


Creating a package for single malignant cell simulation

Bachelor's Thesis

Georg Ye
georye@ethz.ch

Computational Cancer Genomics
ETH Zürich

Supervisors:
Prof. Valentina Boeva,
Josephine Yates

August 20, 2023

Abstract

This thesis presents a novel approach to simulating malignant single cell RNA sequencing data, contributing to the rapidly evolving field of bioinformatics with implications for cancer research.

The study commenced with a comprehensive examination of existing methodologies for non-malignant single cell RNA sequencing data simulation. A previous project from our laboratory successfully utilizes the Splatter [LZ17] package for these simulations, which is used in CanSig [JY22], a tool developed to analyze shared transcriptional heterogeneity of malignant cells of different genetic backgrounds. Specifically, Splatter generates de novo simulations with default parameters.

This project explored the use of the scDesign2 [TS21a] package, which was originally implemented in R. Unlike Splatter, scDesign2 endeavors to maintain the correlation between different genes. Recognizing Python's extensive adoption in bioinformatics, scDesign2 was transposed into Python, facilitating broader accessibility and so that it could be used in CanSig to have a comparison.

Just like the previous simulation, Copy Number Variations (CNVs) - a genomic feature prevalent in malignant cells - were integrated into the scDesign2 simulation to simulate malignant cells. The final Python based scDesign2 tool was then compared to the Splatter variation. Results from this study showed successful preservation of group effect in the simulations. However, the inclusion of CNVs proved challenging, especially when integrated similarly to Splatter, possibly due to the inherent differences between scDesign2 and Splatter. This warrants further investigation into optimizing the incorporation of CNVs into the simulation.

Acknowledgements

I would like to express my sincere gratitude to my advisor, Josephine Yates, for the continuous support of my bachelor thesis study and her immense knowledge. Her guidance helped me in all the time of research and writing of this thesis and showed me how to do science.
Then I thank Valentina Boeva for providing me the opportunity to contribute to her lab. A special note of thanks goes to the dedicated members of the lab, who shared their work with enthusiasm and revealed the and relentless efforts required to confront the challenges of cancer research.

Contents

1 Introduction

Single-cell RNA sequencing (scRNA-seq) allows researchers to investigate gene expression at the level of individual cells. The advent of scRNA-seq has provided unprecedented insights into cellular heterogeneity, cell differentiation, and the complex interplay of molecular pathways in various biological contexts. However, the analysis and interpretation of scRNA-seq data present unique challenges due to the inherent noise, sparsity, and technical variability associated with single-cell measurements.

1.1 Single-cell RNA sequencing

RNA sequencing (RNA-seq) is a powerful technique for analyzing gene expression. It works by converting RNA, extracted from cells or tissues, into sequenceable complementary DNA (cDNA). This process offers insights into gene expression levels, alternative splicing, post-transcriptional modifications, and transcriptome-wide profiles. It is particularly useful for identifying differentially expressed genes and studying gene expression patterns in various conditions, aiding our understanding of cellular functions and diseases. RNA-seq can identify cancer-related chromosomal rearrangements, disease-associated transcriptomic profiles, and potential predictive biomarkers.

However, RNA-seq is not without its challenges. High variability in RNA expression can limit sensitivity for low-expressed genes or isoforms. It is also subject to technical biases like sequencing depth bias and batch effects. The former refers to variations in the number of reads obtained for different genomic regions, while the latter arises from inconsistencies in sample handling, processing, or environmental conditions. These issues can complicate data analysis and interpretation [AH17].

1.2 Why do we need to simulate single cell RNA sequencing data

Simulations are pivotal in bioinformatics, providing a method to test the efficacy of analysis techniques against a "known truth". This allows us to understand the limitations and assumptions of a method, providing an environment where multiple datasets with varying parameters can be swiftly generated at a low cost.

One might wonder why simulations are needed when real datasets are already out there, especially when a simulation needs a reference dataset. Real RNA sequencing data, while valuable, is not considered the "ground truth" due to potential distortions from biological variability, technical noise, and batch effects. These factors can

cloud the identification of genuine differences between samples or conditions.

In contrast, simulated data offers a "known truth" for comparison, allowing us to assess the performance of analysis methods without the confounding influences. It is crucial to remember, however, that simulated data is not a perfect mirror of real biological data, and it might not capture all the complexities of biological systems.

Therefore, it is essential to validate analysis methods using both simulated and real data, keeping in mind the limitations and assumptions of each approach.

For malignant scRNA-seq data, simulations can assist in the development of precise analysis methods for discerning gene expression patterns and cellular heterogeneity related to cancer. By creating realistic scRNA-seq datasets, researchers can evaluate the performance of various analysis methods against a known ground truth [HLC23].

2 Literature

A plethora of simulations for non-malignant cells already exists in the field. In the following section, we shall delve into a selection of these simulation methods, exploring their unique attributes and potential for adaptation. Concurrently, we will explore various strategies on how these existing models may be effectively modified to simulate malignant cells.

2.1 Simulations

According to the scDesign2 paper a simulator should have 6 properties:

1. The simulator is adaptable, as it can be trained by real data to suit various experimental protocols and biological conditions.
2. It preserves genes and their expression level distributions in synthetic data, essential for benchmarking differential gene expression analysis.
3. The simulator captures gene correlations, maintaining a similar structure to real data, crucial for benchmarking multi-gene analyses like cell dimensionality reduction, clustering, rare cell type detection, and cell trajectory inference.
4. It generates synthetic data with varying cell numbers and sequencing depths under the same biological conditions, aiding experimental design and computational method robustness benchmarking.
5. The simulator is transparent, with easily understood and adjustable model parameters. This is crucial for model diagnostics, customized simulation, and identifying well-captured or missed gene correlations.
6. It is computationally and sample efficient, not requiring expensive hardware, excessive computational time, or a large number of real cells for training, making it user-friendly and adaptable to full-length protocols [TS21a].

There are many packages which can simulate single cell RNA sequencing data and most use a statistical distributions, such as the Negative Binomial (NB), to create synthetic data similar to real datasets. The NB distribution is particularly popular in simulating scRNA-seq data due to its ability to capture overdispersion. This property, where the observed variance in gene expression counts exceeds the mean, is frequently observed in scRNA-seq data. The NB distribution, with an additional dispersion parameter for variance modeling, provides more flexibility and precision than simpler models like the Poisson distribution, which assumes the mean and variance to be equal [YC21]. The result of both models is a matrix of counts, where one axis represents the cell and the other the gene. The value of each cell is the number of reads that were simulated for that gene in that cell.

2.1.1 Splatter

Splatter [LZ17] offers six simulation models where five are implementations of previously published models. Only the in house developed one, Splat, will be discussed here. Splat is a proprietary model designed to mimic features found in real scRNA-Seq data, such as high expression outlier genes, variable sequencing depths, trended gene-wise dispersion, and zero-inflation. These characteristics are represented using hyperparameters estimated from actual data within a Gamma-Poisson (mathematically identical to the Negative Binomial) hierarchical model framework.

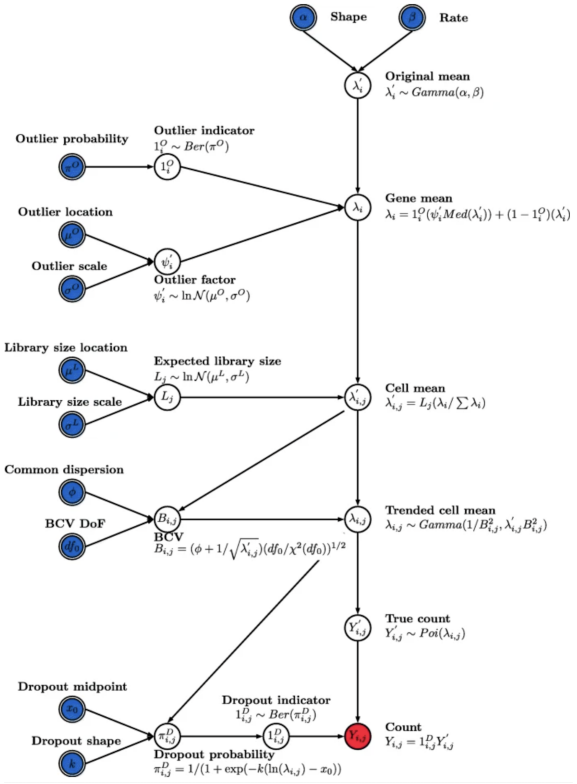


Figure 1: Splat simulation model. Double borders indicate input parameters. The final output is shaded in red. [LZ17] (Fig. 1)

In the Splat model, the mean expression level for each gene is simulated from a gamma distribution, with modifications introduced for expression outliers and a mean-variance trend enforcement. The extreme expression levels, not always captured by the gamma distribution, are addressed by the introduction of a high expression outlier probability, π^O. Outliers are incorporated by replacing the simulated mean with the median of the simulated gene means, adjusted by an inflation factor.

Library sizes are modeled with a log-normal distribution, allowing the adjustment of gene means for each cell proportionally. A significant mean-variance trend observed in RNA-Seq data is enforced in Splat through the simulation of the biological coefficient of variation (BCV) from a scaled inverse chi-squared distribution, with the scaling factor dependent on the gene mean.

The high proportion of zeros observed in scRNA-seq data, often attributable to technical dropout, is also modeled in Splat through a logistic function defined by a midpoint and a shape parameter. This function helps simulate the relationship between a gene's mean expression and its zero count proportion, effectively replacing some simulated counts with zeros.

The final output of the Splat model is a count matrix, with axes representing the cell and the gene. Splat can simulate more complex scenarios including groups, batches, and paths for continuous differentiation or lineages. For example, to simulate paths,

Splat defines expression levels of start and end cells, assigning cells to intermediate steps, and models complex, non-linear gene expression changes. Users can control the proportion of non-linear genes and their variability, offering more versatility.

A comparison study [YC21] found that Splat performs well in most scenarios, although it struggled with complex datasets like CAMP and ENGEL. The results emphasized the importance of choosing the right simulation model for a given dataset, a process simplified by the Splatter framework. The study also highlighted the value of examining each simulated gene against known distributions.

2.1.2 Other simulation methods

Another simulation named scVI [RL18] works as follows: The observed expression x_{ng} of each gene g in a cell n is modeled as a sample drawn from a ZINB distribution $p(x_{ng}|z_n, s_n, l_n)$ where s_n are batch annotations, l_n is an one-dimensional Gaussian which represents nuisance variation and z_n is a low-dimensional vector of Gaussian representing the remaining variation which should better reflect biological differences between cells.
Each cell is represented in a low-dimensional latent space. A neural network maps the latent variables to the parameters of the ZINB distribution. The latent variables are mapped to parameters of the ZINB distribution.

Another paper [Ger20] proposes the idea of instead of generating data from theoretical models, to add signal to real datasets. This method ensures that the simulated data exhibits realistic attributes found in actual data, which are often not captured by theoretical models. Traditional evaluation pipelines involve simulating data based on theoretical models and comparing methods on the simulated data, which can be useful for understanding performance in ideal scenarios. However, real data frequently deviate from these ideal scenarios and exhibit unwanted variation beyond model assumptions. A study [YC21] benchmarked 12 simulations where the top five performing ones used statistical approaches. The underlying models where mostly Zero Inflated Negative Binomial (ZINB), Negative Binomial, and Poisson distributions.
While the Poisson distribution is a widely used basic model for count data, its assumption of equal mean and variance often doesn't hold true for scRNA-seq data. This tends to result in under-dispersion and a poor fit when the variance is higher than the mean.
In contrast, the Negative Binomial distribution offers a broader approach. It is essentially a generalization of the Poisson model, incorporating an additional parameter to account for overdispersion - a common trait in scRNA-seq data where the variance surpasses the mean. This model has found significant use in bulk RNA-seq and subsequently extended to scRNA-seq.
To account for the surplus zeros often observed in scRNA-seq data due to biological variability and technical noise such as dropouts, the ZINB distribution is employed. This model presumes the data to be a blend of counts distributed in accordance

with the Negative Binomial distribution, complemented with additional zeros.

Intriguingly, despite the profound impact of deep learning methodologies across diverse domains, cscGAN [MM20], which is a deep learning model designed for scRNA-seq data simulation, showed only a middling performance relative to other models. This could potentially be attributed to the high cell count necessary to train the deep neural network within their model. But since the start of this project a storm of generative AI tools have been released and some might be of use for simulating scRNA-seq data.

2.2 Malignant adaptation

Simulating malignant scRNA-seq data is different from simulating normal scRNA-seq data due to several factors which influence the gene expression profiles and the complexity of the data. Malignant cells exhibit a higher degree of cellular heterogeneity compared to healthy cells. Cancer cells can have various genetic mutations, epigenetic alterations, and different stages of differentiation. This increased heterogeneity makes it challenging to accurately simulate malignant scRNA-seq data.

2.2.1 Point Mutation

Point mutations, DNA changes replacing one nucleotide with another, result in silent mutations (which do not affect the protein produced), missense mutations (which alter the amino acid sequence of the protein) and nonsense mutations (which create a premature stop codon, often resulting in nonfunctional or truncated proteins). An instance is the Sickle Cell disease-causing mutation. A 2013 study [CK13] scrutinized 3281 tumors across 12 types, finding 127 significantly mutated genes. The mutation count per tumor varied, with the highest averaging around six mutations (e.g., UCEC) and the lowest about two (e.g., AML). Remarkably, 93% of samples had at least one non-synonymous mutation in these genes. TP53 was the most frequently mutated gene. Clustering analysis showed 72% of tumors shared similar mutations with the same tissue type. However, notable mutation diversity was observed within specific cancer types, such as breast, endometrial, and kidney cancers. In these cancers, different sets of mutations drove distinct groups. This study underscores the complexity and variability of gene mutations across different cancer types, contributing to our understanding of cancer genetics.

2.2.2 Copy Number Variations

Copy Number Variations (CNVs)/Copy Number Alterations (CNAs) refer to changes in the number of copies of a particular gene in a genome. These variations can occur

as a result of duplications, deletions, or rearrangements of large segments of DNA. In many cancers, certain genes are amplified (increased in copy number) or deleted (reduced in copy number), leading to an increase or decrease in the products (usually proteins) they encode. For example, amplifications of oncogenes (genes that have the potential to cause cancer) can lead to overproduction of proteins that promote cell growth and proliferation, contributing to cancer development. Conversely, deletions in tumor suppressor genes (which normally help regulate cell growth and prevent cancer) can lead to a lack of these important regulatory proteins, also contributing to cancer development. An increased copy number usually increases the expression of the gene, while a decreased copy number decreases the expression of the gene but genes outside the changed region are also affected.

One paper [RJ11] presents a method to model the impacts of Copy Number Aberrations (CNAs) in tumours using gene expression data from the Cancer Genome Atlas (TCGA). The study defines a mathematical representation to link changes in copy numbers to changes in gene expression levels. This relationship is modelled as a system of linear equations at steady-state, capturing the influence of a gene's own CNA and its effect on other genes.
Two main network representations are proposed: the transcriptional interaction network (A) and the CNA-driven network (G). The former captures direct transcriptional interactions, adjusted for the influence of a gene's own CNA.
For example, let's consider a simplified scenario with three genes: Gene A, Gene B, and Gene C. In this example, Gene A directly regulates the expression of Gene B, and Gene B directly regulates the expression of Gene C. Matrix A would capture these direct interactions, such as the effect of Gene A on Gene B and Gene B on Gene C.
The latter models how the effects of CNA perturbations propagate through the transcriptional network to produce steady-state responses. The CNA-driven network captures cascading effects, allowing identification of key disease-driving CNAs and their downstream targets.
Now, let's say there's a CNA affecting Gene A. This CNA could indirectly influence the expression of Gene C through the regulation of Gene B. Matrix G would capture this indirect effect, showing how the CNA in Gene A propagates through the system and influences other genes, like Gene C, even though there is no direct interaction between Gene A and Gene C.

3 Method

As mentioned a simulation was already written for the CanSig paper. To evaluate that simulation it was proposed to build another simulation which uses a different underlying simulator. Although Splatter can use a reference set, the rewritten package in Python for the paper does not have that capability.

It was decided that scDesign2 should be used for this simulation. This model needs a reference set to estimate the marginal distributions of genes and leverages a copula to model the joint gene-expression distribution, thereby capturing complex correlations amongst genes.

The decision to utilize scDesign2 was influenced by its good performance in various benchmarks [YC21], where it compared favorably with Splatter and other simulators.

In the forthcoming sections, we delve deeper into the inner workings of the scDesign2 simulator, illustrating how it models the distribution of gene expression to produce realistic synthetic single-cell RNA sequencing data.

3.1 ScDesign2's statistical framework

ScDesign2 treats cells of different cell types independent. Therefore given a count matrix $X \in \mathbf{N}^{n \times p}$, where n are the number of cells of that specific cell type and p the number genes, the model parameters will be estimated.
Jointly for the p genes, it is assumed that each cell $X_{.j}$ independently follows a p-dimensional distribution F, which is specified by a copula later. Marginally for each gene X_{ij}, it is assumed that it independently follows a univariate count distribution F_i. F_i can either be a ZINB, NB, Poisson, or Zero Inflated Poisson (ZIP) distribution. For example if F_i is the ZINB distribution then $X_{ij} \overset{ind}{\sim} ZINB(p_i, \psi_i, \mu_i)$. And then the parameters p_i, ψ_i and μ_i are estimated, where p_i is the probability of a zero component (zero inflation), ψ_i the scale/dispersion and μ_i the mean.
The Poisson, ZIP, and NB distributions are three special cases of the ZINB distribution, where $p_i = 0$ for Poisson and NB, and $\psi_i = \infty$ for Poisson and ZIP.

Either one manually chooses which distribution to use for all genes or one lets the model systematically select one for each gene. How the automatic mode determines which model to use is done by first looking if the mean of the gene is greater than the variance, and if that is the case either the Poisson or ZIP distribution are used and the parameters are estimated using MLE.
Next, a likelihood ratio test is performed, comparing the goodness of fit of the Poisson and the ZIP model. This is done by computing a χ^2 (chi-square) value from the log likelihoods of both models and finding the p-value (significance level). If this p-value is less than a specified cutoff (0.05 by default), the ZIP model is preferred.

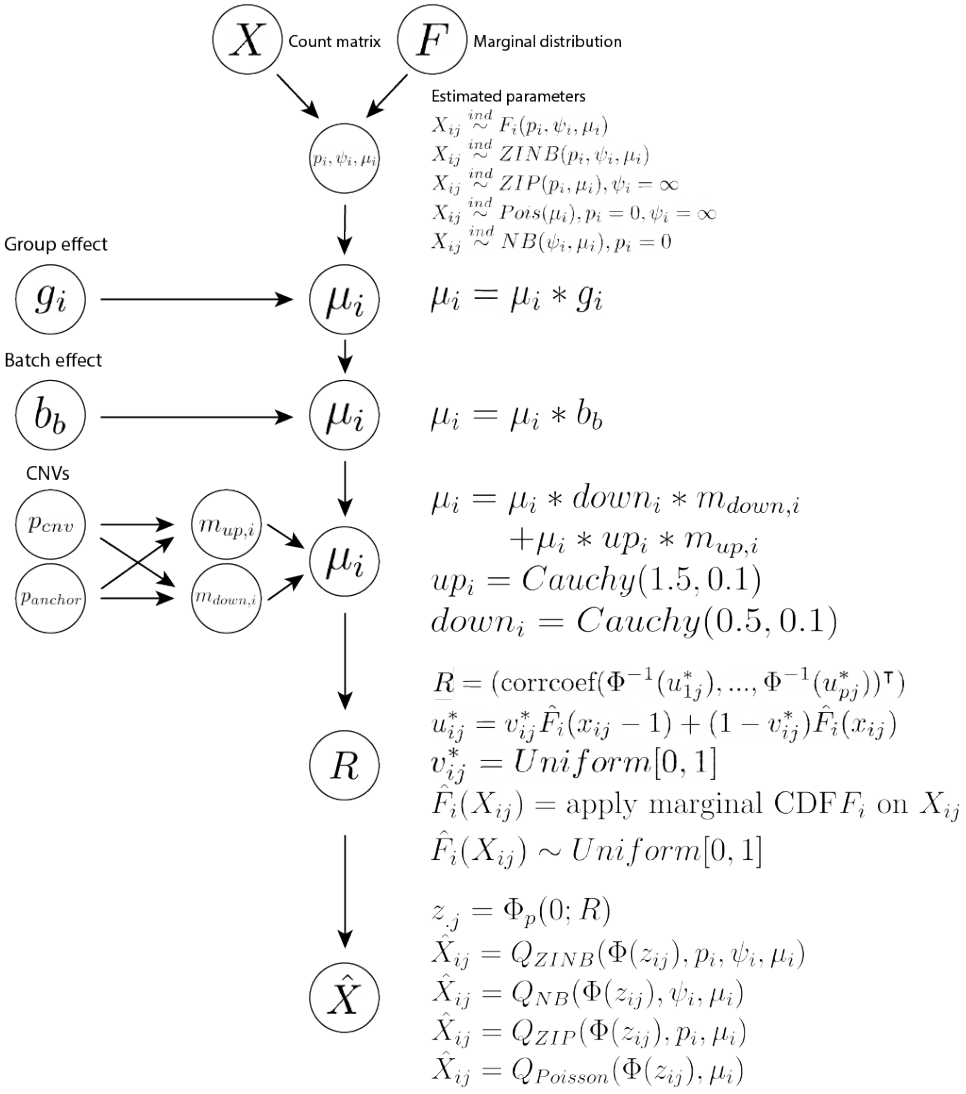


Figure 2: ScDesign2's flowchart This is a simplified chart of scDesign2's simulation model. A reference count matrix and optionally a preferred marginal distribution is provided. From the count matrix, the parameters: p_i (zero inflation), ψ_i (scale/dispersion) and μ (mean) are estimated by fitting the counts to a distribution (ZINB, ZIP, Poisson, ZIP) using MLE. Then its possible to add a group effect, so that cells of the same cell type or grouped together. It is also possible to add a batch effect, so that cells of the same batch are grouped together. For each batch a copy of the parameters is made and those are adjusted with the batch effect. Afterwards the CNVs get added and the covariance matrix for the Gaussian copula gets computed. At last the synthetic count matrix is generated by sampling values from the copula.

If not, or if the ZIP model fails to fit, it defaults to using the Poisson model. The dispersion parameter is set to 0 for both distributions.

If the variance is greater than the mean, a NB model and ZINB is fitted to the data, again by using MLE. These model is similar to the Poisson model but it includes a parameter to model overdispersion (greater variance than the mean), which is common in count data like gene expression.
As before, a likelihood ratio test using the χ^2 value from the log likelihoods of both models is performed, to determine which model fits better.

The model fitting is done by using the Python package statsmodel (v0.14.0) [Sea10]. When comparing the estimated mean, dispersion/scale and zero inflation between our simulation and the R simulation, the estimated mean is always the same while the scale and zero inflation parameter differ.
Note that by default the scale parameter for statsmodel is set to 1 by default. One has to set the scaletype parameter to "x2" (Pearson's chi-square) as this is the one used in the original scDesign2 implementation.

After estimating the marginal distributions of the p genes, i.e., $F_1, ..., F_p$ we move on to add the other effects before calculating the covariance matrix [TS21a].

3.1.1 Group effect

The original scDesign2 implementation does not provide a way to simulate group effect. In this implementation the group effect is simulated by using the method which was used in Splatter (and already implemented in the simulation used for CanSig). Following parameters need to be set:

- p_de_list: the probability of a gene to be differentially expressed
- p_down_list: the probability of a gene that is differentially expressed to be downregulated
- de_location_list: the location factor for each cell type for the differential expression parameters
- de_scale_list: the scale factor for each cell type for the differential expression parameters

When there are 5 groups/cell types, each number in the list corresponds to a specific group. For instance, the first number in the list pertains to group 1. Each group has a proportion of genes (p_de) that are differentially expressed, and among these, a proportion (p_down) is downregulated. A Binomial distribution is used to decide which genes are differentially expressed and which of these are downregulated. It

then generates log-fold change values from a log normal distribution with mean set to de_location and variance set to de_scale. These values are then multiplied to the DE gene means. If a gene is downregulated, the fold-change is inverted ($\frac{1}{fold-change}$).

3.1.2 Batch effect

There is no way to simulate batch effect in the original implementation either so it also uses the method Splatter uses. The batch effect is done in the same fashion as the group effect just instead of the p_de_list, p_down_list, de_location_list and de_scale_list parameters, the parameters pb_de_list, pb_down_list, bde_location_list and bde_scale_list are used (each value in the list affects a given batch).

3.1.3 CNVs

The procedure on how CNVs are simulated was already implemented in the simulation used for CanSig, it was merely adjusted so it can be used with scDesign2.
To apply CNVs, specific details about the genes or genome must be provided. This includes the name of each gene, the chromosome where it is located, and its specific position on that chromosome.
Then several parameters must be established, including: the chromosomes where gains and losses for a clone or subclone may occur; the likelihood of a chromosomal region being gained or lost for both the clone and its subclones; a set of anchor genes, and the probability of gaining a small region around an anchor.
Other parameters are the number of batches simulated, the number of possible malignant states of a batch, the number of possible subclones, and the number of healthy and malignant cells to simulate.
More parameters need to be set but those will be introduced later.
For each batch the subclones are then generated. Using the set parameters each chromosome then has a probability to get mutated, where the length of the region can be controlled by setting a minimum and maximum mutation length. Afterwards the child subclones are generated by inheriting the changes of the parent and repeating the process just mentioned, but it is not possible to have another mutation of the same type (gain/loss) on a chromosome which has already been mutated.
Then for each batch the probability of a cell belonging to which program is generated. This is done with the help of the Dirichlet distribution. At first the alphas for the Dirichlet distribution are set by a parameter. Then by looking at the gains/losses around the anchors, the alphas get adjusted (how they change the alphas can also be set via parameters). Furthermore it is possible for a program and subclone to dropout (probability is set to 0). Finally, each cell get assigned a program and a subclone (the probability to which subclone a cell belongs is also computed with the Dirichlet distribution with the alphas set by parameters).

The computed means of the genes are then changed, although the means of lowly

and highly expressed genes are changed differently. A gene counts as overexpressed if its mean is in the top 30%. If the gene is identified as overexpressed, a value gets sampled from the truncated Cauchy distribution with the value lying in between 0 and 10. The mean of the gene then gets multiplied with that value. The location and scale parameter of the distribution depend on whether the gene is located in a region which is gained or lost. If it is gained, the mean of the Cauchy distribution is set to 1.5 and the scale to 0.1 and otherwise the mean is set the 0.5.

On the other hand, if a gene counts as lowly expressed and is gained, we utilize a Gaussian Mixture Model (GMM).
The GMM used in this context is a two-component mixture, denoted by the mixing coefficients (weights) $\pi = [0.2671, 0.7329]$. These weights specify the probabilities of each Gaussian being chosen to generate a sample. The means and standard deviations of the two Gaussian distributions are $\mu = [3.0553, 0.9422]$ and $\sigma = [2.2546, 0.6179]$ respectively. Sampling from this GMM involves two steps. Initially, a categorical variable is generated, which decides from which Gaussian the sample should be drawn. This variable is sampled from a categorical distribution where the categories are the two components of the GMM and the probabilities are given by π.
Following the selection of the Gaussian component, a sample is drawn from a truncated normal distribution corresponding to the chosen Gaussian. The truncation is applied to limit the range of values between 0 and 10. The parameters of this distribution, namely the location (mean) and scale (standard deviation), are taken from the μ and σ values of the chosen Gaussian component.
The very same thing is done when a gene experiences a decrease but with the means set to $\mu = [2.1843, 0.5713]$ instead.

3.1.4 Computing the covariance matrix

After choosing the marginal distributions, i.e., $F_1, ..., F_p$ and estimating the parameters of the p genes and adjusting the means, a copula is used to model the joint p-dimensional distribution F.
A copula is defined as a joint cumulative distribution function (CDF), $C(.) : [0, 1]^p \rightarrow [0, 1]$, which includes p uniform marginal distributions on [0,1].
That is, C is the CDF of a random vector $U = (U_1, ..., U_p)^\intercal \in [0, 1]^p$, with
$U_i \sim$Uniform$[0, 1]$, $i = 1, ..., p$.
A gene X_{ij} in the cell count vector $X_{\cdot j}$ may not follow the Uniform $[0, 1]$ distribution, but one can transform X_{ij} by applying the marginal CDF F_i so that $F_i(X_{ij}) \sim Uniform[0, 1]$ (probability integral transform).

We were a bit skeptical about this part. The probability integral transform is primarily designed for continuous random variables. When one takes the CDF of such a continuous random variable, the resulting distribution is uniform on [0, 1]. However, for discrete random variables, applying the CDF as a transformation does not yield a perfect uniform distribution on [0, 1]; it merely maps the values of the random

variable into this range. We suspect that, in the context of copula modeling, the objective is to map the range of the discrete random variable to [0, 1], even if the result is not a perfectly uniform distribution [vF22].

Moving on, this allows us to express the joint distribution F in terms of the copula C and the marginal distributions $F_1, ..., F_p$ as follows:

$$F(x_{ij}, ..., x_{pj}) = C(F_1(x_{1j}), ..., F_p(x_{pj})).$$

One important thing to note is that this decomposition is unique only for continuous distributions. For discrete distributions, the copula still exists, but it may not be unique (according to Skalar's theorem). And since x_{ij} is discrete, taking the marginal CDF $F_i(x_{ij})$ will also be discrete.
Therefore the technique of distributional transform is used: generate a random variable V_{ij} ∼Uniform$[0, 1]$ independently for $i = 1, ..., p$ and $j = 1, ..., n$; then define U_{ij} as: $U_{ij} = V_{ij}F_i(X_{ij} - 1) + (1 - V_{ij})F_i(X_{ij})$. By applying this transformation, scDesign2 converts their initial gene counts, denoted X_{ij}, into uniform variables U_{ij} suitable for the copula model. In essence, for a discrete random variable X_{ij} characterized by its CDF F_i, this method spreads the non-zero probability mass of X_{ij} at each specific value x uniformly over the interval [x, x+1). As a result, the discrete CDF F_i is transformed into a continuous one.

The copula chosen in scDesign2 is the Gaussian copula and therefore we get:

$$F(x_{1j}, ..., x_{pj}) = \Phi_p(\Phi^{-1}(u^*_{1j}), ..., \Phi^{-1}(u^*_{pj}); R).$$

The covariance matrix R is set to corrcoef$((\Phi^{-1}(u^*_{1j}), ..., \Phi^{-1}(u^*_{pj}))^\intercal)$ where the correlation function corrcoef calculates the Pearson product-moment correlation coefficients, and Φ^{-1} denotes percent point function, or the inverse of the cumulative distribution function (CDF), of the standard normal distribution.
Also $u^*_{ij} = v^*_{ij}\hat{F}_i(x_{ij} - 1) + (1 - v^*_{ij})\hat{F}_i(x_{ij})$, $v^*_{ij} = Uniform[0, 1]$, and $\hat{F}_i$ is the CDF of the marginal F_i.

The number of genes for which copula correlations need to be estimated is controlled by filtering out the genes whose zero proportions exceed a user-specified cutoff (0.8 by default). That is done so the matrix does not get too big.

A variant without the copula is also offered. In this variant, the primary divergence lies in the assumption of independence amongst the marginal distributions of the p genes, denoted as $F_1, ..., F_p$.

3.2 Generation of the Count Matrix in ScDesign2

In this section p denotes the p parameter of a given distribution (for example the success probability in each experiment for the NB distribution) and p_i the zero inflation.

ScDesign2 first estimate proportions of K cell types from the real scRNA-seq count matrix X, for which the number of cells of a given cell type k are denoted as $n^{(k)}$, their total read count as $N^{(k)}$, the total number of reads of all cell types as N and the total number of cells as n. The number of reads is the total count of RNA sequences that have been identified. So given a count matrix, if we look at gene x and cell y, the value in that cell is how many times x was read in cell y. The cell type proportions are notated as $\pi = (\pi^{(1)}, ..., \pi^{(K)})$ where $\pi^{(k)} = \frac{n^{(k)}}{n}$.

The user can freely choose the number of generated cells for each cell type $n^{(k)'}$ by setting the cell_sample parameter in the code to false, or $n^{(k)'}$ can be sampled from a multinomial distribution, i.e., $(n^{(1)'}, ..., n^{(K)'})^\intercal \sim \text{Multinomial}(n^{'}, \pi)$. The total number of cells generated and the total number of expected reads for all cell types are set by the user, denoted as $n^{'}$ and $N^{'}$. The expected number of reads assigned to cell type k should then be: $N^{(k)0} = \frac{N^{(k)}}{n(k)} n^{(k)'}$. In the generated data in this work the cell_sample parameter was set to false.
Since the expected total number of reads is constrained to $N^{'}$, $N^{(k)0}$ needs to be rescaled to: $N^{(k)'} = \frac{N^{(k)0}}{\sum_{s=1}^{k} N^{(s)0}}, k = 1, ..., K$
Therefore the scaling factor is: $r = \frac{N^{(k)'}}{N^{(k)0}} = \frac{N^{'}}{\sum_{s=1}^{k} N^{(s)0}}$, which does not depend on the cell type and is then used to rescale the mean parameter of every gene. Note that r is set to 1 for all cell types if cell_sample is set to false.

After rescaling the means, $n^{(k)'}$ vectors (cells) are drawn, denoted as $z_{.j}^{(k)'} \in \mathbf{R}^p; j = 1, ..., n^{(k)'}$, independently from our copula $\Phi_p(0; R^{(k)})$ ($R^{(k)}$ denotes the covariance matrix of cell type k for the subset of genes mentioned in the previous section). In words, $n^{(k)'}$ vectors are drawn from the cumulative distribution function of the standard normal distribution with a mean vector of zeroes and covariance matrix $R^{(k)}$.
Then $z_{ij}^{(k)'}$ is converted to $x_{ij}^{(k)'}$ by setting $z_{ij}^{(k)'}$ to be the $\Phi(z_{ij}^{(k)'})$-th quantile of $\hat{F}_i$, i.e. $ZINB(p_i, \psi_i, r\mu_i)$ (or Poisson, ZIP, and NB depending on the estimated parameters). Some packages (like statsmodels) only allow to set a p and not a μ for a distribution, then $p = \frac{\psi}{\psi+\mu}$.

For genes which are not captured in the covariance matrix the following is done: generate a number from the Binomial distribution with parameters $n = 1$ and $p = 1 - p_i$ (p_i denotes the zero inflation) and then either generate a number from a Poisson or NB distribution.
It is possible to set the marginal distribution of the genes to follow a Gamma distribution by setting the marginal parameter to 'Gamma'. The synthetic gene expression counts are generated by first simulating a binary random variable from a Binomial distribution, and then multiplying this by a random variable from a Gamma distribution.

Doing this for all K cell types then gets us the generated count matrix.

3.3 Preprocessing

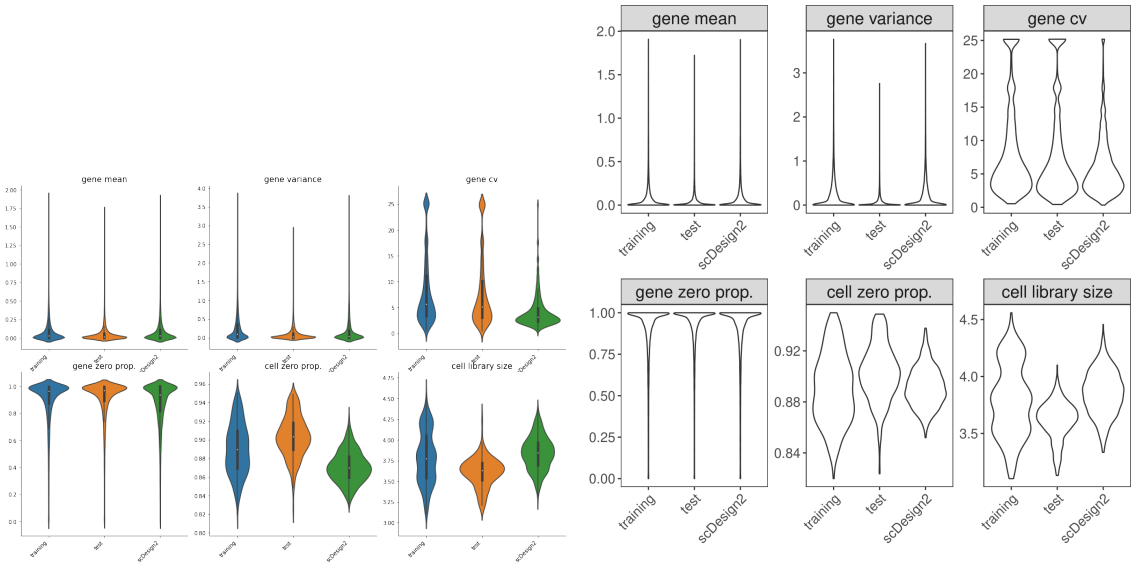
ScDesign2 expects a count matrix with the genes on the rows and the cells on columns. Cells of the same type should have the same label. When loading data into pandas [pdt23], pandas will automatically append a number on repeating indexes. That is when multiple cells have the index Stem, one will stay Stem and the other will be changed to Stem.1, Stem.2, ... and so on. So after importing the data it is important to fix this issue.

In the vanilla scDesign2 implementation evaluation all genes available in the dataset are used but in the extension with group effect, batch effect and CNVs only 5000 genes are used. It was set to 5000 because in the Splatter variation the simulation performed well with 5000 genes. How the 5000 genes are selected, and how they affect the simulation will be discussed in the Evaluation section.

4 Evaluation

4.1 The basic scDesign2 model

In this part our implementation will be compared to the original one. To compare them we follow the official tutorial [TS21b] and see if we get the same results. We first fit and simulate a single cell type. The reference dataset used is a single cell RNA-seq dataset that profiles the transcriptome of mouse small intestinal epithelium. The GEO number for that dataset is GSE92332. The only change that is made to the method in the tutorial is that instead of sampling the training and validation set randomly, we take the first half as training set, and the other half as test set. This is done so we have the same training and test set for the R and Python code.
We use the training set as the reference set and simulate a count matrix. Then violin plots for the gene mean, gene variance, gene coefficient of variation, gene zero proportion, cell zero proportion and cell library size are constructed. The data plotted are from cells of type Stem (Figure 3) and Goblet (Figure 4). The violin plots for Tuft, TA.Early, Enterocyte Progenitor and Enterocyte Progenitor Early cells can be found in the supplementary section (Figures 27, 28, 29, 30).

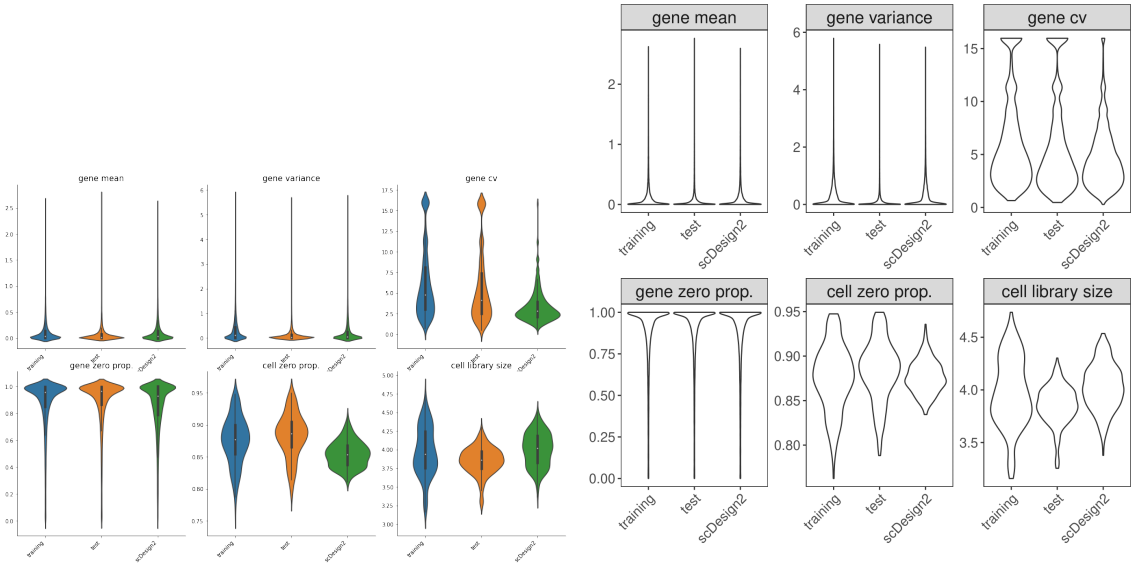


(a) Python: Stem cell type

(b) R: Stem cell type

Figure 3: Violin plots of the training set, test set and simulated set of Stem cells from the R and Python package

One can see that the Python implementation does not do too well when comparing it to the test set, just like the R implementation. But one should actually not compare the Python implementation to the test set but to the R implementation since that is the one we are trying to mimic. Therefore the count matrix of the R implementation was imported into Python so they can be compared side by side. Again the plots for the cell types Stem (Figure 38) and Goblet (Figure 6) are plotted here and the



(a) Python: Goblet cell type

(b) R: Goblet cell type

Figure 4: Violin plots of the training set, test set and simulated set of Goblet cells from the R and Python package

other four can be found in the supplementary section (Figures 31, 32, 33, 34).

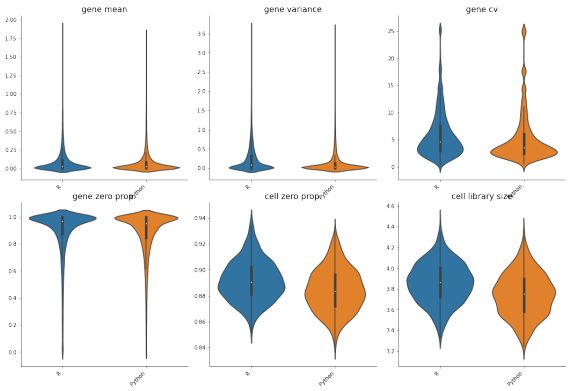


Figure 5: Violin plots of Stem cells

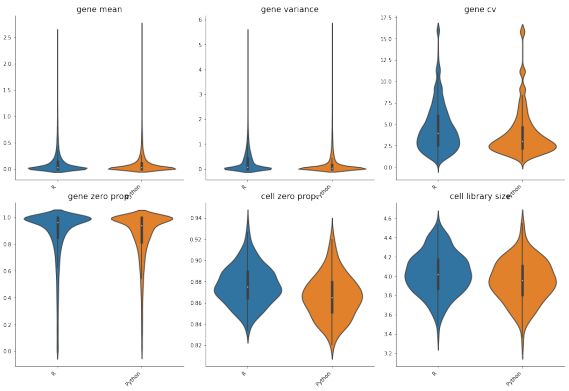


Figure 6: Violin plots of Goblet cells

Upon inspection, we find the distributions of gene mean, gene zero proportion, cell zero proportion and cell library size to be notably comparable between the two analyses. However, differences can be seen in the distributions of gene variance and gene coefficients of variation. This is due to the scale being different when estimating the parameters because the packages used to estimate the parameters work in slightly different ways in Python and R. Despite these variances, we thought the results were close enough and decided to carry on with our project.
The means distributions look very similar but rather than just relying on the visual plots it is better to compare them using actual numbers. So the square error of each gene between the implementations is calculated and the histograms of the errors are plotted in figure 7 and figure 8.

There are instances where the error magnitude exceeds one, which may initially raise

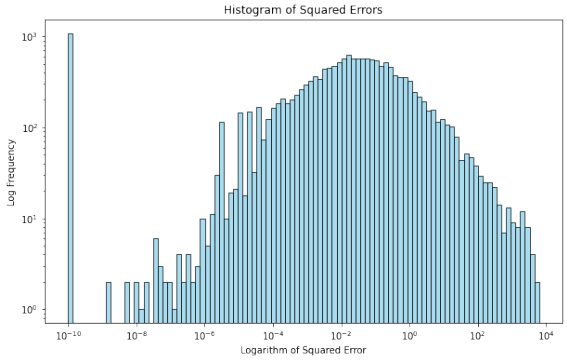


Figure 7: Stem: Square error of means histogram

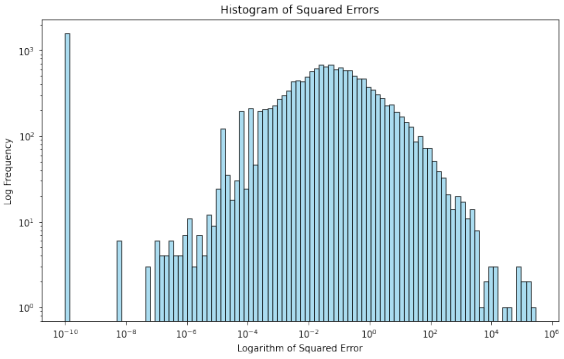


Figure 8: Goblet: Square error of means histogram

concerns. However, on closer inspection, we observed that this typically occurs for genes with an already high mean value. For instance, if the mean is around 20 in one case, it might differ slightly and be 16 or 24 in the other case.
While the discrepancy is noticeable, it is relatively insignificant given the overall high mean value, thus it is not a major concern and can be disregarded. Moreover, it is important to note that there were no instances where a gene was deemed insignificant in one implementation but significant in the other. For instance, the gene with the highest error in Stem cells had a mean of 73.35 in the R count matrix, while the Python count matrix yielded a mean of 56.95. For the Goblet cell type, the means were 394.67 and 527.73 in the R and Python implementations, respectively.
In conclusion, the Python clone performed satisfactorily when benchmarked against the R implementation, thus proving its efficacy and reliability.

The second test is done by fitting and simulating multiple cell types, those are: Stem, Goblet, Tuft, TA Early, Enterocyte Progenitor and Enterocyte Progenitor Early. They test how well their simulated data can mimic real data by projecting the cell to low dimensional plots by using PCA and t-SNE (Figure 10).

When comparing the plots of the simulations from R and Python they look alike. We can see that the PCA plots look very similar and although the t-SNE plots do not look identical one can see their resemblance. Also, note that t-SNE is non-deterministic.
These results give us confidence in the Python version. It is doing well in our tests so far, so we are ready to move on to the next testing steps.

4.2 Comparing the scDesign2 variation to the Splatter variation

The Uniform Manifold Approximation and Projection (UMAP) algorithm was utilized on the simulated gene expression data to inspect whether it could disclose any noteworthy groupings or clusters. In particular, we were interested in observ-

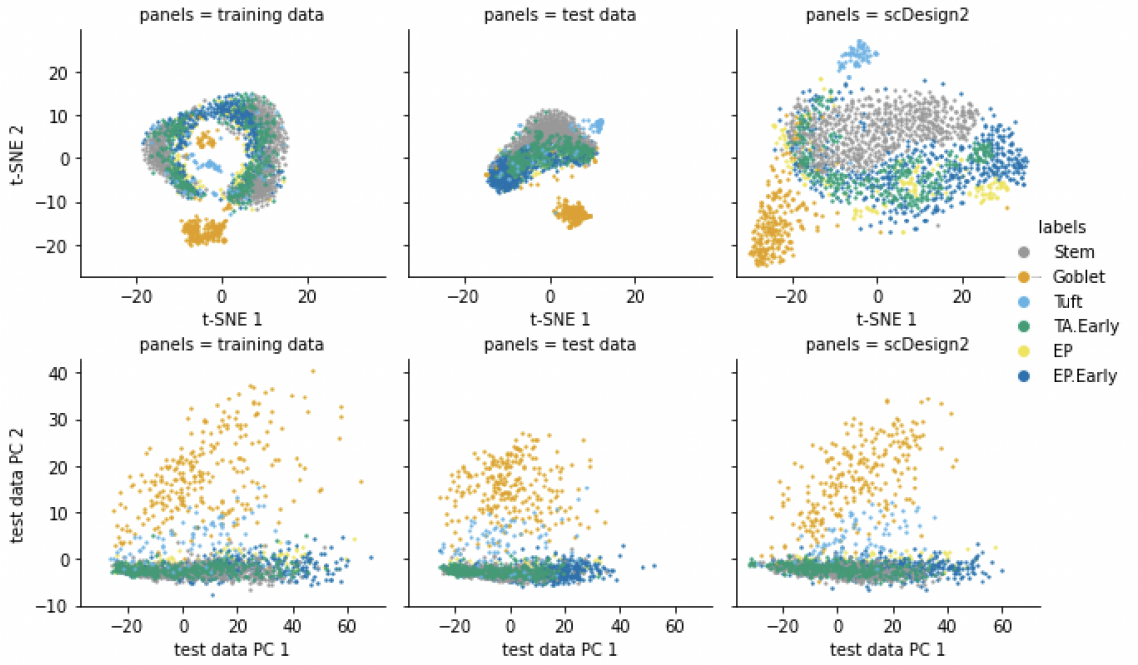


Figure 9: t-SNE and PCA of the Python implementation

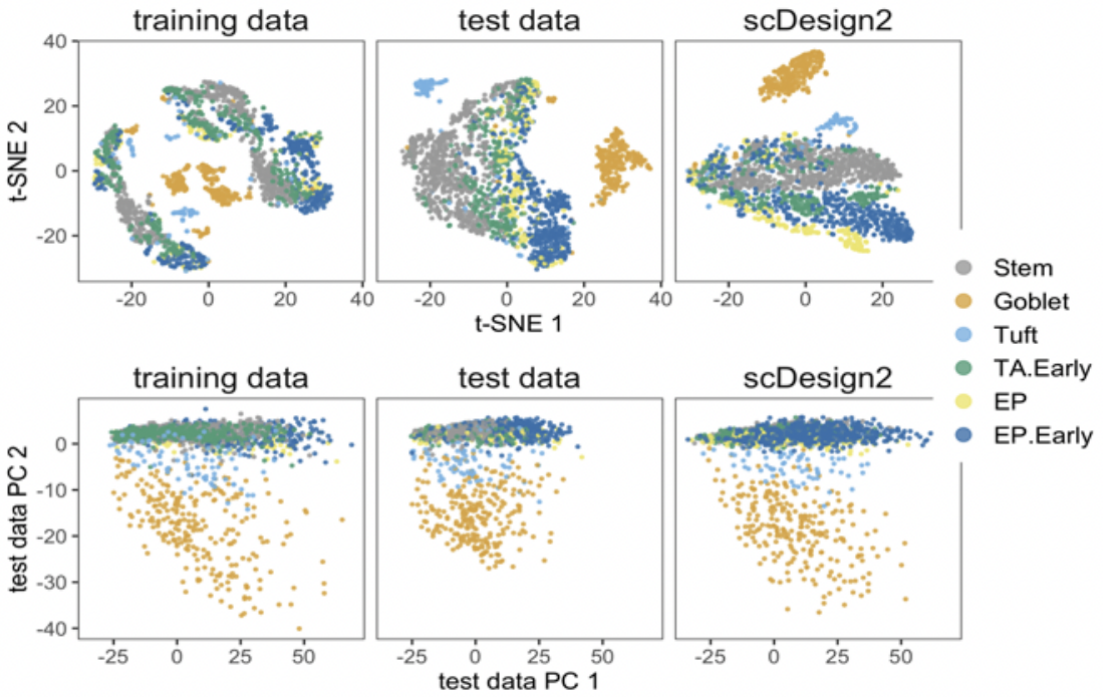


Figure 10: t-SNE and PCA of the R implementation

ing whether malignant cells would cluster together, thus indicating a similar gene expression profile. Similarly, we looked for possible clusters among cells from the same batch or group, which would indicate that the simulation was successful in preserving batch-specific or group-specific features in the gene expression data.
The UMAP analysis was conducted through a series of steps using the Scanpy li-

brary [FAW18]. Initially, the data were normalized to a target sum of 10000. Then highly variable genes were identified, limiting the selection to the top 2000 genes. This process was already performed on the simulation which was used in the CanSig paper. It was used on the scDesign2 implementation to validate the simulation and compare the two implementations.

The reference set used here is the same as the previous one (mouse GEO GSE92332) where 5000 genes were randomly chosen, because the implemented UMAP pipeline requires 5000 genes. It is also worth noting that in the old implementation, 5 cell types/groups are present, those are: Macro, Plasma, Program1, Program2 and Program3. Macro and Plasma cells are healthy and the Program cells are malignant. We randomly assigned the given labels to the actual cell types as follows: Stem as Macro, Enterocyte Progenitor Early as Plasma, Tuft as Program1, Goblet as Program2, and Enterocyte Progenitor as Program3.

Utilizing the group effect, batch effect, and CNVs with specific parameters, we executed the UMAP algorithm for the Splatter variation and show it in figure 11. Each parameter in the lists, namely p_de_list, p_down_list, de_location_list, de_scale_list, and pb_de_list, corresponds to a group in the following order: Macro, Plasma, Program1, Program2, Program3. The same parameter values are applied across all batches for the parameters affecting the batch. The parameters used were as follows:

- p_de_list = [0.2,0.2,0.1,0.1,0.1]
- p_down_list = [0.5,0.5,0.5,0.5,0.5]
- de_location_list = [0.4,0.4,0.25,0.25,0.25]
- de_scale_list = [0.1,0.1,0.1,0.1,0.1]
- pb_de_list = 0.1
- pb_down_list = 0.5
- bde_location = 0.1
- bde_scale = 0.1

When looking at the fourth plot one can see that group effect is present and also when looking at the second plot one can tell that the effects of the CNVs are shown since each color is separated. Looking at the first plot it can be seen that some colors are clustered together and one might conclude that batch effect is partly present; however, it is not. While it may appear that batch effect is also present, a closer examination of the healthy cells, which are not clustered according to their respective batches, indicates otherwise. The clustering observed among the malignant cells can be attributed to the CNVs rather than batch effect.

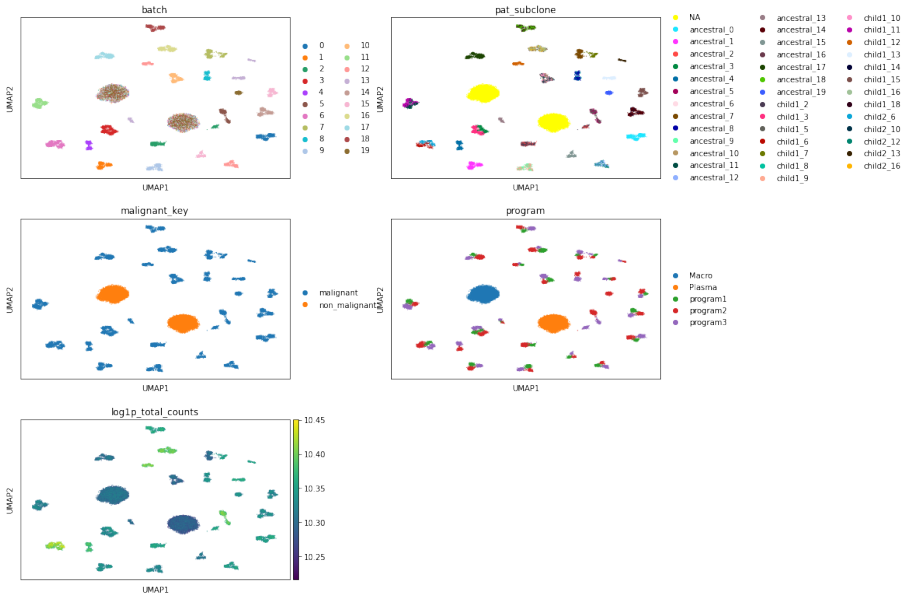


Figure 11: UMAP of the Splatter variation In the first picture each color represents a batch. In the second they represent a subclone, that is ancestral, subclone1, subclone2, or a healthy cell. Then in the third image the colors represent the state of a cell (healthy/malignant). In fourth image a color represents a group/cell type and in the last picture the shade represent the counts.

When looking at the UMAP (Figure 12) without CNVs one clearly can see that no batch effect is present. For it to show, the bde_location parameter needs to be set higher. After running the visualization with different parameters, it was found that the bde_location variable needs to be set to around 0.45 for all batches for it to show (Figure 13).

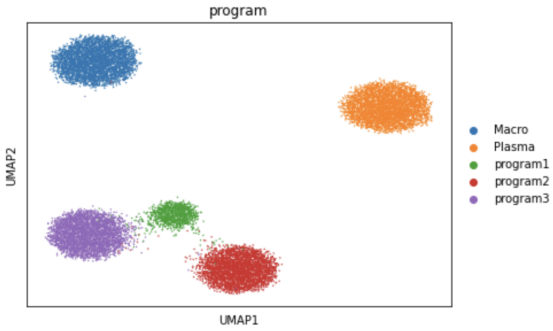


Figure 12: Without CNVs

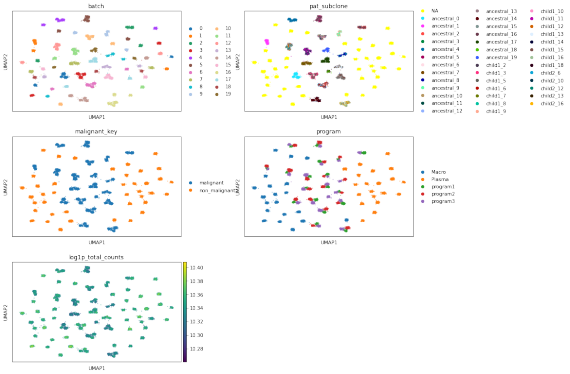


Figure 13: bde_location set to 0.45

4.2.1 Group effect

The goal is to observe all 3 effects in the scDesign2 variation also. It was already demonstrated in the basic test of scDesign2 that it manages to keep different cell types separate (using t-SNE) but here using UMAP, without any group effect, batch effect and CNVs, one can see again that most groups cluster together even in the

absence of additional group effect (Figure 14). The two which overlap each other are Plasma and Program3 but this just might be because they are closely related. Remember Plasma cells are Enterocyte Progenitor Early cells and Program3 cells are Enterocyte Progenitor cells.

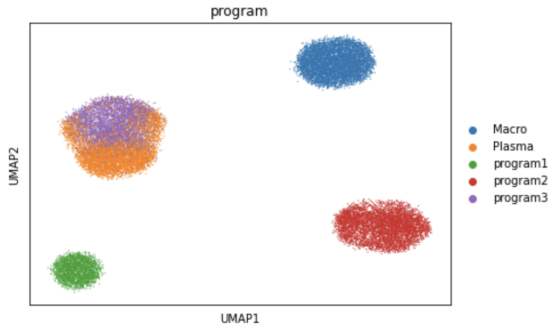


Figure 14: UMAP of the vanilla scDesign2 implementation (mouse dataset)

To combat this issue we changed the reference dataset and used a breast dataset (GSE195665) with the following mapping of the groups: IgG plasma cell → Macro, IgA plasma cell → Plasma, unswitched memory B cell → Program1, naive B cell → Program2, class switched memory B cell → Program3. We will refer to the cells from the breast dataset as 'bcells'. One could also have just swapped the Enterocyte Progenitor cells with the Tuft ones but we believed that including human cells would provide a more meaningful or representative context. When choosing the 5000 genes arbitrarily, it becomes apparent that two groups are still lightly clustered together in certain areas (Figure 15).

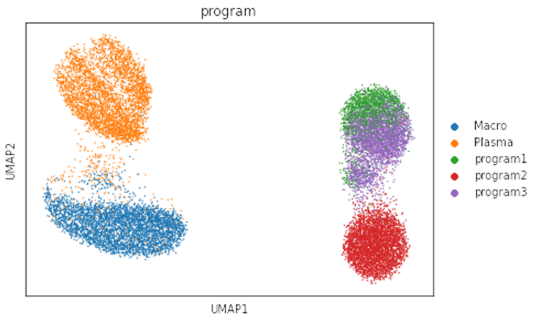


Figure 15: Bcells with 5000 genes chosen arbitrarily. Some cells of different cell types are overlapping.

When examining the count matrix of the reference dataset, we noticed a substantial number of zeros. This abundance of zeros could potentially blur the differences between some cells, leading to similarities in their gene expression profiles. Therefore, to mitigate this effect, instead of randomly choosing 5000 genes, we adopted a more targeted approach. Specifically, we selected 1350 genes that have a mean count of less than 0.1, 1900 genes with mean count $\in [0.1, 0.2)$, 1400 genes with mean count $\in [0.2, 0.5)$, 250 genes with mean count $\in [0.5, 0.9)$ and 100 genes with mean count greater or equal 0.9. We tried to have as few non-expressed genes as possible and

that is how the numbers were chosen. Looking at the UMAP in figure 16 it is evident that the groups are clearly separated even without any addition of group effect.
It can be concluded that the group effect is dependent on the provided dataset. Adding the group effect from Splatter with the same parameters does not make a big difference and in the following experiments the group effect is added (Figure 17).

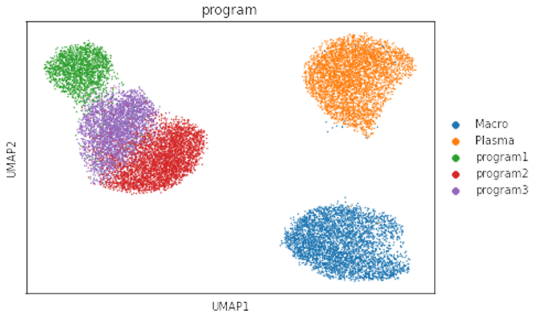


Figure 16: Bcells without group effect

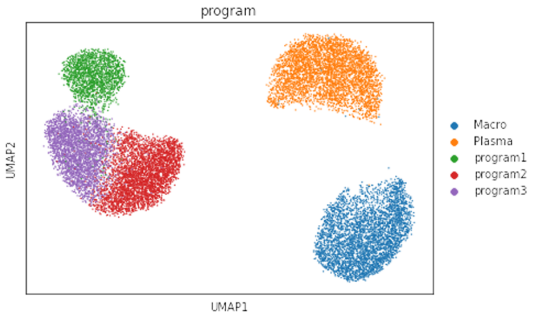


Figure 17: Bcells with group effect

4.2.2 Batch effect

When adding the batch effect with the same parameters where the batches first start to separate in the Splatter variation (bde_location set to 0.45), the batches do not cluster together (Figure 18). In fact nothing noticeable happens.

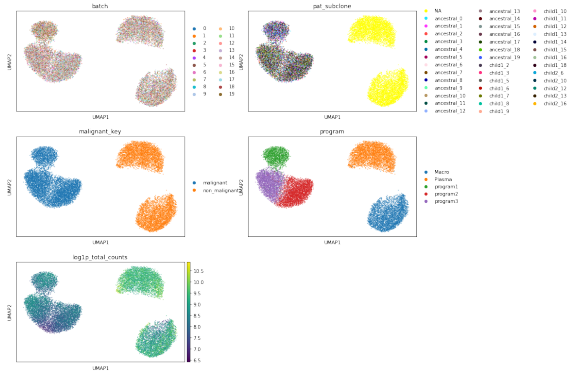
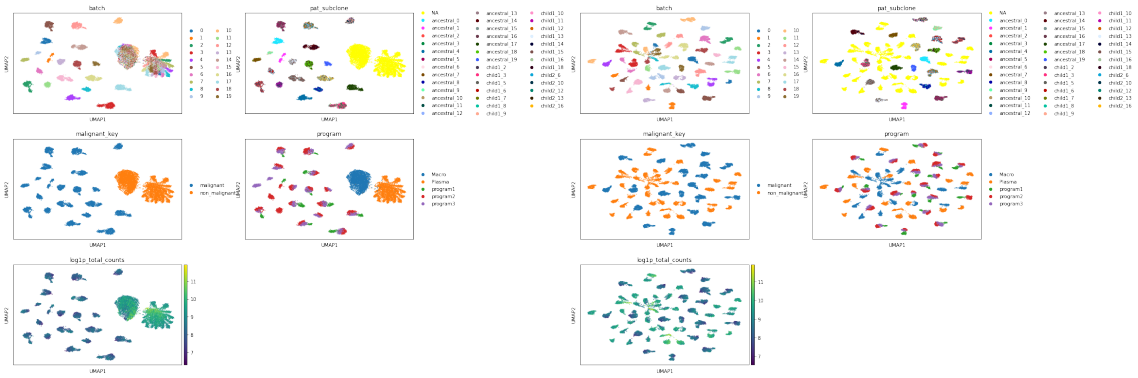


Figure 18: No batch effect observable

By incrementing the bde_location_list parameter by 0.10 at a time and plotting the UMAPs in figure 19 (more in Figure 39), it can be observed that the variable needs to be set to 1.9 in order to see the batches cluster together like in the Splatter variation.

Interestingly, malignant cells require a weaker batch effect to cluster together compared to healthy cells. It is important to mention that in this simulation, these cells are only labeled as malignant, but no CNV effect has been activated. To check if the reason was due to the malignant and healthy labels, we swapped them and we discovered that this phenomenon was unrelated to the labels.



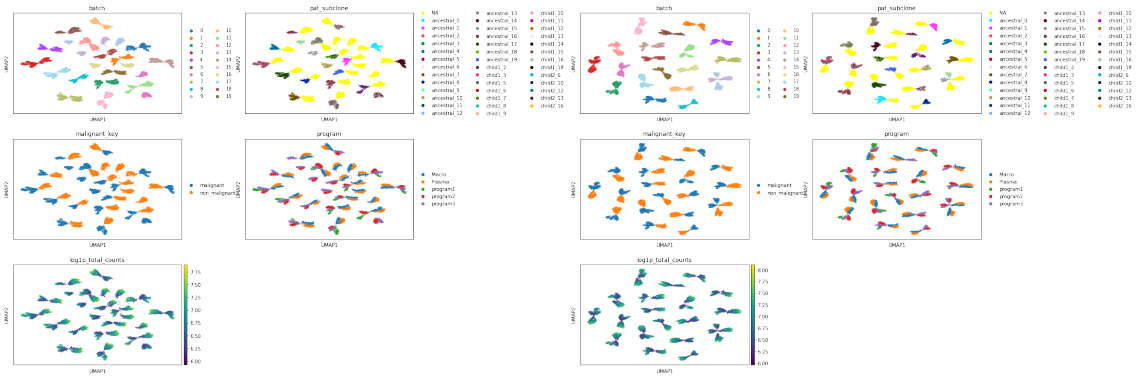
(a) bde_location_list = 1.2 **(b)** bde_location_list = 1.9

Figure 19: Variations of bde_location_list

Afterwards we hypothesized that these cell types might have a stronger identity compared to the remaining three. This hypothesis was based on the fact that these two types required a stronger batch effect to form separate clusters, implying a more robust identity that was able to withstand higher levels of batch-induced variation. To test this hypothesis, we neutralized the effects of differential gene expression, which is a key determinant of cellular identity. Specifically, we set the expression values of the genes, which were differentially expressed in these two cell types relative to the others, to zero.

The differential analysis was performed using the "rank_genes_groups" function of Scanpy [FAW18] to compare the "IgG plasma cell" and "IgA plasma cell" groups to the combined group of "unswitched memory B cell", "naive B cell", and "class switched memory B cell". The log-fold changes, as well as p-values and adjusted p-values, are calculated for this comparison. Log-fold changes represent the magnitude of the difference in gene expression between the two groups being compared, with a higher absolute value indicating a larger difference.
In our case, genes were considered differentially expressed if the adjusted p-value was less than 0.01 where genes with absolute log-fold changes of more than 2 are considered significant. We first required the adjusted p-value to be less than 0.05 and the log-fold change to be greater than 1.5 but it would filter out more than 1000 genes which we considered too many and that is how we came up with our values. With them we only got around 400 genes which were considered differentially expressed.
Next, we conducted the UMAP analysis again, using the batch effect that previously caused clustering in the other three cell types, as shown in figure 20.

The two cell types, which formerly had maintained their distinct clusters in the face of these batch effects, are now clustering in batches like the other three types at the same rate. This behavior strongly suggests that the differentially expressed genes we identified and nullified were indeed a significant contributor to the robust identity of these two cell types.
Therefore, our initial hypothesis seems to hold true: the two cell types in question initially demonstrated a stronger identity, as evidenced by their resistance to clus-

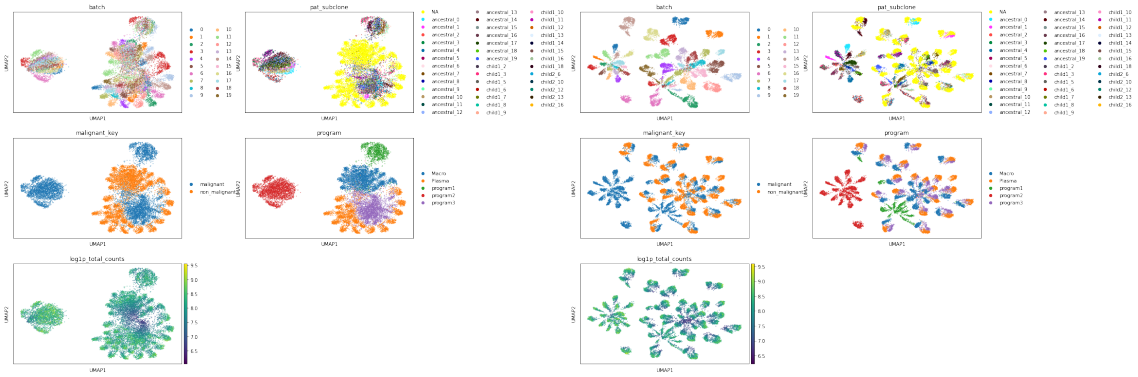


(a) bde_location_list = 1.2 **(b)** bde_location_list = 1.4

Figure 20: After setting the expression values of differentially expressed genes to 0

tering under weaker batch effects. This identity, however, was largely due to the presence of differentially expressed genes, and when these factors were eliminated, the cell types lost their distinctiveness. These findings underscore the critical role of gene expression profiles in defining cellular identity and provide new insights into how batch effects can influence our ability to discern these identities.

Consequently, the intensity of the batch effect required is dependent on the reference set provided. We applied the UMAP algorithm to the mouse set, but instead of randomly selecting 5000 genes, we chose genes such that about 2000 were not expressed. For that dataset, it's evident from figure 21 (more in Figure 40) that the batch effect appears at a consistent rate across all cell types and that it requires a different parameter value.

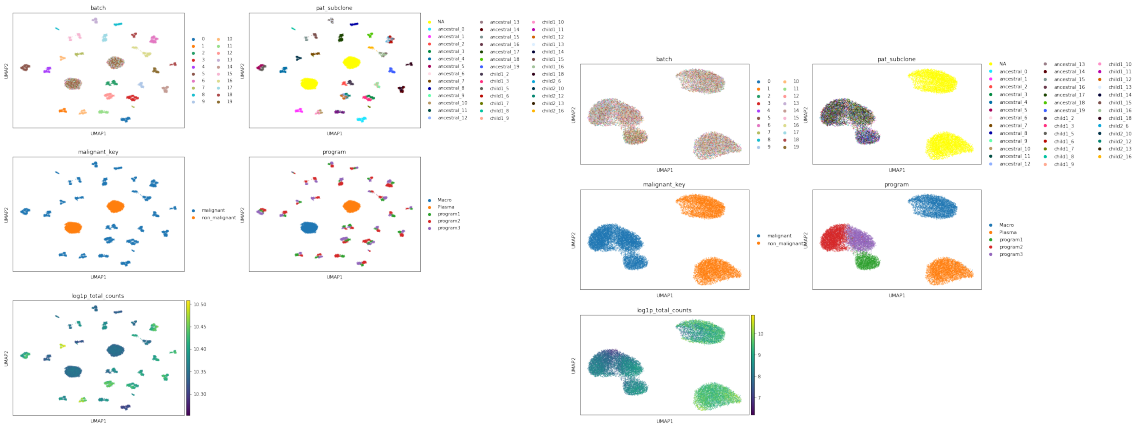


(a) bde_location_list = 1.3 **(b)** bde_location_list = 1.6

Figure 21: Batch effect is observed at the same rate for the mouse set

4.2.3 CNVs

As shown in a previous section, the Splatter variation clusters malignant cells of the same batch together with just CNVs enabled. Unfortunately this is not the case for the scDesign2 variation (Figure 22).



(a) Splatter **(b)** scDesign2

Figure 22: Effect of CNVs on the UMAP

One can see that the CNVs do not change the UMAP of it. Maybe a stronger effect of the CNVs is needed, that is for example a gain doubles the gene expression instead of a 1.5 fold or a loss decreases the gene expression to a third instead of a half. Through experimentation with different parameter values, we identified the settings needed for the malignant cells to cluster together in batches. Specifically, we found that a value of 4 for the location parameter of the truncated Cauchy distribution during a gain and a value of 0.3 during a loss event gave satisfactory results (recall that these are for highly expressed genes). The means (μ) of the Gaussians of the GMM had to be set to [3.5553, 1.2422] for a gain and [1.6843, 0.3513] for a loss (Figure 23). The simulator captures gene correlations

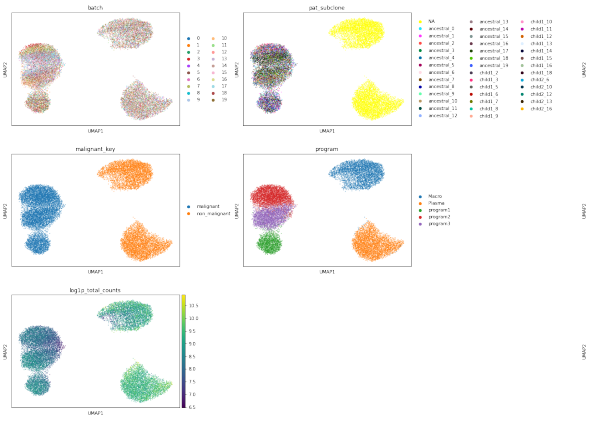
However, these values are unrealistically high and do not reflect real world data, prompting further investigation. We then realized that, contrary to our earlier proposal, we had not recalculated the covariance matrix after altering the means; we had only computed it post the original mean fitting. Accordingly we fixed this issue and ran UMAP analysis again with different values for the parameters (Figure 24).

Unfortunately, the difference is not very pronounced. For the parameters used in b) it is slightly worse and for parameters used in c) and d) it is slightly better.

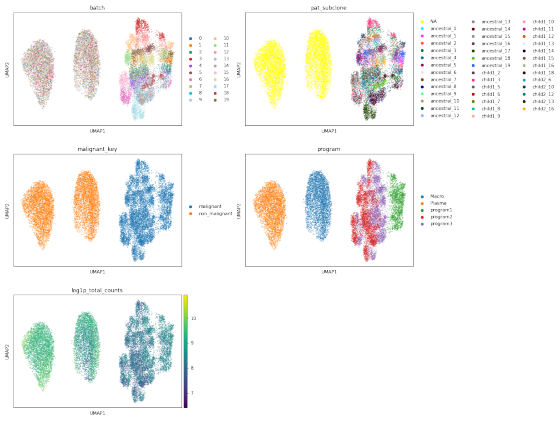
We next we adjusted the probability of a CNV occurring on a chromosome, increasing it from 20% to 50%, to assess its influence on the clustering of batches (Figure 25).

And it did cause them to cluster together, not very strong but better than before.

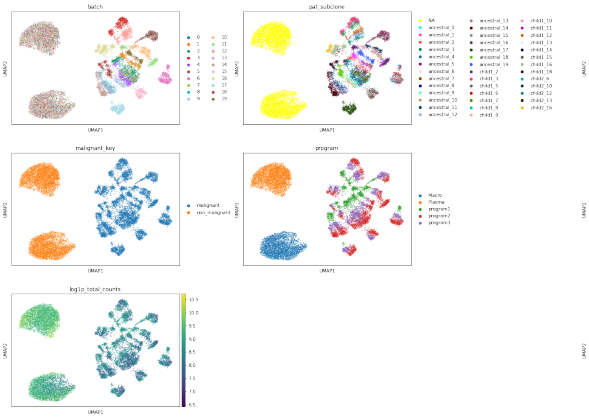
To check if our implementation of scDesign2 had any flaws we first generated a count matrix from the Splatter simulation of only healthy cells with group effect and without batch effect. Then we used this as the reference set and ran the simulation with CNVs (Figure 26).



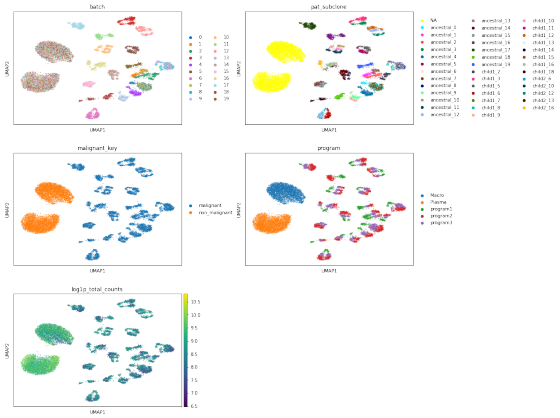
(a) gain: $x_0 = 2, \mu = [3.3553, 1.0422]$ loss: $x_0 = 0.4, \mu = [1.8843, 0.5013]$



(b) gain: $x_0 = 2.5, \mu = [3.4553, 1.1422]$ loss: $x_0 = 0.4, \mu = [1.7843, 0.4013]$



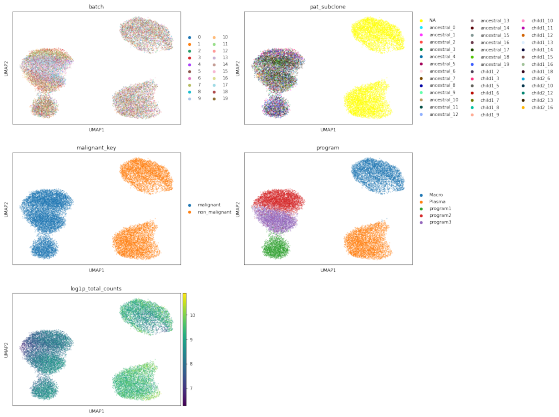
(c) gain: $x_0 = 3, \mu = [3.4553, 1.1422]$ loss: $x_0 = 0.3, \mu = [1.7843, 0.4013]$



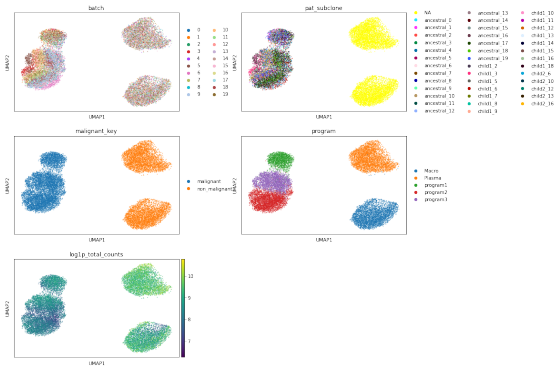
(d) gain: $x_0 = 4, \mu = [3.5553, 1.2422]$ loss: $x_0 = 0.3, \mu = [1.6843, 0.3513]$

Figure 23: Experimentation with different CNV parameter values

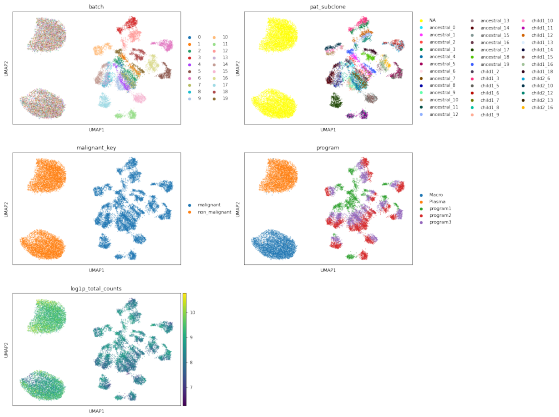
It can be seen that most batches are clustered together. It is not as well clustered as the Splatter variation since the subclones of are batch are not totally separate. But nonetheless it is far better than what we have seen so far.
We conclude that our implementation of scDesign2 is correct, but an underlying issue appears to be hindering the ability of the CNVs to cause cells to cluster together. This may be due to an incompatibility between the manner in which CNVs and their effects are implemented and the chosen simulation.
Due to time constraints, it is at this time that we have to come to an end for our project.



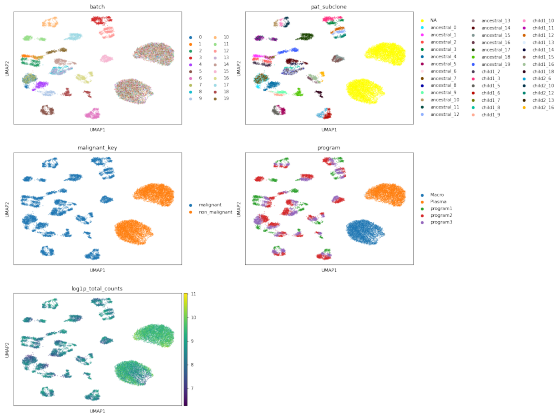
(a) gain: $x_0 = 2, \mu = [3.3553, 1.0422]$ loss: $x_0 = 0.4, \mu = [1.8843, 0.5013]$



(b) gain: $x_0 = 2.5, \mu = [3.4553, 1.1422]$ loss: $x_0 = 0.4, \mu = [1.7843, 0.4013]$



(c) gain: $x_0 = 3, \mu = [3.4553, 1.1422]$ loss: $x_0 = 0.3, \mu = [1.7843, 0.4013]$



(d) gain: $x_0 = 4, \mu = [3.5553, 1.2422]$ loss: $x_0 = 0.3, \mu = [1.6843, 0.3513]$

Figure 24: Experimentation with different CNV parameter values with adjusted covariance matrix

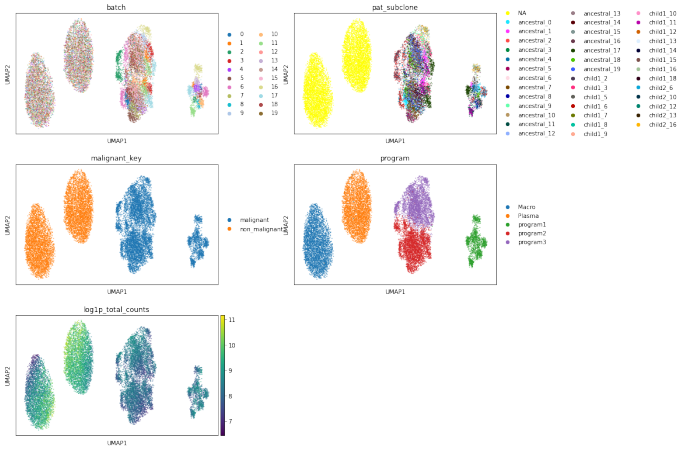


Figure 25: Increased number of CNVs

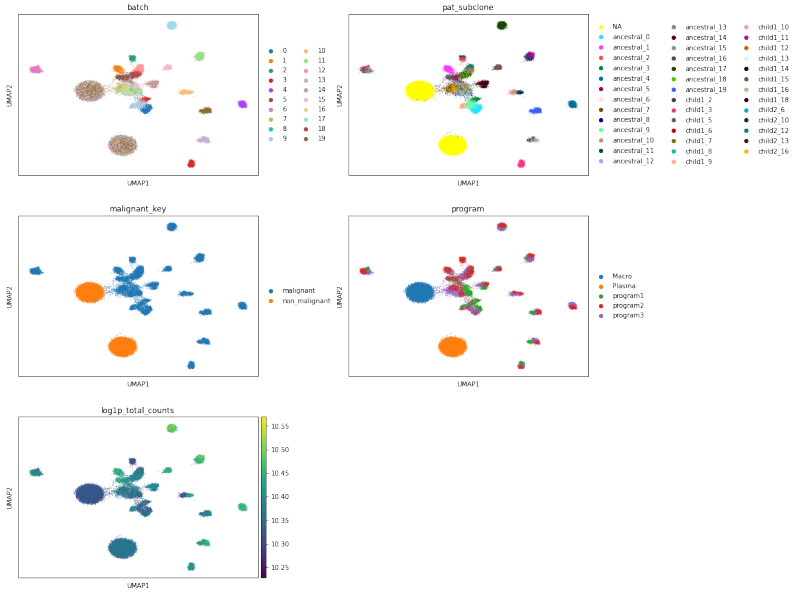


Figure 26: Using the Splatter generated count matrix as reference.

5 Results

In the current study, we successfully converted scDesign2 from R to Python. Additionally we modified it to incorporate CNVs so it can simulate malignant cells, and compared it to the simulation used for the CanSig paper. For the analysis the UMAP algorithm was utilized to visualize the high-dimensional gene expression data and potentially reveal any hidden patterns or structures within the cells.

We found that the group effect, in which each group or cell type is clustered together, was effectively maintained by the simulation. Interestingly, the strength of the batch effect had to be significantly increased to become apparent. This requirement, however, was heavily influenced by the provided reference dataset. It was observed that cell types with a stronger identity needed a stronger batch effect to differentiate them and vice versa for weaker identities.

However, the incorporation of CNVs into the simulation proved challenging, especially when integrated in the same way as in the Splatter variation. We hypothesize that this difficulty might be attributed to the inherent differences between the scDesign2 and Splatter frameworks, particularly with regard to the maintenance of gene-to-gene correlation via a covariance matrix in scDesign2.

6 Discussion

In the course of our study, we faced significant challenges in effectively incorporating CNVs into the simulation, as per the methodology employed in the Splatter variation. Our results suggest that resolving these incompatibilities might require alternative approaches for integrating CNVs into scDesign2.

Several such strategies have been proposed in the existing literature. One notable method has been discussed in a publication mentioned earlier [RJ11], where the authors present a different model for incorporating CNVs into cell simulation frameworks. This approach may offer new perspectives on how CNVs can be modeled in single-cell RNA sequencing simulations, potentially overcoming the limitations we encountered here.

Beyond adjusting the CNV integration, another potential solution could involve a shift in the foundational simulation framework. During the course of our project, other simulations such as scMultiSim [HL23] have been published. These novel simulation tools offer different modeling approaches and may inherently accommodate CNVs more effectively than scDesign2, thus providing a possible alternative path for future research.

While CNVs have been the primary focus in our study, they represent just one facet of the genetic changes that can occur in malignant cells. The integration of point mutations, chromosomal rearrangements, and epigenetic changes could provide a more suited approach to simulating malignancy. Each of these elements offers unique insights into the complex biology of cancerous cells and could contribute to more accurate and detailed simulations. The challenge lies in developing suitable models to represent these features accurately within the simulation framework. Thus, future research might benefit from not only addressing the CNV integration challenges identified in our work but also extending the scope to incorporate these additional genetic and epigenetic dimensions.

Ultimately, this study revealed the complexity involved in simulating malignant single-cell RNA sequencing data. This paves the way for future research and advances in this field.

References

[AH17] Sarah A. Teichmann & Tapio Lönnberg Ashraful Haque, Jessica Engel. Rna sequencing: new technologies and applications in cancer research. *https://jhoonline.biomedcentral.com/articles/10.1186/s13045-020-01005-x*, 2017.

[CK13] Benjamin J. Raphael & Li Ding Cyriac Kandoth, Michael D. McLellan. Mutational landscape and significance across 12 major cancer types. *https://www.nature.com/articles/nature12634*, 2013.

[FAW18] Philipp Angerer & Fabian J. Theis F. Alexander Wolf. Scanpy: large-scale single-cell gene expression data analysis. *https://genomebiology.biomedcentral.com/articles/10.1186/s13059-017-1382-0*, 2018.

[Ger20] David Gerard. Data-based rna-seq simulations by binomial thinning. *https://bmcbioinformatics.biomedcentral.com/articles/10.1186/s12859-020-3450-9*, 2020.

[HL23] Michael Squires Xi C Hechen Li, Ziqi Zhang. scmultisim: simulation of multi-modality single cell data guided by cell-cell interactions and gene regulatory networks. *https://www.biorxiv.org/content/10.1101/2022.10.15.512320v3*, 2023.

[HLC23] Charlotte Soneson & Mark D. Robinson Helena L. Crowell, Sarah X. Morillo Leonardo. Rna sequencing: new technologies and applications in cancer research. *https://genomebiology.biomedcentral.com/articles/10.1186/s13059-023-02904-1*, 2023.

[JY22] Paweł Czyż Marc Glettig Frederieke Lohmann Richard von der Horst Elia Saquand Nicolas Volken ProfileAgnieszka Kraft Valentina Boeva Josephine Yates, Florian Barkmann. Cansig: Discovering de novo shared transcriptional programs in single cancer cells. *https://www.biorxiv.org/content/10.1101/2022.04.14.488324v1*, 2022.

[LZ17] Alicia Oshlack Luke Zappia, Belinda Phipson. Splatter: simulation of single-cell rna sequencing data. *https://genomebiology.biomedcentral.com/articles/10.1186/s13059-017-1305-0*, 2017.

[MM20] Vikas Bansal Christoph Kilian Daniel S. Magruder Christian F. Krebs Stefan Bonn Mohamed Marouf, Pierre Machart. Realistic in silico generation and augmentation of single-cell rna-seq data using generative adversarial networks. *https://www.nature.com/articles/s41467-019-14018-z*, 2020.

[pdt23] The pandas development team. pandas-dev/pandas: Pandas. *https://github.com/pandas-dev/pandas*, 2023.

[RJ11] Teresia Kling Linnéa Schmidt Erik Johansson Torbjörn E M Nordling Bodil Nordlander Chris Sander Peter Gennemark Keiko Funa Björn Nilsson Linda Lindahl & Sven Nelander Rebecka Jörnsten, Tobias Abenius. Network modeling of the transcriptional effects of copy number aberrations in glioblastoma. *https://www.embopress.org/doi/full/10.1038/msb.2011.17*, 2011.

[RL18] Michael B. Code Michael I. Jordan & Nir Yosef Romain Lopez, Jeffrey Regier. Deep generative modeling for single-cell transcriptomics. *https://www.nature.com/articles/s41592-018-0229-2*, 2018.

[Sea10] Perktold Josef Seabold, Skipper. statsmodels. *https://github.com/statsmodels/statsmodels*, 2010.

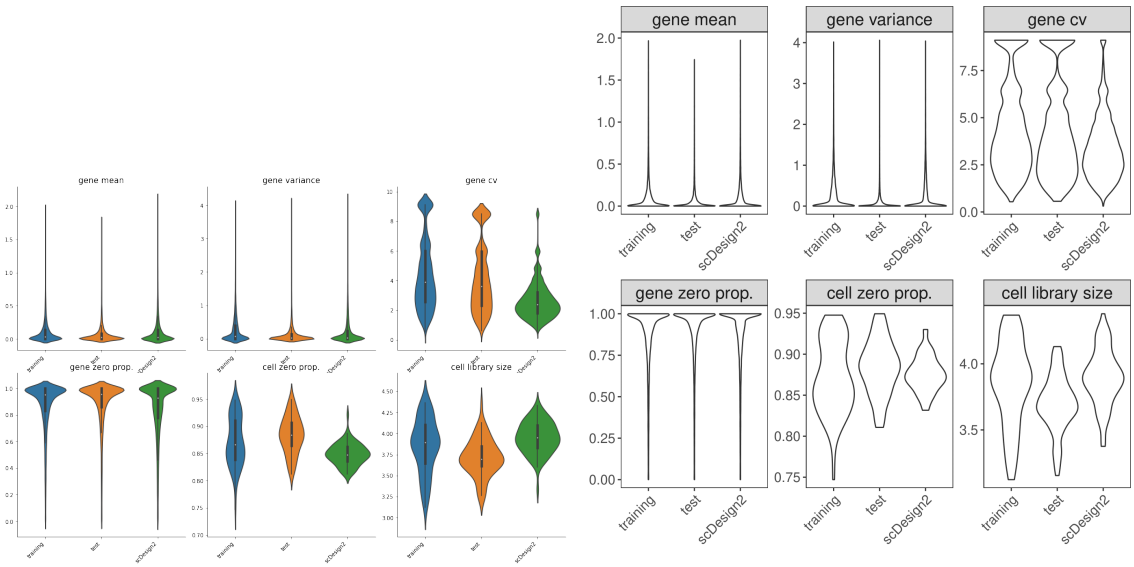
[TS21a] Wei Vivian Li & Jingyi Jessica Li Tianyi Sun, Dongyuan Song. scdesign2: a transparent simulator that generates high-fidelity single-cell gene expression count data with gene correlations captured. *https://genomebiology.biomedcentral.com/articles/10.1186/s13059-021-02367-2*, 2021.

[TS21b] Wei Vivian Li & Jingyi Jessica Li Tianyi Sun, Dongyuan Song. scdesign2 tutorial. *https://htmlpreview.github.io/?https://github.com/JSB-UCLA/scDesign2/blob/master/vignettes/scDesign2.html*, 2021.

[vF22] Alexander von Felbert. Probability integral transform; quantile function theorem. *https://www.deep-mind.org/2020/11/29/probability-integral-transform-quantile-function-theorem/*, May 2022.

[YC21] Jean Yee Hwa Yang Yue Cao, Pengyi Yang. A benchmark study of simulation methods for single-cell rna sequencing data. *https://www.nature.com/articles/s41467-021-27130-w*, 2021.

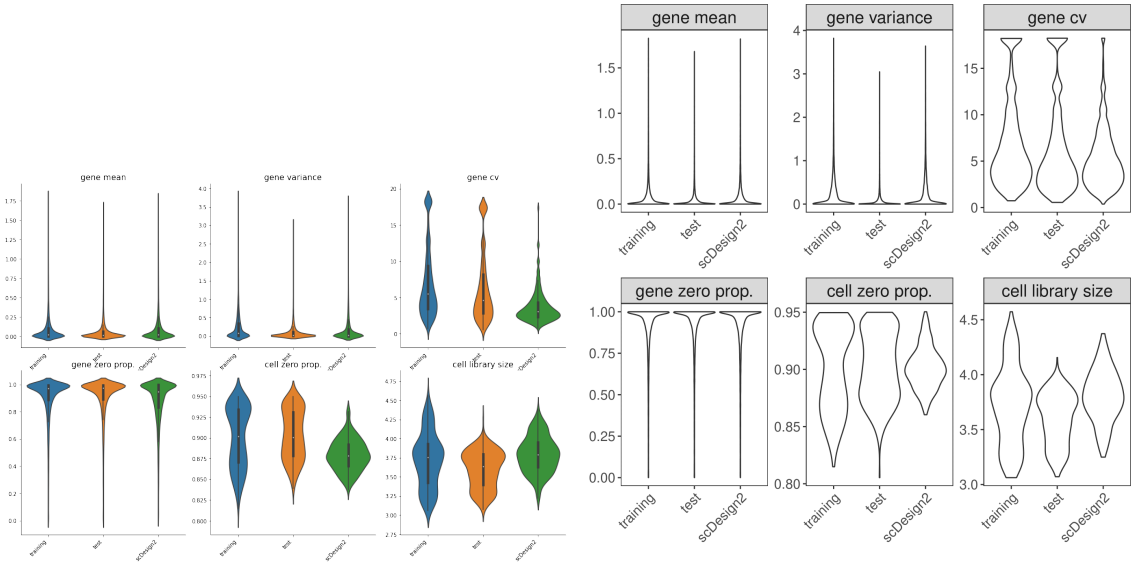
Supplementary



(a) Python: Violin plot of Tuft cells

(b) R: Violin plot of Tuft cells

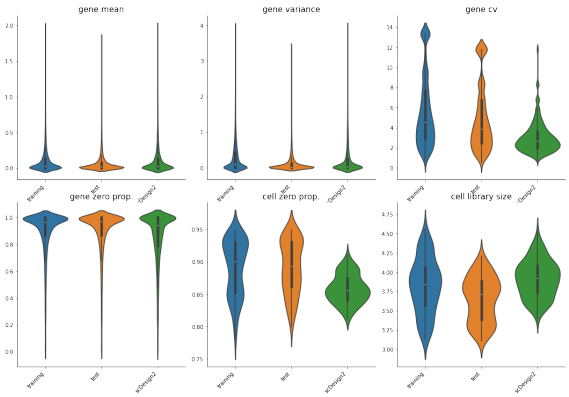
Figure 27: Violin plot of the training set, test set and simulated set of Tuft cells from the R and Python package



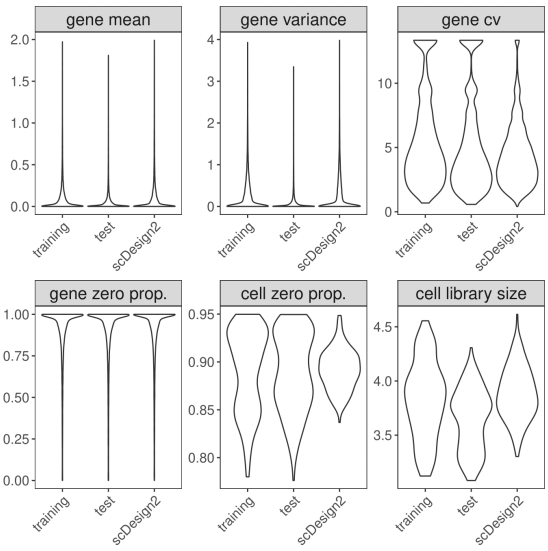
(a) Python: Violin plot of Ta Early cells

(b) R: Violin plot of Ta Early cells

Figure 28: Violin plot of the training set, test set and simulated set of Ta Early cells from the R and Python package

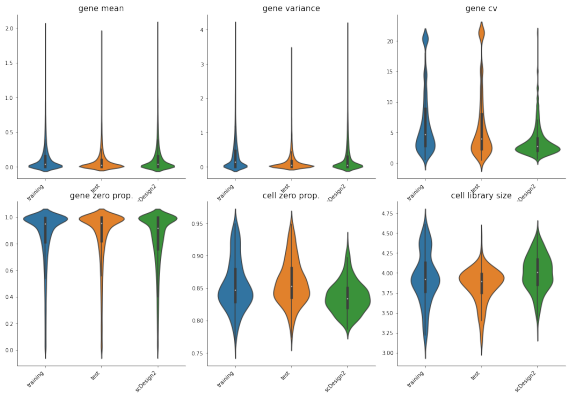


(a) Python: Violin plot of Enterocyte Progenitor cells

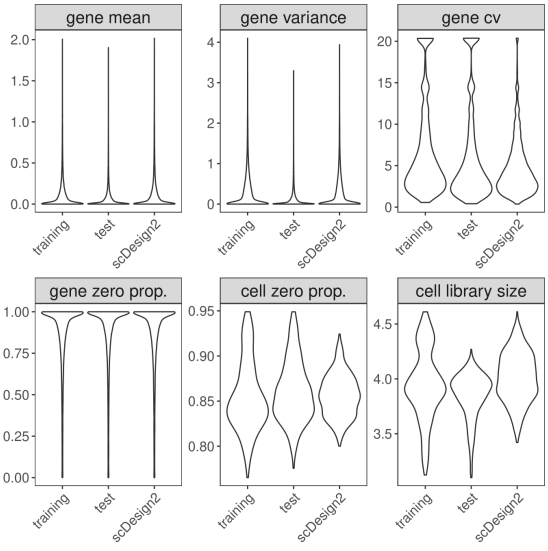


(b) R: Violin plot of Enterocyte Progenitor cells

Figure 29: Violin plot of the training set, test set and simulated set of Enterocyte Progenitor cells from the R and Python package



(a) Python: Violin plot of Enterocyte Progenitor Early cells



(b) R: Violin plot of Enterocyte Progenitor Early cells

Figure 30: Violin plot of the training set, test set and simulated set of Enterocyte Progenitor Early cells from the R and Python package

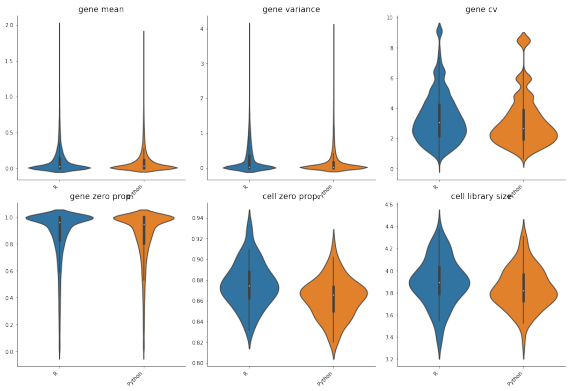


Figure 31: Violin plot of Tuft cells

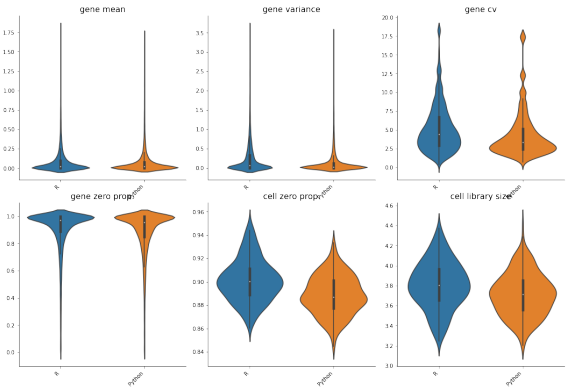


Figure 32: Violin plot of TA Early cells

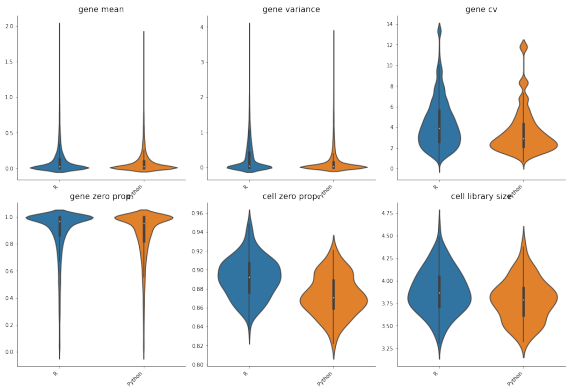


Figure 33: Violin plot of Enterocyte Progenitor cells

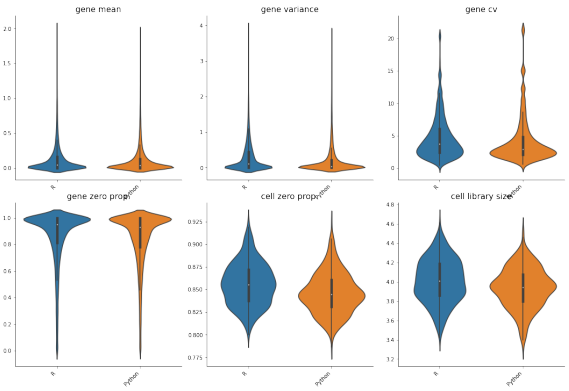


Figure 34: Violin plot of Enterocyte Progenitor Early cells

Figure 35: Stem: Square error of means histogram

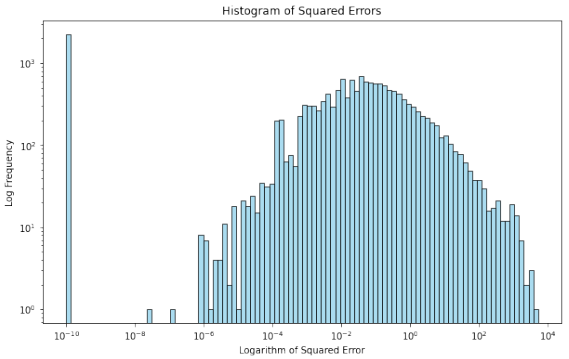


Figure 36: Goblet: Square error of means histogram

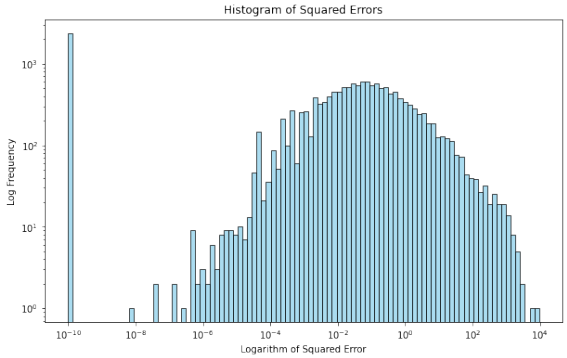


Figure 37: Stem: Square error of means histogram

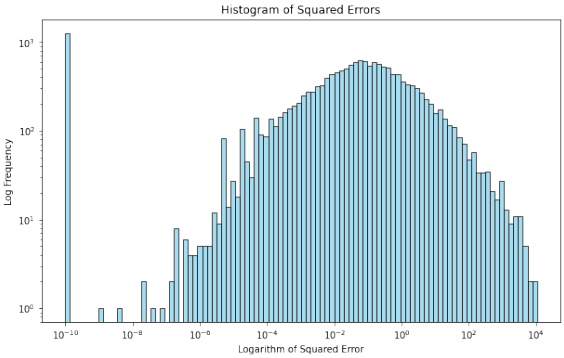
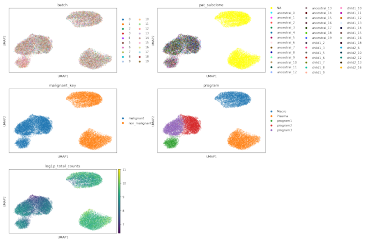
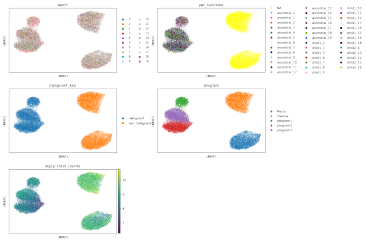


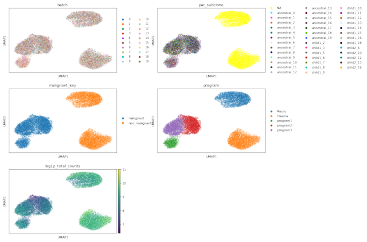
Figure 38: Goblet: Square error of means histogram



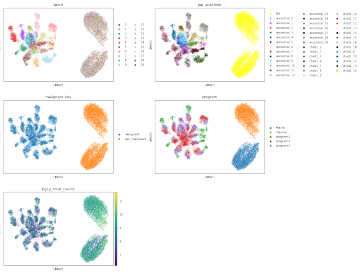
(a) bde_location_list = 0.7



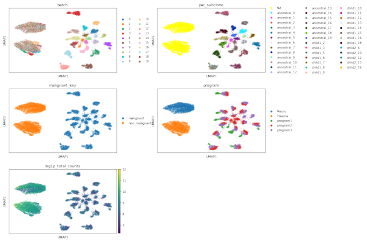
(b) bde_location_list = 0.8



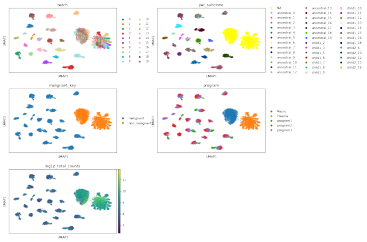
(c) bde_location_list = 0.9



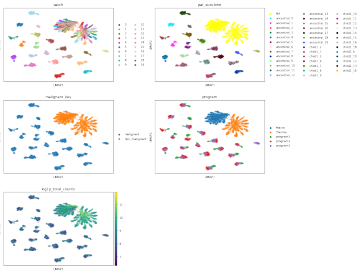
(d) bde_location_list = 1.0



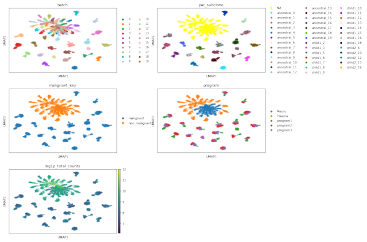
(e) bde_location_list = 1.1



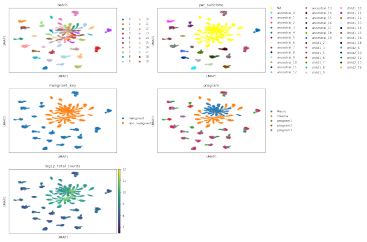
(f) bde_location_list = 1.2



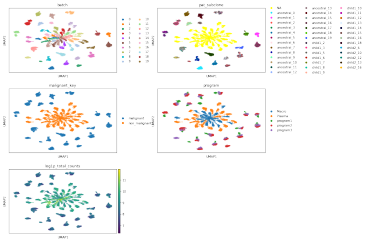
(g) bde_location_list = 1.3



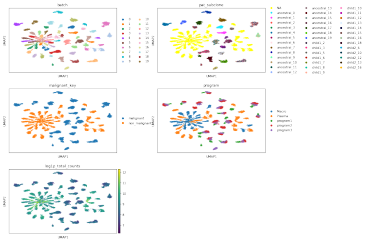
(h) bde_location_list = 1.4



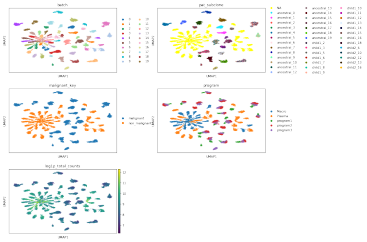
(i) bde_location_list = 1.5



(j) bde_location_list = 1.6

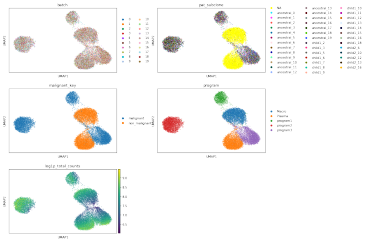


(k) bde_location_list = 1.7

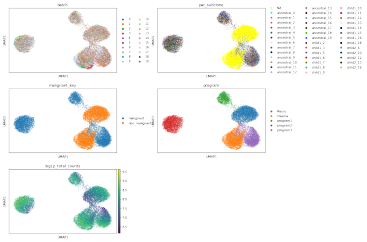


(l) bde_location_list = 1.8

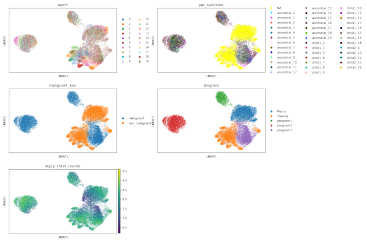
Figure 39: Batch effect on bcells



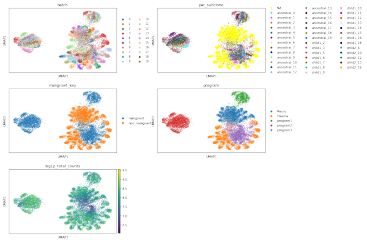
(a) bde_location_list = 1.0



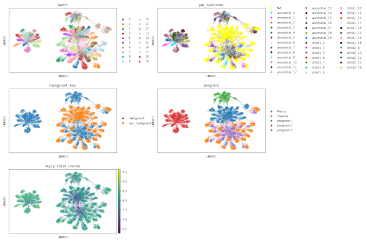
(b) bde_location_list = 1.1



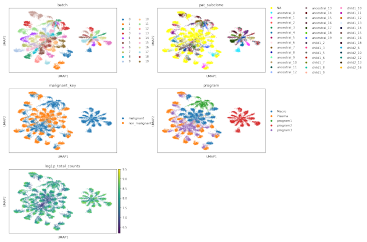
(c) bde_location_list = 1.2



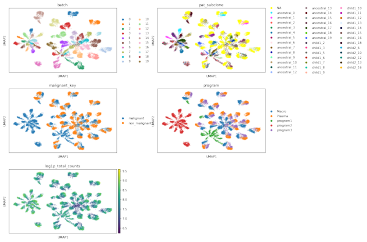
(d) bde_location_list = 1.3



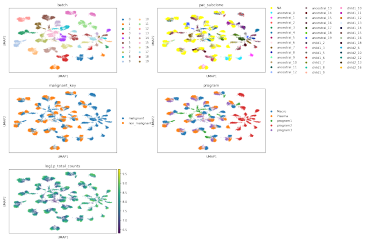
(e) bde_location_list = 1.4



(f) bde_location_list = 1.5



(g) bde_location_list = 1.6



(h) bde_location_list = 1.7

Figure 40: Batch effect on mouse cells