Internship dissertation

Cross-species comparison of cell types using scRNA-seq data integration

Anaëlle Cossard

M1 Bioinformatics and Biostatistics
Paris-Saclay Univeristy

Supervisors : Yuyao Song, Dr. Irene Papatheodorou

EMBL-EBI, Wellcome Genome Campus, Hinxton, Cambridge, UK

8 May 2023 - 27 July 2023

Contents

1 Introduction

This internship takes place at the European Bioinformatic Institute (EBI) situated in the Wellcome Genome Campus, Hinxton, United Kingdom, funded in 1994 which is a part of the European Molecular Biology Laboratory (EMBL). The EMBL is an intergovernmental organisation funded in 1974 by Leó Szilárd, James Watson and John Kendrew and currently supported by 28 different states. The research conduct are site-specific, the EMBL includes six different sites :

- Grenoble (France) : structural biology research and services
- Hamburg (Germany) : structural biology research and services as well
- Heidelberg (Germany) : main laboratory and the head quarters, focusing on molecular biology research and services
- Rome (Italy) : epigenetics and neurobiology
- Barcelona (Spain) : tissue biology and disease modelling
- Hinxton (United Kingdom) : bioinformatics research and services.

The aim of the EBI is to provide freely available data, bioinformatics services and training. This involves providing and maintaining databases, such as Expression Atlas, or tools like Clustal Omega. Collaborations are strongly encouraged, in particular by the presence of an Elixir hub at the EBI, or the Wellcome Sanger Institute also situated in the Wellcome Genome Campus, but collaborations with laboratories all over the world are made as well. For example, EBI has collaborated with DeepMind to create the AlphaFold DataBase and is keeping it updated.

For my internship, I am working with the Papatheodorou Group, led by Dr. Irene Papatheodorou, which focuses on gene expression, decoding phenotypes and developing tools to analyse bulk and single-cell data. I am working under the direction of both Dr. Papatheodorou and Yuyao Song, who is a predoctoral fellow working on cross-species comparison of cell types using single-cell RNA sequencing data (scRNA-Seq data). She develops and applies various computational approaches on publicly available multi-organ scRNA-seq data from different species, to study the conservation and divergence of gene expression programs between homologous cell types. A gene program (GP) is a group of genes that play a role in the same function.

Thanks to the increase in the number of single-cell atlases for multiple species, we can now perform cross-species analysis with the aim of studying the difference between the cell types across those species. However it is a challenge to perform a good cross-species integration due to the difference between the species that is higher than the difference between the cell types. In addition, the earlier the species have diverged, the larger the difference between the two species. Consequently, the existing tools developed for single-cell data integration within the same species have difficulties to well-mixed the species to group the same cell types together. Nevertheless, Shafer [5] shows that some of them have been adapted for cross-species analysis and are promising. Being able to achieve a cross-species integration would be a great step to understand cell types evolution over time and would be useful to transfer knowledge between different species. In particular, in the case of a disease, if a drug is tested on the mouse and has been proven effective, we could know if it would possibly be effective on the human by looking at the similarity between the targeted cell types. Even though the same cell type across species is performing the same function, this does not mean that it is achieved by the same gene expression.

By following this objective, Song et al. [6] performed a benchmarking of different cross-species integration strategies of scRNASeq data, by testing different homology mapping[1], different tools and parameters. They found that tools based on neural networks were able to deliver the best results. Not long after this study, a new scRNASeq data analysing tool called expiMap was published by Lotfollahi et al. [3]. expiMap is based on a neural network, designed to integrate data from diseased and healthy patients, that is intended to be biologically explainable. It is an autoencoder neural network, taking into account the study from which the data comes. expiMap does not only use the count matrices[2], but also a binary matrix indicating which gene belongs to which GP[3]. It is performing a first training on the reference dataset (healthy patient) and then a second one on the query dataset (diseased patient), using the knowledge of the first training and the ability to learn new GPs. This is already promising and in addition, it has shown to be more efficient than scVI, one of the most robust tool to perform cross-species integrations according to Song et al.[6].

For these reasons, the aim of my internship is to test the ability of expiMap to perform cross-species integrations. To achieve this, I will use brain scRNASeq data from *Homo sapiens* (human), *Mus musculus* (mouse) and *Drosophila* (fly). In the first place, I will start by integrating the human with the mouse because they diverged 87 million years ago while human and fly diverged 680 million years ago. The objective is to have the same cell types from the two species grouped together while keeping the species specific information. In a second time, I will perform the integration between the mouse and the fly and in the end if it is possible, try to integrate the human, the mouse and the fly together.

[1]Only a certain type of homologous genes, all types of homologous genes, all genes.

[2]In a count matrix, the cell (i,j) indicate how many time the gene i has been found in the cell j.

[3]In a GP matrix, if the cell (i, j) is 0 then the gene i is not in the GP j and on the contrary, the gene i is in the GP j if the cell (i, j) is equal to 1.

2 Methods

2.1 Dataset

We have selected the following datasets for analysis:

- Human : the dataset is from Allen Brain Map named *Human M1 10X* [1]. It contains single-nucleus transcriptomes from 76533 cells derived from 2 post-mortem human brain specimens in the primary motor cortex. I used the raw count matrix (matrix.csv), the cell metadata (metadata.csv) and the gene metadata (human_MTG_2018-06-14_genes-rows.csv). The matrix and cell metadata files are available here[4] and the gene metadata file can be downloaded here[5].
- Mouse : the dataset is from the database CellxGene named *An integrated transcriptomic and epigenomic atlas of mouse primary motor cortex cell types: 10X_nuclei_v3_Broad* [8]. The data comes from the Allen Mouse Brain Atlas and contains single cell transcriptomics from 159738 cells from the mouse primary motor cortex. I used the .h5ad (AnnData v0.8) file available here[6].
- Fly : the dataset is The Fly Cell Atlas : single-cell transcriptomes of the entire adult Drosophila - 10x [4]. It contains 527530 cells from 15 different individuals. I used the file E-MTAB-10519.project.h5ad available here[7].

2.2 Working environment

All the scripts are based on Python (v3.10.10) and have been written in jupyter lab (v3.5.3), are either jupyter notebook or python file, with the following packages :

- scanpy : v1.9.3
- anndata : v0.9.1
- umap : v0.5.3
- numpy : v1.24.3
- scipy : v1.10.1
- pandas : v2.0.1
- scarches : v0.5.8
- biomart : v0.9.2

Since the data is voluminous, I used the cluster of the EBI via an ssh access in the terminal to run my scripts. First, I set up my working directory in the cluster with miniconda, created a new environment and installed all the necessary packages. Once it was working, I downloaded the data on the cluster. Finally, I asked the cluster for 500GB of memory in a long run job, that way I was able to launch jupyter lab and my notebooks were running on the cluster. To run expiMap I sent the job to a high performance computing node in order to get it done faster, where the job took between 3 and 6 hours to run.

[4]https://portal.brain-map.org/atlases-and-data/rnaseq/human-m1-10x
[5]http://celltypes.brain-map.org/api/v2/well_known_file_download/694416044
[6]https://cellxgene.cziscience.com/collections/ae1420fe-6630-46ed-8b3d-cc6056a66467
[7]http://ftp.ebi.ac.uk/pub/databases/microarray/data/atlas/sc_experiments/E-MTAB-10519/

2.3 Data preprocessing

In order to analyse the datasets, the first step was to create the anndata objects (cf. Annexes Fig.7) to work with. The main attributes of the anndata object are the count matrix, the var dataframe which is a dataframe storing all the relative information to the genes, hereafter refer to as the gene dataframe, the obs dataframe storing the relative information to the cells, hereafter refer to as the cell dataframe. To create and manipulate the objects I used the scanpy and anndata [7] packages. These two packages are built jointly and are used to analyse single-cell gene expression. The anndata object are stored in .h5ad files, thus for the fly and the mouse the objects were already well constructed. For the human, I constructed it with the different files previously downloaded, adding the matrix, the gene dataframe and the cell dataframe.
The mouse and the human objects were complete with all the cell types annotation and the gene names, the data had already been sorted as well. The fly object needed more manipulation, first I only keeped the cells from the head. Second, I removed all the cells which were not annotated, finally resulting in an object of 23332 cells and 14078 genes. I also had to remove genes from the raw count matrix which were not in the count matrix.

Once the objects were well constructed, I performed the scanpy tutorial on preprocessing and clustering[8] for each of those datasets to analyse them and perform quality control. In the tutorial, the data are normalised and scaled before performing a Principal Component Analysis (PCA) and a neighbourhood graph then we look for the marker genes. Since the anndata objects are complicated to understand in the beginning, by performing the tutorial, it was also a way to get familiar with them and understand their structure more. For the mouse, we had to try several dataset before having one suitable for the next stage.
Each object was constructed, analysed and saved (in .h5ad file) in a separate notebook. The notebooks created and the new objects are saved in the data folder.

2.4 Getting the homologous genes

Once the objects are created, we need to do the link between the human object and the mouse one. In order to do that, we query the biomart server and get the *hsapiens_gene_ensembl* dataset thanks to the package biomart. To prepare the data for Expimap, I constructed an anndata object with all the homologous genes available in the dataset between the human and the mouse.

2.5 expiMap

To run expiMap I followed the basic tutorial for query to reference mapping using expiMap[9]. The main steps of the tutorial are : first getting the new matrix with the GPs based on a file, second training expiMap on the reference object, third training expiMap a second time with the query object and fourth visualising the results. I started with the default values and later tried other ones to adapt the training to my data.

[8]https://scanpy-tutorials.readthedocs.io/en/latest/pbmc3k.html
[9]https://docs.scarches.org/en/latest/expimap_surgery_pipeline_basic.html

3 Results

3.1 Data preprocessing

To make sure the selected data were suitable for further analysis, I looked at the PCA plot and the Uniform Manifold Approximation and Projection (UMAP) plot, both dimension reduction techniques.
This allow us to see whether the object are well constructed or not, meaning whether the annotation are matching the right cells. I started with the human, in Figure 1 this is a UMAP plot colored by the cell type class labels[10] and they are well separated from each other, being promising for the subclass labels[11] plot. Indeed in Figure 2 the UMAP plot shows the cell type subclass labels that are also well distinguished from one another. Meaning that the cell types annotation is well associated with the cells.

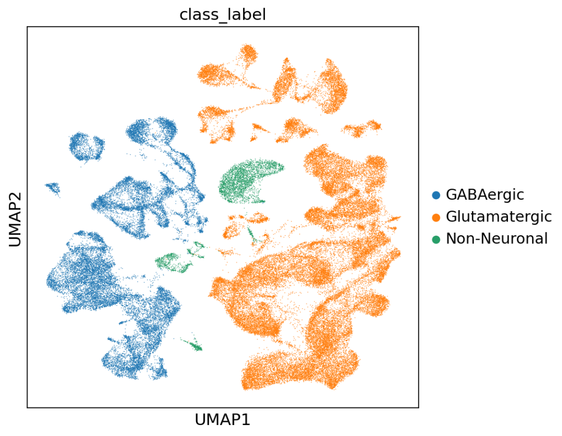


Figure 1: UMAP plot of the cell type class labels for the human dataset

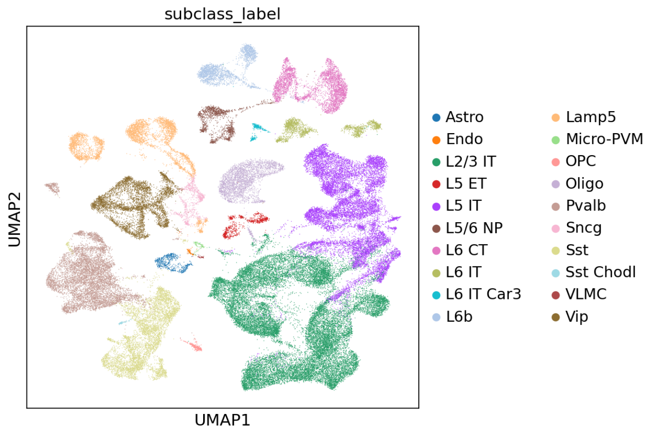


Figure 2: UMAP plot of the cell type subclass labels for the human dataset

For the mouse, the first two datasets I tried needed to be constructed from different files and required filtering, but they were not giving satisfactory plots (cf. Annexes Fig 8 & 9). Meaning that I was not able to match the cell annotation to the right cell. However, the last one we chose gave better results (cf. Fig 3). The fly's result of the analysis is the Figure 10 in the annexes.

3.2 Creating new anndata object with homologous genes

The following information is collected from the ensembl query : the ensembl gene id of the human gene and its name, and if available the mouse homologous gene information such as the ensembl id and the name. Moreover, we have access to the orthological relationship between the human and mouse genes which could be either one to one (o2o) : meaning one gene of the human has a single homologous gene in the mouse, one to many (o2m) : one human gene has

[10]the class label is one of the three following cell types : GABAergic, glutamatergic or non-neuronal.

[11]the subclass label is one of the following cell types, being more precise than the class label : Sst, L5/6 NP, L5 IT, L2/3 IT, Oligo, L6 CT, L5 ET,L6b, Lamp5, L6 IT, Pvalb, Vip, Astro, Micro-PVM, L6 IT Car3, OPC, Sncg, Endo, Sst Chodl, VLMC, Macrophage, Peri, CR, Prog/IP, SMC

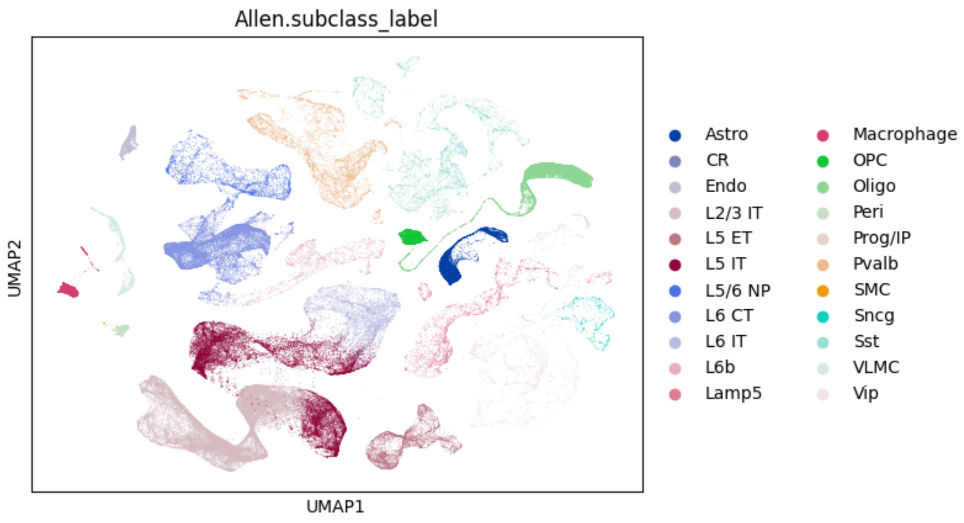


Figure 3: UMAP plot of the cell type subclass labels for the mouse dataset

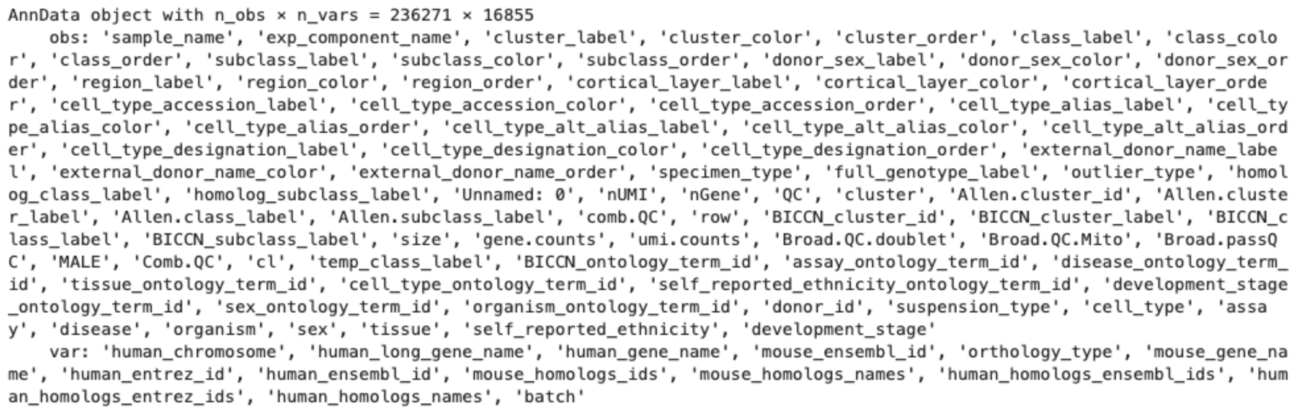
many homologous genes in the mouse, or many to many (m2m) : many genes in the human are homologous with many genes in the mouse. The result of the query is stored in a dataframe.

First I restricted the dataframe to the human genes present in my human object. To do that, I queried again the biomart server to get the correspondence between the ncbi id, which was the reference in the human object, and the ensembl id. From there I constructed a dataframe with only the human genes and the mouse genes present in my objects, with all the previous information including the ncbi id. However, some ncbi id had many ensembl id matching and some ensembl id had many ncbi id matching, so we chose to keep only the first one for each. In the end, I had a dataframe with almost 22000 homologous relationships. The dataframe is saved in a .csv file in the repository data/human_mouse.

The next step was to create the count matrices. In order to do so, I started by constructing the o2o one. We chose to start from there because it was the most intuitive one, progressing from the simplest to the most complex case. I concatenated the two anndata object with a function of the anndata package[12], giving me a new anndata object with the two matrices combined. I concatenated the cell and the gene dataframes individually before putting everything together in the o2o anndata object.
Next for the o2m, we realised that we had o2m but also many to one (m2o), meaning multiple human genes in connection with one mouse gene. I splitted the two cases to treat them individually. Here, to construct the count matrix, we choose to concatenate the count line of the single gene with the average for each cell of the count matrix from all the homologous genes associated. At the same time, I constructed the gene dataframe by saving all the information for homologous genes from the many side and adding one of these genes as reference[13] for the relation. The construction of the count matrix and the gene dataframe is done by a function that I wrote in a first time for the o2m and that I have later adapted for the m2o. For the o2m,

[12]anndata.concat()

[13]We took the first one in the alphabetical order given by the method .sort().



```
AnnData object with n_obs × n_vars = 236271 × 16855
    obs: 'sample_name', 'exp_component_name', 'cluster_label', 'cluster_color', 'cluster_order', 'class_label', 'class_colo
r', 'class_order', 'subclass_label', 'subclass_color', 'subclass_order', 'donor_sex_label', 'donor_sex_color', 'donor_sex_or
der', 'region_label', 'region_color', 'region_order', 'cortical_layer_label', 'cortical_layer_color', 'cortical_layer_orde
r', 'cell_type_accession_label', 'cell_type_accession_color', 'cell_type_accession_order', 'cell_type_alias_label', 'cell_ty
pe_alias_color', 'cell_type_alias_order', 'cell_type_alt_alias_label', 'cell_type_alt_alias_color', 'cell_type_alt_alias_ord
er', 'cell_type_designation_label', 'cell_type_designation_color', 'cell_type_designation_order', 'external_donor_name_labe
l', 'external_donor_name_color', 'external_donor_name_order', 'specimen_type', 'full_genotype_label', 'outlier_type', 'homol
og_class_label', 'homolog_subclass_label', 'Unnamed: 0', 'nUMI', 'nGene', 'QC', 'cluster', 'Allen.cluster_id', 'Allen.cluste
r_label', 'Allen.class_label', 'Allen.subclass_label', 'comb.QC', 'row', 'BICCN_cluster_id', 'BICCN_cluster_label', 'BICCN_c
lass_label', 'BICCN_subclass_label', 'size', 'gene.counts', 'umi.counts', 'Broad.QC.doublet', 'Broad.QC.Mito', 'Broad.passQ
C', 'MALE', 'Comb.QC', 'cl', 'temp_class_label', 'BICCN_ontology_term_id', 'assay_ontology_term_id', 'disease_ontology_term_
id', 'tissue_ontology_term_id', 'cell_type_ontology_term_id', 'self_reported_ethnicity_ontology_term_id', 'development_stage
_ontology_term_id', 'sex_ontology_term_id', 'organism_ontology_term_id', 'donor_id', 'suspension_type', 'cell_type', 'assa
y', 'disease', 'organism', 'sex', 'tissue', 'self_reported_ethnicity', 'development_stage'
    var: 'human_chromosome', 'human_long_gene_name', 'human_gene_name', 'mouse_ensembl_id', 'orthology_type', 'mouse_gene_na
me', 'human_entrez_id', 'human_ensembl_id', 'mouse_homologs_ids', 'mouse_homologs_names', 'human_homologs_ensembl_ids', 'hum
an_homologs_entrez_ids', 'human_homologs_names', 'batch'
```

Figure 4: Final anndata object. The object contains 236271 cells (n_obs) : 76533 cells from the human and 159738 cells from the mouse, and 16855 genes (n_vars). The dataframe associated with the cells (obs dataframe) contains 86 columns, 38 from the human, 45 from the mouse and 3 new columns I added. The 3 new columns are : *sample name* which contains all the sample names from the human and the mouse, *homolog_class_label* which contains all the class labels from the human (*class_label*) and from the mouse (*Allen.class_label*) and *homolog_subclass_label* which contains all the subclass labels from the human (*subclass_label*) and the mouse (*Allen.subclass_label*). The dataframe for the genes (var dataframe) contains 14 columns from the human, the mouse and the ensembl query. The *batch* column keeps track of the original object between o2o, o2m, m2o, m2m.

there are 356 relations and for the m2o, 212 relations.
The last one to create was the m2m matrix. I adapted the function I created for the o2m to concatenate the average counts of the multiple human genes with the average counts of the multiple mouse genes, saving the genes information in the same manner as the o2m. You can find an example of a function in the Annexes (cf. Fig 11). For the m2m, 129 relations were found. For each object, I concatenated the cell dataframe from the human and the mouse before adding it to the new object.

Finally, I concatenated the four objects in one using the same function from anndata, adding the gene dataframe and the cell dataframe separately, to obtain the final object including all the homologous relationships (cf. Fig 4). For each object I kept all the information from both the mouse and the human in the dataframes for the cells and the genes. Each object (o2o, o2m, m2o, m2m and the fourth concatenated) is saved in the data/human_mouse repository.

3.3 expiMap

The first integration I did was using the default parameters of the expiMap tutorial, the reactome.gmt annotation file for the GPs (also used in the tutorial), the mouse counts; of the final object previously obtained, as the reference and the human counts as the query. The default parameters were not giving a satisfying integration, the two species were not mixed, so I changed them several times gradually.

The first parameter to look at in the training is *alpha_kl* : this parameter is responsible for the correction of the integration. If the final plot looks like a glob, then it is better to decrease the parameter and conversely, if the integration is bad it should be increased. With that information, I changed *alpha_kl* from 0.5 to 0.6, which turned out to be the most suitable value. Second, we had a few GPs ($\sim$250) while the training is set for 300-500 GPs. This is controlled by the *ALPHA* parameter, which I decreased from 0.7 to 0.6. After that, the training on the mouse data was showing a great result (cf. Fig 5). However, the results on Figure 6 after the second train on the human data are not showing a good integration between the human and the mouse. I tried to decrease to 6 the minimum number of genes required for a GP to be selected as well, which was set to 12, in order to get more GPs ($\sim$350) but it did not change the results of the training.

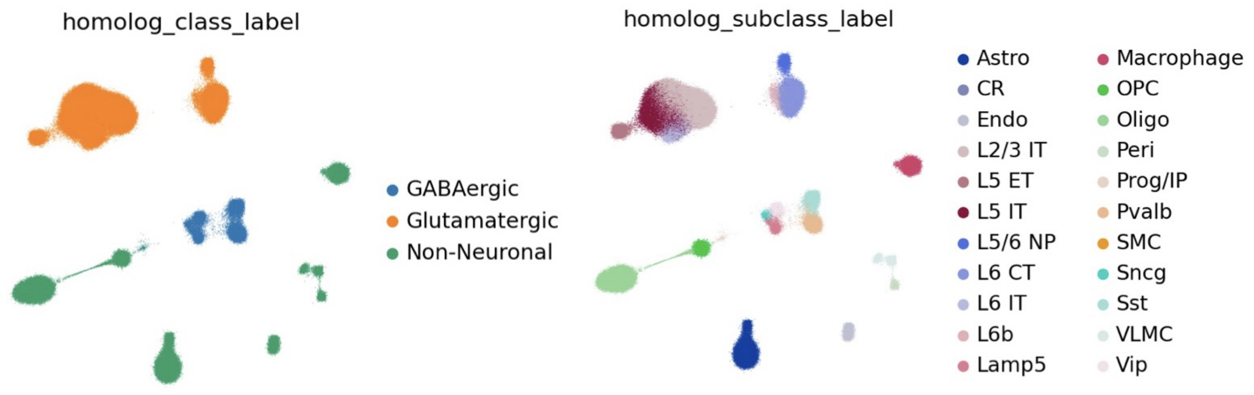


Figure 5: UMAP plots of the mouse after training expiMap. The parameters used for the training were *ALPHA = 0.6, alpha_kl = 0.6.* The first plot is the UMAP plot coloured by the class labels and the second plot is the UMAP plot coloured by the subclass labels of the cells.

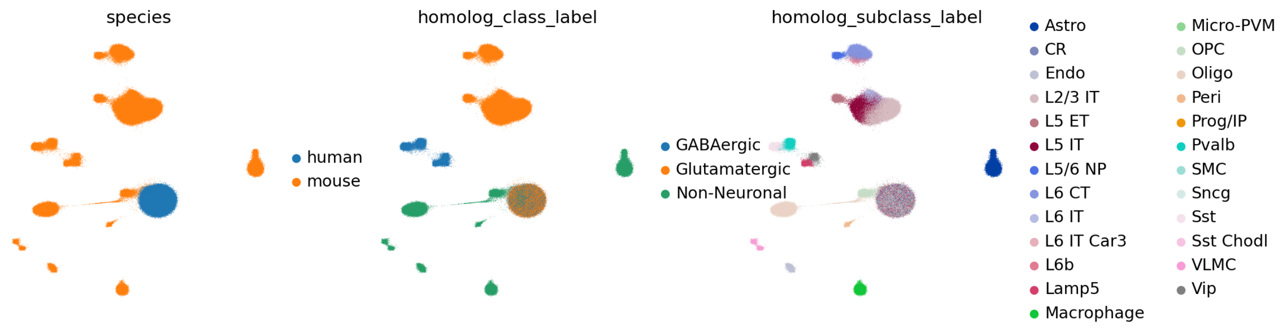


Figure 6: UMAP plots of the mouse and the human after training with expiMap. The parameters for the first training are *ALPHA = 0.6, alpha_kl = 0.6*, for the second training I used the default ones. The first plot is the UMAP plot coloured by species, the second one is coloured by cells class labels and the third one by the cells subclass labels.

4 Discussion

4.1 Dataset

We chose those datasets because the human and the mouse ones come from the same institute and are done with the same techniques. This allows to minimise the batch effect due to the possible differences on how the data have been collected. Moreover, for both datasets the cell type annotation was the same for the class labels and the subclass labels, making preprocessing easier.
For the fly dataset, we used the fly cell atlas because it is a complete and reliable dataset that also uses the same techniques for collecting data. However, since at this step of my work I have not work with this dataset yet, I will not discuss the results (cf. Annexes Fig 10) because I cannot affirm that this dataset does not contain any hidden problems to perform an integration with the other datasets.

4.2 Homologous genes anndata object

To construct the final object we made choices for its structure. First, we kept only the first gene when there were problems in the ids matching. In the first place we wanted to keep only the most highly expressed genes. Unfortunately the only way to do it was by hand and since there were 266 mismatched genes id, we simply choose to take the first one. Second, while constructing the matrices, we averaged the counts when we had multiple homologous genes to match together. This is called doing a “meta-gene”, meaning we are combining many genes together, this has been used by Geirsdottir et al. [2] and has proven to be working.

Next, many manipulations of the objects have been done, leading to several verification of the construction :

- o2o : since I used the anndata.concat() function, no big error of construction could have been made. I made sure it was concatenated along the right axis. To add the genes dataframe, the genes names had been kept so I checked that the index of the new dataframe I was adding was equal to the genes names in the o2o object.
- o2m : this object is constructed line by line, therefore no mismatch between the line in the matrix and in the genes dataframe should have been done. To verify that the names of the cells and the genes were indeed matching, I compared the human part of the o2m object to the original human object, all the counts were equal for the same cell and gene names.
- m2o : same thing that the o2m, but I checked the mouse counts. They are all equal as well.
- m2m : for this one, it was not really possible to verify the counts, since they are all averages. However, the function I used is an adaptation of the one I made for the o2m and m2o and those objects are apparently well constructed, the m2m should also be.
- final object : the matrix is constructed with the anndata.concat() function, the result should be the one expected. For the gene dataframe and the cell dataframe, I checked that the indexes were equal between the dataframe and the object before adding them to the

final object.

If time permits during the next month of my internship and I can move on to the mouse and fly integration, I will adapt the function from the m2m to be a general function that could directly be applied to all the orthologous types at once. At least the same function could be applied for the o2o, o2m, m2o and m2m, even if we treat each case separately. However, the function needs to be adapted on a case-by-case basis depending on the original dataset we are using and the ensembl query result, because each one has different attributes in their dataframes.

4.3 expiMap

The first integration results using expimap (cf. Fig 6) are not encouraging because the human cells are not well integrated with the mouse ones. It is well known that it takes time to adapt the algorithms' parameters to our data, so we thought about the ones that could influence the result. First, in the default parameters they are only using 2000 highly variable genes for the training, this is a common value but often needs to be increased when analysing brain data. Second, we have few GPs for the training as well compared to their default values. To solve this problem, we want to try using another file, the Gene Ontology annotation. This file is a binary matrix instead of a .gmt file (like the one used in the tutorial), therefore I will need to create a new matrix which corresponds to the format of the one used in expiMap training.

In parallel, to verify it was not a construction problem of the final object, we runned Yuyao's integration scripts using scANVI, currently the best tool to perform corss-species integration according to Song et al.[6]. The integration was not giving great results as well. Then we tried only for the GABAergic cells, and with the default parameters, it finally start showing satisfacotry results : the same cell types are starting to be map together (cf. Annexes fig 12). This proves that the object is indeed well constructed, and there are no matching problems between the cells and their annotation. Moreover, the difference between two species that have diverged for so long is already difficult to map for an algorithm, but here, it seems that the cross-species difference across several cell classes are also too challenging for the algorithm to handle at once.

The next step of this project is to try to integrate the three cell classes independently, first with the default files for the GPs and increasing the number of highly variable genes from 2000 to 3000. Second, I will also perform these integrations by using the Gene Ontology annotation matrix. This could be a solution to our problem because the reactome file that is used is creating GPs based on the biological pathways, meaning the genes are involved in the same output, whereas Gene Ontology annotation is based on biological processes, meaning the genes are grouped together when they have similar function. Moreover, Gene Ontology database is more precise and more annotated than the reactome one. Therefore, the use of Gene Ontology annotation could provide a better understanding of our data than reactome annotation. Finally, if the results are conclusive, I will repeat the operation with the mouse as the query and the fly as the reference. The integration could be more challenging since the two species have diverged a very long time ago.

5 Code availability

All my scripts are available on Github here[14].

6 Acknowledgements

I would like to thank my supervisors, Irene Papatheodorou and Yuyao Song, for allowing me to do this internship in a such supportive environment. I have learned a lot in various subjects: scRNASeq data integration, homologous genes, neural network and how to use new tools. However, the most important knowledge for me is what I learned about research in general: I am now aware that everything takes time, it is always helpful to talk with other people and it is normal to make mistakes, more we can learn a lot about it. Those are what I will keep in mind for, perhaps, my future PhD. I am very grateful to Yuyao Song for her attention and her precious advice throughout the entire internship. I would also like to thank Nadja Nolte, Anna Vathrakokoili-Pournara, Francesca Nadalin and Paula Cardenas for our enriching conversations. Finally, I am truly thankful to the Embassy of France in London for the financial support, that allowed me to get the most of this internship in Cambridge, hopping this partnership will benefit many other students that are looking to get into the research world.

[14]https://github.com/anaellle/CrossSpeciesIC

7 References

[1] T. E. Bakken, N. L. Jorstad, and Q. Hu et al. Comparative cellular analysis of motor cortex in human, marmoset and mouse. *Nature*, 2021.

[2] Laufey Geirsdottir, Eyal David, Hadas Keren-Shaul, Assaf Weiner, and Stefan Cornelius Bohlen et al. Cross-species single-cell analysis reveals divergence of the primate microglia program. *Cell*, 179(7):1609–1622.e16, 2019.

[3] M. Lotfollahi, S. Rybakov, and K. Hrovatin et al. Biologically informed deep learning to query gene programs in single-cell atlases. *Nat Cell Biol*, 25:337–350, 2023.

[4] De Waegeneer Maxime, Jasper Janssens, Hongjie Li, and Stein Aerts. The Fly Cell Atlas: single-cell transcriptomes of the entire adult Drosophila - 10x dataset. https://www.ebi.ac.uk/biostudies/arrayexpress/studies/E-MTAB-10519, 2021.

[5] Shafer M.E.R. Cross-species analysis of single-cell transcriptomic data. *Front. Cell Dev. Biol.*, 7, 2019.

[6] Yuyao Song, Zhichao Miao, Alvis Brazma, and Irene Papatheodorou. Benchmarking strategies for cross-species integration of signle-cell rna sequencing data. bioRxiv, April 2023.

[7] Isaac Virshup, Sergei Rybakov, Fabian J. Theis, Philipp Angerer, and F. Alexander Wolf. anndata : Annotated data. bioRxiv, December 2021.

[8] Z. Yao, H. Liu, and F. Xie et al. A transcriptomic and epigenomic cell atlas of the mouse primary motor cortex. *Nature*, 2021.

8 Annexes

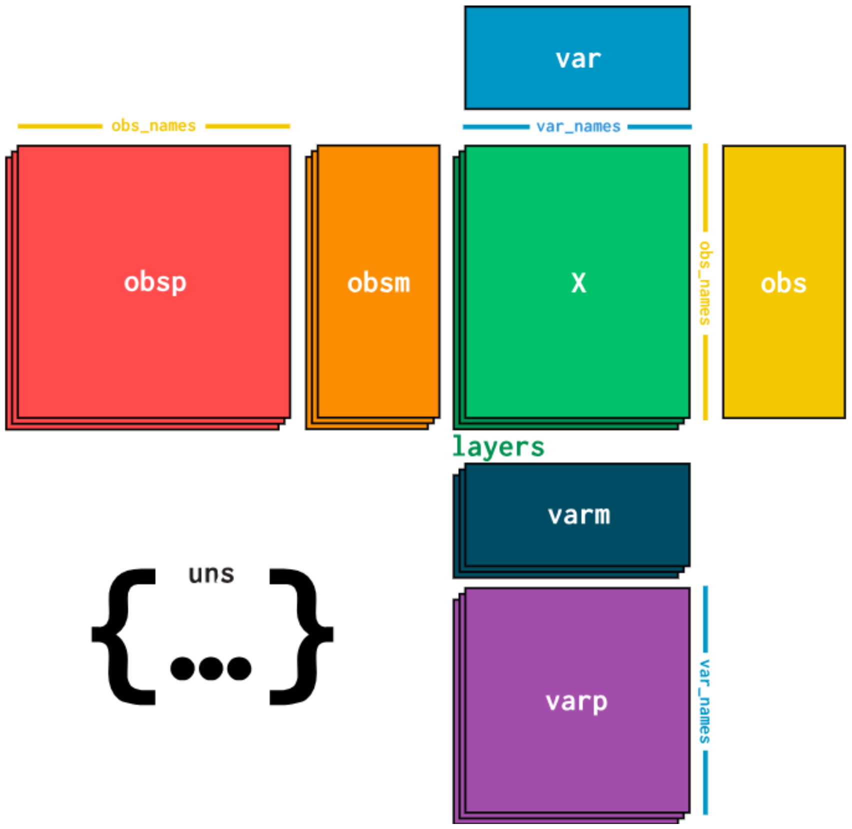


Figure 7: anndata object. X: count matrix. obs: observation dataframe containing the information about the cells. obs_names: index of the obs data frame. var: dataframe containing the information about the genes. var_names: index of the var dataframe. obsp, obsm, varm, varp, layers: other attributes containing information after manipulation of the data like the Principal Component Analysis or the highly variable genes.

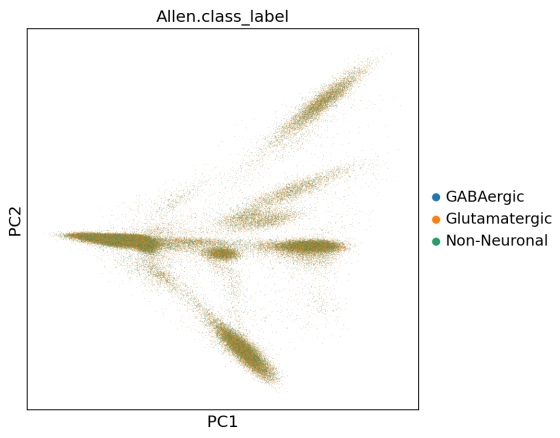


Figure 8: PCA plot of the cell type class labels for the first mouse dataset

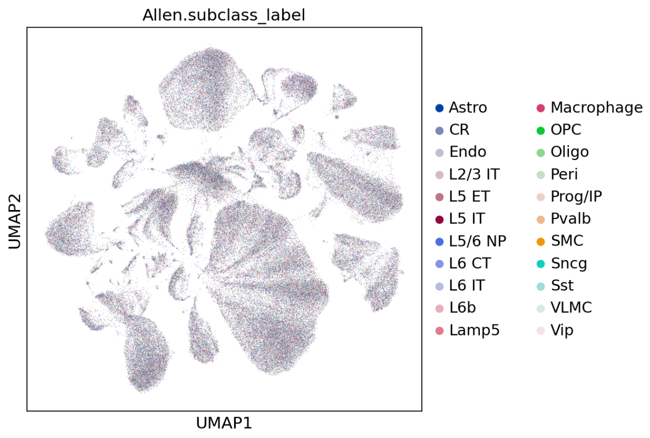


Figure 9: UMAP plot of the cell type subclass labels for the first mouse dataset

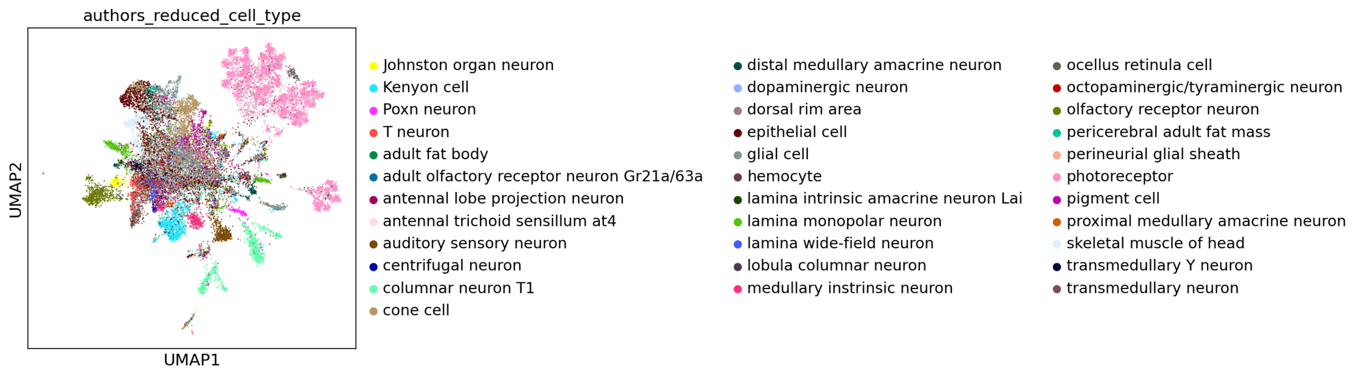


Figure 10: UMAP plot of the cell type annotated by the authors for the fly dataset

```
def construct_m2m_matrixanddf():
    # def the variables to return
    matrix = []
    todo = many2many_genes
    var_df = []

    # start the loop throught the m2m dataframe
    for index, row in many2many_genes.iterrows():
        #get the human id
        human_id = row.human_ensembl_gene_id

        # check if it has already been done, if not we go into the if
        if str(human_id) in todo.human_ensembl_gene_id.values:
            # starts new lines for dataframe
            var_line = {}
            var_line['orthology_type'] = row.orthology_type

            # get all the mouse and human homologous genes
            working_df = get_gene_list(human_id)

            # def mouse arrays for stocking info
            mouse_names = []
            mouse_ensembl_ids = []
            mouse_index = []

            # def human arrays for stocking info
            human_names = []
            human_ensembl_ids = []
            human_entrez_ids = []
            human_index = []

            # pass throught all the homologous genes
            for index2, row2 in working_df.iterrows():
                if row2.human_ensembl_gene_id not in human_ensembl_ids :
                  # save the information about current human gene
                    human_names.append(row2.human_external_gene_name)
                    human_ensembl_ids.append(row2.human_ensembl_gene_id)
                    human_entrez_ids.append(row2.human_entrezgene_id)

                    # get the index of the gene in the human matrix
                    human_index.append(np.where(m2m_human.var_names ==
                                                          m2m_human.var.loc[
                                                          m2m_human.var.
                                                          entrez_id == row2.
                                                          human_entrezgene_id
                                                          ].index[0])[0][0])

                if row2.mouse_homolog_ensembl_gene not in mouse_ensembl_ids :
                    # save the information about current mouse gene
                    mouse_names.append(row2.mouse_homolog_gene_name)
                    mouse_ensembl_ids.append(row2.mouse_homolog_ensembl_gene)
                    # get the index of the gene in the mouse matrix
                    mouse_index.append(np.where(m2m_mouse.var_names ==
                                                          m2m_mouse.var.loc[
                                                          m2m_mouse.var.
                                                          index == row2.
                                                          mouse_homolog_ensembl_gene
                                                          ].index[0])[0][0])

            ######### Get mouse matrix #########
            # get the cells counts for each gene
            mouse_homolog_matrix = np.asarray(m2m_mouse.X.todense()[:,
                                                       mouse_index].T)
            # get the mean for each cell
            mouse_counts = np.mean(mouse_homolog_matrix, axis=0)

            ######### Get human matrix #########
            # get the cells counts for each gene
            human_homolog_matrix = m2m_human.X[:, human_index].T
            # get the mean for each cell
            human_counts = np.mean(human_homolog_matrix, axis=0)

            ######### Add new line to matrix #########
            # add the newline to the count matrix
            if matrix == []:
                matrix = np.array([np.concatenate([human_counts,mouse_counts]
                                                          )])
            else :
                matrix = np.append(matrix, [np.concatenate([human_counts,
                                                          mouse_counts])], axis
                                                          = 0)

            # Remove all genes from todo
            todo = todo[~todo.mouse_homolog_ensembl_gene.isin(
                                                       mouse_ensembl_ids)]
            todo = todo[~todo.human_ensembl_gene_id.isin(human_ensembl_ids)]

            # add the information for the genes to the df
            var_line['human_homologs_ensembl_ids'] = human_ensembl_ids
            var_line['human_homologs_entrez_ids'] = human_entrez_ids
            var_line['human_homologs_names'] = human_names
            human_ensembl_ids.sort()
            var_line['human_ensembl_id'] = human_ensembl_ids[0]
            var_line['human_entrez_id'] = working_df.loc[working_df.
                                                       human_ensembl_gene_id ==
                                                       human_ensembl_ids[0]].
                                                       human_entrezgene_id.iloc[0
                                                       ]
            var_line['human_gene_name'] = working_df.loc[working_df.
                                                       human_ensembl_gene_id ==
                                                       human_ensembl_ids[0]].
                                                       human_external_gene_name.
                                                       iloc[0]
            var_line['human_long_gene_name'] = m2m_human.var.loc[m2m_human.
```

```
                                                       var.entrez_id == var_line[
                                                       'human_entrez_id']].
                                                       gene_name.iloc[0]
            var_line['human_chromosome'] = m2m_human.var.loc[m2m_human.var.
                                                       entrez_id == var_line['
                                                       human_entrez_id']].
                                                       chromosome.iloc[0]

            var_line['mouse_homologs_ids'] = mouse_ensembl_ids
            var_line['mouse_homologs_names'] = mouse_names
            mouse_ensembl_ids.sort()
            var_line['mouse_ensembl_id'] = mouse_ensembl_ids[0]
            var_line['mouse_gene_name'] = working_df.loc[working_df.
                                                       mouse_homolog_ensembl_gene
                                                        == mouse_ensembl_ids[0]].
                                                       mouse_homolog_gene_name.
                                                       iloc[0]

            # add the newline to the rows of the df
            var_df.append(var_line)

    return matrix.T, var_df, todo
```

Figure 11: m2m construction function : function in the many2many_human_mouse.ipynb notebook to create the m2m count matrix between the human and the mouse as well as the gene dataframe for the m2m anndata object.

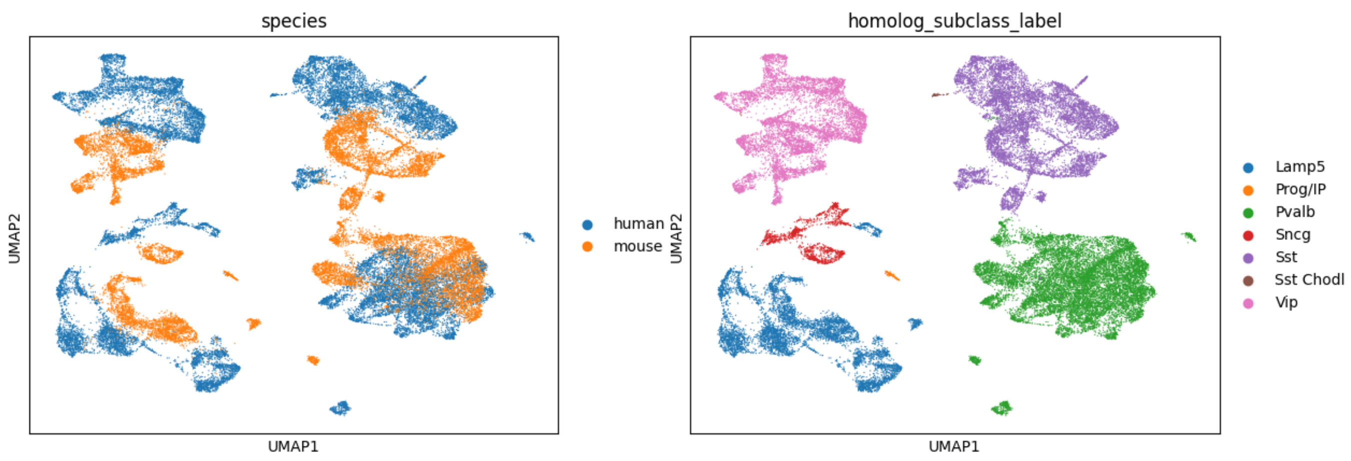


Figure 12: scANVI UMAP plots : UMAP plots of the human and the mouse integration using scANVI, taking only the GABAergic cells. The first UMAP plot is colored by species and the second plot is colored by cell subclass labels.