doing so opens up all MLscAN's sophisticated functionalities to a wider user audience. Users can effortlessly input their data, adjust parameters through sliders and dropdown menus, and visualize complex results in real-time. This streamlines the analytical workflow and enhances the overall research experience by allowing scientists to concentrate on interpreting their findings instead of wrestling with computational hurdles.

3. MLSCANAPP: DESIGN AND IMPLEMENTATION

This chapter provides a detailed description of the technical aspects involved in its development, including the integration with the existing MLscAN R package and an overview of the app's interface and user experience.

3.1 Overview of MLscANApp

The MLscANApp is an RShiny application that directly employs the existing MLscAN R package, offering users an intuitive interface to interact with its functions.

RShiny is a web application framework for R that facilitates the creation of interactive, data-driven applications [23]. It allows users with R programming knowledge to develop applications without delving into complex web development frameworks. The code is typically organized into two main components: the user interface (UI) and the server function. The UI defines the layout and input controls, while the server function manages the logic and outputs in response to user interactions [24]. In our case, the server-side processes the inputs, communicates them to MLscAN, and retrieves the results to present them back to the user through the UI.

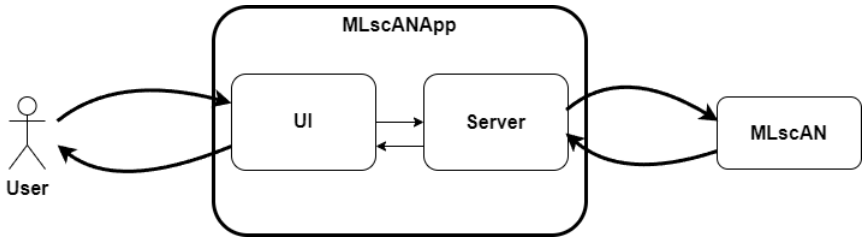

Figure 3.1: Basic architecture of the MLscANApp, illustrating the flow of interactions between the user and its various components.

Figure 3.1 illustrates the connection between these components. A more detailed explanation will follow in the next two sections.

The MLscANApp is designed to guide users step-by-step through the analysis process, allowing them to intervene at each stage and efficiently obtain relevant results. Each step in the MLscANApp corresponds directly to a step in the MLscAN R package, as outlined in 2.1. This structure ensures a seamless integration of the user experience with the analytical capabilities of MLscAN, facilitating a more intuitive and effective data analysis workflow. A more detailed description of the interface will follow in the section 3.4.

3.2 Modular Design

The MLscANApp is built following a modular design approach, a key principle in `RShiny` development that helps manage complex applications efficiently. In `RShiny`, **modularity** involves breaking the app into smaller, self-contained components, or ”modules,” each

responsible for a specific part of the app's functionality. This approach enhances reusability, scalability, and maintainability by isolating different app parts into independent units, making it easier to troubleshoot, update, or extend the app as needed [25].

In the MLscANApp, each phase of the MLscAN pipeline—dimensionality reduction, clustering/modeling, trajectory inference, and reconstruction of gene regulatory networks (GRNs)—is implemented as a separate module. This modular structure ensures that each step in the MLscAN R package is handled independently by its respective module. Each module is solely responsible for executing the tasks related to that particular analysis step, ensuring clean separation of concerns and streamlined workflow management.

To implement this modular design, the MLscANApp leverages the `golem` package, which simplifies the process of building modular and scalable Shiny applications. `Golem` provides a structured framework that helps organize the app into distinct components, separating each module's Interface (UI) and server logic. This approach allows each module to function independently while maintaining a cohesive workflow.[25].

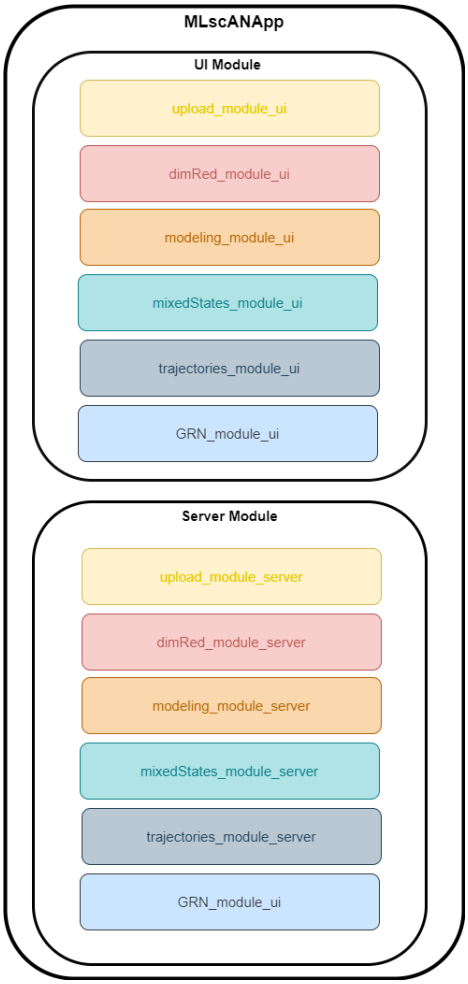

Figure 3.2: Modular design of the MlscANApp. The modules outlined in black represent the default modules provided by the golem framework. The colored modules are custom-built, with each color corresponding to a specific step in the pipeline. Each step has its corresponding module, which is divided into two sub-modules: one that handles the user interface (UI) and another that manages the server-side logic for that particular step.

The structure of the application using `golem` framework is given below:

```
.
  DESCRIPTION
  NAMESPACE
  R/
      app_ui.R                    # Defines the user interface (UI)
      app_server.R                # Contains server-side logic
      run_app.R                   # Main function to run the app
      app_config.R                # App-specific configuration
      global.R                    # Global variables and setup
      dimRed_module_ui.R          # UI for dimensionality reduction module
      dimRed_module_server.R      # Server logic for dimensionality reduction module
      upload_module_ui.R          # UI for upload module
      upload_module_server.R      # Server logic for upload module
      modeling_module_ui.R        # UI for modeling module
      modeling_module_server.R    # Server logic for modeling module
      trajectories_module_ui.R    # UI for trajectories module
      trajectories_module_server.R# Server logic for trajectories module
      GRN_module_ui.R             # UI for Gene Regulatory Network (GRN) module
      GRN_module_server.R         # Server logic for GRN module
      mixedStates_module_ui.R     # UI for mixed states module
      mixedStates_module_server.R # Server logic for mixed states module
  inst/
      app/
          www/
              bootstrap.css       # CSS for styling the app
          app.R                   # Main entry for deployment
      golem-config.yml            # Configuration file
  tests/
      testthat/                   # Automated test scripts
  man/                             # Documentation for app functions
  dev/
      01_start.R                  # Setup script
      02_dev.R                    # Script to add modules/functions
      03_deploy.R                 # Script to deploy the app
  data/
       data-raw/                   # Raw data for included datasets
```

As we can see from the structure above and Figure Figure 3.2, we have integrated our modules, that implement specific functionalities of the pipeline, into the modular design of the `golem` framework.

3.3 Server Side and Integration with MLscAN

As mentioned earlier, each module/tab in the MLscANApp corresponds to a specific step in the MLscAN pipeline. To achieve this, each module triggers a separate run of the `MLscAN` function, passing the corresponding parameters required for that step. The app utilizes the `stopAt` parameter of the `MLscAN` function from the MLscAN R package to control the flow and stop the process at the appropriate phase.

Each module is responsible for reading and saving the parameters the MLscAN function requires for that particular step. These parameters are stored in global structures, allowing other modules to access them as needed when running the `MLscAN` function. That means the user should understand which parameters are necessary at each step, refer to the corresponding tab whenever they want to modify them, and observe how they affect the analysis. However, once the parameters for a particular tab are set, they remain stored and accessible for the subsequent steps, ensuring consistency and ease of use throughout the analysis process.

When a user gives the command to run the analysis, the module executes the MLscAN function using its own parameters and those of the previous steps, which are stored in the global structures - implemented in the `global.R`. Importantly, the function only uses parameters from the current and previous steps of the pipeline, excluding any parameters related to later steps, even if those parameters have been set by the user in previous runs. This ensures that each run remains consistent with the logical progression of the MLscAN pipeline and avoids the use of premature inputs from later steps.

After the MLscAN function returns the results, the server side of the MLscANApp reformats them into messages and plots, which are then displayed in the interface. Figure 3.3 provides a more detailed view of the app's architecture.

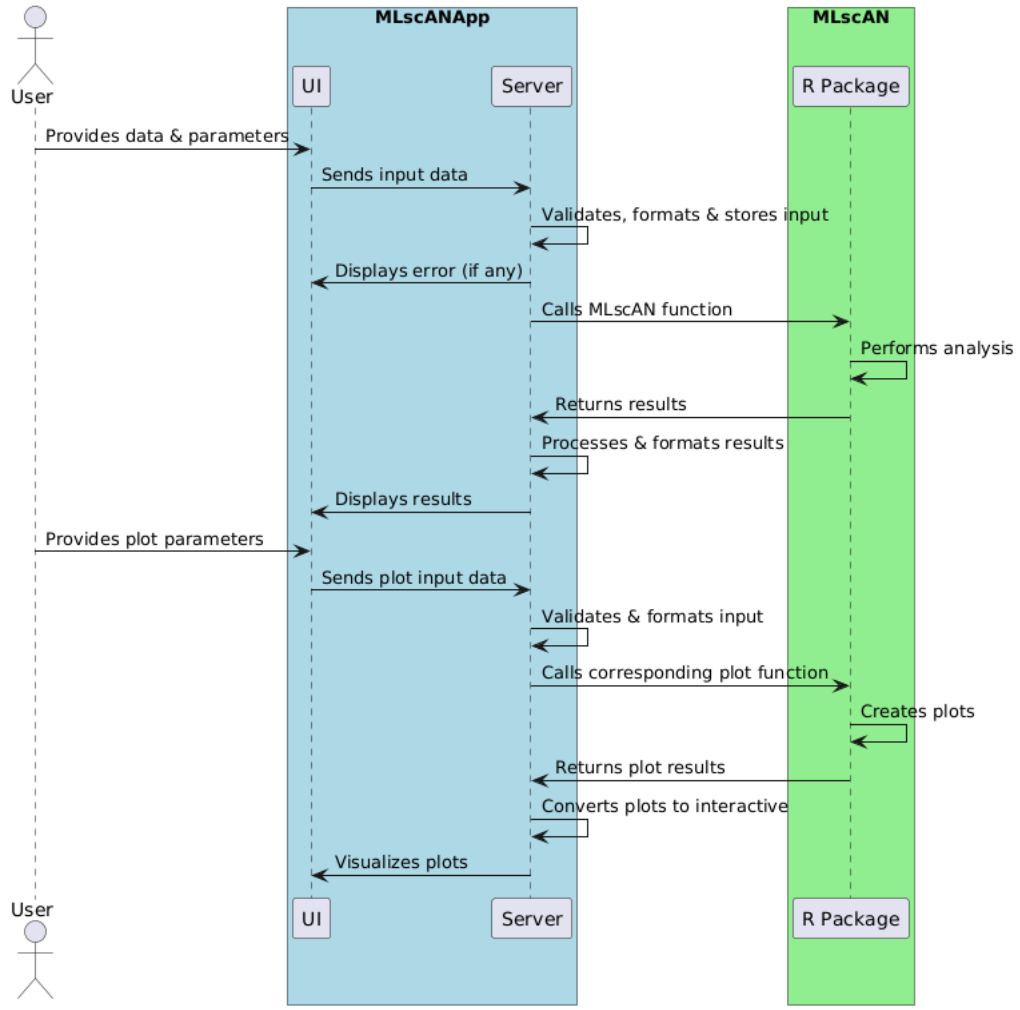

Figure 3.3: Detailed Architecture of MLscANApp and interaction with the MLscAN R package. Users provide input data and parameters through the UI. The server processes these inputs, validates them, and stores them before sending the data to the MLscAN functions, where the analysis is performed. The results are then returned to the server, processed, and displayed to the user through the UI. For plot generation, users input plot parameters, which are validated and sent to the corresponding MLscAN plot functions. The resulting plots are converted to interactive visualizations and displayed in the app's UI.

3.4 User Interface

The MLscANApp features a simple and intuitive user interface, organized around a navigation bar where each tab corresponds to a distinct phase of the downstream data analysis process outlined by the MLscAN R package. This tab-based layout guides users through each step of the MLscAN pipeline, enabling seamless interaction with the complex analytical processes involved.

In each tab, users provide parameters through various input fields, including text inputs, sliders, and dropdown menus. These fields are designed to accept the specific parameters the MLscAN functions require. Info messages are generated automatically to inform and encourage the user to modify parameters if they enter values that are unacceptable to

the MLscAN functions, ensuring that only valid inputs proceed to analysis. If the user insists on providing such parameters, the server handles this by applying valid defaults and informing the user about the parameters used.

To initiate a data analysis step, users press buttons tied to particular MLscAN functions or workflows. For example, a button might trigger the execution of a single MLscAN function or initiate a sequence of steps, such as e.g., removing mixed-state cells (more on chapter chapter 4), which runs several pre-processing steps before calling the `MLscAN` function itself.

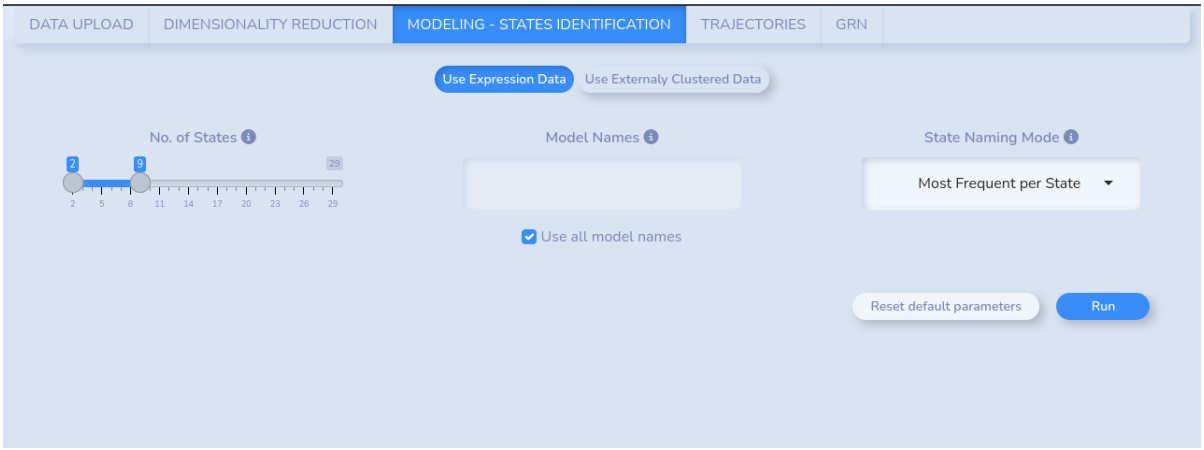

Figure 3.4: Example of the app interface for a specific step, showcasing the input fields and buttons the user interacts with to perform the analysis

As shown in Figure 3.4, the interface allows users to input the required parameters through various fields and then press the 'Run' button to perform the analysis,

The app leverages **reactivity** [24] to enhance user interaction. When users adjust any input field (e.g., sliders or dropdown selections), the new value is instantly saved and is ready to be applied the next time a "Run" button is pressed. This ensures that users can tweak their parameters without triggering unnecessary computations, promoting efficient workflow management.

Upon pressing the "Run" button, the app generates visualizations based on the most up-to-date set of parameters. The app displays corresponding representative plots for each pipeline phase, illustrating the analysis results. Each plot is wrapped within an accordion-style component, allowing users to collapse or expand them as needed. Additionally, some plots require further input parameters to modify their output, which can be adjusted using additional input fields within the same interface. Alongside these visualizations, a brief informational "box" is provided to give users a concise description of the results, aiding in interpreting each analysis step.

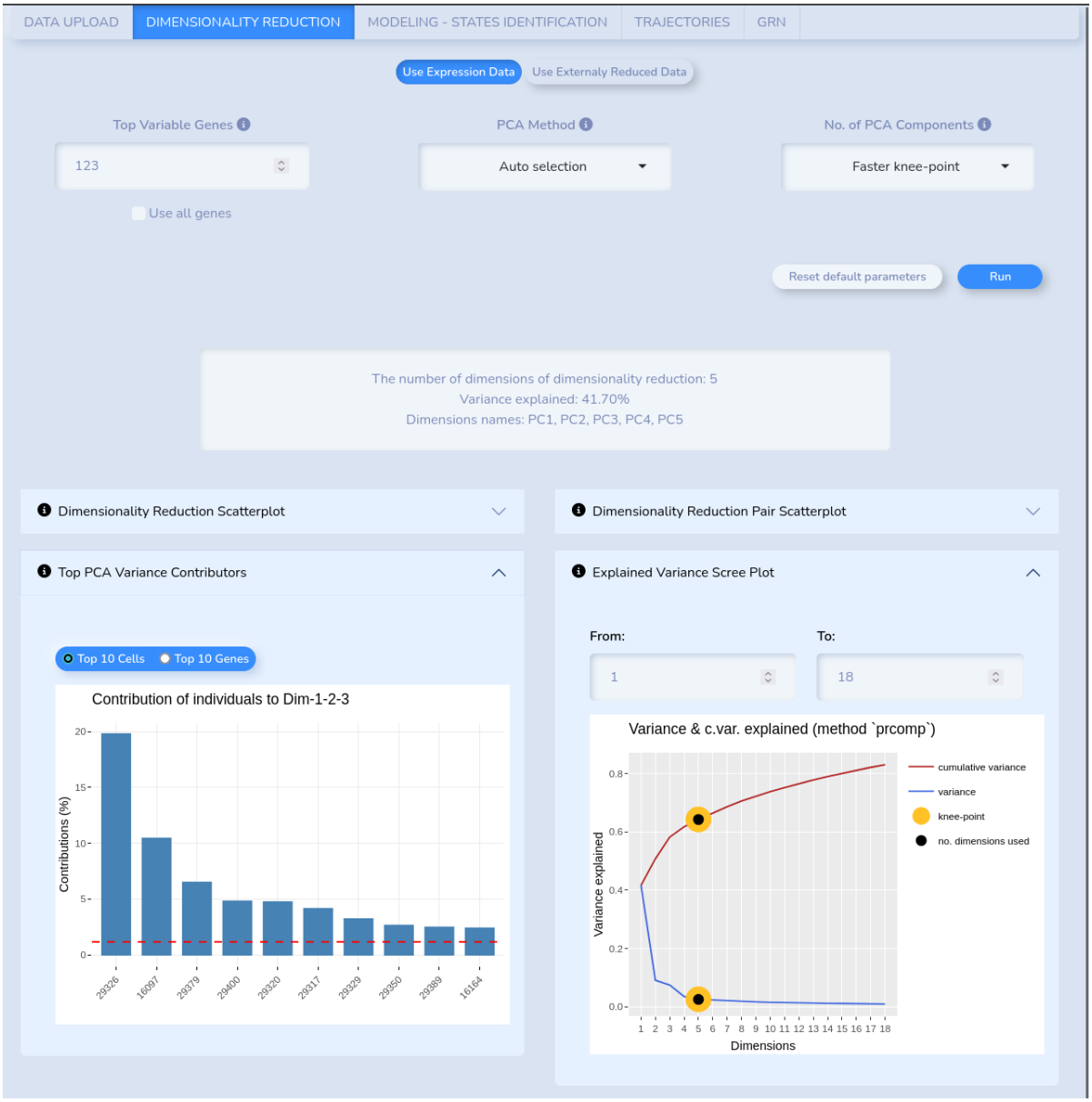

Figure 3.5: MLscANApp example showcasing tab appearance after the user runs an analysis step. The app includes plots within accordions, which can be collapsed or expanded for easier navigation. Some plots require additional user-provided parameters to modify the output. An ”info box” is also displayed, summarizing the results of each step, providing users with a quick overview of the analysis.

3.5 Data Visualization

One of the main improvements in the MLscANApp, compared to the original MLscAN package, is the way data is visualized. While the MLscAN package provides static visualizations, most of the plots in the MLscANApp are interactive, allowing users to zoom in and out or hover over specific points to reveal additional details, such as gene names or expression values. These interactive features provide a more flexible data exploration, allowing users to inspect results in greater depth without manually modifying plot parameters.

Additionally, the plots are organized within specific tabs corresponding to each analysis step in the pipeline. This ensures that the visualizations are presented in the relevant context, helping users stay aligned with the workflow.

The app allows users to generate different versions of certain plots based on their needs. In some cases, complex plots are split into multiple visualizations for clarity, or radio buttons provide options for users to select specific plot versions. These options trigger different functions or modify parameters in the back-end to tailor the output to the specific analysis step, without overwhelming the user with unnecessary complexity.

The corresponding functions of the MLscAN package are called to generate all plots displayed after each analysis step. These visualizations use the parameters provided by the user through the app's interface or, in some cases, predefined parameters set by the developer to ensure optimal presentation of results.

In the next chapter, we will explore specific workflows that demonstrate how the app can be used to perform all MLscAN-supported scRNAseq data analyses.

4. MLSCANAPP EXAMPLE WORKFLOWS

4.1 Overview of the Application

The app's structure mirrors the steps of the MLscAN pipeline [17, 16], with each tab in the navigation bar corresponding to a specific step of the pipeline. Users can run each step in isolation through the respective tab, but all steps should be followed in the recommended order for the complete data analysis.

Users can freely go back and rerun any of the previous steps, and they can stop the analysis at any point. However, they cannot skip steps;. For example, they must complete DATA UPLOAD before moving on to DIMENSIONALITY REDUCTION and later steps. Moreover, they cannot skip DIMENSIONALITY REDUCTION and go directly to the MODELLING or TRAJECTORIES steps of the MLscAN data analysis workflow.

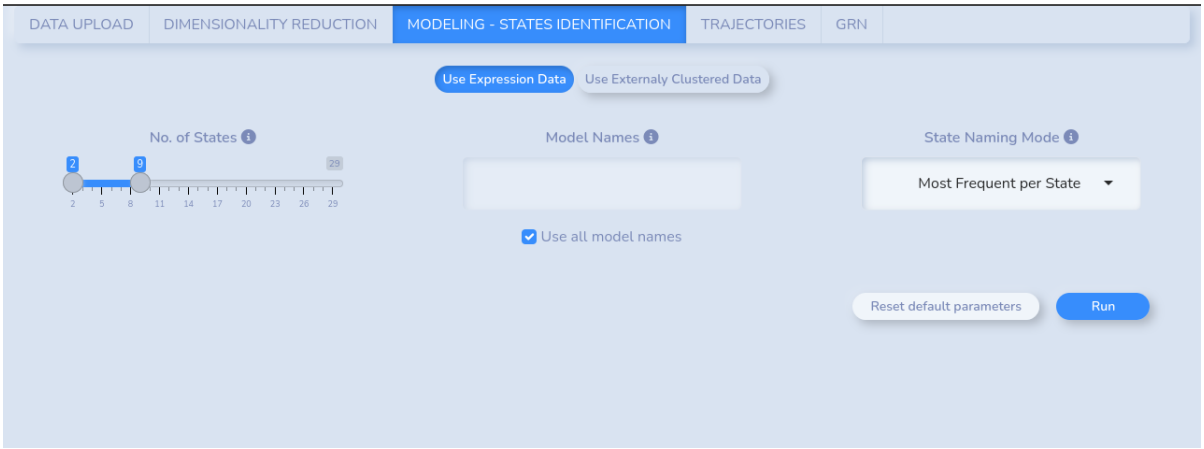

Figure 4.1: MLscANApp tab example illustrating the app's organization and parameters with pre-selected values.

Each tab displays the parameters required for that pipeline step. Default values are provided in placeholders; these values will be used during execution if not changed. However, users can modify default values through dropdown menus, text fields, sliders, and other input controls as needed. For more details about a specific parameter, users can hover over the info icon next to it, and this action will trigger the display of a brief description. (see Figure 4.2)

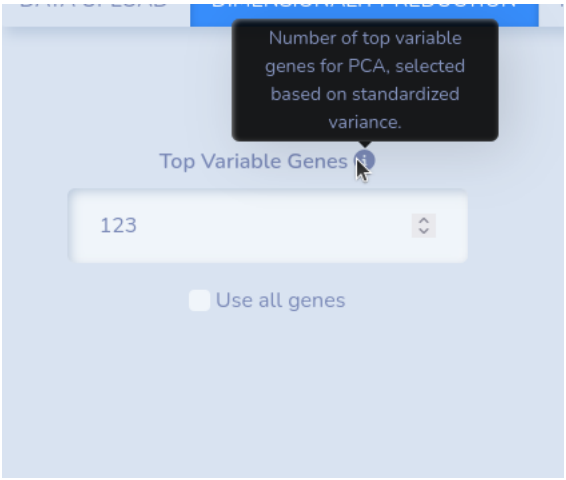

Figure 4.2: Hover feature example: The user can hover over the info icon to view a brief description of the parameter field.

Each step generates a variety of plots to visualize the results. These plots are contained within accordions, which remain collapsed by default. Users can expand the accordion to view a particular plot. The app also allows users to interact with the plots by zooming in/out or downloading them for further use. In Figure 4.3 we can see how the app's interface regarding plots looks like by default. Figure 4.4 shows how the plots show up after the user has expanded some of the accordions.

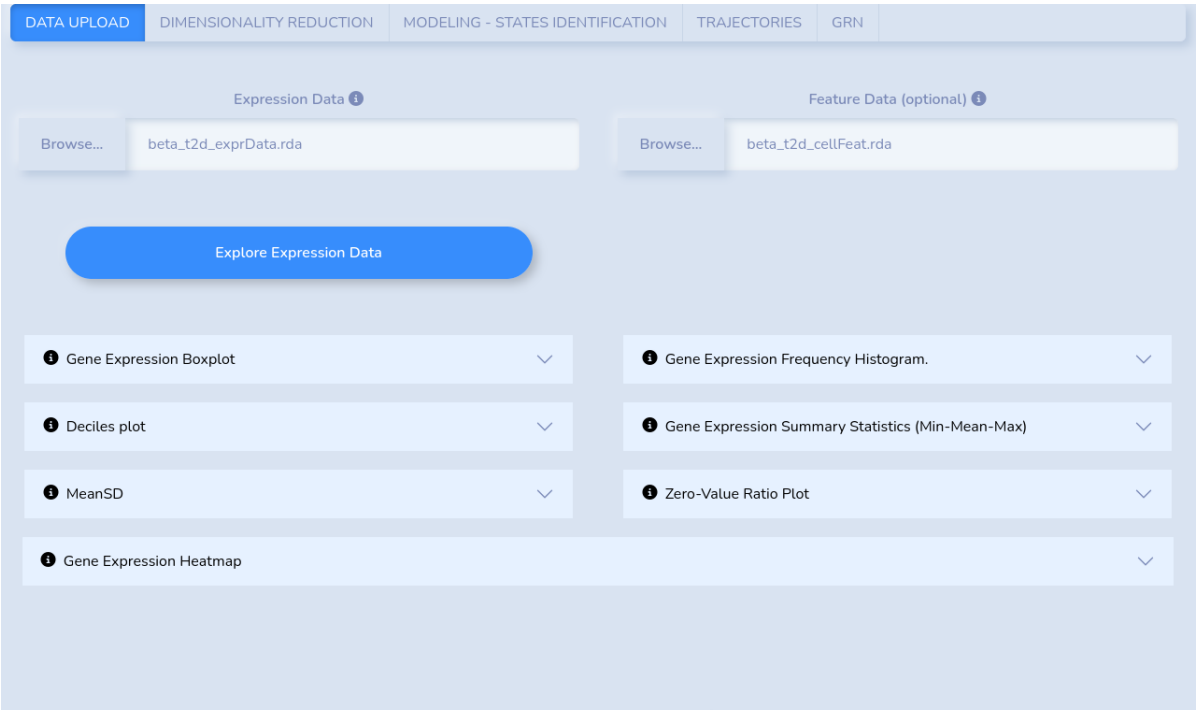

Figure 4.3: An MLscANApp example to showcase what the collapsed accordions look like.

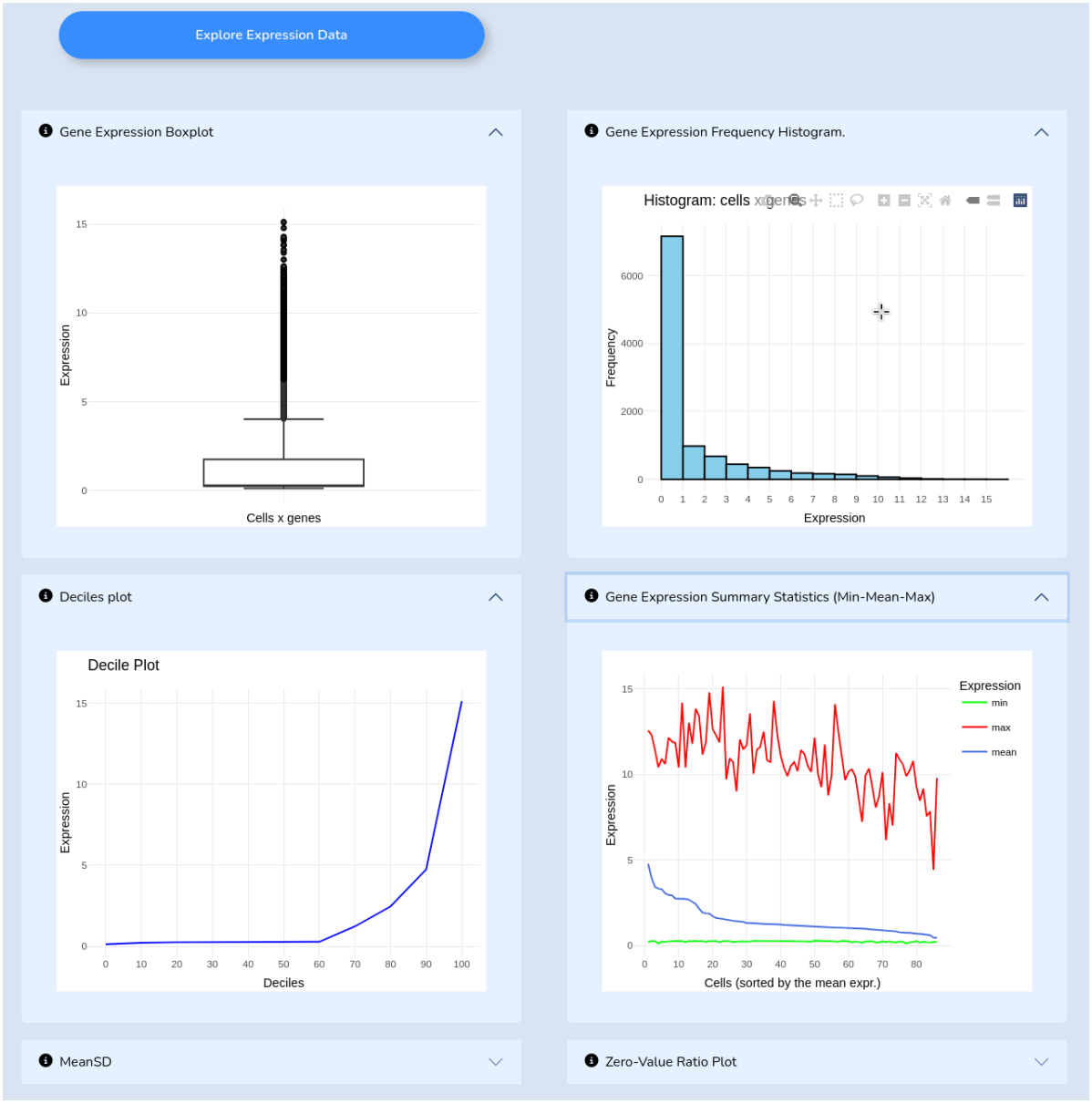

Figure 4.4: An MLscANApp example to showcase what the expanded accordions look like.

Additionally, there are ”boxes” that provide a concise summary of the results after each step, which we name info boxes. (see Figure 4.5).

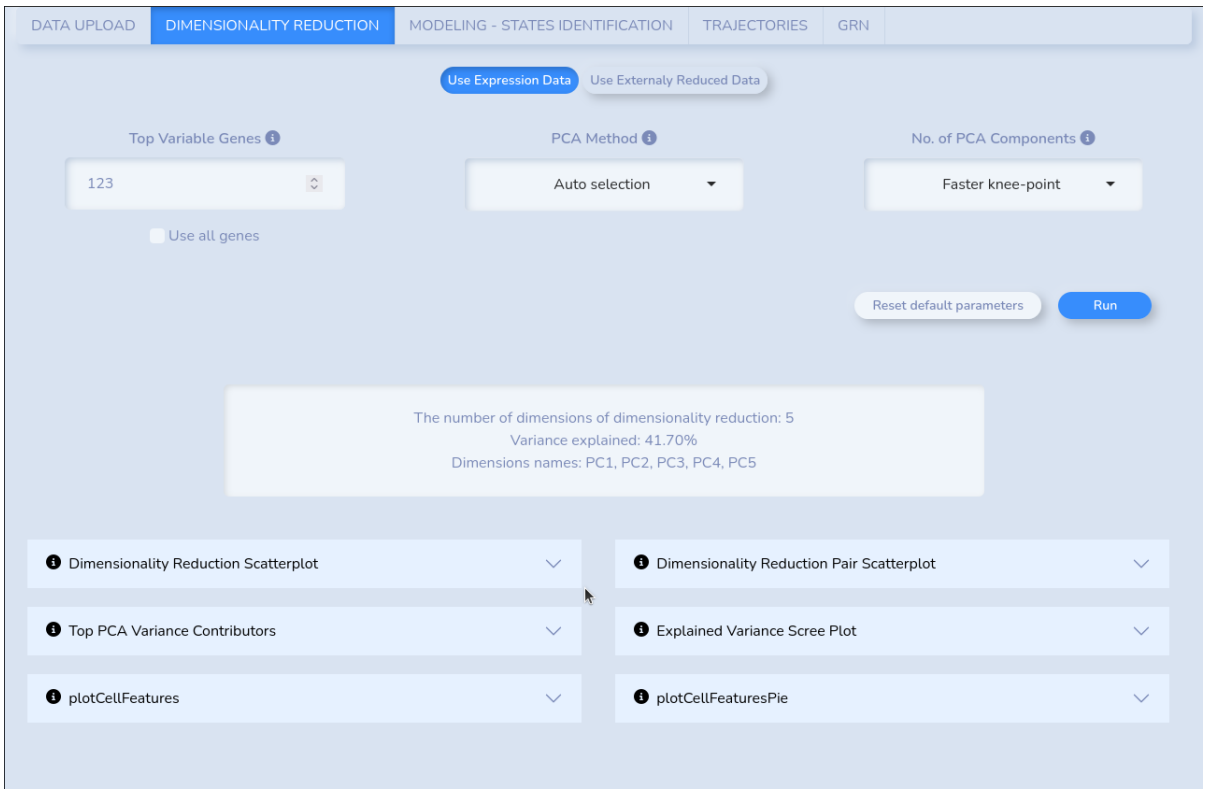

Figure 4.5: An MLscANApp execution displaying the info boxes. The light-colored rectangle beneath the buttons serves as an info box, summarizing the results for the specific analysis step.

Users can also manually intervene in the analysis by supplying their own results instead of the app-generated outputs. This is particularly applicable in the dimensionality reduction and modeling steps, where users can upload files containing their custom-reduced data or clustering results produced outside MLscANApp. Figure 4.6 illustrates the layout of the tab where the user can perform analysis with the provided data. There are two options under the navigation bar. The default selection is ”Use Expression Data”. If the user wishes to analyze their own data that has been reduced outside the app, they should select the alternative option, ’Use Externally Reduced Data,’ as shown in Figure 4.7. The same concept applies to the **MODELING** tab as well.

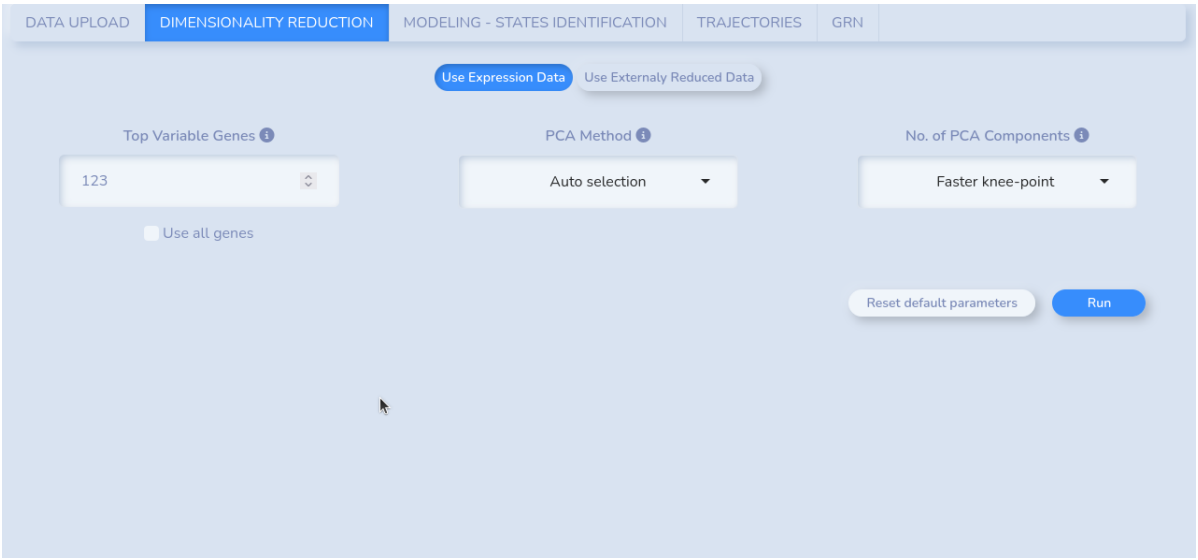

Figure 4.6: DIMENSIONALITY REDUCTION tab showcasing the interface for performing this step using the uploaded expression data. The 'Use Expression Data' radio button is selected by default.

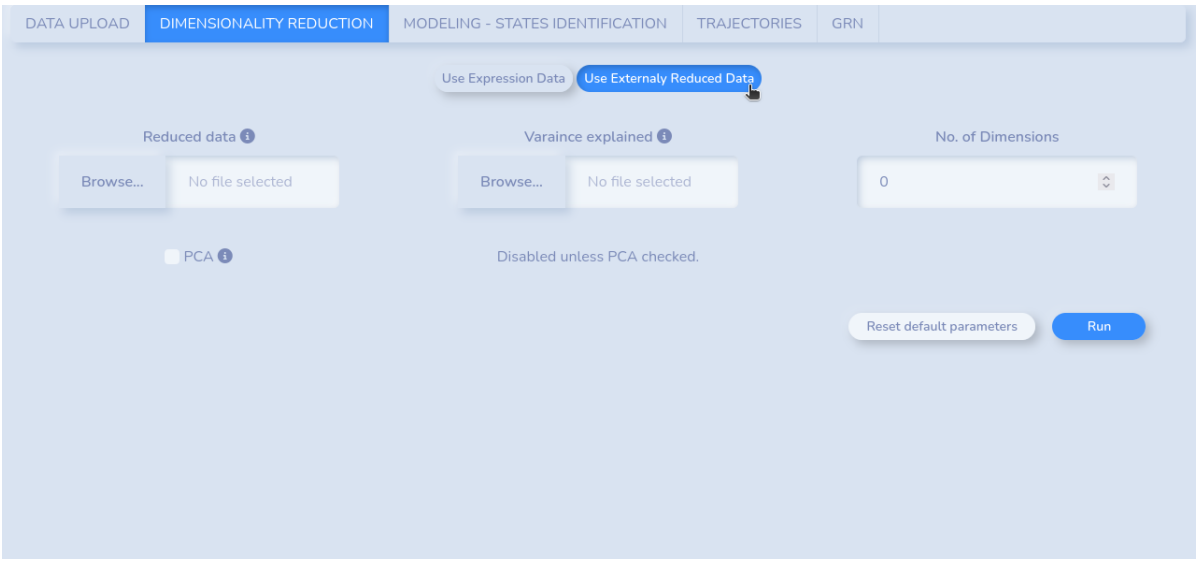

Figure 4.7: DIMENSIONALITY REDUCTION tab showcasing the interface for performing this step using the externally reduced data. The 'Use Externally Reduced Data' radio button is selected.

4.1.1 Instructions to Run the App

Currently, the app is unpublished and not hosted on any server. Therefore, users must download the code and run the app locally to host it on their own machine. To do so, they need to follow these steps:

1. Download the folder containing all the MLscANApp files.
2. Open the *MLscANApp.Rproj* in the Rstudio.
3. In the console:

```
golem::document_and_reload()
MLscANApp::run_app()
```

4.1.2 Basic MLscANApp workflow

This workflow describes a complete analysis using real data and how the user can utilize each interface tab to perform a specific analysis and inspect the results.

4.1.3 Dataset

For this example analysis, we utilized the Gene Expression Omnibus data under the accession GSE83139 In this study, the authors performed single-cell RNA sequencing on pancreatic islet cells, of types α, β, δ, PP, ductal, and acinar from diabetic and non-diabetic donors [26].

The MLscAN pipeline requires a pre-processed expression matrix as input. For details on the pre-processing steps, please refer to the "Data Acquisition and Pre-processing" MLscAN use case in [16]. We focused here on the β-cells, selecting them from three groups: type 2 diabetic adults, non-diabetic adults, and non-diabetic children. After applying gene filtering, the expression matrix contains 86 cells (two outlier cells were removed) and 123 genes. We have used this preprocessed matrix in our example.

The input for the MLscANApp must be a `.rda` file containing a cell-by-gene expression matrix. In this matrix:

- Rows represent individual cells, with the first element in each row being the cell identifier.
- Columns represent genes, with the first element in each column corresponding to the gene identifier.

An example:

```
          CLDN2     CXCL8    COL1A1     DUOX2    COL6A3
12763 0.2072646 1.2186902 2.2118990 0.2072646 1.9455168
16088 0.2648785 0.2648785 0.2648785 0.2648785 0.2648785
16096 0.2654744 0.2654744 0.2654744 0.2654744 0.2654744
16097 0.1228687 0.1228687 7.7795616 0.1228687 7.1065616
16099 0.2357508 0.2357508 0.2357508 0.2357508 0.2357508
```

4.1.4 Prior Information (Optional)

While the MLscANApp does not require any prior information to perform its unbiased data analysis, users have the option to upload such information, if available, to be used to enhance the quality of produced visuals. We should emphasize that prior information provided will not influence the data analysis but will be used exclusively to enrich the produced plots, offering a more detailed and informative representation of the data analysis

results. Prior information should be uploaded in the `Feature Data` field as an `.rda` file containing a cell-by-feature matrix. In this matrix:

- Rows represent individual cells, with the first element of each row being the cell identifier.
- Columns represent features, with the first element of each column being the feature name.

A valuable prior information for visualization purposes is the possibly available ground truth labels of each cell (cell type). This information should be provided in the cell features matrix under a column labeled `cellType`. Although this input will not influence the analysis, it will impact how the inferred cell states are named in the MLscANApp results (i.e., states will be named based on the most common cell type within that state), -enhancing the visualization and providing more context to the results.

An example such matrix:

```
     cellName cellType treatment
[1,] "12763"  "adult"  "control"
[2,] "16088"  "adult"  "control"
[3,] "16096"  "adult"  "control"
[4,] "16097"  "adult"  "control"
[5,] "16099"  "adult"  "control"
[6,] "16107"  "adult"  "control"
```

For our example, since the β-cells were sourced from three groups—diabetic adults, non-diabetic adults, and non-diabetic children—we assigned labels during the preprocessing of the expression matrix (as described in the ”Data Acquisition and Preprocessing” mentioned above). These labels identified each cell based on its origin: healthy adult (`adult` label), healthy child (`child` label), or adult with type-2 diabetes (`T2D` label). This created a cell features matrix containing additional information for each cell. The `cellType` feature in this matrix represents the ground truth that will guide our analysis and interpretation of the pipeline results.

Let’s walk through an example of the entire pipeline, following the app’s structure and examining each tab in detail.

4.1.5 DATA UPLOAD tab

In the first step, the user can explore the uploaded data through a series of corresponding plots.

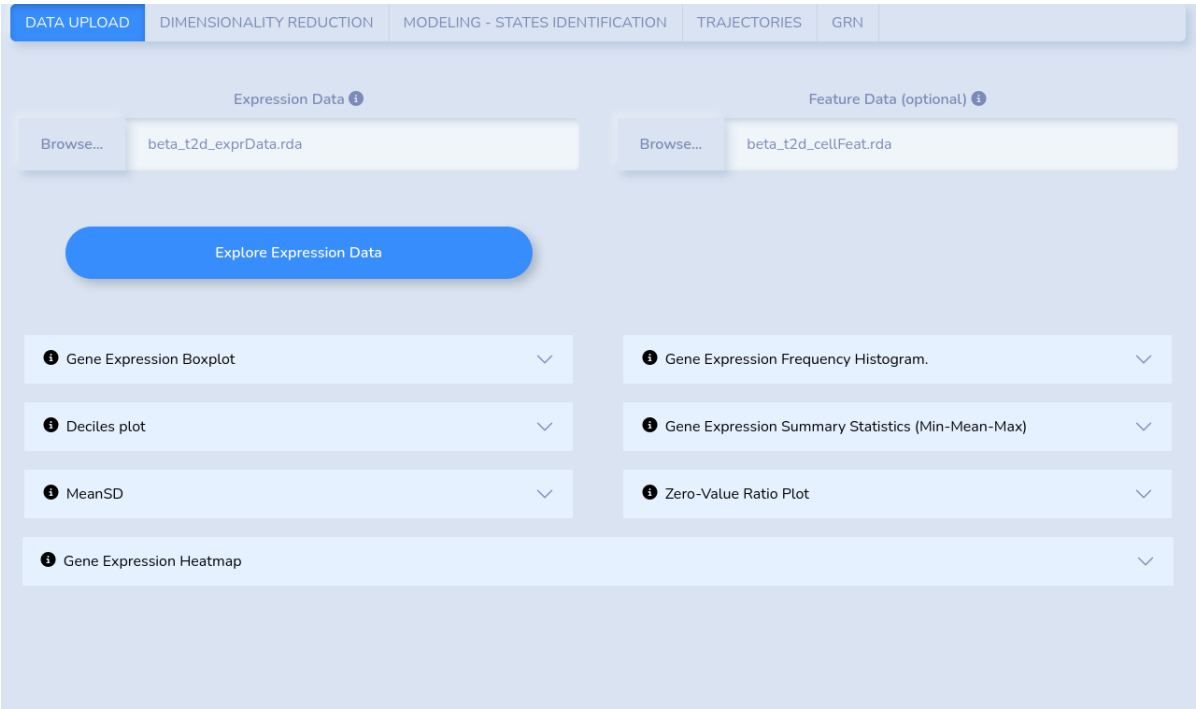

Figure 4.8: DATA UPLOAD tab after the ”Explore Expression Data” button is pressed

Figure 4.8, shows what the tab looks like after the user has uploaded their dataset and chosen to press the ”Explore Expression Data” to interpret their data. There are several plots the user can utilize for the exploration.

For example, Figure 4.9 shows a boxplot of gene expression across all selected β-cells. The horizontal x-axis represents the (cells x genes) data matrix, while the y-axis represents gene expression levels. The boxplot indicates that most data falls within a lower range of expression values, with the majority concentrated around a median close to zero. A large number of outliers (higher expression levels) are visible, indicating that there are specific genes with notably high expression in some cells. The long whiskers suggest that there is significant variability in the expression data, with several genes exhibiting high variance across different cells.

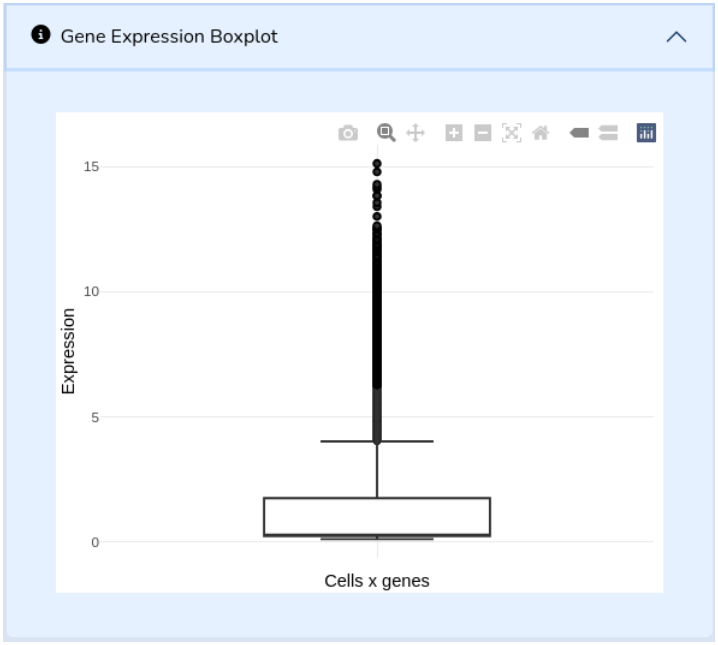

Figure 4.9: Gene Expression Boxplot illustrating gene expression levels across selected cells. The horizontal x-axis represents the cells x genes expression matrix, while the y-axis shows gene expression levels. The boxplot highlights data concentration around the median, variability in expression, and the presence of outliers indicating genes with higher expression.

Another useful plot the users can explore in this tab is the **Gene Expression Frequency Histogram** shown in Figure 4.10. The histogram shows that most gene expression values are very low, which is consistent with the sparse nature of single-cell RNA-seq data. The small number of higher expression values suggests the presence of key genes that may be driving cellular behavior or defining cell types.

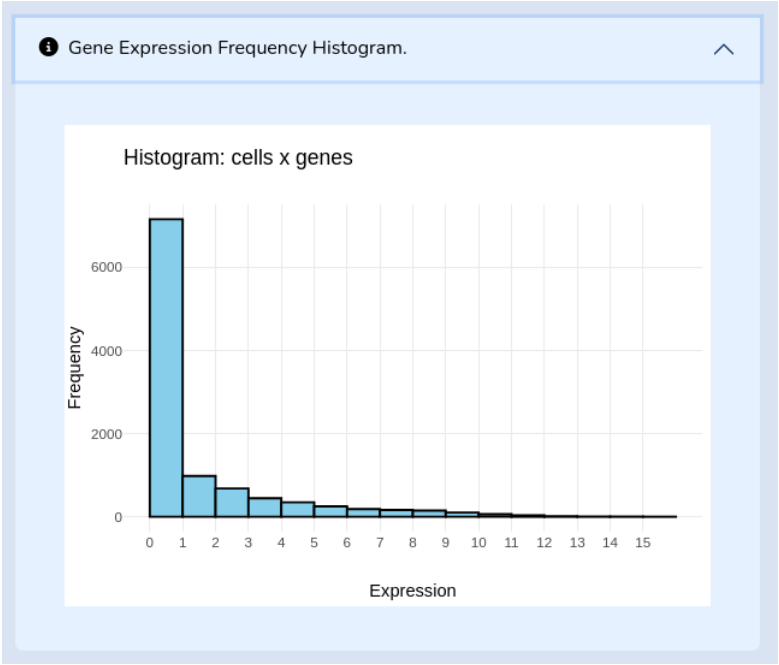

Figure 4.10: Gene Expression Frequency Histogram illustrating the distribution of gene expression values, highlighting the frequency of low and high expression levels within the dataset.

Figure 4.11 provides a comparison of the minimum, maximum, and mean gene expression levels for each cell across the genes in its profile (expression matrix row). The data here

reveals a heterogeneous population of cells with a subset displaying high gene activity. The variability in maximum expression suggests that certain cells have unique or active expression gene patterns, potentially marking them as different subtypes or functional states.

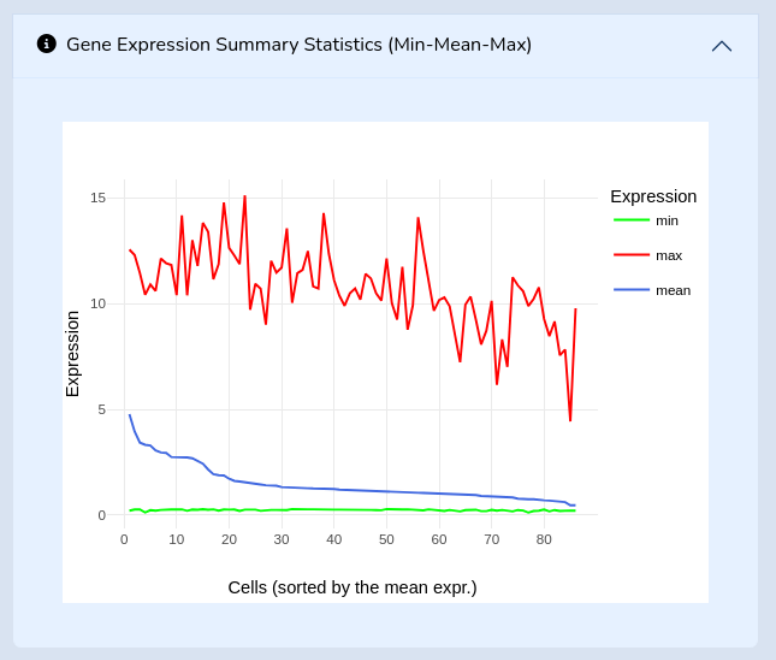

Figure 4.11: Gene Expression Summary Statistics comparing the minimum, maximum, and mean expression levels across cells, providing an overview of the variability in gene expression within the dataset

4.1.6 DIMENSIONALITY REDUCTION tab

The next tab corresponds to the dimensionality reduction step of the MLscAN package. Figure 4.12 gives an overview of what this tab looks like after running this step. In the parameters fields, we can see the selected parameters for running this step.

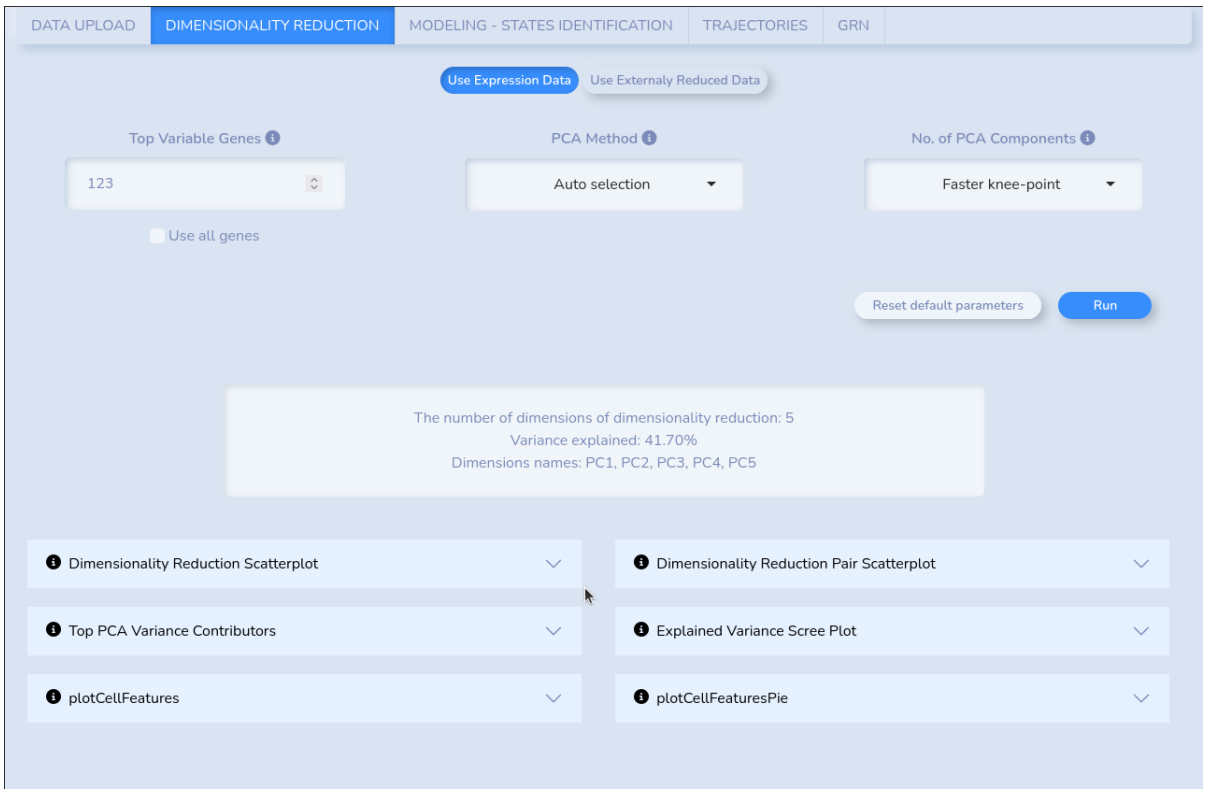

Figure 4.12: Overview of the DIMENSIONALITY REDUCTION tab after executing the step, showcasing the parameter fields with the selected settings used for the analysis.

By default, Principal Component Analysis (PCA) is applied to the provided dataset to reduce dimensionality. PCA is performed using either the `prcomp` a (built-in R function) or `irlba` (part of the `irlba` package [27]), where, by default, the appropriate function is automatically selected based on the size of the expression matrix. `irlba` is used when both the number of genes and cells exceeds 100, while `prcomp` is applied for smaller matrices, which is why auto-selection is pre-configured in the `PCA Function` field.

The dimensionality reduction step can return many components, (i.e. 20 PCA vectors) but MLscANApp recommends selecting a specific number of principal components (PCs) based on the knee-point method. The `No. of PCA Components` parameter ultimately determines how many PCs will be retained for the clustering step.

By default, the app aims to determine the optimal number of PCs by analyzing points on a plane, where each point represents the number of components considered (horizontal axis) and the variance explained by using n PCs compared to n-1 (vertical axis). The optimal point, known as the knee-point, is where the rate of explained variance starts to level off. The default method, `Faster knee-point`, identifies this knee-point as the point that maximizes the perpendicular distance to a line connecting the first and last points on the plane (representing the minimum and maximum number of components considered). For more details on alternative options for this parameter, refer to the MLscAN R package documentation.

A brief overview of the selected number of PCs, the variance explained, and the names of the principal components are provided in the info box. (see Figure 4.13).

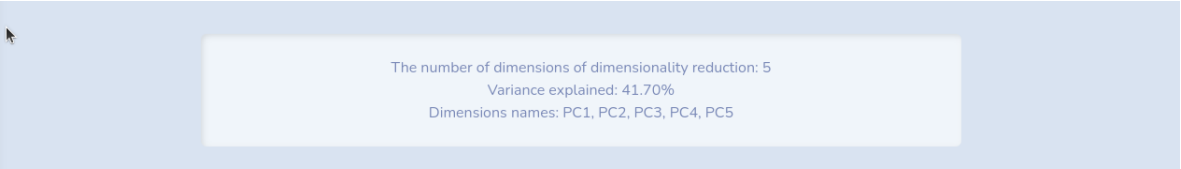

Figure 4.13: Information box in the DIMENSIONALITY REDUCTION tab displaying the number of principal components, their names, and the cumulative variance explained

To gain insights into the process, we can refer to the **Explained Variance Scree Plot**, which visualizes the variance explained by each PC component along with the cumulative variance explained. This plot also highlights the number of PCs MLscANApp suggests keeping based on the knee-point method. By analyzing this plot in Figure 4.14 we can observe that in our case the optimal number of components to retain is 5.

Figure 4.14: Explained Variance Scree Plot showing the variance explained by each principal component (PC) and the cumulative variance. The plot indicates the optimal number of PCs to retain, suggested by MLscANApp using the knee-point method. The fields ”From:” and ”To:” are used to set the limits for the Dimensions axis.

Figure 4.15 provides an overview of the dimensionality-reduced data. Using the cell types (ground truth) from the features matrix allows us to annotate each cell according to its actual type, coloring each type differently and visually assessing the quality of the dimensionality reduction operation. Although the prior information in the cell features matrix does not affect the analysis, it helps us observe that cells of the same type tend to be positioned close to each other. Additionally, this prior information highlights that PC1 seems to capture age-related variation by separating child cells from adult normal and T2D cells, while PC2 seems to distinguish between adult normal and adult T2D cells.

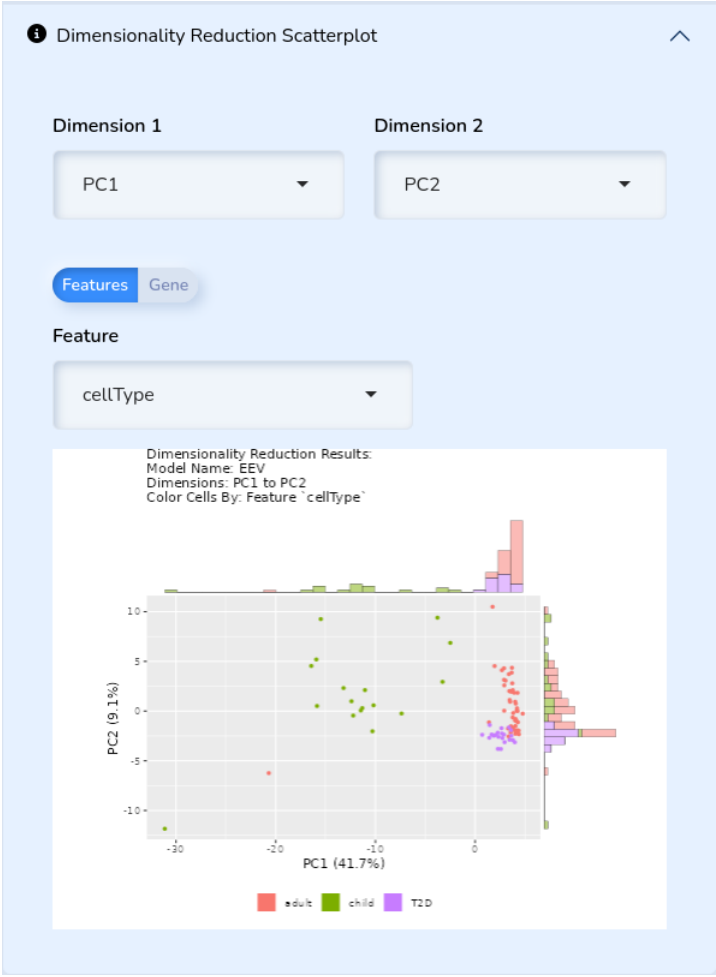

Figure 4.15: Dimensionality Reduction Plot. The user can specify which principal components (PCs) to plot using the Dimension 1 and Dimension 2 fields. Radio buttons allow the selection of cell coloring, either based on the provided features (specified in the field below the radio button) or on gene expression levels.

4.1.7 MODELING - STATES IDENTIFICATION tab

This tab includes clustering the data utilizing Gaussian Mixture Modeling (GMM) using the `EM algorithm` and selecting the best model (as implemented by the `mclust`[28] package in R).

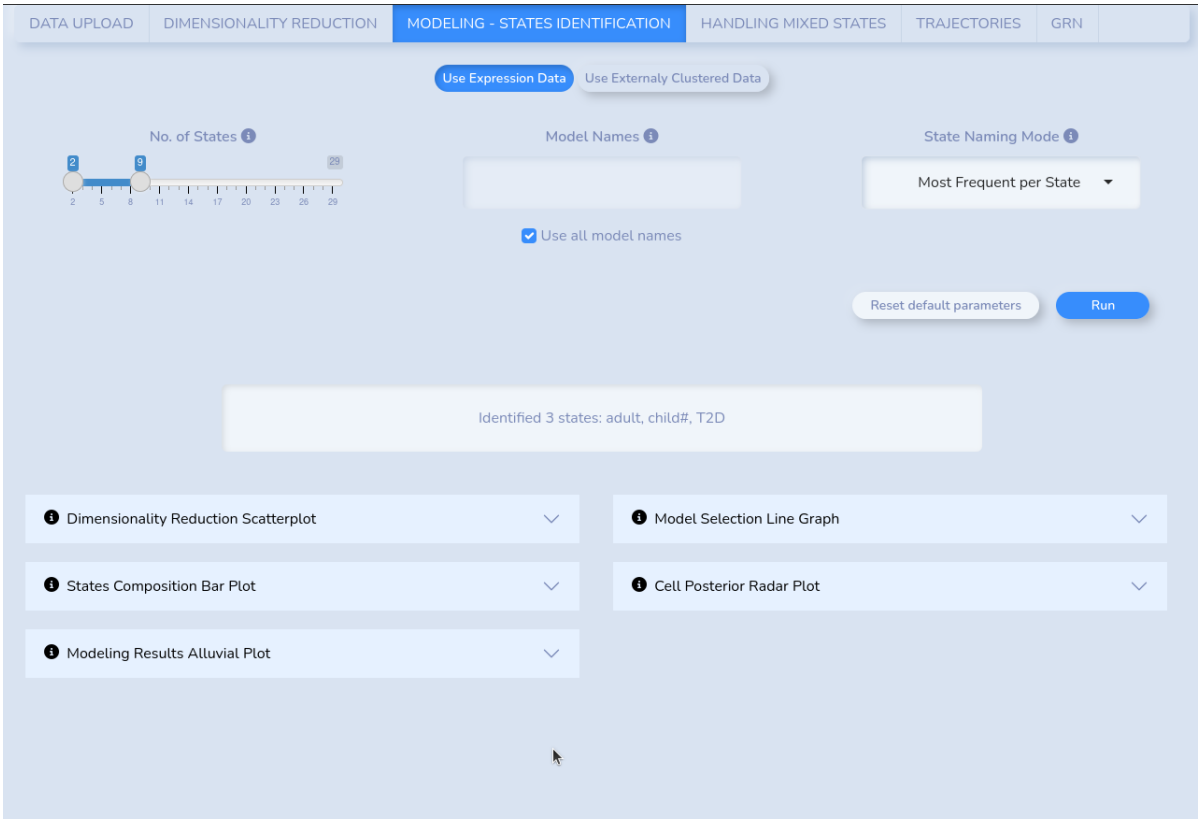

Figure 4.16: Overview of the MODELING - STATES IDENTIFICATION tab after executing the step, showcasing the parameter fields with the selected settings used for the analysis.

The number of states to use in GMM is chosen by evaluating the Bayesian Information Criterion (BIC). As the number of states increases, the difference in BIC between the best-performing model for each number of states and its neighboring models (those with one more or one fewer state) is computed. This is called ΔBIC. A threshold is used to determine when the change in BIC becomes negligible. Following the parsimonious modeling principle, the simplest model with ΔBIC values below this threshold for both neighboring models is selected as the ”best” model.

By default, MLscANApp evaluates all possible types of models (with different covariance matrix structures), which is why the `Use all model names` option is checked. The range for the possible number of states in the Gaussian Mixture Model (GMM) is provided as a parameter, with the default range set to $[2:9]$. Since feature data has been provided, the (`Most Frequent per State`) option is preselected for naming the inferred states. This means that each inferred state will be named after the cell type that is most prevalent within that state according to the ground truth information provided.

The candidate models and their BIC values are visualized in the **Model Selection Line Graph**, which provides a comparison of all GMM model types (based on different covariance matrix structures) and the number of states considered.

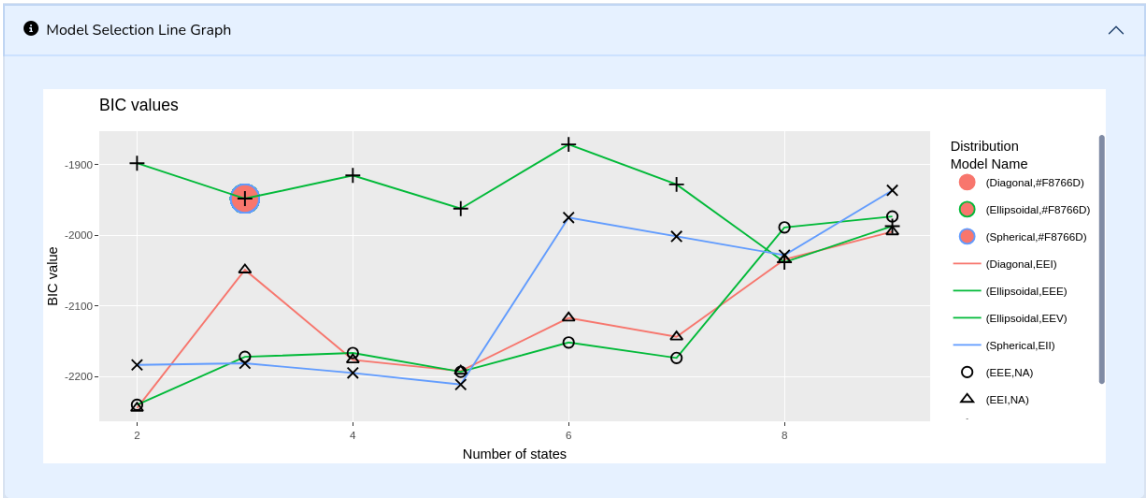

Figure 4.17: Model Selection Line Graph comparing candidate models and their BIC values for various GMM model types, based on different covariance matrix structures and the number of states considered, also highlighting the selected model.

In this case, the optimal model identified was the GMM with `EEV` structure, which assumes an ellipsoidal distribution with equal shape and volume but variable orientation. The model with only three states was found to be the most appropriate (parsimonious). The GMM with three components (states) was selected based on ΔBIC criterion applied by MLscAN. Even though the plot in Figure 4.17 shows that there is a model with a higher BIC value and six states, the use of ΔBIC (described previously) as a parsimonious model selection criterion led to MLscAN choosing the simpler 3-state model. This model has a lower complexity, and the ΔBIC values suggest that adding more states does not provide a significant improvement in model fit, while it increases significantly the risk of overfitting.

Now that we have identified the clusters (cell states) inferred by MLscAN, we can revisit the **Dimensionality Reduction Plot** Figure 4.18. The ellipses represent the covariance structure of the inferred states (GNN components), and as we can see, the cell types (ground truth) are well-aligned with the inferred states, indicating a strong correspondence between the actual cell types and the unsupervised clustering results.

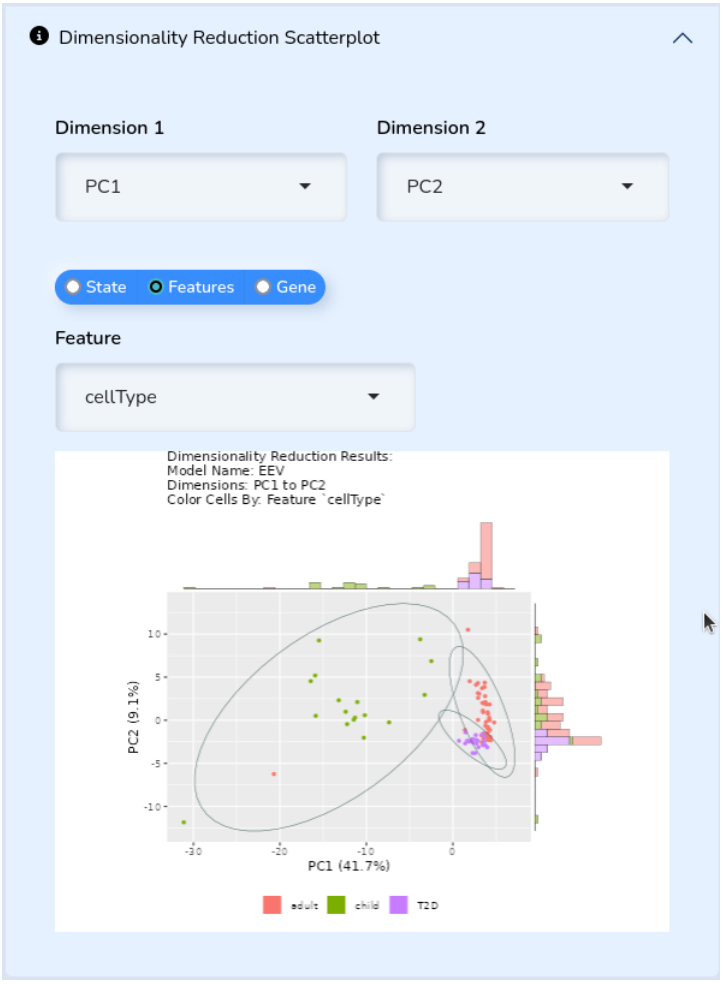

Figure 4.18: Dimensionality Reduction Plot after modeling. The ellipses represent the inferred cell states. The cells are colored according to their respective cell type. The user can select which principal components (PCs) to plot using the Dimension 1 and Dimension 2 fields. Radio buttons allow cell coloring based on the provided features (specified in the field ”Feature” below the radio buttons), gene expression levels, or inferred states.

By analyzing the **States Composition Bar Plot** Figure 4.19 we see that the unsupervised clustering results are faithful to the ground truth since the inferred states are “pure,” consisting primarily of cells of the same type.

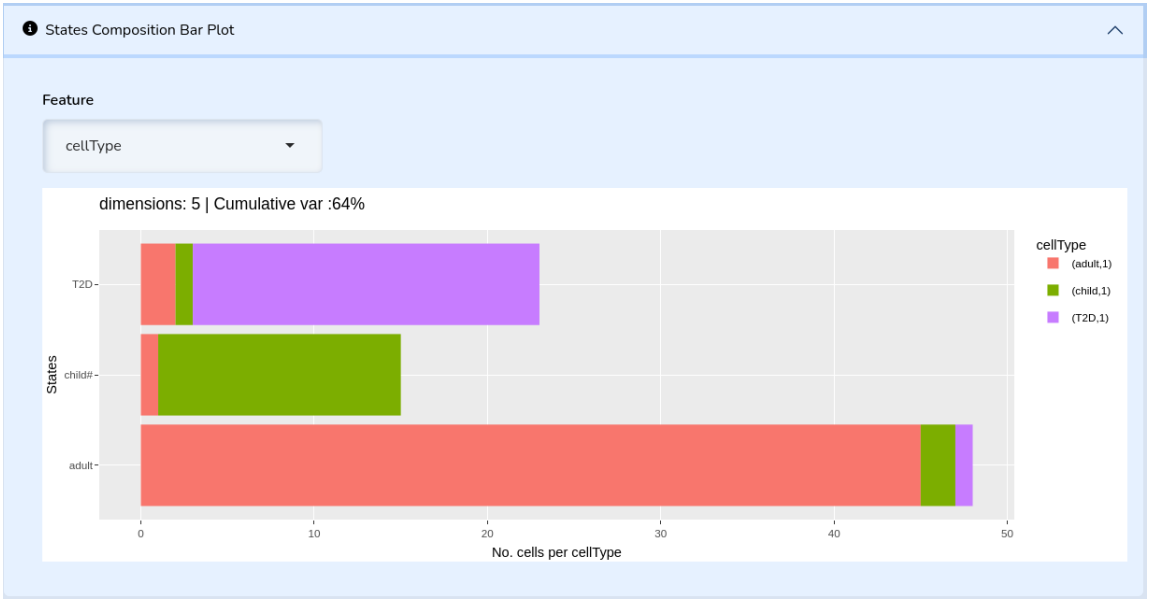

Figure 4.19: States Composition Bar Plot illustrating the composition of each state. Each color represents a cell type. The 'Feature' field allows the user to select which provided features will be used to color the bars. We observe that inferred states have high purity, each one mainly representing one cell type.

4.1.7.1 Mixed-States

An MLscAN innovation is that it can identify and analyze "mixed -states". Mixed states are sets of cells with an unusually large expression variance relative to other inferred states. Their existence may indicate the presence of outlier cells or the existence in a state of interesting cell sub-populations that warrant more attention. A new tab has appeared in the navigation bar called **HANDLING MIXED-STATES,** which serves to execute different actions the user can perform over the mixed-states. This is a dynamic tab that appears only if mixed -states are identified after the modeling. In our use case, a mixed state exists (with its name ending with a hash, namely child#), as apparent from its large variance (large green ellipse) in Figure 4.18. A detailed discussion on how MLscAN and the app handle such states is provided later in section section 4.2. MLscAN favors parsimonious modeling to reduce the risk of overfitting, but if it feels that it underfits the data (as may be the case when mixed states appear), it can selectively focus and analyze these states further and possibly recursively decompose them.

4.1.8 TRAJECTORIES tab

To infer trajectories between pairs of states, MLscAN first identifies potential transitions between the states. However, only identified transitions with solid support by the data are considered valid for trajectory construction. Moreover, the cells participating in each trajectory are partitioned into three consecutive subsets (in pseudotime), referred to as "micro-states". Micro-states inference for trajectories and data analysis at the micro-state level is another MLscAN innovation. But let's take things one at a time and provide a more detailed explanation of these concepts below.

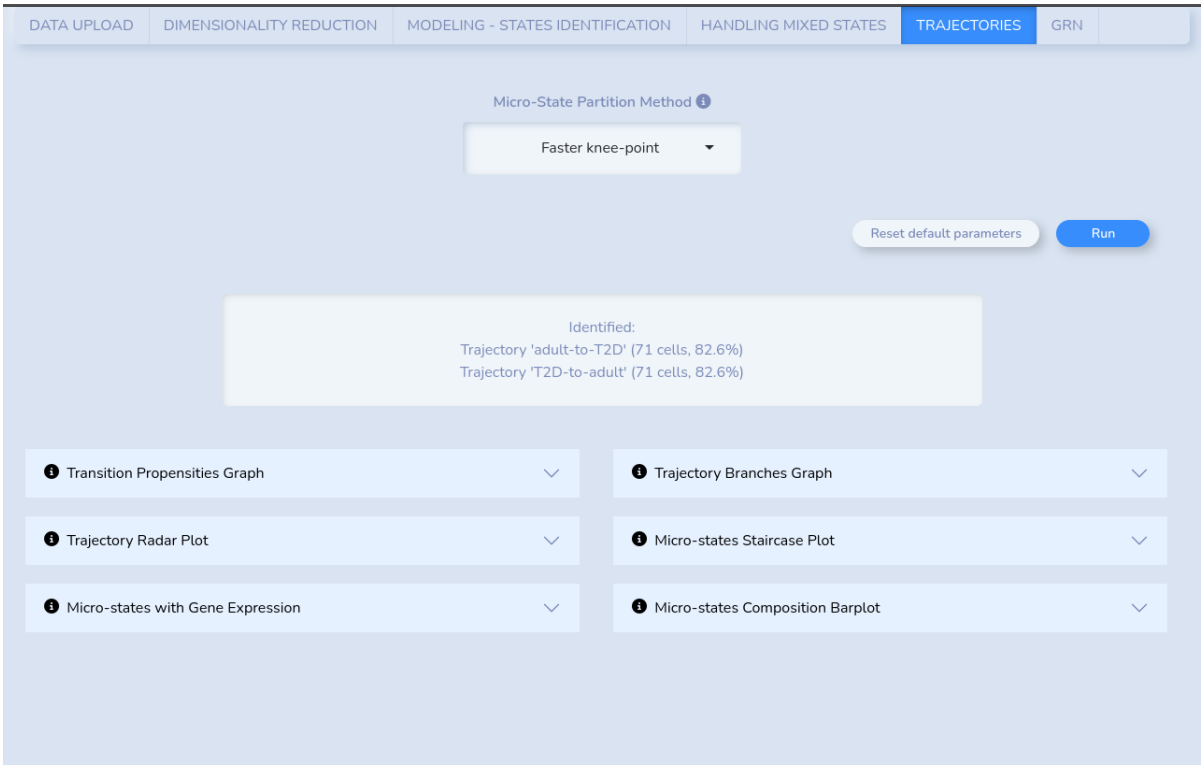

Figure 4.20: Overview of the TRAJECTORIES tab after executing the step, showcasing the parameter fields with the selected settings used for the analysis.

4.1.8.1 Transitions

Using MLscAN, after GMM modeling, each cell has a posterior distribution to all inferred states. The cells with their two largest posterior probability values to a specific pair of states, e.g., A and B, form a set. After sorting these cells in descending posterior order to state A we obtain the A-to-B transition, which is actually a list of ordered cells. We can think of this sorted cells' list as a model of the biological progression with cells organized along a path from the departing (ground) state (with the highest posterior) to the arriving (landing) state (with the second highest posterior) in pseudotime. This directed path is called a (state-to-state) transition in MLscAN. Similarly, we can construct a B-to-A transition by sorting the cells in descending posterior B order.

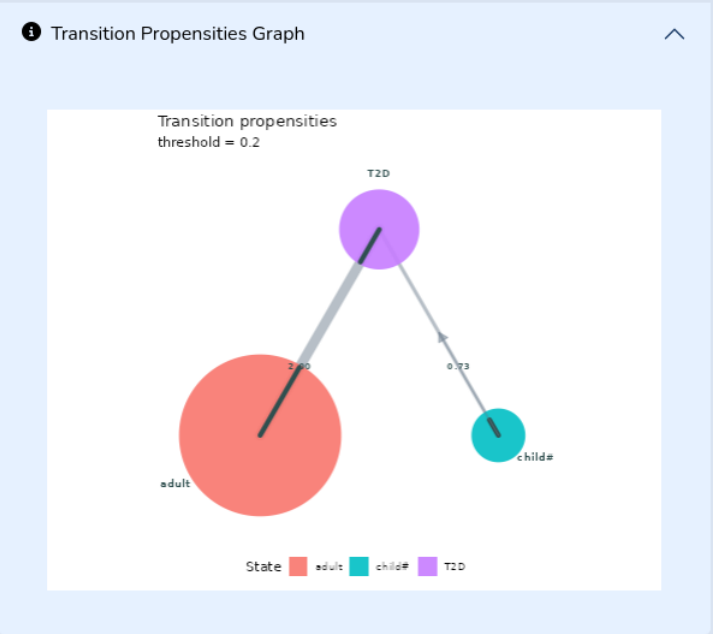

Figure 4.21: Transition Propensities Graph. Plot illustrating states as circles, with sizes corresponding to the number of cells assigned to each state. Transitions between states are represented by gray lines, with line thickness indicating the strength of the transition. Dark segments on the circle edges reflect the proportion of cells with a second-highest probability assigned to another state.

In Figure 4.21, each inferred state is shown as a circle, with its size reflecting the number of cells assigned to it, meaning cells whose highest posterior probability indicates membership (ground) to that state. Transition pairs between states are depicted by gray lines (edges) connecting two circle centers. This indicates that a subset of cells have their two largest posterior probabilities assigned to those two connected states. The weight of the edge (and the thickness of the line) indicates the percentage of cells in the two states that favor this interaction. The interaction ”propensity” is calculated by adding the percentages of cells from each state involved in their interaction. For example, if 95% of the cells in state A have their second-highest posterior probability in state B, and 55% of state B’s cells have their second-highest probability in state A, the interaction propensity value would be 1.5 (i.e., 0.85 + 0.55). Based on this logic, the maximum possible propensity value (strength of interaction) is apparently 2.

Note that the line connecting the centers of two circles has dark black segments emanating from the two circle centers. The length of each segment is proportional to the percentage of cells whose second-highest posterior probability is assigned to the other state in the pair. For instance, if 50% of state A’s cells have their second-highest posterior probability in state B, half the radius length of state A’s circle will be shown as dark black.

In our example, most adult state cells have their second-highest posterior probability pointing to state T2D, and vice versa. As a result, the blackened portions of the radii of both circles nearly reach all the way to the circles’ circumference, indicating visually that the propensity value of this interaction is close to 2.

Moreover, the Figure shows a one-sided transition from the child# state to the T2D state, with a propensity of 0.73. This is so because 73% of child# state cells have their second-highest probability in state T2D, while no T2D cells have a second–highest probability in the child state. Interestingly, in this scenario, MLscANApp did not detect a direct transition from the child to the normal adult state. Instead, the evolutionary pathway progresses

through the T2D state, which aligns with a hypothesis suggesting that T2D represents a ”remembered” state passed through during cellular development from the child to the adult state. This supports the theory that a dedifferentiation process could be triggered later in life, reactivating this remembered path and explaining the transition to state T2D [26].

4.1.8.2 Trajectories

An A-to-B state transition is considered a “trajectory branch” (to be called “trajectory” for brevity) in MLscAN if there is enough support for it in the data after modeling. That means it contains at least 6 cells, where 3 of them belong to state A and the other 3 to state B.

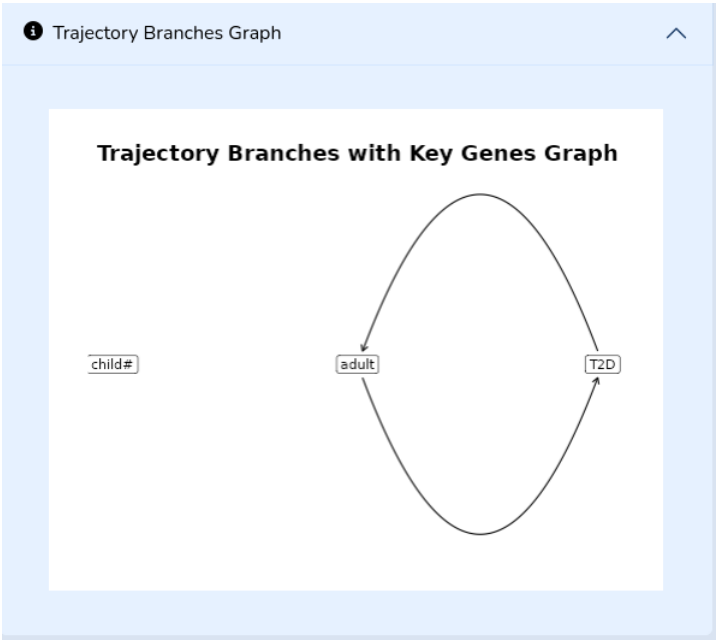

Figure 4.22: Trajectory Branches Graph. Plot displaying the subset of transitions identified as valid trajectories by MLscANApp. Arrows indicate the directionality of transitions, highlighting which transitions form valid trajectories.

Figure 4.22 shows the subset of transitions, shown on Figure 4.21 that MLscANApp has identified as valid trajectories.

4.1.8.3 Micro-States

To establish a view of biological progression along the A-to-B state trajectory branch MLscAN partitions this cell list into three consecutive sub-lists, called ground, transitory, and landing, and referred to here as “micro-states”. The micro-states in MLscAN recapitulate the consecutive phases of the biological progression in pseudotime. The assignment of each cell to a micro-state is determined by its posteriors to the two states defining the trajectory branch. For instance, a cell is assigned to the ”ground” micro-state if its posterior for the originating (or ”ground”) state is significantly higher than its posterior for the destination (or ”land”) state. Conversely, if the posterior probability favors the destination state, the cell is placed in the ”land” micro-state. Additionally, some cells may be assigned to an intermediate ”transitory” (or ”trans”) micro-state, where the posterior probabilities for both states are relatively similar in value. So MLscAN has a probabilistic view of a trajectory as

a “staircase” in posterior probability space, where each cell in the modeled biological process is placed at a certain “step” of this staircase. The higher the cell’s posterior probability to the departing state (ground), the higher the cell’s step in that staircase of probabilities.

We can use the corresponding plot to observe the partition of cells into the consecutive micro-states of a trajectory branch. As shown in Figure 4.22 in our example, there are only two valid trajectory branches: `adult-to-T2D` and `T2D-to-adult`. Figure 4.23 shows the number of cells assigned to each micro-state in these branches. Interestingly, the boundaries between micro-states can differ between the two branches. For instance, in the plot, we notice that the middle transitory micro-state (green) contains more cells in the `adult-to-T2D` trajectory than in the reverse `T2D-to-adult` trajectory. While both branches have a transitory micro-state, this is not guaranteed for all trajectories.

Figure 4.23: Micro-States Composition Barplot. Plot illustrating the partition of cells into consecutive micro-states along trajectory branches. It shows the number of cells assigned to each micro-state in the specified branches, highlighting potential differences in boundaries between them.

In Figure 4.24, the cells are arranged in decreasing order of their highest posterior probability for the first (or ”departing”/ground) state `adult`, represented by the red curve, from left to right. MLscANApp applies an algorithm to establish thresholds for posterior probabilities to partition the trajectory into three sequential micro-states: ground, transitory, and landing (from left to right). These micro-state regions are visually represented on the plot using color shading. The plot shows that as cells transition from the `adult` ground micro-state to the `T2D` landing micro-state, their posterior probability for the `adult` state monotonically decreases (red curve) while their posterior probability for the `T2D` state generally increases (purple curve). At some point as the cells are approaching the landing state their highest posterior becomes the one to the `T2D` state.

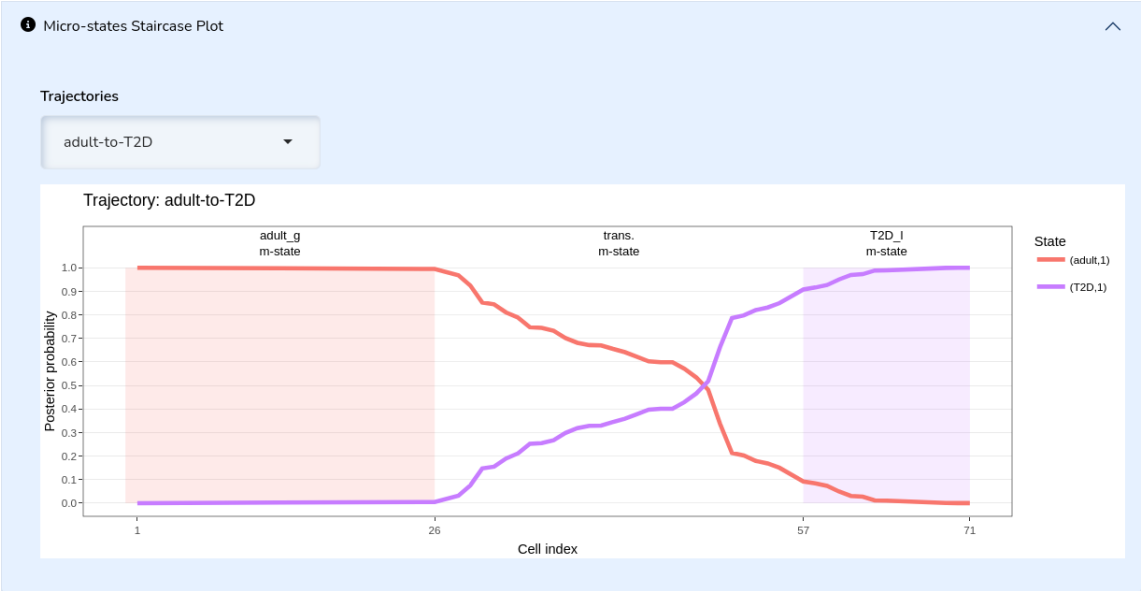

Figure 4.24: Micro-States Staircase Plot displaying the three micro-states—ground, transitory, and landing—marked by shaded regions. As cells transition from the ground to the landing state, the probability for the `adult` state decreases (red curve), while the probability for the `T2D` state (purple curve) increases, becoming dominant in the landing state.

The circular plot in Figure 4.25 provides an alternative visual representation of the cells organized of a trajectory branch in posterior probability space. Each cell (going from the top counter-clockwise) is depicted as a node on the circle, with its color corresponding to its state (highest posterior probability) and radius colored based on its transition state (second highest posterior probability). The length of the radius corresponds to the highest posterior probability value for the cell. As we move counterclockwise along the circle, the highest posterior probability (radius length) decreases. Black radius markers (protruding line segments) mark the micro-state boundaries. The first black marker is the boundary between the ground and the transitory micro-state, while the second is the boundary between the transitory and landing micro-states. We observe that as we enter the transitory micro-state there are adult state cells whose first posterior is rapidly dropping (red nodes). Moreover, beyond a certain point, the trajectory has cells of the T2D state (purple nodes) with a dominant posterior to the T2D state (landing) and a secondary posterior to the adult state (denoted by the red radii).

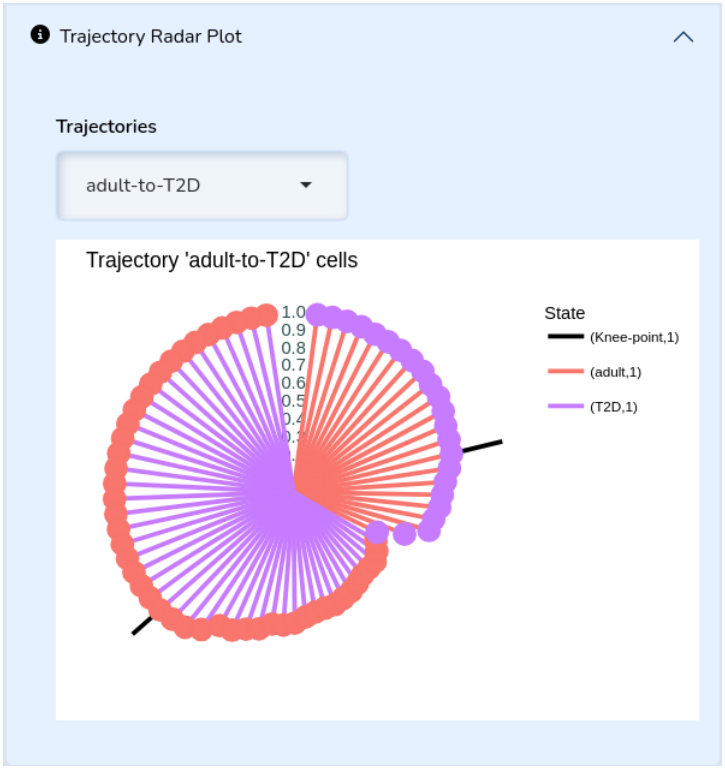

Figure 4.25: Trajectory Radial Plot visualizing cells along a trajectory branch (from the top going counterclockwise) in posterior probability space. Nodes represent individual cells colored by their highest posterior probability state. The radius length indicates the highest posterior value and its color is the transition state (landing state). Black markers identify the micro-state boundaries.

4.1.9 Gene Regulatory Networks (GRN) tab

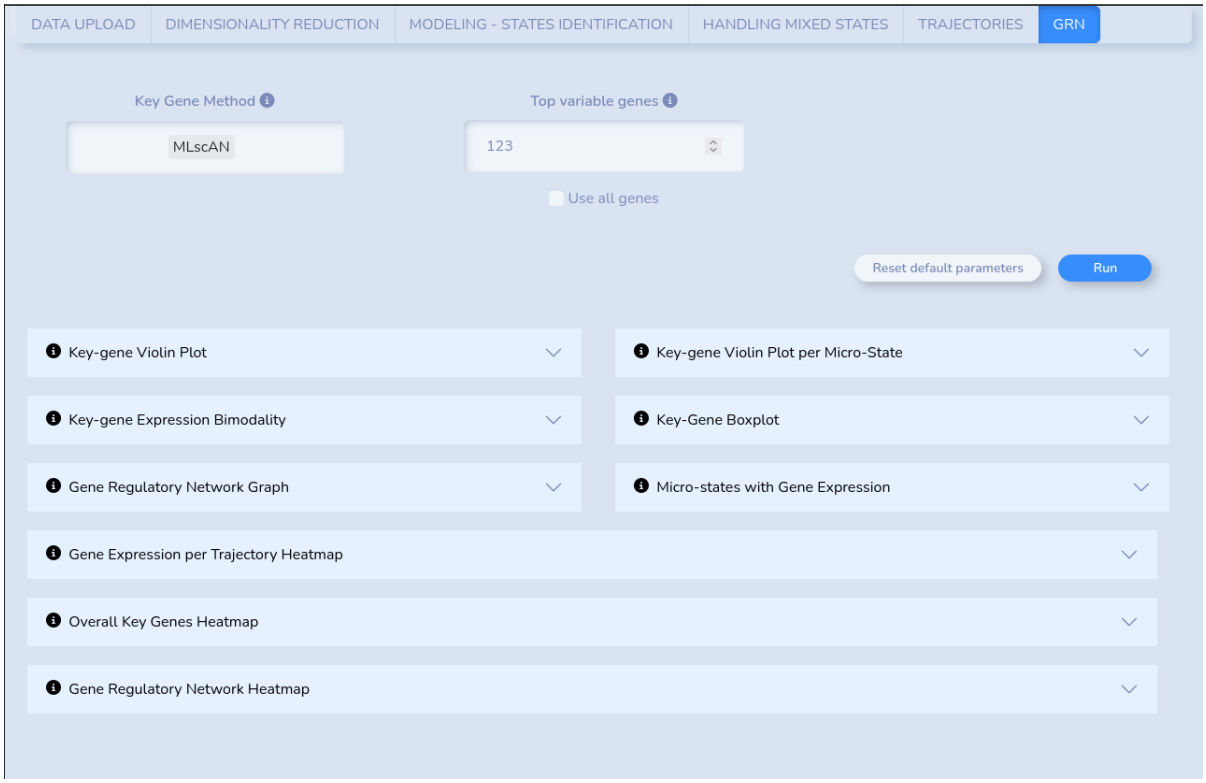

Figure 4.26: Overview of the GRN tab after executing the step, showcasing the parameter fields with the selected settings used for the analysis.

In this step, MLScAN reconstructs a Gene Regulatory Network (GRN for each micro-state of every selected trajectory branch inferred. The process of reconstructing GRNs begins with identifying genes governing the state-to-state transition biological process modeled by a trajectory. We refer to them as the ”key-genes” of the trajectory. The parameters in this tab relate to the procedure used to identify these key-genes, as explained below.

4.1.9.1 Key-Genes

Key-genes are genes that exhibit “interesting” behavior along the cells of a trajectory and can be considered “key-players” for the corresponding state-to-state transition. By default, MLscANApp utilizes the method described in the MLscAN publication [17] to determine whether a gene qualifies as a key-gene for a given trajectory. Intuitively, a gene is considered a key-gene if it exhibits bimodal expression and shifts its expression mode (from high to low or vice versa) as cells transition along the trajectory micro-states. The default method for key-gene identification, named `MLscAN`, is pre-selected in the `Key Gene Method`. Identification of key-genes limits the analysis to the most variable genes, as analysis can be computationally very demanding for large datasets. By default, this is set to 1000 genes, but the maximum number of genes is used if fewer than 1000 genes are available. In our example, we use all the available genes, which are 123.

Users can focus on any specific trajectory to gain deeper insights into the identified key-genes. This can be done using the plot in Figure 4.27, which generates a violin plot per key-gene, showing whether the gene shows high or low expression in each micro-state (ground and landing). This allows us to confirm visually the bimodal expression pattern

and the mode-switching behavior of each key gene along the microstates of a trajectory branch.

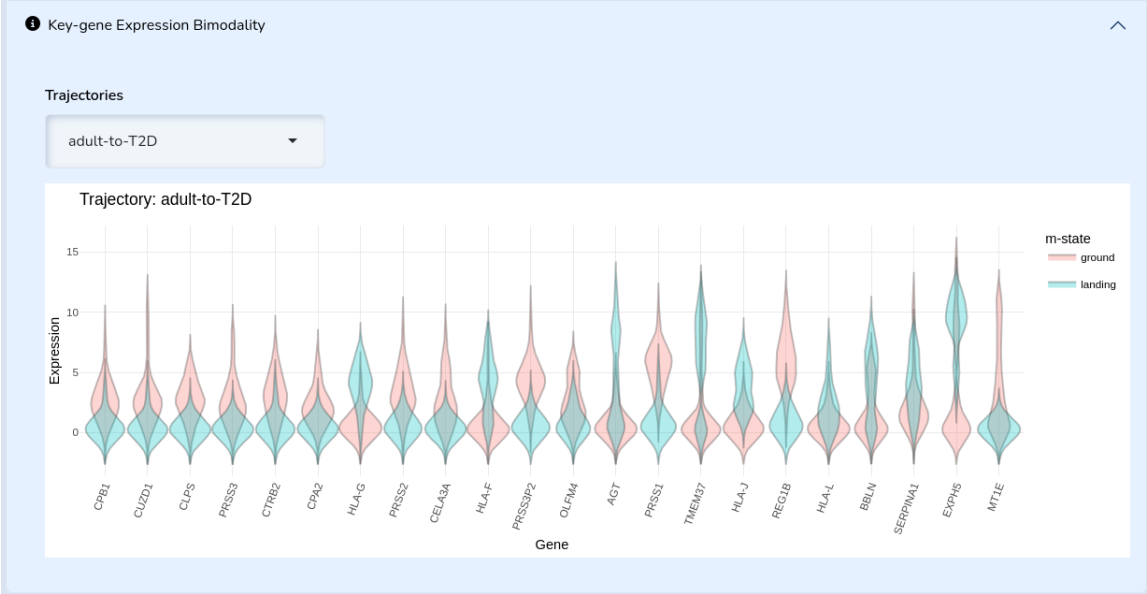

Figure 4.27: Key Genes Expression Bimodality. Plot featuring two overlaid violin plots per key gene, displaying high and low expression levels in each micro-state of a specific trajectory branch. This visualization helps to identify bimodal expression patterns and the switching behavior of key genes along trajectory micro-states. In the field "Trajectories" the user can choose one of the available trajectories.

In our example, for the `adult-to-T2D` trajectory, key-genes display high expression in the ground micro-state and low expression in the landing micro-state, or the reverse. This confirms the bimodal and model-switching behavior (high-to-low or low-to-high) of key-genes as cells progress along the trajectory connecting the two states.

Another useful plot for visualizing gene expression patterns along trajectory branches is the **Key-gene Expression Heatmap** shown in Figure 4.28, where cells are arranged from left to right according to their position in the trajectory list. This allows us again to observe the bimodal expression and the mode-switching behavior of key-genes as cells move from the ground micro-state (left) to the landing micro-state (right) of the trajectory, with some genes displaying more distinct patterns than others.

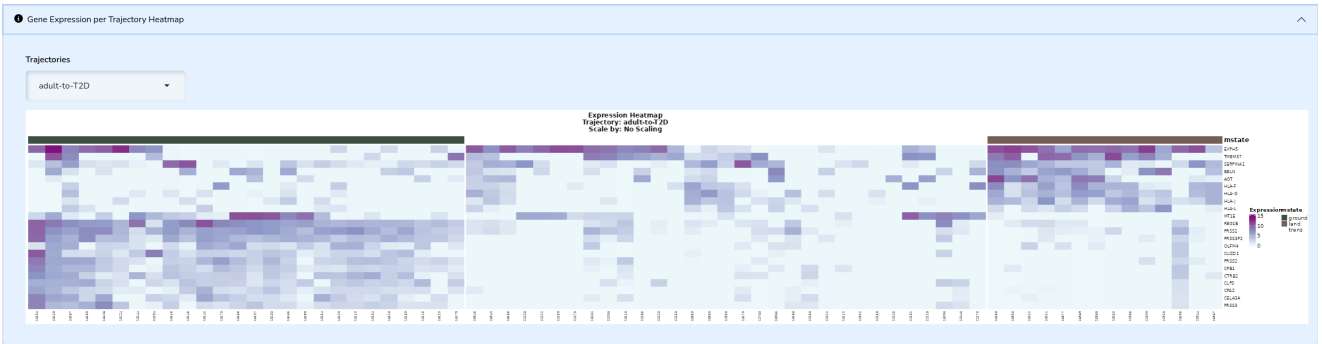

Figure 4.28: Gene Expression per Trajectory Heatmap displaying cells arranged from left to right based on their micro-state. This plot visualizes bimodal expression patterns and the switching of dominant expression modes as cells transition from the ground to the landing micro-state. In the field "Trajectories" the user can choose one of the available trajectories.

We delve even further into the visualization of a specific key-gene expression pattern using

the **Micro-States with Gene Expression plot** in Figure 4.29. This plot shows how the expression of a **single** key-gene changes as we progress through the cells of the different micro-states along a trajectory. The horizontal axis corresponds to the list of the trajectory cells ordered from left to right in decreasing posterior probability value to the departing state (adult). The left vertical axis, combined with the two colored lines, provides the posterior probability values of each cell to each one of the two states of the trajectory. The right vertical axis, combined with the vertical bars, provides the expression levels of the specified gene across all cells in the trajectory.

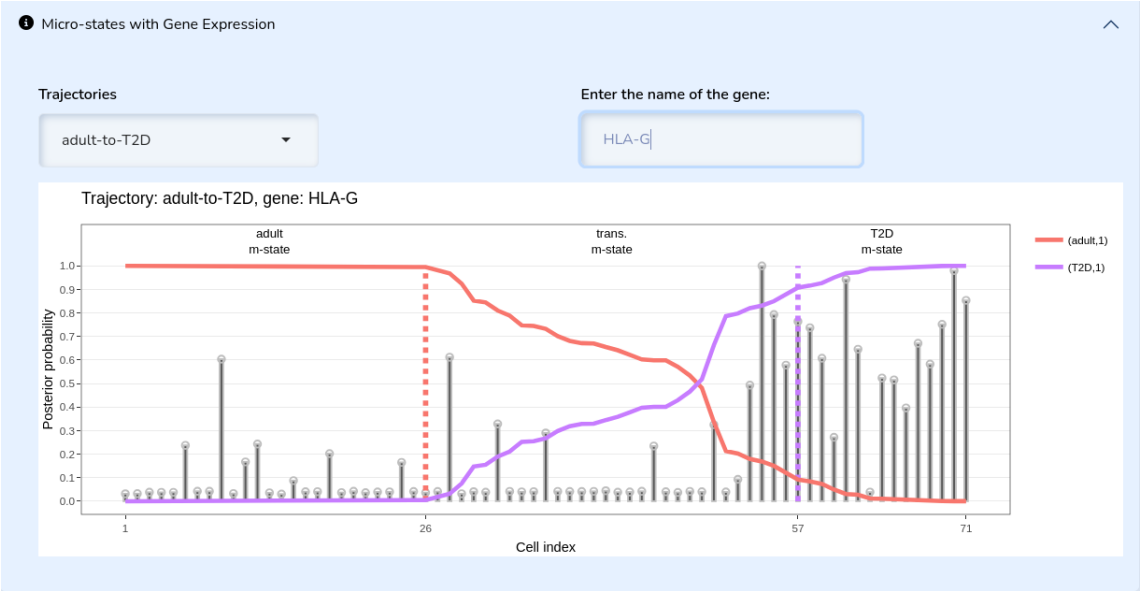

Figure 4.29: Micro-States with Gene Expression plot illustrating how the expression of a single gene changes across cells in different micro-states within a trajectory branch. The horizontal axis represents the cells, while the left vertical axis with colored lines shows the posterior probabilities for each state, and the right vertical axis with bars indicates the gene expression levels.

For example, looking at the behavior of gene `HLA-G` along the trajectory `adult-to-T2D`, we notice a difference in expression patterns between micro-states (separated, by vertical dotted lines). The `adult` (ground) m-state cells have strikingly lower gene expression than the `T2D` (landing) m-state cells. This is also reflected in part by the cells' posterior probabilities. Higher gene expression is correlated with a higher posterior probability to the landing state in this case. In summary, we observe a clear OFF-to-ON expression mode switching pattern of the HLA-G gene happening along the trajectory, which justifies the decision to consider it as a "key-player" when it comes to modeling the dynamics of the `adult-to-T2D` state transition using GRNs. We remark that MLscAN uses only the cells relevant to the trajectory of interest to infer its key-genes and not all cells that belong to the two states this trajectory connects. This, along with the partitioning of trajectories to micro-states makes the modeling of transition dynamics parsimonious and very focused.

4.1.9.2 GRN reconstruction

After partitioning trajectories into micro-states and identifying the key-genes for each branch, Gene Regulatory Networks (GRNs) can be reconstructed. A GRN is built for each trajectory micro-state, utilizing only the key-genes associated with that branch. The goal is to explore potential changes in the regulatory mechanisms of key-genes as cells transition

from one micro-state (ground) at the departing state to another (landing) at the transitioned state.

To create GRNs, MLscAN `GENIE3` algorithm [29] for network inference. This algorithm infers gene regulatory networks by using random forests to predict the expression of each gene based on the expression of all other genes (non-linear regression). It computes importance scores for each gene pair, identifying potential regulatory relationships based on these scores.

A GRN is a fully connected graph with nodes the key-genes, and weighted edges among them that model gene regulation. Figure 4.301 shows an example of how to visualize the inferred GRN weights generated by the GENIE3 algorithm in a matrix format. For any target gene (column), we can examine its regulator genes (rows) and determine whether their influence is positive (excitatory) or negative (inhibitory) to the target, as well as gauge the strength of their regulatory effect. Larger weight magnitudes (positive or negative) indicate stronger support from the data for the regulatory relationship.

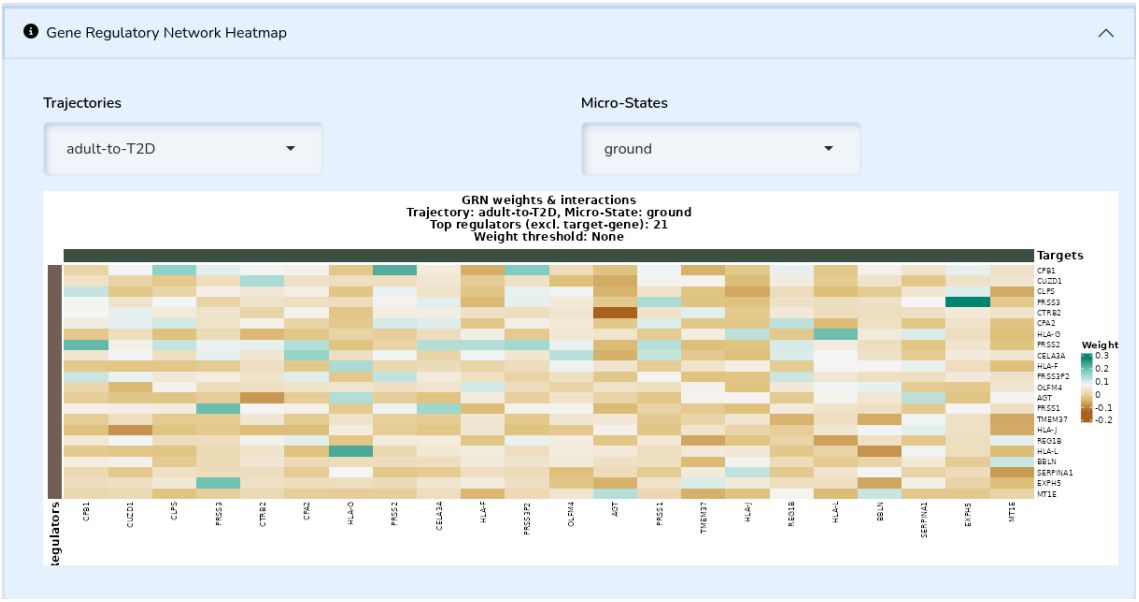

Figure 4.30: Gene Expression per Trajectory Heatmap visualizing the inferred Gene Regulatory Network (GRN) weights, generated by the GENIE3 algorithm, between key genes in a matrix format. For each target gene (columns), the plot displays its regulators (rows), indicating whether their influence is positive or negative and the strength of their regulatory effect. In the field "Trajectories" the user can choose one of the available trajectories whereas in the field "Micro-States" they can choose one of the three micro-states

Figure 4.30 shows the weights and interactions of key genes for the ground micro-state of the adult-to-T2D trajectory. It is important to note that the inferred GRNs can differ significantly between a trajectory's ground and landing micro-states even though the key-genes are the same.

The **Gene Regulatory Network Graph** visualizes the GRN for each micro-state of each trajectory as a graph (Figure 4.31–4.32). This approach is particularly useful for comparing the distinct micro-states along a trajectory branch.

In the graph, each arrow signifies an interaction between genes, where the arrow starts at the regulator gene and points to the target gene. Green edges indicate excitatory interactions, while red edges represent inhibitory ones. To maintain clarity and avoid excessive clutter, **Gene Regulatory Network Graph** by default only displays edges with weights

above a certain threshold, calculated as $max(all\ weights) - std(all\ weights)$. Additionally, only genes involved in valid interactions—either as regulators or targets—are plotted. As a result, the number of genes shown may differ between micro-states, as some are excluded due to not meeting the interaction weights threshold or due to limitations on the number of top regulators per target gene. The user has the ability to manually define the weight threashhold and the number of regulators considered.

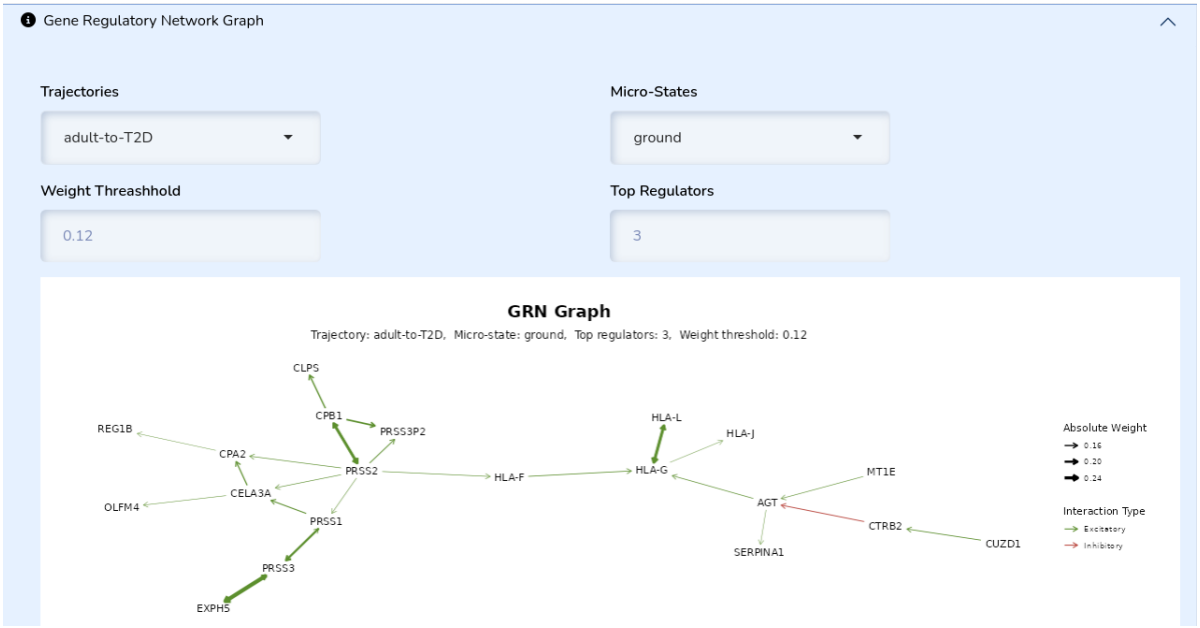

Figure 4.31: Gene Regulatory Network Graph for `adult-to-T2D` ground micro-state

As shown in the graph in Figure 4.31, PRSS3 exhibits a strong excitatory relationship with the gene EXPH5, as indicated by the thick green arrow. When we observe the same relationship in the heatmap in Figure 4.30, the interaction between PRSS3 and EXPH5 is shown with a weight near the top of the scale, represented by the blue color. In general, thicker arrows in the graph indicate stronger interactions, which are validated by the higher weight values in the **Gene Regulatory Network Heatmap**.

In Figure 4.32, we plot the same graph for the landing micro-state and we can see how the relationships between genes have shifted. For instance, **BBLN** now exhibits a strong inhibitory relationship with **PRSS1** (thick red arrow), while **AGT** maintains an excitatory relationship with **SERPINA1** (green arrow). This suggests that as we move between micro-states (from ”ground” to ”land”), the regulatory interactions between the same or different genes can change significantly, highlighting the dynamic nature of gene regulatory networks in different cell states.

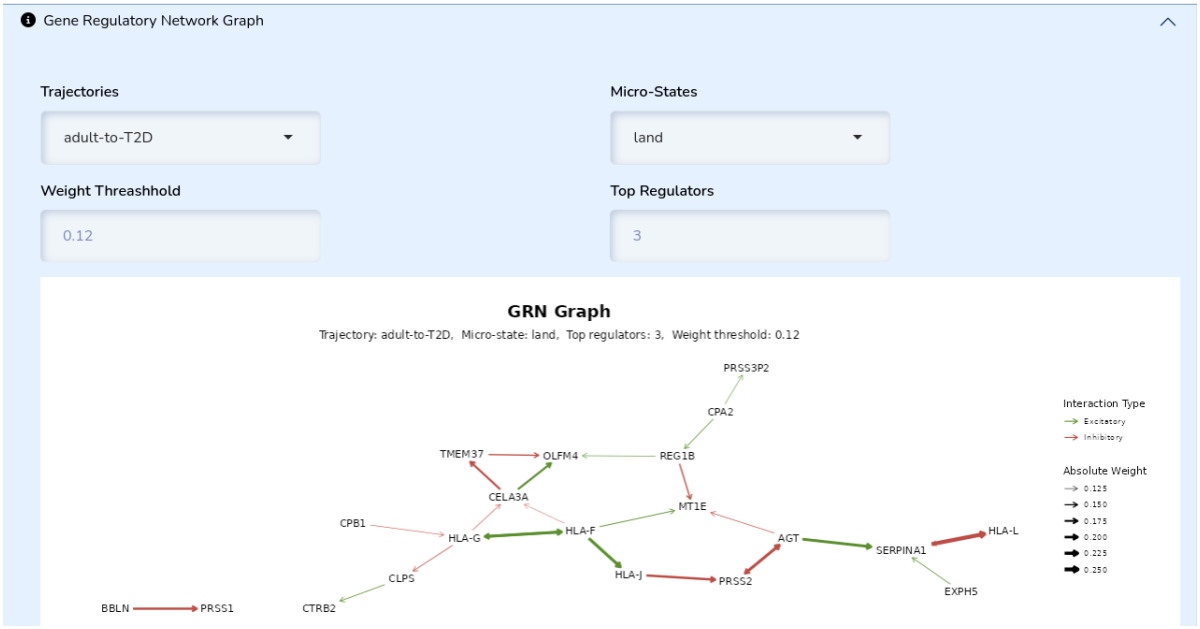

Figure 4.32: Gene Regulatory Network Graph for `adult-to-T2D` land micro-state

MLscAN introduces the concept of micro-states to account for dynamic changes in gene expression along a trajectory (pseudo-time). The regulatory patterns of key genes may differ significantly across these micro-states, a phenomenon that plays a crucial role in various cellular processes. This dynamic regulation has been frequently observed in developmental biology, cancer research, and other vital cellular functions [30].

4.2 Handling Mixed-States

As mentioned before, MLscAN detects mixed-states, i.e., states with an unusually large variance relative to the other inferred states. This suggests the possibility of outliers or interesting cell subpopulations within a state that warrants more attention. There is a specific tab on the navigator bar dedicated to the mixed-states. (see Figure 4.33)

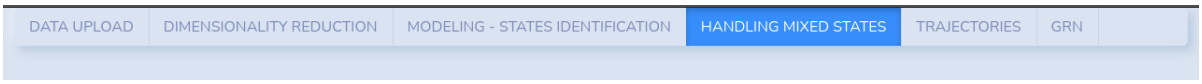

Figure 4.33: Mixed-state tab in the navigation bar

The structure of this tab differs from the others. Briefly, the tab is divided vertically into two sections. In the first part,Figure 4.34, the user can see general information about the mixed-states, how to handle them, and some plots to get insights on the mixed-states.

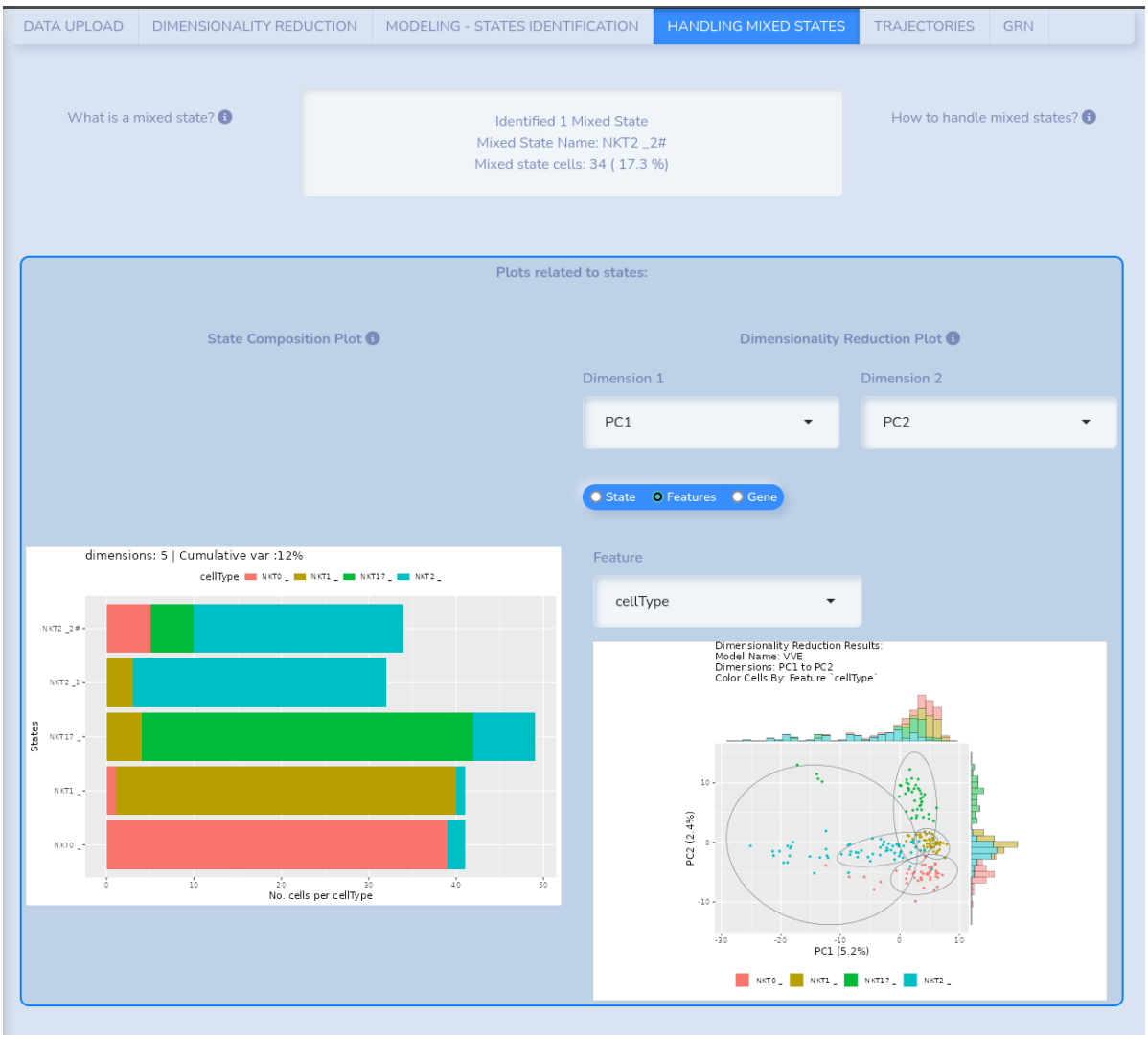

Figure 4.34: Mixed-state tab first section. Information regarding the identified mixed-state. An info box with general information about the identified mixed-state. To the left and right of the info box, there are extra info fields that the user can hover over to get insights on the procedure to be followed to handle mixed states. Below these, are two plots to visualize the inferred states including the mixed ones.

The second section, Figure 4.35, shows the actions that can be performed to handle the mixed-states. First, the user has to choose the mixed-sate they want to analyze. The user can either remove the mixed-state, which means eliminating the cells that cluster in that state entirely from the dataset and make a new run without them, or further analyze them to possibly divide them into subpopulations. Let's see an example for both scenarios.

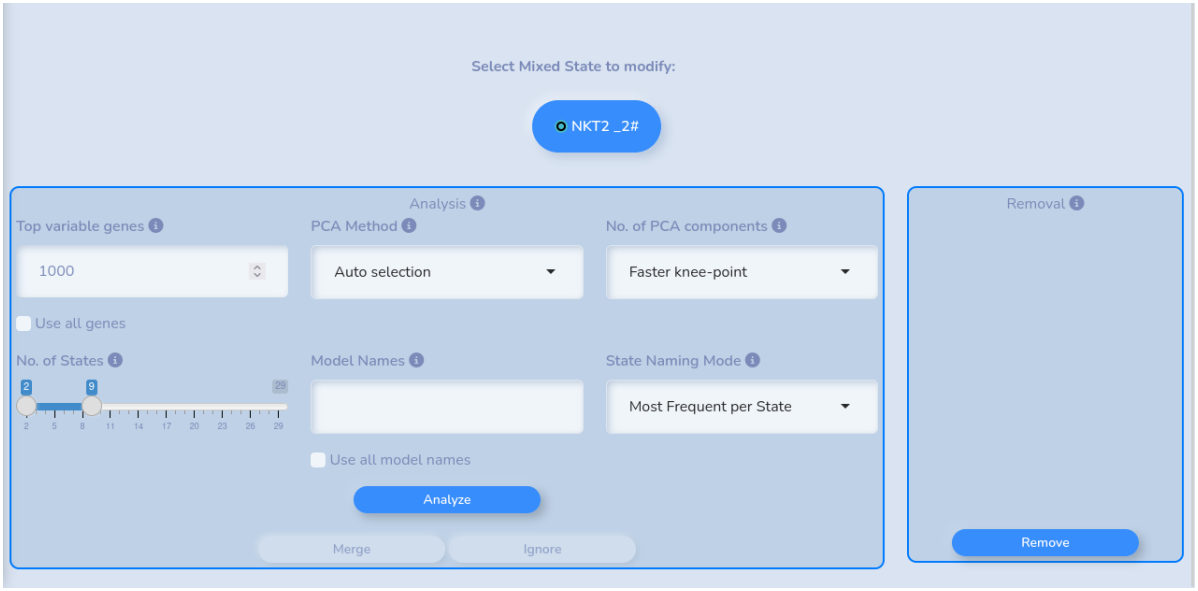

Figure 4.35: Mixed-state tab second section. Radio buttons that correspond to the inferred mixed-states, fields, and buttons related to the actions that can be taken over the selected mixed-state.

4.2.1 Mixed-state removal

In this example, we assume that the detected mixed-state consists of outlier cells, and we will show how to remove it entirely from downstream data analysis.

4.2.1.1 Dataset and Parameters Used

We will utilize the `nkt_thymic_exprData` dataset [31], comprising 197 thymic Natural Killer T (NKT) cells and expression data for 6,799 genes. In their research, Engel's group characterized these thymic NKT cells into four highly divergent subsets: NKT0, NKT1, NKT2, and NKT17. Despite their antigen similarity, these subsets exhibited numerous gene expression and epigenetic differences. Apart from the expression data, we will also be using the cell features of the dataset (`nkt_thymic_cellFeat`), which contain the ground truth labels of each cell, used here for visualization purposes only. First, we upload the dataset, see Figure 4.36. Then, we will perform the following two steps, Dimensionality reduction and Modelling.

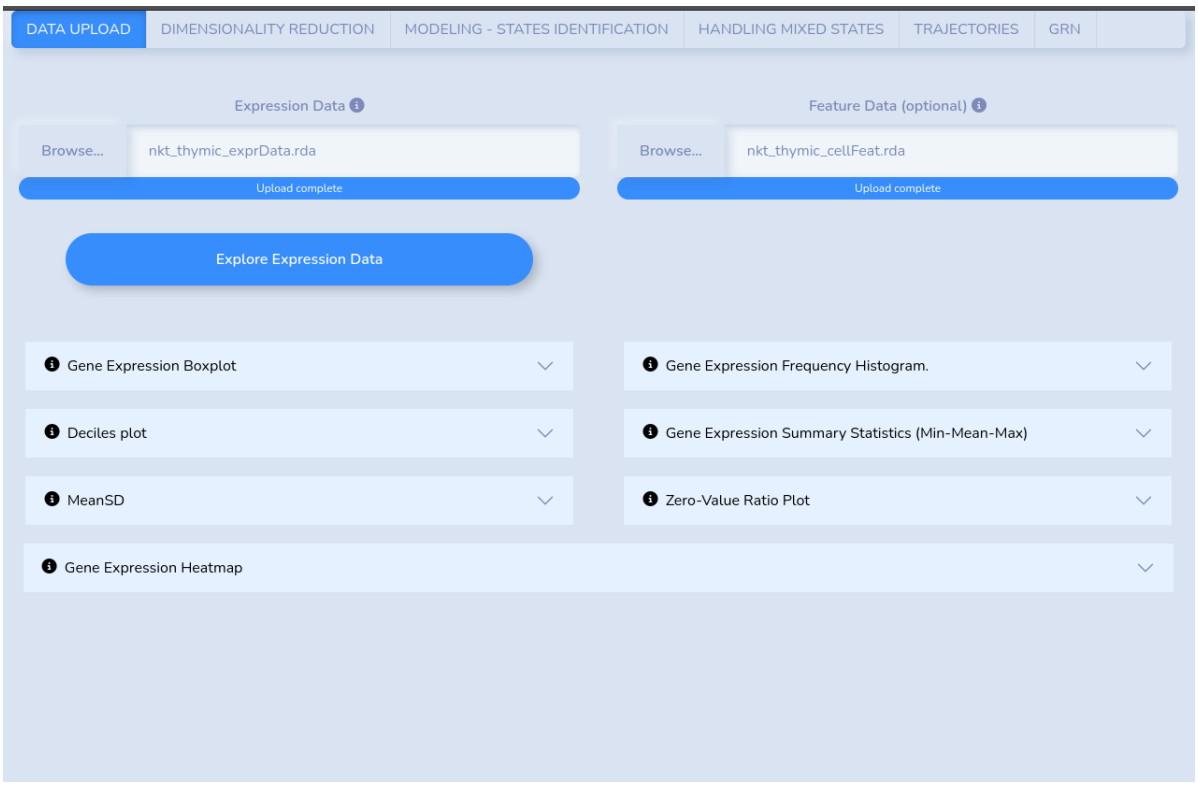

Figure 4.36: DATA UPLOAD tab after hitting the ”Explore Expression Data” button

We will only use the 800 most variable genes for the Dimensionality Reduction step; see Figure 4.37.

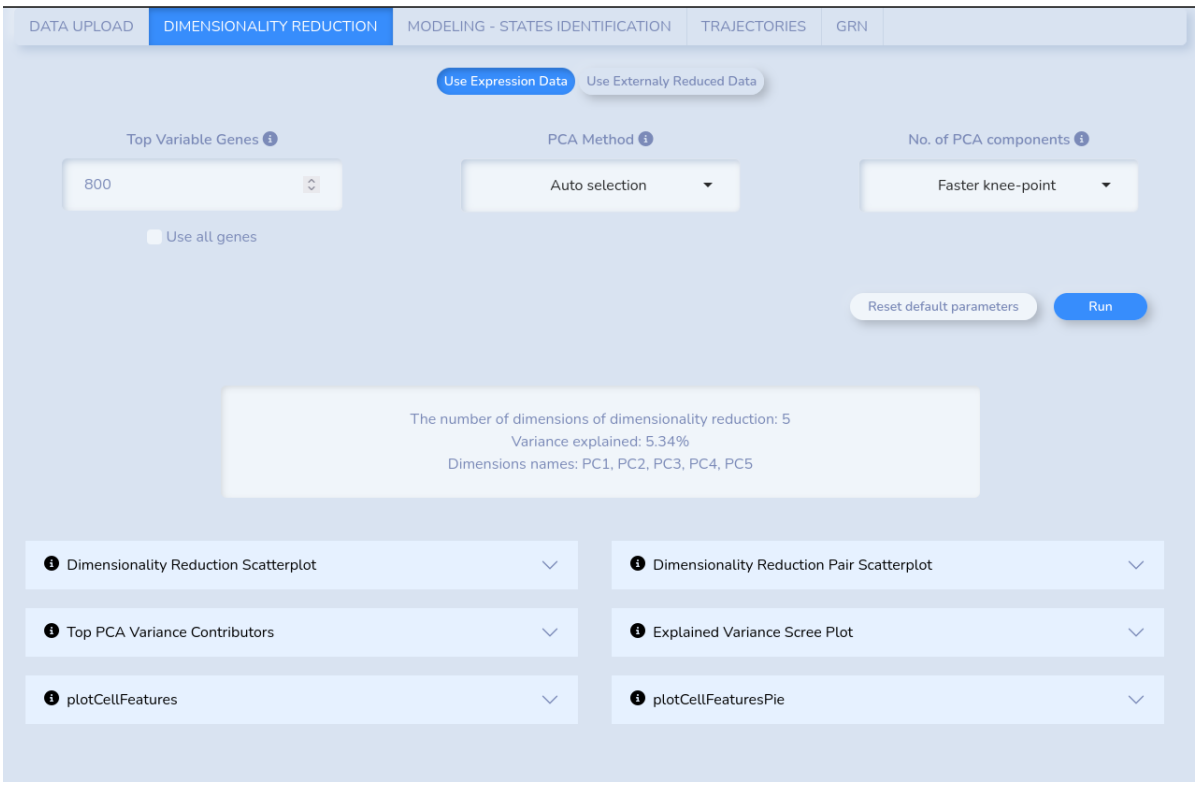

Figure 4.37: DIMENSIONALITY REDUCTION tab after hitting the ”Run” button. 800 most variable genes are selected from the Most Variable Gene field.

For the clustering/modeling step, we will use the default parameters of the MLscANApp, Figure 4.38.

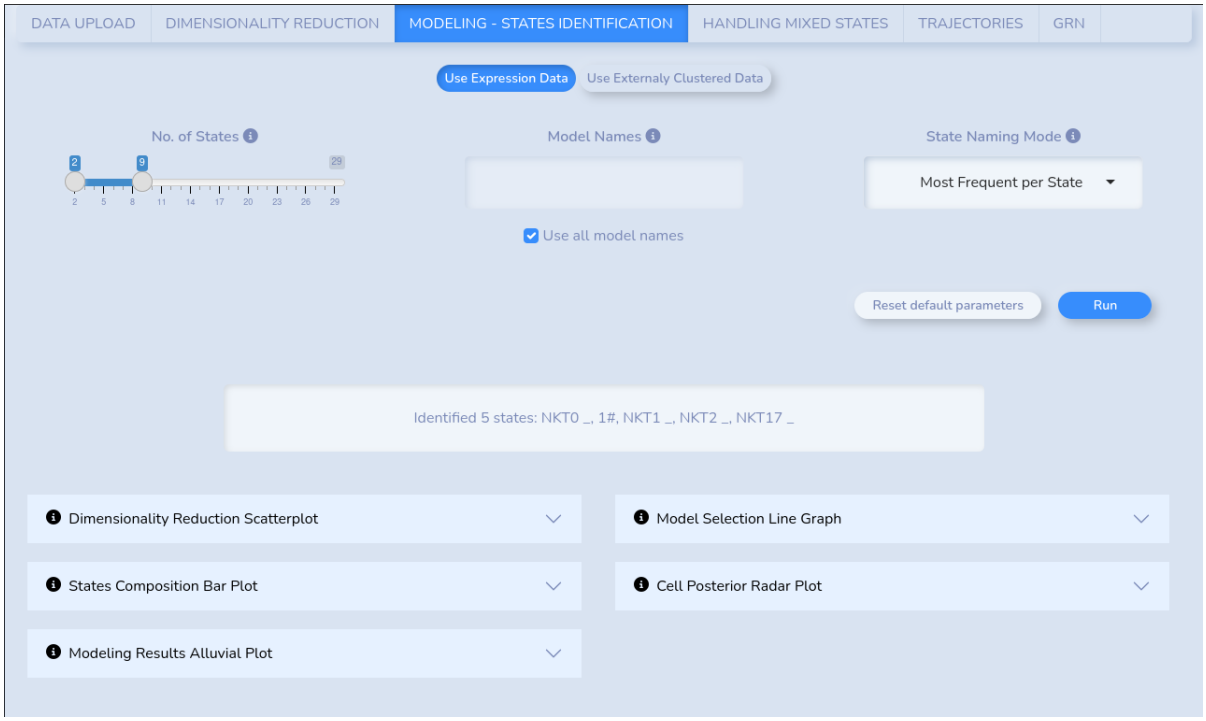

Figure 4.38: MODELING -STATES IDENTIFICATION tab after hitting the ”Run” button.

4.2.1.2 Inspection of the Identified Mixed-state

On the **HANDLING MIXED STATES** tab, the info box in Figure 4.34 indicates that one mixed-state was identified, labeled as 1#. MLscAN adds to the names of mixed-states a hash ”#” suffix.

By inspection of the Dimensionality Reduction Scatterplot,Figure 4.39, we can easily detect a mixed state (1#) with a very large PC1 variance.

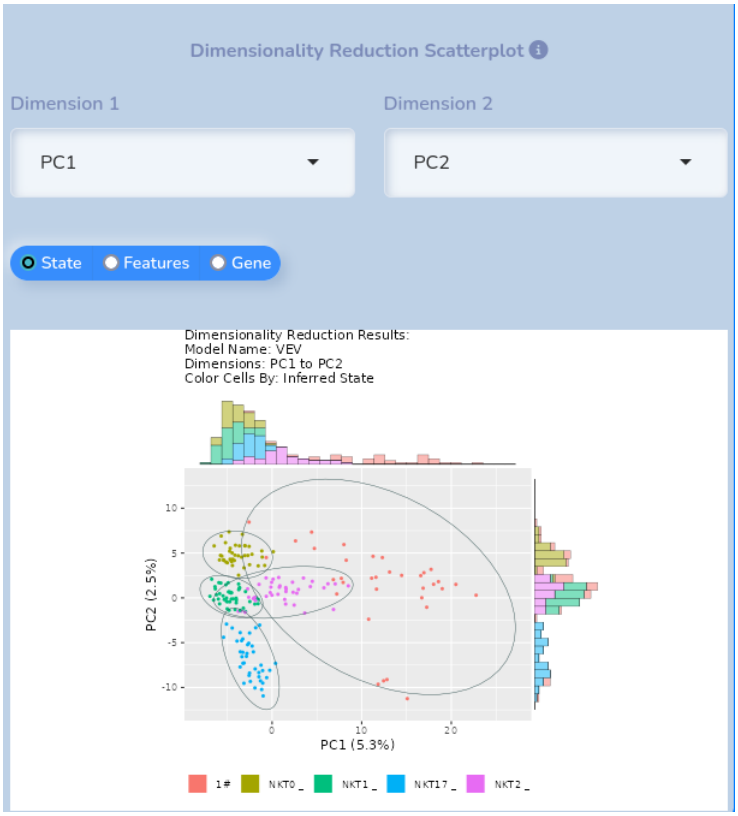

Figure 4.39: Dimensionality Reduction Scatterplot, with cells colored according to their respective state. Ellipses represent the inferred states.

When coloring the cells by their type in Figure 4.40, we observe that one state (highlighted by the large ellipse) contains cells of multiple types.

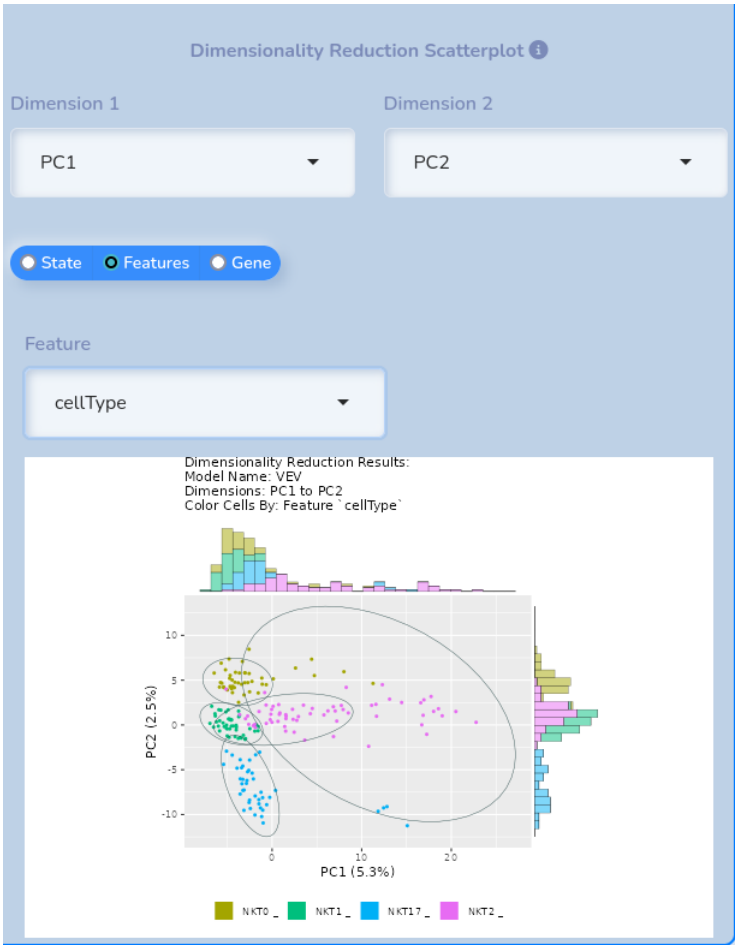

Figure 4.40: Dimensionality reduction scatterplot, with cells colored according to their respective type. Ellipses represents the inferred states.

To analyze the states composition we can refer to the **State Composition Bar plot**

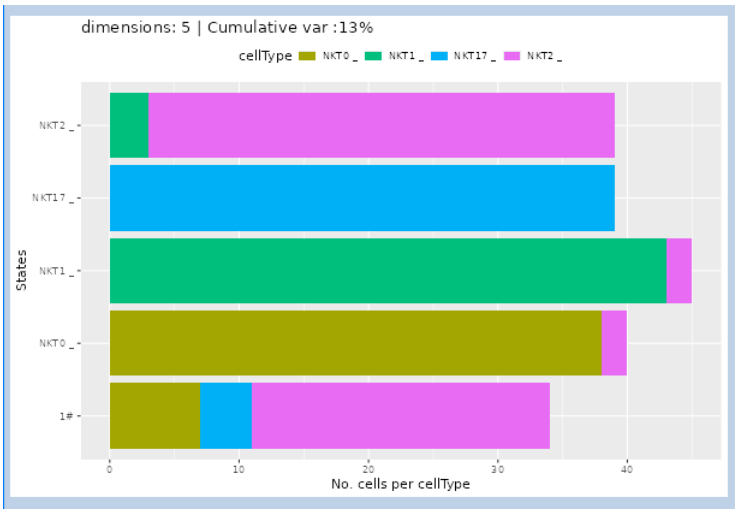

Figure 4.41: State Composition Bar plot

In Figure 4.41, we can see that MLscAN creates a mixed state (1#) comprising a few NKT0, NKT17, and NKT2 type cells. Based on the info from the info box, the mixed state contains 37 out of 187 total cells (17.3%). Since this mixed state is heterogeneous and represents only a small fraction of the total cells, one option is to consider it as a set of distributed outlier cells and proceed to remove it.

4.2.1.3 Removal

As only one mixed-state is identified, it is preselected in the radio button list. We simply click the *Remove* button, and MLscANApp will execute the corresponding workflow. If additional mixed-states were identified, the **HANDLING MIXED STATES** tab would update with the latest results. In our case, however, no new mixed-states are identified after removing the selected cells, and the tab is removed from the navigation bar. A dialog will notify us that no more mixed-states are present, and we will be directed to the next step of the pipeline (trajectory inference). The dialog will also display the **States Composition Bar Plot** to reflect the current state composition.

Figure 4.42: Pop-up dialog, notifying the user that there are no more mixed-states to explore, including the States Composition Bar plot showing the composition of the inferred states after removing the mixed-state.

In our case, removing the mixed state has improved the clustering, as shown in the plot of Figure 4.42. *Note:* If the results are unsatisfactory, we can return to the **MODELING-**

STATES IDENTIFICATION tab and rerun the modeling with the same or adjusted parameters.

4.2.2 Alternative Approach

We can utilize the navigation bar to go back and forth in the pipeline. For example, if we have inspected a mixed-state and are unsure whether we want to remove it, we can visit the **TRAJECTORIES** tab, Figure 4.43, execute the step, and observe the **Transition Propensities** graph

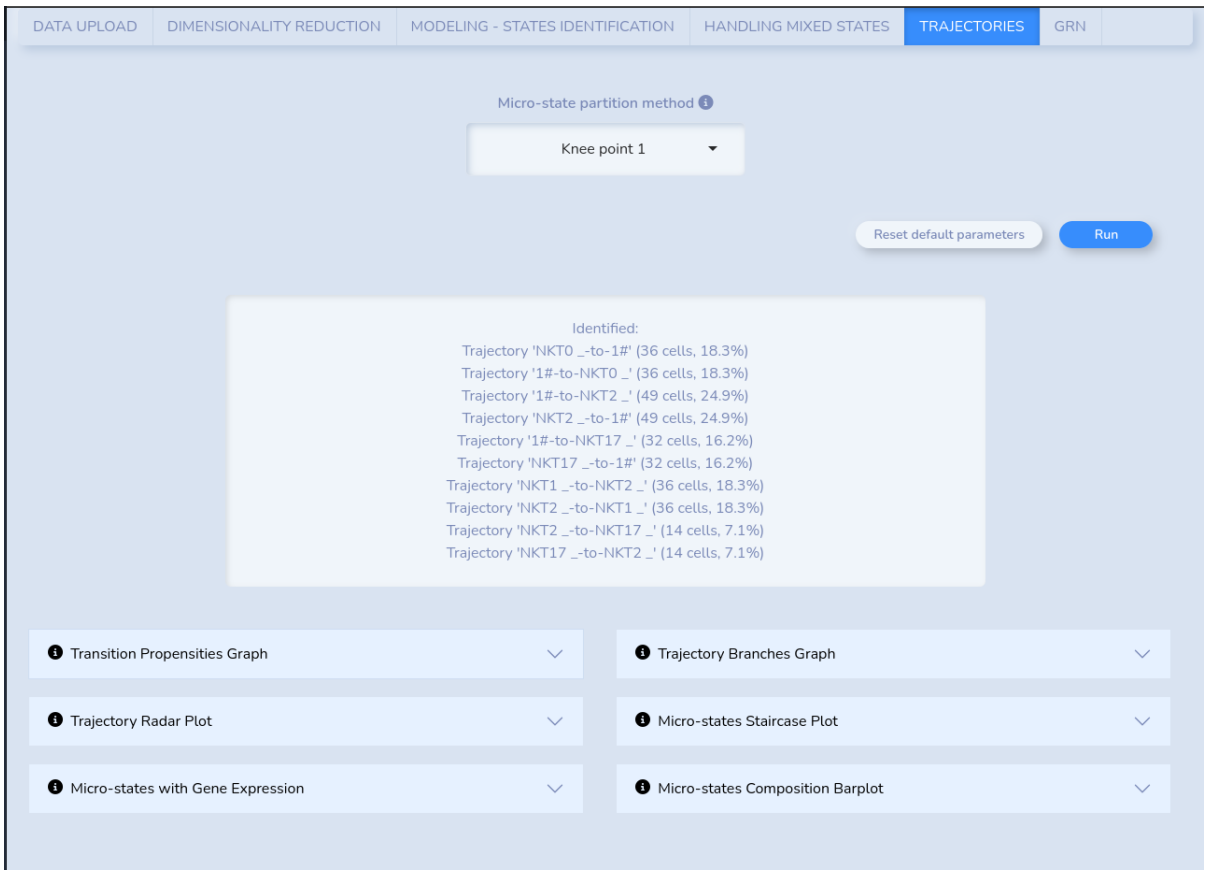

Figure 4.43: TRAJECTORIES tab showcasing how the tab looks like after executing the step.

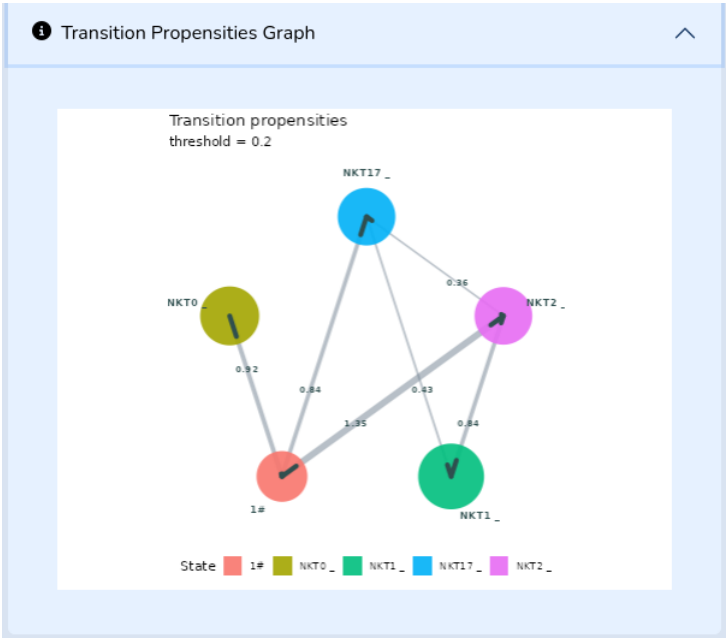

Figure 4.44: Transition Propensities graph, before removing the mixed-state 1#. Plot illustrating states as circles, with sizes corresponding to the number of cells assigned to each state. Transitions between states are represented by gray lines, with line thickness indicating the strength of the transition. Dark segments on the circle edges reflect the proportion of cells with a second-highest probability assigned to another state.

As we can see in Figure 4.44, nearly every other state interacts with the mixed state. This is expected because this state covers a broad range in the posterior probabilities space due to its high variance.

We can now go back to the **HANDLING MIXED STATES** tab, remove the mixed-state 1#, and re-execute the Trajectory Inference step in the **TRAJECTORIES** tab. Removing the mixed state allows MLscANApp to generate a transition network of higher quality, see Figure 4.45. From Engels' analysis [31], we know that NKT0, which contains precursor cells, is closer to NKT2 cells than to the other subsets. This seems to be confirmed by our transitions plot. We may also conclude that NKT2 is an intermediate subset, as it connects to all the other subsets.

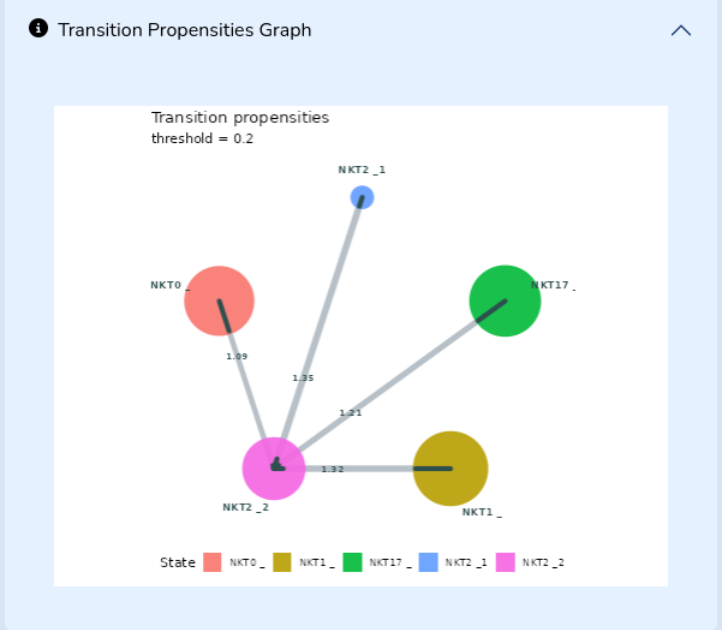

Figure 4.45: Transition Propensities graph, after removing the mixed-state 1#. Plot illustrating states as circles, with sizes corresponding to the number of cells assigned to each state. Transitions between states are represented by gray lines, with line thickness indicating the strength of the transition. Dark segments on the circle edges reflect the proportion of cells with a second-highest probability assigned to another state.

4.2.3 Mixed State Analysis

The initial mixed-state contains a large proportion of the cells, and in many such cases, it might not be appropriate to remove the entire mixed-state and lose those cells from our analysis; instead, we might want to isolate and analyze the mixed-states further on their own, as the next example shows.

In this use case, a detected mixed-state is further analyzed to identify possible sub-populations of cells within it, which are then incorporated back into the ongoing analysis as new entities.

4.2.3.1 Dataset and Parameters Used

We will utilize the `mesc_pre_exprData` dataset [32], which contains single-cell RNA-sequencing (scRNA-seq) data from mouse Embryonic Stem Cells (mESCs) collected at 0, 12, 24, 48, and 72 hours after inducing differentiation into Primitive Endoderm cells.

We will also be using the cell features of the dataset, which contain the ground truth labels of each cell, for visualization purposes only. Additionally, we will use the `prcomp` function for the Principal Components Analysis of the dimensionality reduction step and explore the GMM models with diagonal covariance structures only (i.e., EEI, VEI, EVI, VVI) for the model selection step.

Figure 4.46 and Figure 4.47 below provide an overview of what the **DIMENSIONALITY REDUCTION** tab and **MODELING - STATES IDENTIFICATION** tab will look like.

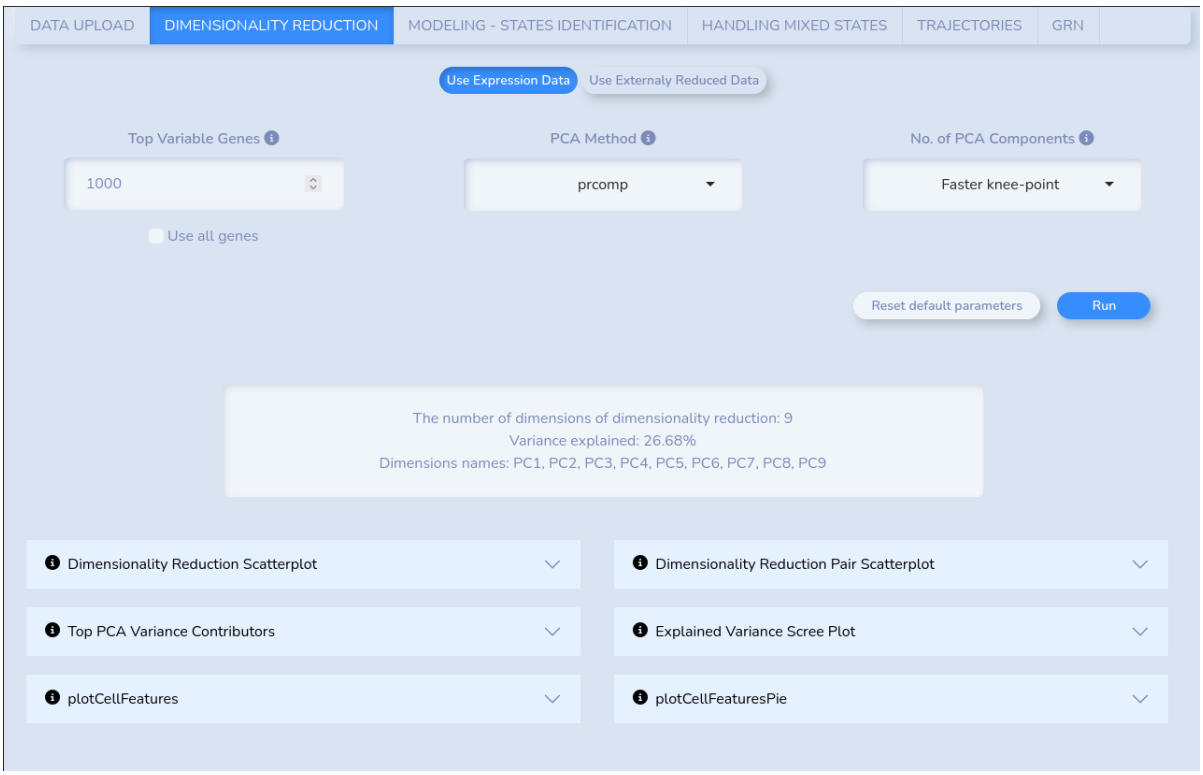

Figure 4.46: DIMENSIONALITY REDUCTION tab after hitting the ”Run” button. The parameters fields show the selected settings used for the ”Mixed States Analysis” use case. All parameters field has the default values except the PCA Method where we have selected ”prcomp”.

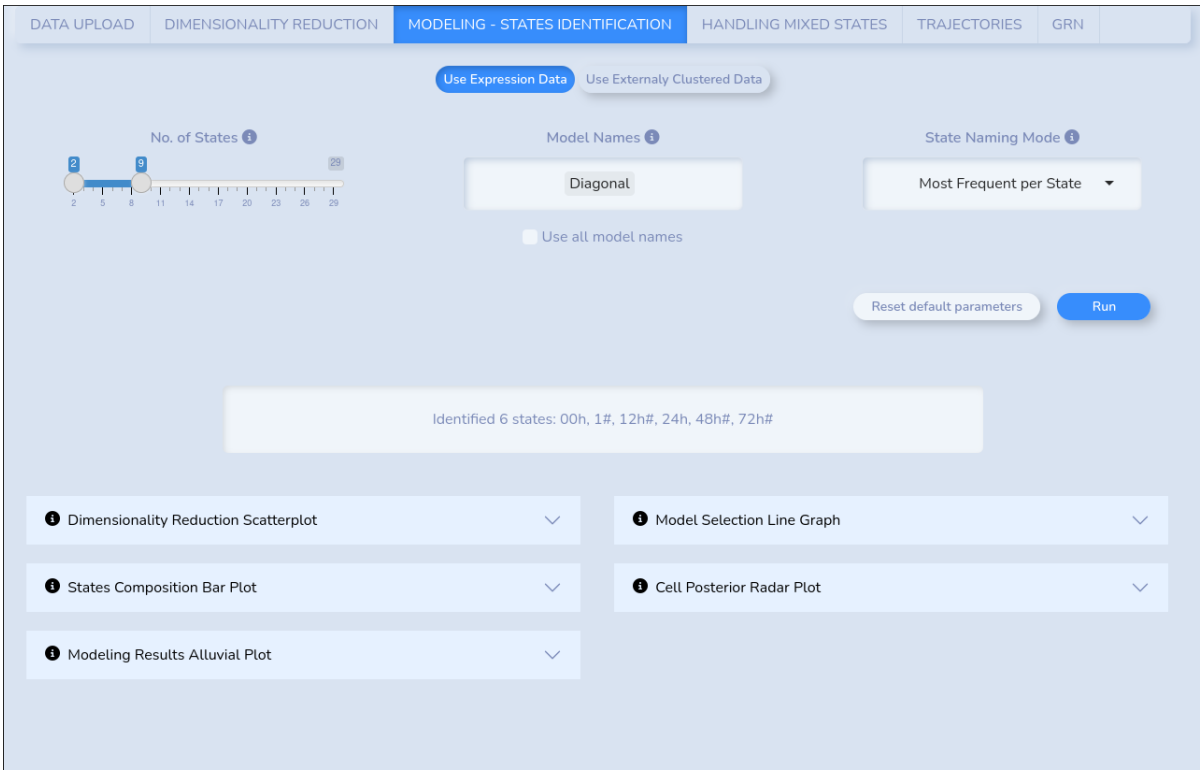

Figure 4.47: MODELLING - STATES IDENTIFICATION tab after hitting the ”Run” button. The parameters fields show the selected settings used for the ”Mixed States Analysis” use case. All parameters field has the default values except the PCA Method where we have selected ”Diagonal”.

The Dimensionality Reduction plot (Figure 4.48) and states composition plot (Figure 4.49) give us a clear picture of the clustering results. We can see four mxed-states inferred, which were named 1#,48h#, 12h#, and 72h#. The last three consist of cells of one cell type (ground truth) by at least 70%, so we will not consider them further in the analysis. In contrast, 1# is a large variance state consisting of a mixture of 48h and 72h type cells, suggesting the possible existence of sub-populations. MLscANApp can help us investigate this possibility by further analyzing this particular mixed state's cells.

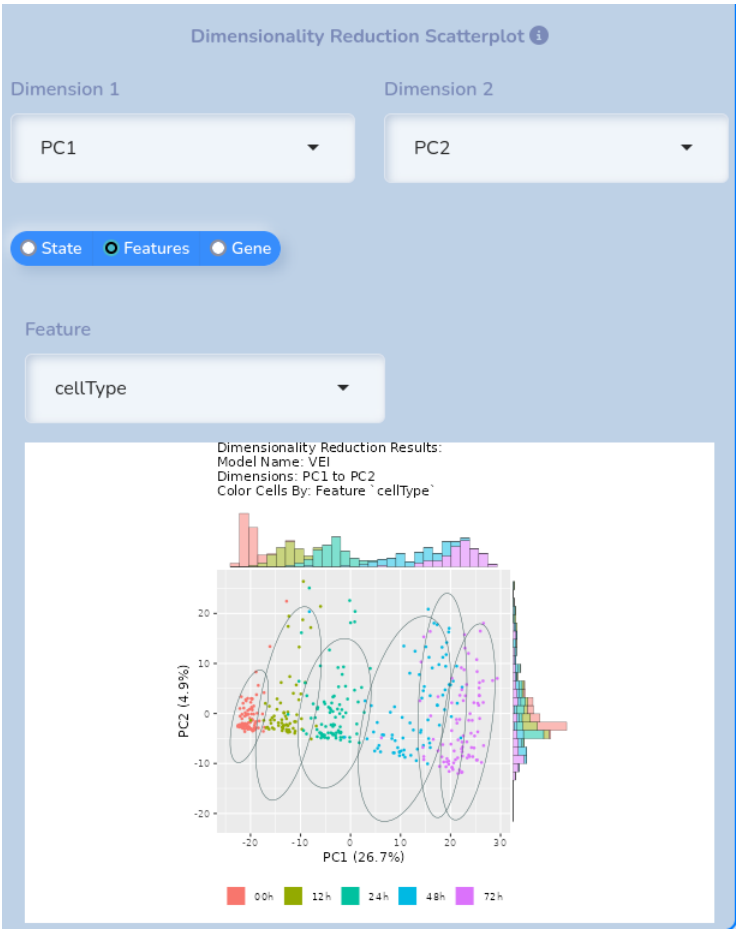

Figure 4.48: Dimensionality Reduction Scatterplot, with cells colored according to their respective type. The ellipses represent the inferred states. The user can select which principal components (PCs) to plot using the Dimension 1 and Dimension 2 fields. Radio buttons allow for cell coloring based on the provided features (specified in the field "Feature" below the radio buttons), gene expression levels, or inferred states.

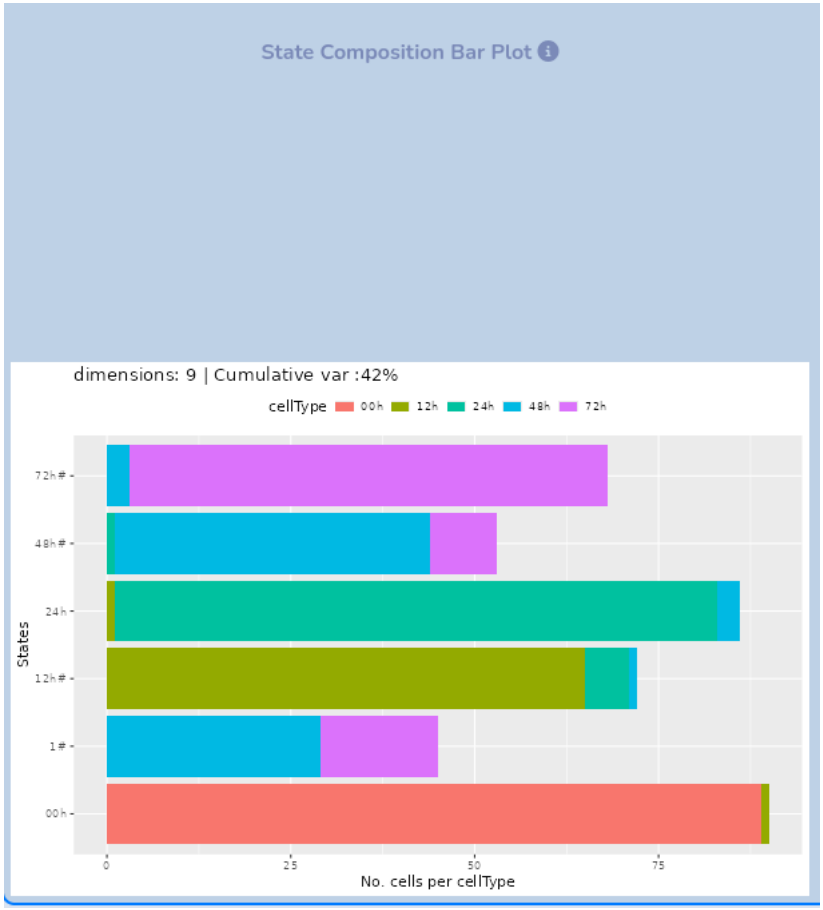

Figure 4.49: States Composition Bar plot illustrating the composition of each state. Each color represents a cell type. The 'Feature' field allows the user to select which one of the provided features will be used to color the bars.

To analyze the mixed-state 1#, we first select it from the radio button list, adjust the necessary parameters for the analysis, and click the "Analyze" button. In this case, we will use the `prcomp` function for PCA, leaving the rest of the parameters at their default values. MLscANApp will isolate the cells of the selected mixed-state and run the pipeline up to the modeling step for this subset of cells.

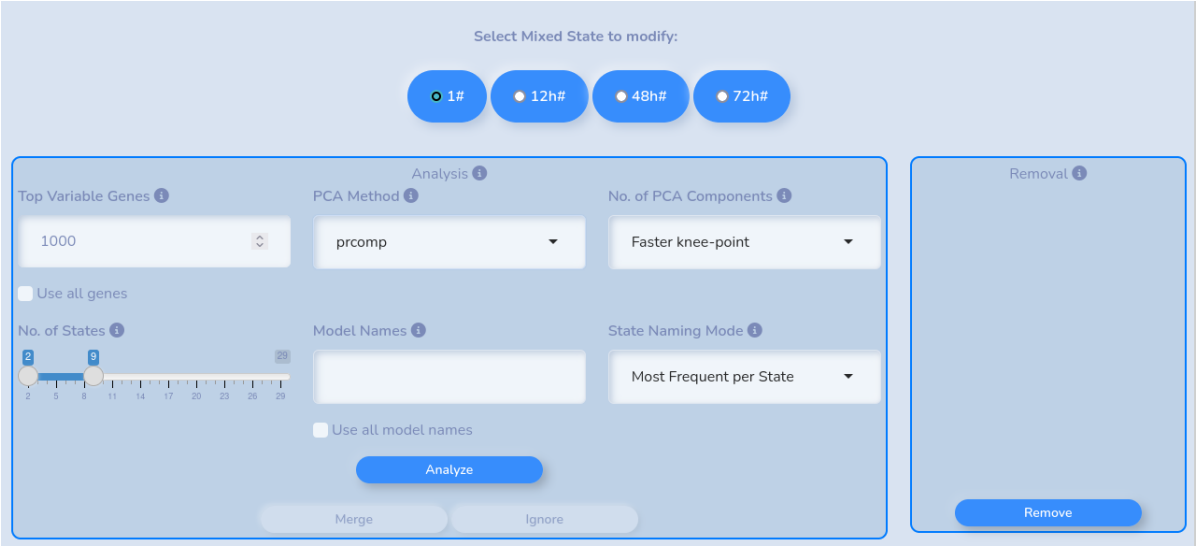

Figure 4.50: Second section of the HANDLING MIXED STATES tab. The parameters fields show the parameters used for the ”Mixed States Analysis” use case.

To view the analysis results, we scroll down to the section following the second part of the **HANDLING MIXED STATES** tab, where the plots corresponding to these cells will dynamically appear.

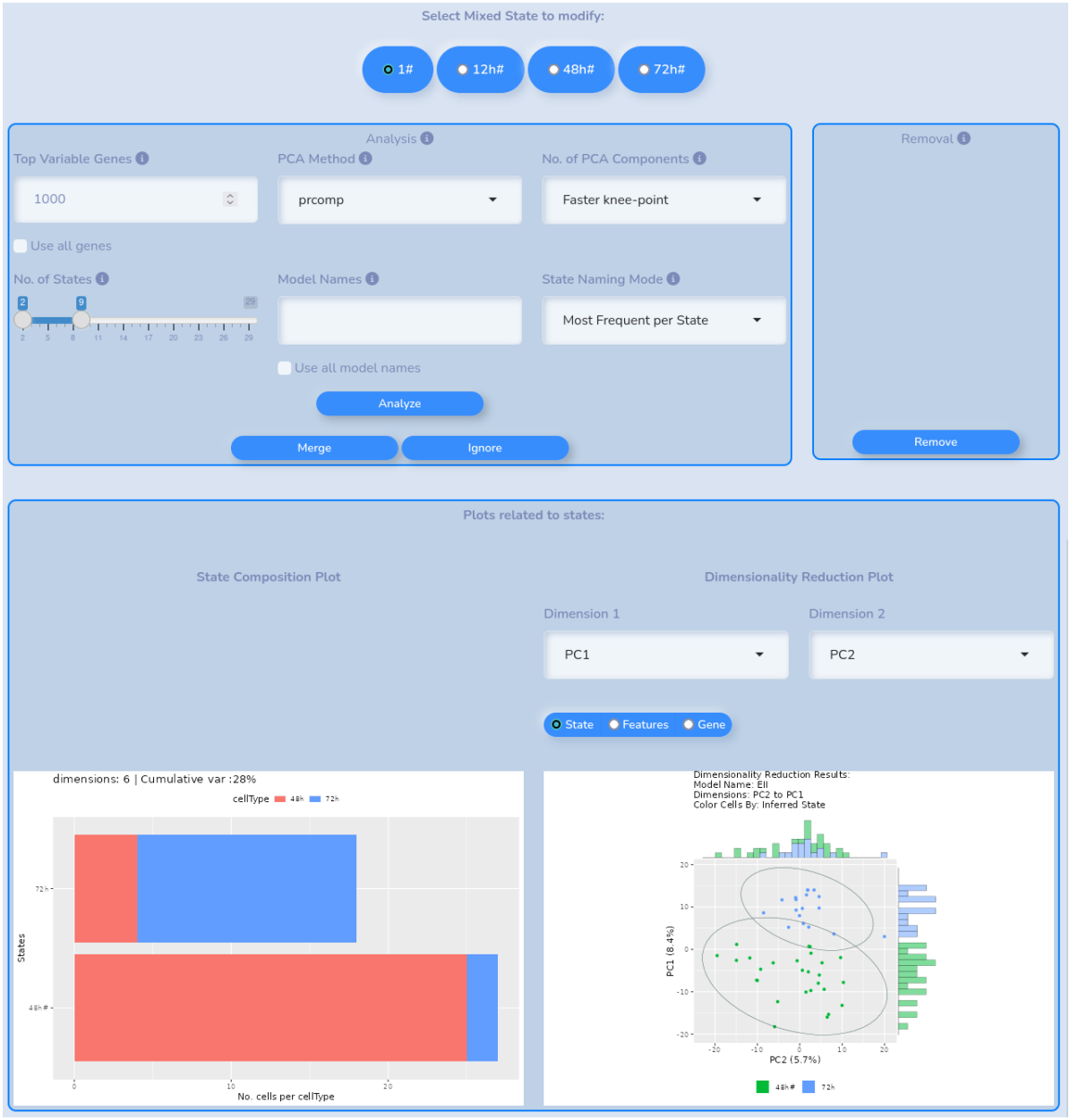

Figure 4.51: HANDLING MIXED STATE TAB after analyzing mixed-state 1#. The second section of the tab has now the ”Merge” and ”Ignore” buttons active. Below this, in the third section, we see the corresponding plots for the analysis of the selected mixed-state.

From the plots in Figure 4.51 we can see that the MLscANApp run of the mixed-state in isolation reveals two sub-populations, one containing mainly 48h type cells and one containing mainly 72h type cells.

As we can see in the Figure 4.51, the **”Ignore** and **”Merge”** buttons became active. This is because after analyzing the mixed-states, we have two options: either ignore the performed mixed-states analysis, leaving the mixed-state intact, or decompose it and incorporate its two sub-populations into the ongoing downstream analysis.

We will use the latter option, so we hit the **”Merge”** button and MLscANApp will perform the merging procedure for us and update the tab. Now, we scroll up to the first section of the tab to see the new plots, see Figure 4.52

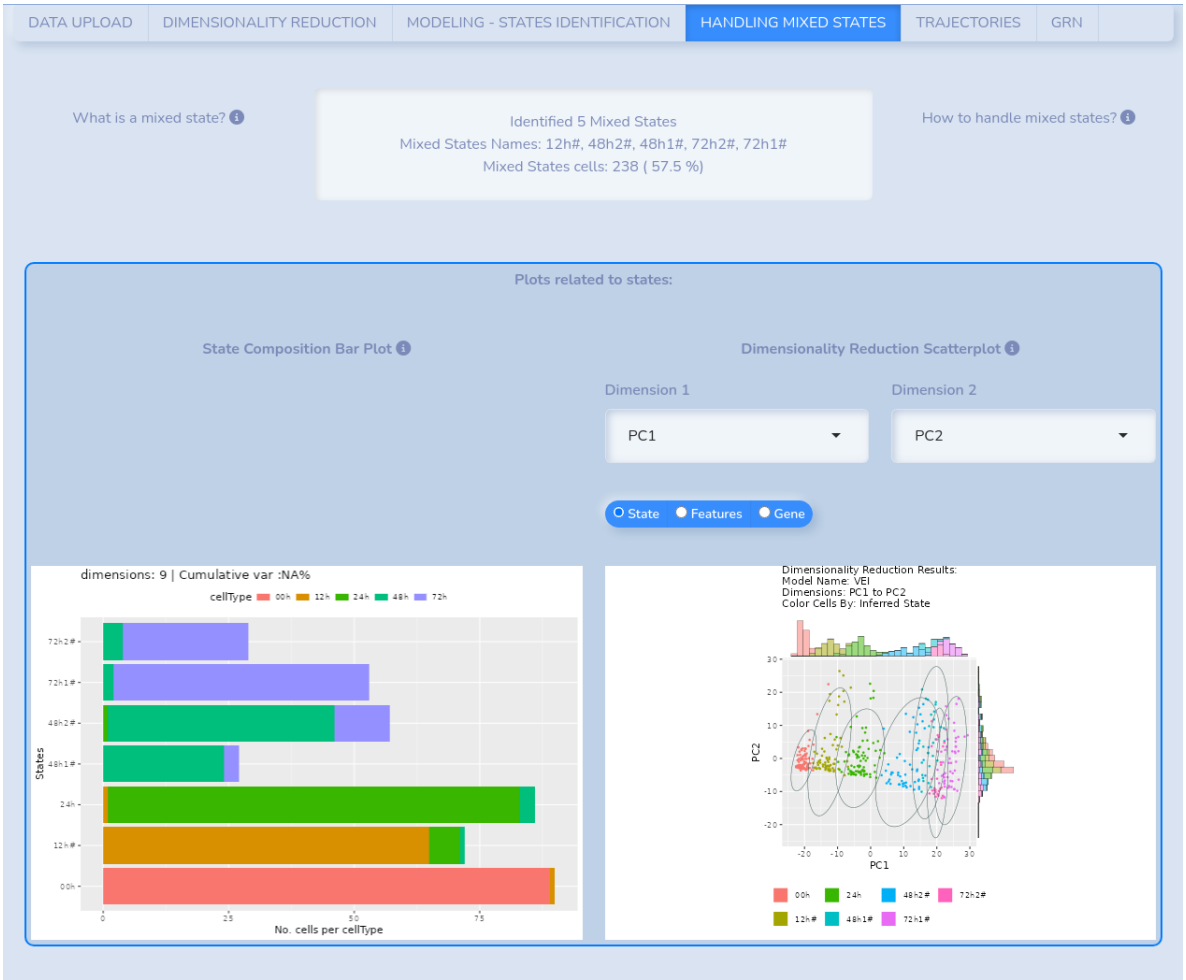

Figure 4.52: The first section of the HANDLING MIXED STATES tab after hitting the ”Merge” button to include the sub-populations inferred from the analysis of a mixed-state in the main analysis as separate states.

MLscANApp now detects two new states for sub-populations from 72h and 48h cells after decomposing the originally mixed-state. This procedure can be repeated for any mixed state and as many times as the user wants. To the best of our knowledge, the ability to identify, examine, remove entirely, or split cell states into sub-populations systematically, recursively, and interactively is a unique feature of MLscANApp among similar applications.

5. CONCLUSIONS AND FUTURE WORK

The widespread use of single-cell RNA sequencing (scRNA-seq) in transcriptomics research highlights the necessity for accessible and efficient data analysis platforms. This thesis focused on the design and development of MLscANApp, an RShiny application designed to simplify the scRNA-seq analysis process by offering a comprehensive, user-friendly interface built on top of the comprehensive MLscAN R package.

The primary goal of this thesis was to design and implement an interface for the MLscAN R package that would open its use to non-programmers. We successfully designed and developed the app's first version, which fully supports all the unbiased and, in some aspects, unique analysis steps that MLscAN supports. This interface was crafted to guide users through each step of the pipeline, effectively communicating the results of each analysis phase. It offers flexibility, allowing users to intervene at various points and upload their own custom data for steps like clustering and modeling.

Furthermore, the results are presented in many visually engaging and interactive plots, supporting both data exploration and results interpretation. We converted most of the original MLscAN plots from static to interactive, enhancing the visualization of the analysis results. This improvement allows researchers to explore their data more intuitively, furthering their ability to extract meaningful biological insights.

In previous chapters, we delved into the computational architecture of MLscANApp, analyzed in detail all the software engineering aspects, and demonstrated its ability to streamline the scRNA-seq pipeline with specific, detailed use cases.

The development of MLscANApp successfully met the primary goals of this research, providing a user-friendly interface for the MLscAN R package and facilitating the scRNA-seq analysis process for non-programmers. However, there are some limitations that, if addressed, could further boost the application's usability. Currently, not all plots are fully interactive, which may restrict users' ability to explore data in full depth. Future work will involve customizing the remaining MLscAN functions to integrate with interactive libraries in Shiny, allowing for a more intuitive exploration of the results.

Additionally, the absence of a pause-and-resume functionality presents a challenge for those conducting lengthy or computationally intensive analyses. To address this limitation, the application's modular design can be leveraged to support the development of a feature that allows users to save their progress and resume at a later time. This improvement would grant greater flexibility in managing workflows and make the app more versatile for researchers at all levels of programming expertise.

While our primary goal was to tailor the app for scientists without programming skills, some additional features could make the app valuable for scientists with programming expertise as well. One potential future development is enabling users to upload model runs for visualization, allowing them to incorporate analyses performed outside the app, such as in RStudio. This enhancement would let users explore and display their results within the interactive environment of MLscANApp without needing to rerun the entire MLscAN pipeline. By adding this feature, the app could be used not only for generating analyses but also for interpreting and presenting results from other tools, broadening its appeal to those familiar with R and seeking efficient ways to showcase their findings.

ABBREVIATIONS - ACRONYMS

MLscAN	Machine Learning for Single Cell Analytics
PCA	Principal Components Analysis
PCs	Principal Components
GMM	Gaussian Mixture Model
scRNA-seq	Single Cell RNA-sequencing
TI	Trajectory Inference
GRN	Gene Regulatory Network
BIC	Bayesian Information Criterion
EM	Expectation Maximization
m-state	Micro-State
irlba	Implicitly Restarted Lanczos Bidiagonalization Algorithm
T1D	Type I Diabetes Mellitus
T2D	Type II Diabetes Mellitus
GUI	Graphical User Interface

BIBLIOGRAPHY

[1] Paolo Mazzarello. A unifying concept: the history of cell theory. *Nature cell biology*, 1(1):E13–E15, 5 1999.

[2] Cole Trapnell. Defining cell types and states with single-cell genomics. *Genome research*, 25(10):1491–1498, 10 2015.

[3] Rory Stark, Marta Grzelak, and James Hadfield. RNA sequencing: the teenage years. *Nature reviews. Genetics*, 20(11):631–656, 7 2019.

[4] Zhi Wang, Mark Gerstein, and Michael Snyder. Rna-seq: a revolutionary tool for transcriptomics. *Nature Reviews Genetics*, 10(1):57–63, Jan 2009.

[5] Aviv Regev and et al. The human cell atlas. *eLife*, 6, Dec 2017.

[6] Single-cell transcriptomics of 20 mouse organs creates a tabula muris. *Nature*, 562(7727):367–372, Oct 2018.

[7] S. J. Altschuler and L. F. Wu. Cellular heterogeneity: do differences make a difference? *Cell*, 141(4):559–563, 2010.

[8] P. Van Galen and et al. Single-cell rna-seq reveals aml hierarchies relevant to disease progression and immunity. *Cell*, 176(6):1265–1281.e24, Mar 2019.

[9] Elana Der and et al. Single cell rna sequencing to dissect the molecular heterogeneity in lupus nephritis. *JCI Insight*, 2(9), May 2017.

[10] Bart Van De Sande and et al. Applications of single-cell rna sequencing in drug discovery and development. *Nature Reviews Drug Discovery*, 22(6):496–520, Apr 2023.

[11] Fuchou Tang, Constantin Barbacioru, Yiyue Wang, Eric Nordman, Christopher Lee, Ning Xu, Xinghua Wang, Jessica Bodeau, Brian B. Tuch, Arsalan Siddiqui, et al. mrna-seq whole-transcriptome analysis of a single cell. *Nature Methods*, 6:377–382, 2009.

[12] Gang Chen, Bo Ning, and Tao Shi. Single-cell rna-seq technologies and related computational data analysis. *Frontiers in Genetics*, 10:317, Apr 2019.

[13] ΜΑΛΕΣΙΟΥ ΕΥΘΥΜΙΑ. Δημιουργία πακέτου R ανάλυσης γονιδιακής έκφρασης κυττάρων που αναγνωρίζει τις κυτταρικές καταστάσεις και ανακατασκευάζει ρυθμιστικά δίκτυα για τις πιθανές μεταβάσεις καταστάσεων με μη-εποπτευόμενη μηχανική μάθηση. Master's thesis, National and Kapodistrian University of Athens, Athens, Greece, 2019.

[14] Arsenios P. Chatzigeorgiou. MLscAN: A flexible tool for single-cell data analysis pipelines and model selection using unsupervised machine learning methods. Master's thesis, National and Kapodistrian University of Athens, Athens, Greece, 2021.

[15] George A. Koliopanos. Flexible single-cell RNAseq data analysis pipelines using MLscAN. Master's thesis, National and Kapodistrian University of Athens, Athens, Greece, 2021.

[16] Ioannis D. Mystakidis. Enhancements of the mlscan package for the computational analysis of single-cell rna sequencing data. Master's thesis, National and Kapodistrian University of Athens, Athens, Greece, 2024.

[17] P. Tsakanikas, D. Manatakis, and E. S. Manolakos. Machine learning methods to reverse engineer dynamic gene regulatory networks governing cell state transitions. *bioRxiv*, 2 2018.

[18] Abdullah Yousif, Niveen Drou, Julian Rowe, Mohammed Khalfan, and Kristin C. Gunsalus. Nasqar: a web-based platform for high-throughput sequencing data analysis and visualization. *BMC Bioinformatics*, 21(1), Jun 2020.

[19] Azimuth. `https://azimuth.hubmapconsortium.org/`. Accessed: September 15, 2024.

[20] Robert Hillje, Pier G. Pelicci, and Luca Luzi. Cerebro: interactive visualization of scrna-seq data. *Bioinformatics*, 36(7):2311–2313, Nov 2019.

[21] Amir Khodadadi-Jamayran and et al. Icellr: Combined coverage correction and principal component alignment for batch alignment in single-cell sequencing analysis. *bioRxiv*, Apr 2020.

[22] K. Davie and et al. A single-cell transcriptome atlas of the aging drosophila brain. *Cell*, 174(4):982–998.e20, Aug 2018.

[23] Joe Cheng [aut] JJ Allaire [aut] Carson Sievert ORCID iD [aut] Barret Schloerke ORCID iD [aut] Yihui Xie [aut] Jeff Allen [aut] Jonathan McPherson [aut] Alan Dipert [aut] Barbara Borges [aut] Posit Software PBC [cph fnd] jQuery Foundation [cph] (jQuery library Winston Chang ORCID iD [aut, cre] and cph] (jQuery library; authors listed in inst/www/shared/jquery-AUTHORS.txt) jQuery UI contributors [ctb cph] (jQuery UI library; authors listed in inst/www/shared/jqueryui/AUTHORS.txt) Mark Otto [ctb] (Bootstrap library) Jacob Thornton [ctb] (Bootstrap library) Bootstrap contributors [ctb] (Bootstrap library) Twitter Inc [cph] (Bootstrap library) Prem Nawaz Khan [ctb] (Bootstrap accessibility plugin) Victor Tsaran [ctb] (Bootstrap accessibility plugin) Dennis Lembree [ctb] (Bootstrap accessibility plugin) Srinivasu Chakravarthula [ctb] (Bootstrap accessibility plugin) Cathy O'Connor [ctb] (Bootstrap accessibility plugin) PayPal Inc [cph] (Bootstrap accessibility plugin) Stefan Petre [ctb cph] (Bootstrap-datepicker library) Andrew Rowls [ctb cph] (Bootstrap-datepicker library) Brian Reavis [ctb cph] (selectize.js library) Salmen Bejaoui [ctb cph] (selectize-plugin-a11y library) Denis Ineshin [ctb cph] (ion.rangeSlider library) Sami Samhuri [ctb cph] (Javascript strftime library) SpryMedia Limited [ctb cph] (DataTables library) John Fraser [ctb cph] (showdown.js library) John Gruber [ctb cph] (showdown.js library) Ivan Sagalaev [ctb cph] (highlight.js library) R Core Team [ctb cph] (tar implementation from R) jQuery UI library), jQuery contributors [ctb. shiny: Web application framework for r. *CRAN*, 2024.

[24] Hadley Wickham. *Mastering Shiny*. O'Reilly Media, 2021.

[25] V. Guyader C. Girard C. Fay, S. Rochette. *Engineering Production-Grade Shiny Apps*. Chapman Hall/CRC, 2021.

[26] Yue J. Wang, Jonathan Schug, Kyoung-Jae Won, Chengyang Liu, Ali Naji, Dana Avrahami, Maria L. Golson, and Klaus H. Kaestner. Single-Cell transcriptomics of the human endocrine pancreas. *Diabetes*, 65(10):3028–3038, 6 2016.

[27] James Baglama and Lothar Reichel. Augmented implicitly restarted lanczos bidiagonalization methods. *SIAM journal on scientific computing*, 27(1):19–42, 1 2005.

[28] Luca Scrucca, Michael Fop, T Brendan Murphy, and Adrian E Raftery. mclust 5: clustering, classification and density estimation using gaussian finite mixture models. *The R journal*, 8(1):289, 2016.

[29] Vân Anh Huynh-Thu, Alexandre Irrthum, Louis Wehenkel, and Pierre Geurts. Inferring Regulatory Networks from Expression Data Using Tree-Based Methods. *PloS one*, 5(9):e12776, 9 2010.

[30] Ancheng Deng and Xiaoqiang Sun. Dynamic gene regulatory network reconstruction and analysis based on clinical transcriptomic data of colorectal cancer. *Mathematical biosciences and engineering*, 17(4):3224–3239, 1 2020.

[31] Isaac Engel, Grégory Seumois, Lukas Chavez, Daniela Samaniego-Castruita, Brandie White, Ashu Chawla, Dennis Mock, Pandurangan Vijayanand, and Mitchell Kronenberg. Innate-like functions of natural killer T cell subsets result from highly divergent gene programs. *Nature immunology*, 17(6):728–739, 4 2016.

[32] Tetsutaro Hayashi, Haruka Ozaki, Yohei Sasagawa, Mana Umeda, Hiroki Danno, and Itoshi Nikaido. Single-cell full-length total RNA sequencing uncovers dynamics of recursive splicing and enhancer RNAs. *Nature communications*, 9(1), 2 2018.