Contents

Introduction

1.1 Overview

Understanding important characteristics of cell populations and the role of certain genes in cell development mostly relies on the analysis of single-cell data. The identification of relevant genes and effective analysis is essential in many biological realms, notably cancer detection and the development of treatments [1]. This information can be extracted from gene counts, thus with the increasing amounts of data obtained lots of novel analyzing techniques have been developed to aid in this process [2].

Currently, quality assurance, dimensionality reductions, unsupervised clustering methods, and substantial pre-processing are all used in the analysis of the massive amounts of data generated by cell sequencing [3]. Such data may contain thousands of features, which are grouped and reduced using techniques such as PCA, Isomap, tSNE, UMAP and more.

Different cell types express different genes and different sequencing platforms capture different numbers of genes (as well as having different expressions across different sequencing techniques). For that reason, the process of classifying cell types is very susceptible to noise and varied extraction methods - hence, analysis and studies on such datasets is more difficult due to the (often) incomparable nature of datasets coming from different sources [3]. For example, there might be notable variations in the data from the sequencing techniques dropSEQ and smartSEQ (which will be discussed

later) [4]. As such, combining varied datasets may result in significant differences and impair the final model's ability to generalise reliably.

The ideas presented in this thesis are a good step forward for automatic annotation since they make use of new directions such as transformer architectures in the hopes of creating a foundational generalizable model. Moreover, evaluating its performance on cell-type annotation tasks, while also fine-tuning the model on more complex cell-state annotation tasks will give a good overview of the benefits or drawbacks of using transformers and whether they might be a promising solution to a widely applicable standardized annotation procedure.

Specifically, a novel transformer architecture for genomic data will be evaluated in its foundational (pre-trained) form on simpler cell-type annotation tasks on varied cell datasets. Subsequently, it will be fine-tuned on a more complex task of cell-state annotation (specifically Hypoxic cancer states) and evaluated against other baseline models with lower complexities - to discern the utility of such involved aforementioned structures.

1.2 scRNA Data

Transcripts are sequenced into reads using a technique called single-cell RNA sequencing, or scRNA-seq for short [4]. As previously indicated, this technology has been applied to the analysis of data related to a number of disorders, most notably cancer cells. Because of both biological and technological constraints, single-source scRNA-seq data (obtained from a single cell) is more complex and noisy than bulk RNA-seq data (obtained from a sample of cells) [4].

There are multiple approaches for sequencing scRNA-seq data - such as Drop-seq and Smart-seq (each with their pros and cons). For example, Drop-seq quantified mRNA levels with less amplification noise by using unique molecular identities, while Smart-seq identifies a higher number of genes per cell [5]. Drop-seq is a more economical way for measuring transcriptomes of large numbers of cells than other approaches like Smart-seq, which is more effective for smaller counts of cells [5]. The best results

might be obtained by using the data extraction method that best fits the goal, or if that is not possible, by using several sources and skillfully combining them during the study.

Count matrices are often the format in which each of these methods produces its output. More specifically, the rows correspond to individual cells and the columns show the gene expression levels in the dropSEQ and smartSEQ approaches [6]. As previously said, it is important to remember that the rows and columns of these matrices represent separate entities.

	Gene 1	Gene 2	Gene 3
Cell 1	100	140	250
Cell 2	95	24	125
Cell 3	130	45	70

Table 1.1: Example format of a count matrix.

1.3 Data Pre-Processing

Now, this section will go into great detail about the difficult process of preparing scRNA data to give an overview of the issues found in practice.

The presence of noise and artefacts in raw scRNA-seq data can potentially mask the dataset's true relevance (i.e. the biological signal). To convert raw count matrices from scRNA-seq data into accurate and illuminating cellular information, a laborious technique is needed.

During the first phase, quality control, poor quality cells are easily identified and removed. These cells show reduced counts of detected genes, shallow counts at count depths, or excessive expression of mitochondrial genes on average [3]. There are two methods for carrying out the filtering process: either manually by setting appropriate criteria, or automatically by using methods like median absolute deviation computations. Furthermore, instruments like SoupX and CellBender are employed to reduce contamination resulting from ambient RNA by computing and deducting the contami-

nation fraction [3]. In the end, empty droplets and doublets—which could have more than one cell or none at all—are found and removed. When it comes to doublet detection, scDblFinder is a highly recommended choice [3].

Additionally, two essential steps in the data transformation process are normalisation and variance stabilisation. Count normalisation commonly uses modifications like the shifted logarithm (log(x + 1)) [3] to establish comparability amongst cellular profiles. To guarantee that the normalisation approach is optimal, the particular analytical objectives should direct the choice. One method that successfully lessens the impact of aberrant profiles on the overall data structure is variance stabilisation [3]. Confounding variables that are biological or technical are then taken into account. Data integration approaches are then used to address batch effects, which are the result of sample differences [3]. More advanced techniques like Harmony and linear embedding models like canonical correlation analysis are suitable for simpler integration jobs [3]. Deep learning approaches, like scANVI, scVI, and scGen, perform better in more complex scenarios [2].

Ultimately, genes that are physiologically informative—especially those that vary among subpopulations—are highlighted via feature selection. Deviance, for instance, uses gene-wise models to determine which genes are involved. Following this, the data is streamlined and visualised using dimension reduction techniques, such as PCA [2]. Conversely, more flexible visualisations are offered by t-SNE, UMAP, and PHATE [2]; these are vital for cross-checking interpretations using quantitative metrics to avert mistakes. Because of this, transforming scRNA-seq data necessitates a painstaking set of complex steps that ensure data integrity, remove biological artefacts and noise, and magnify the real biological signal before analysis. With the use of this enhanced data, one can start the analysis knowing that the data more closely captures the innate qualities [3].

State of the Art Annotation Techniques

Given that the focus of this thesis is that of evaluating transformer models' performance in the realm of automatic annotation, it is fundamental to have a grasp of the current annotation models and tools available to better understand what current methods have to offer.

2.1 Machine Learning Models

Cell-type annotation has shown results with traditional machine learning algorithms. To categorise cells into recognised kinds, a large number of these models rely on marker genes or pre-established reference databases. One such example is scCATCH, which aggregates information from several sources including the CD Marker Handbook, CancerSEA, the Mouse Cell Atlas project, [7] and CellMarker to determine marker genes for the inputted clusters [8]. A grading algorithm then allocates a certain cell type to each cluster. Similar to this, another model known as SCSA finds marker genes for each cluster population and then uses simple scoring to perform a cell-type categorization [9].

Operating at the cluster level, SCINA departs from single-cell analysis. It enables the identification of differentially expressed genes within each cluster by fitting a bimodal distribution to marker genes [10]. Using a Bayesian probabilistic model, CellAssign divides cells into predefined cell types according to the expression of reference marker genes [11]. This technique can handle complex experimental designs and operates at

the level of individual cells.

The k-nearest neighbour approach is also used in cell-type annotation using scmap-cell and scmap-cluster programmes [12]. Although scmap-cluster concentrates on mapping at the cluster level by calculating the similarity between reference clusters and query cells, scmap-cell quickly approximates nearest-neighbor searches for individual cells [12]. Random forests are another important machine learning technique that is used. Models like SingleCellNet, which identify multiple cell types associated with a single or group of cells by giving a score, are used in this technique [13]. This is very valuable because it enables the identification of cells that are in transition states.

In order to achieve reliable and accurate cell-type annotation, CHETAH presents a multi-round comparison strategy in which candidate cells are iteratively compared to reference subsets using distinct sets of genes in each round [14]. Last but not least, Moana (SVM) employs a novel strategy by using hierarchical classification to maximise the number of cells examined while reducing computation time, providing a good trade-off between resource consumption and result quality [15].

2.2 Deep Learning Models

When large-scale datasets are available, deep learning models frequently outperform conventional machine learning techniques in terms of performance. For example, LAmbDA interprets the task as a transfer learning issue and uses artificial neural networks (ANNs) to perform cell-type categorization [16]. Developed specifically for the Mouse Cell Atlas, SuperCT is a supervised classifier that learns from datasets inside the atlas to reliably identify cellular subtypes while enduring batch effects [17]. Another well-known ANN with a strong reputation for accurately categorising cellular subtypes is ACTINN, which excels at managing extensive research projects [18].

Additional models of deep learning, such as scNym, which makes use of semi-supervised adversarial neural networks, demonstrate the adaptability and promise of DL for cell-type annotation [19]. More specifically, scNym learns rich representations of cell identity that provide efficient annotation transfer by leveraging information from both new,

unlabeled data sets and labelled data sets [19]. Furthermore, because of its broad applicability and learning capacity, it achieves performance better than previous methods by allowing transfers in annotations across experiments despite biological and technical variances [19].

Transformers

3.1 Transformer Architecture

After having analysed current state of the art methods for cell-type annotation, there is a more recent development in the field, specifically the Deep Learning side. Transformers are now being used and applied in various industries due to their great ability to generalise and learn the nuances of very distinct data - especially when sequential.

A transformer block, the core unit of transformer architectures, consists of two primary components: an attention mechanism (cross- or self-, the difference will be touched upon later) and a feedforward neural network. The attention mechanism allows the model to weigh the importance of each element in the input sequence relative to others, capturing long-range dependencies and contextual information effectively [20]. Following the attention mechanism, the feedforward neural network applies nonlinear transformations to the attended representations, enabling the model to learn complex mappings between input and output. These transformer blocks are then stacked together to form the backbone of transformer-based models [20].

When looking specifically at the attention mechanism, it is a novel way to interpret sequential inputs in a coherent and interconnected manner. As the word ”attention” suggests it’s a way for the model to focus on a specific portion of the input sequence in correlation to the inputs received. This structure allows the model to keep in mind the rest of the sequence while only working a specific part of it [20]. In addition, multi-head

attention allows the model to focus on multiple nuances at once (think of it as multiple attentions running in parallel), and hence generalise much better given a specific dataset [20].

Regarding the feedforward neural network added after it, its purpose is to first non-linearly project the output into a larger feature space and then back down into its original size space - this has shown, empirically, a better ability to learn and has helped the model improve its representational capabilities.

Generally speaking, in the original transformer one finds two types of blocks - encoders and decoders. These structures work together to bring the required result by being trained simultaneously through a large corpus of data inputs and outputs. However, it is not always the case that these two types of blocks work together - specifically in BERT architectures or the now well-known GPT (Generative pre-trained transformer) architectures.

In the former, the encoder blocks are the only things kept, and therefore the final part that would come from the decoder section in a classic transformer is not there. scGPT (despite being called GPT) is actually an encoder model meaning that it is very capable at the following two tasks of Masked Language Modelling (MLM) and Next Sentence Prediction. They are respectively, generating a token for a ”hidden” one in a sequence and predicting a sequence coming sequentially after the input one [21].

3.1.1 Mathematical Formulation

In order to understand the inner workings of a transformer architecture and then to understand that of an Encoder, a ”chronological” approach will be taken, starting from the inputs moving through the architecture until the final outputs.

Given a sequence of tokens (length n), these are one hot encoded into a matrix S ($s \times n$) where each column is an item of the sequence and the rows represent the dictionary of that transformer (which contains s elements) - the cells have a 1 in the row that matches the token and 0 everywhere else [22]. The matrix S is subsequently pre-

multiplied by another matrix W^E (of size $m \times s$), also known as the Encoder Matrix, where each column represents the m-dimensional embedding of the token in the sequence [22]. The output is a resulting new matrix X where the dimensions are $m \times n$ - so a sequence of m-dimensional embeddings which can be understood by the model. The reason why dense lower dimensional embeddings are used is that they reduce sparsity and increase representativity of the output matrix [22].

The next theoretical part that needs explaining is Attention, along with its variants such as self-, cross- and even masked as they will allow an understanding of how scGPT actually works - due to the fact that they created their own custom attention mechanism. As aforementioned, attention aims to give contextual relations to sequential data by letting tokens ”attend to” others - despite this arguably abstract explanation, in reality an attention is simply a matrix of weights that are then used to multiply each token and weigh it respectively - with regards to the current step/focus in the model [20]. Specifically (cross-) attention is mathematically formulated as follows (along with its multiplication with the input to produce the ”weighed” output):

$$Q = W^Q X \tag{3.1}$$

$$K = W^K C \tag{3.2}$$

$$V = W^V C \tag{3.3}$$

$$\Omega = \frac{1}{\sqrt{d_k}} K^\top Q \tag{3.4}$$

$$A = \text{softmax along columns } (\Omega) \tag{3.5}$$

$$U = VA \tag{3.6}$$

$$Z = W^O U \tag{3.7}$$

Specifically, let’s look at what the various matrices mean [22]. This is the query-key formulation of the attention mechanism, which most closely resembles how it is actually implemented in code. W^Q, W^K, W^V and W^O are all trainable matrices that help generate the respective query, key, value and output matrices - we can think of this as follows: the query matrix Q creates an embedding of the input sequence that when multiplied

with the key matrix K (an embedding of the context) provides the (rescaled) weights Ω that each token should pay attention to another [20]. Specifically A (the softmax-ed version) has n columns (one for each item in the sequence X) and each one is a vector of normalized weights which represent how much importance (from 0 to 1) that respective row token has on the column token. This weighing is then pre-mulitplied by the value matrix V and the output matrix to produce an output Z of the original dimension of the initial input X, the only difference being that now the embeddings account for contextual clues [20] - we can imagine it as the vector being shifted in the embedding space to better represent the semantic meaning in the input sequence.

Now that the process of attention is generally clear, it is important to note a detail - the presence of the matrices X and C. When attention is done on X and $C = X$ we call this self attention; otherwise (as in the example above) we call it cross-attention [20]. The meaning is simply that we either calculate attention on the input sequence itself (in self-attention A is a $n \times n$ matrix) or on the context generated by the encoder block (in cross-attention A is a $m \times n$ matrix, where m is the length of the context sequence) [20]. It is therefore evident, that the encoder blocks must use self-attention (as they are the ones producing C) and decoder blocks use cross-attention (when possible) on the context produced by the encoders.

Masking in encoder blocks is fundamental for tasks like masked language modeling and sequence-to-sequence tasks. While the encoder processes the entire input sequence at once, we need to prevent it from ”peeking” at future tokens during self-attention, as it may affect the representations learned [20]. To accomplish this, we employ masking, which limits the encoder’s access to only the information it has encountered up to the current position.Therefore, through masking we allow the encoder blocks to only have access to the information they have reached up to by setting the pre-softmaxed attentions of those tokens coming after, to -inf [22]. This means that after the softmax, the attention of the current token on the other ones coming after will be 0 (no contextual relation). This is simply done by adding the following matrix M to Ω:

$$M_{j,i} = \begin{cases} 0, \text{if } j \leq i \\ -\infty, \text{else} \end{cases} \tag{3.8}$$

Now the internal mechanisms are clear lets look at the overarching architecture of the encoder and decoder blocks. Evidently, the usual structure is that the encoder blocks come before the decoder blocks, and often there are multiple encoder blocks followed by multiple decoder blocks - the number is an arbitrary choice and depends on the task at hand (note that the output of the previous layer makes up the input of the current one) [20]. So firstly, looking at a generic encoder structure:

$$\bar{z}^{(i)} = \text{Self-Attention}\left(x^{(i)}; X\right) \tag{3.9}$$

$$\bar{x}'^{(i)} = \text{LayerNorm}\left(\bar{x}^{(i)} + \bar{z}^{(i)}\right) \tag{3.10}$$

$$\bar{z}'^{(i)} = \text{MLP}(\bar{x}'^{(i)}) \tag{3.11}$$

$$\bar{x}''^{(i)} = \text{LayerNorm}\left(\bar{x}'^{(i)} + \bar{z}'^{(i)}\right) \tag{3.12}$$

It is useful to note that usually, the LayerNorm layers (equations 3.2 and 3.4) include a skip connection - this is done to prevent information loss as we travel further into the network [20].

Then, looking at a generic decoder structure:

$$\bar{z}^{(i)} = \text{Masked Self-Attention}\left(x^{(i)}; X\right) \tag{3.13}$$

$$\bar{x}'^{(i)} = \text{LayerNorm}\left(\bar{x}^{(i)} + \bar{z}^{(i)}\right) \tag{3.14}$$

$$\bar{z}'^{(i)} = \text{Cross-Attention}\left(x^{(i)}; C\right) \tag{3.15}$$

$$\bar{x}''^{(i)} = \text{LayerNorm}\left(\bar{x}'^{(i)} + \bar{z}'^{(i)}\right) \tag{3.16}$$

$$\bar{z}''^{(i)} = \text{MLP}(\bar{x}''^{(i)}) \tag{3.17}$$

$$\bar{x}'''^{(i)} = \text{LayerNorm}\left(\bar{x}''^{(i)} + \bar{z}''^{(i)}\right) \tag{3.18}$$

We can see here that firstly masked self-attention is calculated and then cross-attention - however like before, every time there is a Layer Norm, there is also a skip connection from previous layers, this makes sure that the information is never lost deeper into the network [20].

ScGPT is an encoder-only model and as will be shown in the next part, its architecture resembles the respective one above, except for some evident changes that had to

be made due to the nature of the task at hand.

3.2 scGPT

As one may expect, especially having read the previous explanations of scRNA data, unlike words or other input forms commonly used in transformers such genomic data can actually pose some extra difficulties when wanting to implement the transformer architecture in transcriptomic tasks. Specifically, its non-sequentiality and variational differences between extraction methods may hinder performance and prevent an encoder model from generalizing well.

scGPT, a novel model working in just this field trained on scRNA-seq data, has built a compelling transformer architecture combining various distinct yet effective features that together allow it perform really well. Its initial training purpose being that of (1) generating unknown gene expression values based on known gene expression (MLM) and (2) generating whole-genome expression given an input cell type condition (Next Sentence Prediction) [23], furthermore allows it to retain incredible performance when working on diverse downstream tasks such as cell-type annotation as will be shown later on. The output of the foundational pre-trained model is hence that of a latent representation of genes or cells put in input.

3.2.1 Architectural Overview

Working through the architecture, important steps or issues that had to be resolved by changing the architecture (from a basic encoder) are explained. Firstly, one has to convert the gene expressions into an embedded form for the transformer model to be able to do operations with it. This is done by treating each gene in a cell as a token and representing all the genes of a i by $t_g^{(i)} \in N^M$ where:

$$t_g^{(i)} = [id(g_1^{(i)}), id(g_2^{(i)}), ..., id(g_M^{(i)})] \tag{3.19}$$

and id is a lookup function for the id of that gene in the scGPT dictionary [23]. Moreover, additional tokens like $< cls >$ and $< pad >$ where introduced to respectively group gene tokens into a single-cell representation and padding inputs to fixed lengths in the case

they may fall short.

A further problematic point addressed is indeed the input data (scRNA) due to its variational differences in gene counts when considering data taken from different sequencing methods. The solution to this problem has actually been that of moving away from raw gene counts but rather move to a standardized binning method; specifically, when a bin number is specified, bins are created with edges equivalent to quantiles in the range of those counts [23]. Therefore, each row (single cell) in the count matrix has its gene expression value replaced by the respective bin label in which that count would be found, as follows for a single gene count in cell i:

$$x_j^{(i)} = \begin{cases} k, \text{if } X_{i,j} > 0 \text{ and } X_{i,j} \in [b_k, b_{k+1}], \\ 0, \text{if } X_{i,j} = 0. \end{cases} \tag{3.20}$$

The above [23] effectively allows for the input structure to be totally independent from the extraction method and unlike normalization or log transformation ensures that the data points are always in the same denoted range giving much more stability to the model.

A further position-wise condition tokens vector $t_c^{(i)}$ was devised [23] along with the above, to include inherent characteristics that came from experimental alterations. This again defined as a vector of size M where each element represents the condition of that specific input gene [23].

Combining the above the final embedding $h^{(i)} \in R^{M \times D}$ inputted into the transformer is defined as follows:

$$h^{(i)} = \text{emb}_g(t_g^{(i)}) + \text{emb}_x(x^{(i)}) + \text{emb}_c(t_c^{(i)}) \tag{3.21}$$

where emb represents the standard PyTorch embedding layer [23]. It should be noted that fully connected layers where used for the binning expression values to further enhance expressivity. Morevoer, as explained above the output of one layer is the input

of the next and therefore:

$$\begin{aligned} h_0^{(i)} &= h^{(i)} \\ h_l^{(i)} &= \text{transformerblock}(h_{l-1}^{(i)}) \quad \forall l \in [1, n] \end{aligned} \tag{3.22}$$

The next fundamental change that occured in scGPT, architecturally, was the modification of the attention mechanism due to the inherent non-sequential nature of gene expressions, unlike most other NLP tasks. Specifically, unlike for other models, there is no concept of sequentiality. scGPT decides prediction order based on so called 'attention scores'. The general formulation of attention masking has been provided in the previous section, so the focus now will just be that of the newly formulated version.

In the input embedding $h_l^{(i)}$ each token can belong to 3 distinct categories: (1) the <cls> or <pad> tokens, (2) known genes with token and expression value embeddings and (3) unknown genes for which we have to predict expression values [23]. Generally, scGPT only allows attention computations between the specific query gene and 'known' genes [23]. Each element in the attention matrix is reformulated as follows:

$$a_{i,j} = \begin{cases} 0, \text{if } j \notin \text{unknown genes}, \\ 0, \text{if } i = j \text{ and } j \notin \text{unknown genes}, \\ -\infty, \text{if } i \neq j \text{ and } j \notin \text{unknown genes}. \end{cases} \tag{3.23}$$

Hence, scGPT predicts expressions for these unknown genes via stacked transformer blocks with the masked-attention map described above [23]. Moreover, for the gene expression prediction objective, a multi-layer perceptron was used with a mean square loss:

$$L = \frac{1}{U_{unk}} \sum_{j \in U_{unk}} (\text{MLP}(h_n^{(i)}) - x_j^{(i)})^2 \tag{3.24}$$

where U_{unk} is the set of output positions for unknown genes and the $x_j^{(i)}$ is the actual gene expression value to be predicted [23].

3.2.2 Downstream Tasks

In addition to the foundational scGPT model - it was also finetuned on important downstream tasks such as cell-type annotation, perturbation response detection, GRN inference and more - by adding a trainable NN at the end of it.

Specifically, for the cell-type annotation task, a reference set with true labels was used. Gene expressions were log transformed, normalized and binned as explained above - all pre-trained weights were used to initialize the model, except for the weights of feed-forward network that were randomly initialized. The task was trained on the objective of minimizing classification loss [23].

For the perturbation response prediction task, highly-variable genes were selected and instead of using binned values, the perturbation response predictions were done on log-transformed gene expressions to increase performance. Moreover, the condition token was used to state whether a specific gene had been perturbed or not (binary condition). The fine-tuning objective was that of predicting a perturbed cell from a starting control cell, and this was done by randomly pairing a non-perturbed control cell with each perturbed cell.[23].

Two approaches were investigated for the gene regulatory network (GRN) task: zero-shot and fine-tuned. Gene similarity networks were constructed in the zero-shot setting using k-nearest neighbours on pretrained gene embeddings from the scGPT model. Functional gene clusters were then discovered by the network after clustering and gene program extraction. In contrast, the fine-tuned configuration used a scGPT model that had been fine-tuned on immune human data to construct a gene similarity network that was comparable. Then, perturbed versus control cell sets were used to identify the most-influenced genes using attention-based target gene selection to help understand how genes interact with one another [23].

Here we will create a new downstream task of cell-state annotation by adapting the cell-type one and moreover allowing both encoder and NN weights to be trainable - in the hopes of improving performance for a more complex objective.

Fine-Tuning

4.1 Datasets

Before delving into the fine-tuning procedure which took place, it is important to first understand the specific datasets that were used in the modelling process and why they were chosen.

Evidently, to compare the performance of traditional annotation methods versus new transformer architectures, a diverse set of data and tasks had to be employed, focusing on both cell-type and cell-state annotation (the latter was chosen due to the more complex nature of the task, which ultimately aimed to push the limits of the transformer to see whether its true value added would be on tasks where an increased level of abstraction is required). The cell-type annotation data was sourced from various parts of the human body, including foetal cells, PBMC, and PIC. In contrast, the cell-state annotation data came from TNBC cell lines, with labels indicating different states of Hypoxia. The references for the data used can be found in the Appendix A.

Let's delve into the details of each dataset to then discuss about the pre-processing that occurred and how the fine-tuning was carried out. Firstly, looking at the Foetal Dataset it is a dataset with 4504 cells and 29680 gene expressions respectively, these cells are generally blood cells and they are being classified with respect to two different labels: cell-type (13 of them such as Granulocytes, Megakaryocytes, Endothelial...) and cell-subtypes which go one step deeper than the previous labels (23 of them such

as Granulocytes 1, Granulocytes 2 ...).

Subsequently, the PBMC dataset (peripheral blood mononuclear cells) is again another cell dataset - this time much bigger than the previous with 14039 cells and 12762 gene expressions respectively. These were split between control and stimuli samples quite evenly with around 47% stimuli and 53% control. This time there was only one label class which was cell-type (8 of them such as CD14+ Monocytes, Dendritic cells ...).

Moreover, the PIC dataset (pancreatic islet cells) is the further dataset that was used for cell classification - with 6321 cells and 34363 genes respectively. These cells were again classified for cell-type (13 possible labels of the king gamma, acinar, alpha ...)

Finally, the TNBC dataset (triple negative breast cancer cell lines) which is what was then used for the fine-tuning (specifically its hypoxic states) was as follows: 9324 cells and 19046 gene expressions respectively. The labels could either have been Hypoxia and Normoxia, or the stage labels (8 possible stages, split evenly as 4 hypoxia and 4 normoxia stages), however when classifying the performance was checked on the hypoxic states as they are generally those of interest in transcriptomic analysis. It is important to note, as will be shown later, that the fine-tuning on this dataset was done in two different ways (one time training on only the hypoxic data, and one time training on both hypoxic and normoxic data - despite always validating on just hypoxic data).

4.2 Pipeline

The fine-tuning pipeline was adapted from scGPT's own reference of how they performed fine-tuning on an annotation downstream task - therefore what will be discussed in this section is the overall final process that took place in order to train scGPT on the cancer data.

Firstly, the hyperparameters were set - these were left like in the original scGPT fine-tuning as it seemed to give good performances, the only thing that was modified was the number of epochs as after some tests it seemed to converge after around 50 epochs.

The training ran with a batch size of 32 and an initial learning rate of 1e-4. The model architecture was left like the default one which includes 4 layers of TransformerEncoder, each with a layer size of 128 and 4 attention heads, incorporating a dropout probability of 0.2. Advanced features such as Masked Value Prediction (MVC) for cell embedding and Elastic Cell Similarity (ECS) were disabled or set to minimal influence with a threshold of 0.0. The model is also set to leverage fast transformer techniques.

To utilize the full potential of scGPT, its 'human' model was selected, the largest available. The vocabulary files were checked against the dataset's genes to ensure all necessary tokens were included. Missing special tokens were appended to the vocabulary, and genes in the dataset were filtered based on their presence in the model's vocabulary. The model parameters, such as embedding size, number of heads, hidden dimension size, and the number of layers, were evidently extracted from the aforementioned hyperparameters.

The data that was previously loaded now had to be pre-processed, specifically in addition to being sum-normalized (i.e. the gene counts across a cell sum a to specific value, in this case 1e4) they were also log-normalized and subsequently binned into 51 bins (hyperparameters, default of scGPT). The binning procedure was exactly as described in the previous chapter when discussing the architecture. Moreover, now that the input data was created, the output labels were set to be the cell-states of cancer cells and the data was split into train and test (with a test size of around 10%).

The training uses cross-entropy loss as evidently it's trying to match the predicted labels to the true labels (or at least, their one-hot encoded versions as explained in the transformer section). Moreover, an ADAM optimizer is used which is a very performant version of gradient descent which is pretty standard in Deep Learning models.

Other Modelling Techniques

To investigate other modelling approaches, XGBoost and Neural Networks (NNs) were used - they provided baseline performances on the various datasets. Moreover, by doing so one can understand the advantages (or disadvantages) of various model complexity levels while also justifying (or not) the performance of the fine-tuned scGPT model.

5.1 Neural Networks

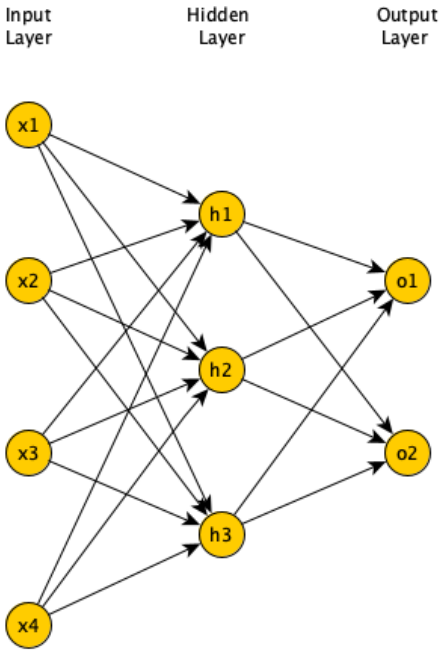
To begin with, Multi-Layer Perceptron (MLP) architectures were developed for the Foetal, PBMC, PIC, and Cancer datasets. These NNs take the full gene expression of a cell as input, a parameter that varies with each dataset. The network architecture resembles a funnel, starting with the wide gene expression input and gradually reducing dimensionality to the label classes (by halfing number of neurons each hidden layer - in total there were 4 hidden layers). ReLU activation functions and batch normalization layers were, in addition, included to stabilize the model and make weight updates more accurate.

Datasets were binned using a consistent procedure to create two versions for each dataset and label type. This approach allows for the assessment of both baseline performance and the impact of binning on model effectiveness, giving insight on its relevance in transcriptomics.

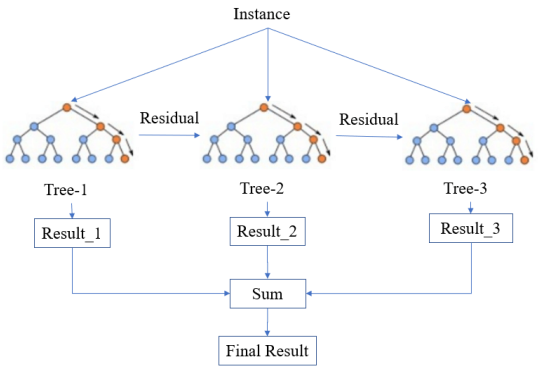
5.2 XGBoost

To further simplify the modeling approach and create a spectrum of complexity levels, a XGBoost model was used (gradient boosted ensemble of decision trees). Training XGBoost on the same datasets provided a comparison against the more complex NNs and scGPT.

As with the NNs, the datasets were binned, producing two versions for each dataset and label type for XGBoost as well. By employing both NNs and XGBoost, we can better understand the trade-offs between model complexity and performance, providing valuable insights for future modeling efforts in the field.



(a) Example Funnel-like Structure of NN Source: [24]



(b) Example XGBoost Structure Source: [25]

Results

It's time to assess how well each dataset and modelling technique performed in different circumstances, having previously discussed the different datasets and modelling strategies used (scGPT, MLP, and XGBoost).

We will first assess the fine-tuned scGPT using the Hypoxia cell-state dataset. This dataset comes in two variations: one where training was conducted using only the Hypoxia data points, and the other where training was conducted using the entire collection of Normoxia-Hypoxia data points. The evaluation was based only on the Hypoxia labels in both instances.

After that, several datasets will be used to evaluate XGBoost's performance, including the Foetal, PBMC, PIC cell-type, and Hypoxia cell-state datasets (in the same two versions as the scGPT fine-tuning). In order to look into how data binning affects performance, both raw and binned data versions will be examined.

In a similar manner, the MLP will be assessed using the same datasets and modifications as XGBoost. Once more, in order to learn more about how data binning affects performance, both raw and binned data variations will be taken into consideration.

Ultimately, embeddings for the PIC, PBMC, Foetal, and Hypoxia datasets (two versions as above) will be constructed in order to compare the other models results with the pre-trained scGPT (more precisely, scGPT-human). Several techniques, includ-

ing DBSCAN, Agglomerative Clustering, and K-means, will be used to cluster these embeddings. Furthermore, a basic 3-layer MLP classifier trained on the unique labels of each dataset will receive the embeddings as input. In order to determine whether the pre-trained scGPT gives any advantages over the neural network and XGBoost methodologies stated above, this will provide a baseline performance.

6.1 Cell Type Classification

The cell type classification task is evidently an easier one as from a set of genes and their respective counts the model merely has to understand the overarching class of that cell. In the table below, the outline of the results achieved is presented. The accuracy is calculated as the right number of labels divided by the total number.

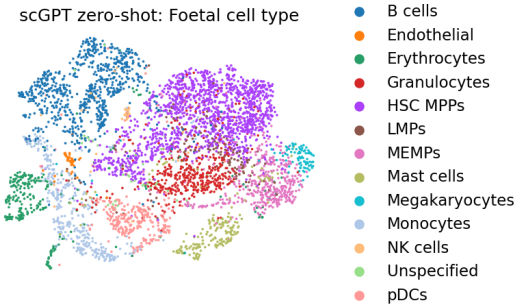
Task	Model	Accuracy (Binned Data)	Accuracy (Raw Data)
Foetal cell type	XGBoost	0.891	0.887
Foetal cell type	MLP	0.859	0.788
Foetal cell type	scGPT-human + classifier	0.850	NA
PBMC cell type	XGBoost	0.949	0.947
PBMC cell type	MLP	0.926	0.887
PBMC cell type	scGPT-human + classifier	0.925	NA
PIC cell type	XGBoost	0.978	0.977
PIC cell type	MLP	0.968	0.877
PIC cell type	scGPT-human + classifier	0.884	NA

Table 6.1: XGBoost, MLP and pre-trained scGPT on cell-type classification.

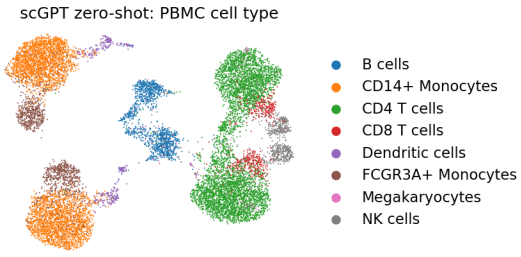
As one can easily see, first of all, the binning procedure largely improves the performances of classification (and despite not being related to the focus of this research, it can be seen as potentially a fundamental building block in implementing more complex architectures such as the transformer one - since it allows more stable and less varied inputs). Moreover, for all datasets, the pre-trained scGPT never achieved the highest performance - the main contenders were always XGBoost and the MLP, with the for-

mer often taking the lead (despite the gap reducing in the ”binned” case for the reason explained above).

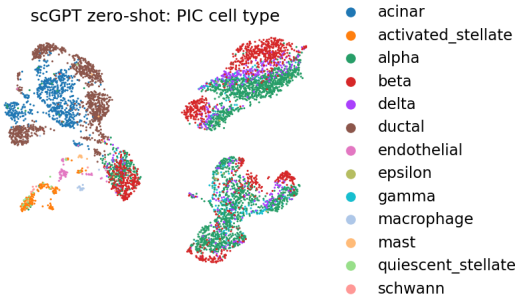
Moreover, for the scGPT predictions (since it produces embeddings) there was also the opportunity to plot a UMAP (dimensionality reduction technique that brings the embeddings down from size 512 to 2 to plot in a scatter graph) for each dataset which will be shown below to see whether visually some results can be seen. It is important to note that the color in the UMAP represent the true labels and they are used to see whether some separability can be found in the dataset.



(a) Foetal Cells UMAP



(b) PBMC Cells UMAP



(c) PIC Cells UMAP

Figure 6.1: UMAP plots for scGPT-human embeddings in 2D

As one can see, the clusters produced by the embeddings, despite looking very nice in terms of correlation and separability, are evidently hard for a clustering algorithm to match due to their uneven structures that often overlap - for this reason their accuracy was quite low and therefore the results for it won’t be included as they don’t add value to the implications obtained (unlike the above classifiers).

6.2 Cell State Classification

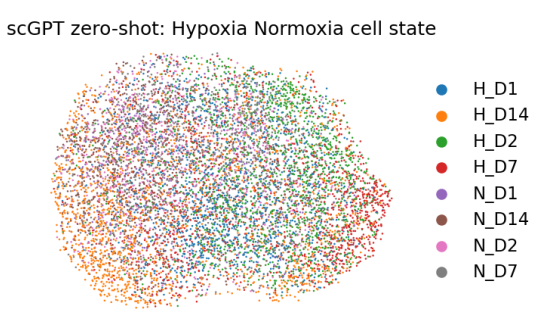
The hypoxic cell state classification task is, unlike the previous one, much harder as from a set of genes and their respective counts the model need to be explained or understood well enough to be able to say not only the macro type but morevoer, its nuances to determine the states. In the table below, the outline of the results achieved is presented. The accuracy is, again, calculated as the right number of labels divided by the total number. The task column, unlike before, was omitted as it is always the same task of hypoxic state classification.

Model	Accuracy (Binned Data)	Accuracy (Raw Data)
XGBoost (H,N)	0.829	0.819
XGBoost (H)	0.839	0.857
NN (H,N)	0.802	0.806
NN (H)	0.817	0.864
Finetuned-scGPT (H,N)	0.763	NA
Finetuned-scGPT (H)	0.673	NA
scGPT-human + classifier (H,N)	0.664	NA
scGPT-human + classifier (H)	0.693	NA

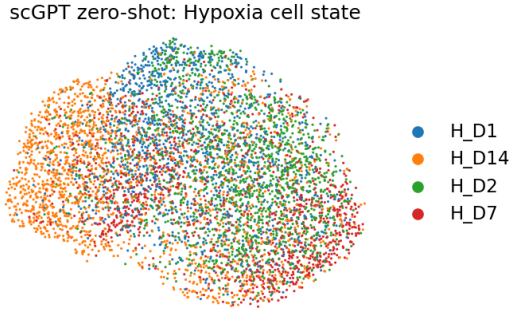
Table 6.2: XGBoost, NN, and fine-tuned scGPT on hypoxic cell state classification. H,N means trained on both Hypoxia and Normoxia data, H means trained only on Hypoxia data - always evaluated only on Hypoxic states.

These time the results are a bit more complex to interpret - specifically, we do not see any clear distinction between Raw vs Binned data, suggesting that potentially for more complex tasks or depending on the situation, binning actually (as one may expect) reduces the information contained in the data by reducing the overall variance (since you are effectively trainig on bin labels, instead of actual gene expressions). Moreover, it is evident how the fine-tuned scGPT on both variants of hypoxic data do not match the performance of the XGBoost and MLP (which are also much simpler models) and the pre-trained scGPT's performance with a classifier added at the end again does not provide decent performances - especially when put against the aforementioned ones.

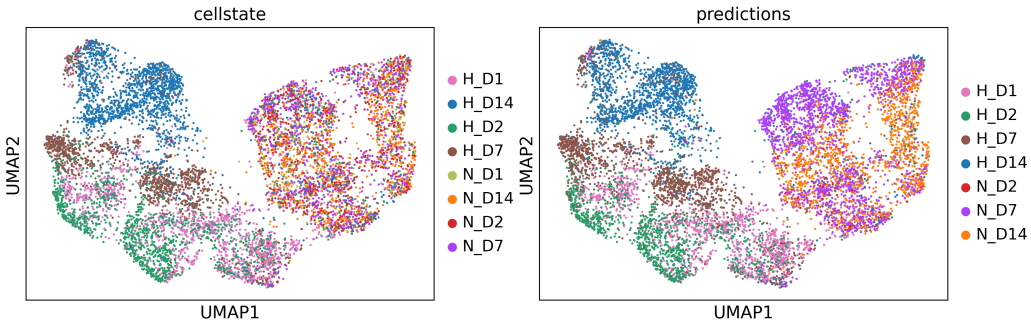
Much like before, for the scGPT predictions there was again the opportunity to plot a UMAP for each dataset which will be shown below to see whether visually some results can be seen. The color will again represent the true label of the data point and in this case the plots will be between the pre-trained scGPT human and the finetuned ones. Moreover, for the finetuned scGPT we will also plot the true cell states to show whether the finetuning actually helped the model better learn the data.



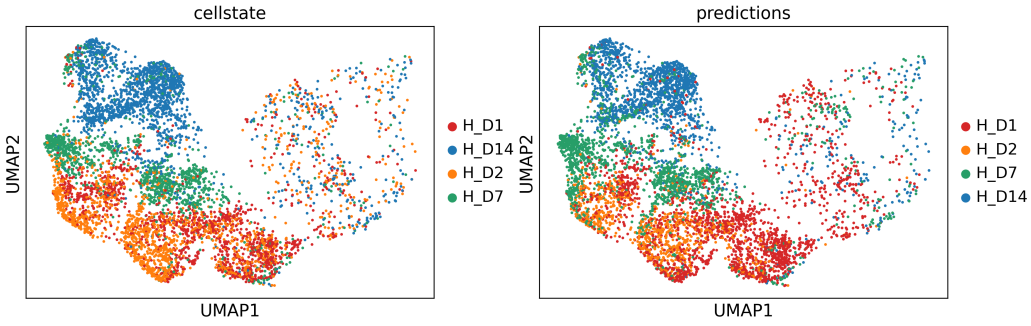
(a) Hypoxia Normoxia Zero Shot UMAP



(b) Hypoxia Only Zero Shot UMAP



(c) Hypoxia Normoxia Finetuned UMAP



(d) Hypoxia Only Finetuned UMAP

One can easily notice the imminent difference between the pretrained and finetuned model - it is clear how by allowing the scGPT encoder to retrain its weights specifically to the task one can achieve a much better performance - where, despite not being as good as the XGBoost and MLP in its accuracy, still very nicely represents the hypoxic cell states. That is, in the zero shot the overall chaotic nature of the points overlapping each other clearly signifies that a pre-trained scGPT struggles with more complex tasks of cell-state classification.

Discussion

7.1 Evaluation

This research started with the aim of understanding how the usage of transformer architectures in the realm of transcriptomics could be of use and whether it made sense. Specifically, by analyzing performances of various models (transformers, MLPs, XGBoost) in the cell type and cell state classification tasks across different datasets the aim was that of providing a strong foundation to determine the relevance of such novel modelling methods in this field.

As shown above, it seems that scGPT can work as a foundational model to produce satisfactory embeddings that can be then used for other tasks - and in certain instances even be finetuned on downstream objectives; yet, its performance still lags behind that of other much simpler models that clearly are much lighter to use and train much faster (namely, XGboost and the MLP architecture used in this paper). Therefore, it poses the question of whether such a complex architecture might be necessary.

From the results obtained, it seems as though in the circumstance of cell state classification the performance of scGPT (both pretrained and finetuned) is nowhere near as good as other alternatives and therefore its choice might not be the best one - that is, it might be that scGPT can only work on macro tasks such as cell type classification and that the model is not yet apt in more nuanced understandings (it should not be excluded that a potential modification to the architecture might actually improve in this

realm).

Nonetheless, when looking at its performance on cell type classification, despite still being slightly worse than the other models used (the gap is much smaller now - often performances are even comparable) scGPT actually makes sense. That is, given its nature and the fact that the pre-trained human model was trained on such vast and varied amounts of data actually allows it work as a ready-to-use foundational model that performs quite well on majority of datasets - giving an easy to use tool that can effectively produce embeddings (as shown by the UMAPs) for the various cell and gene expressions. Arguably, the reason why the name ”GPT” was used was to emulate ”ChatGPT” in the sense that it is a general purpose cell embedding model that can work as a good foundation on various tasks (but again, not the very complex / nuanced ones such as cell-state classification).

Overall, it therefore seems that transformer architecture use in transcriptomic annotation tasks can be useful however as it now stands its usability is limited to broader tasks as the level of understanding needed for more nuanced tasks is something that would either require a more over-parametrized model or an alteration of the current scGPT architecture.

7.2 Extension

As described in the previous portion, there are some interesting directions which could be taken with regards to how to further extend this investigation. Specifically, the most evident one would be that of testing on more varied and distinct datasets for both tasks. That is, in this research cell-types were tested on 3 different datasets and cell-states just on one; this is of course limiting in the sense that in order to fully understand the ”generalizability” of the model one has to have tested it across its full extent on various tasks.

Moreover, as disucssed above, the model despite performing well in cell type classification tasks really struggles with more complex subtype tasks - beside investigating other cases where this might or might not be the case, it would also be interesting and useful to attempt modifications on the architecture to see whether that could bring any

benefits. For example, making use of the conditioning tokens that they have not yet put to the test in any scenarios or changing the embedding sizes or the number of encoding layers are all very interesting ideas that would for sure bring a meaningful impact to the model depending on the task. Of course one has to keep in mind that the objective of the model is still that of being a foundational one - so it is important that its ability to generalize is not lost through such changes.

Finally, given the encoder nature of the model it would be interesting to attempt some sort of quantization or knowledge distillation - that is, given the large parametrical size (and hardware requirements), see whether it would be possible to reduce the size of the model while maintaining performance. A lightweight generalizable model would for sure also be an interesting direction that scGPT could take (much like other encoders) that would benefit the scientific community.

Bibliography

[1] E. Hodzic, "Single-cell analysis: Advances and future perspectives," *Bosnian Journal of Basic Medical Sciences*, vol. 16, pp. 313–314, Nov. 2016, Accessed: 2024-06. DOI: 10.17305/bjbms.2016.1371. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5136769/.

[2] L. Heumos, A. C. Schaar, C. Lance, *et al.*, "Best practices for single-cell analysis across modalities," *Nature Reviews Genetics*, vol. 24, pp. 550–572, Mar. 2023, Accessed: 2024-06. DOI: 10.1038/s41576-023-00586-w. (visited on 06/08/2024).

[3] M. D. Luecken and F. J. Theis, "Current best practices in single□cell rna□seq analysis: A tutorial," *Molecular Systems Biology*, vol. 15, Jun. 2019, Accessed: 2024-06. DOI: 10.15252/msb.20188746. (visited on 06/08/2024).

[4] D. Jovic, X. Liang, H. Zeng, L. Lin, F. Xu, and Y. Luo, "Single□cell rna sequencing technologies and applications: A brief overview," *Clinical and Translational Medicine*, vol. 12, Mar. 2022, Accessed: 2024-06. DOI: 10.1002/ctm2.694. (visited on 06/08/2024).

[5] C. Ziegenhain and et al., "Comparative analysis of single-cell rna sequencing methods," *Molecular Cell*, vol. 65, no. 4, 631–643.e4, Feb. 2017, Accessed: 2024-06. DOI: 10.1016/j.molcel.2017.01.023. [Online]. Available: https://doi.org/10.1016/j.molcel.2017.01.023.

[6] M. P. Kirchner, L. Pantano, M. Mistry, R. Khetani, and Rory. "Generation of count matrix." Accessed: 2024-06. (Feb. 2020), [Online]. Available: https://hbctraining.github.io/scRNA-seq/lessons/02_SC_generation_of_count_matrix.html.

[7] X. Shao, J. Liao, X. Lu, R. Xue, N. Ai, and X. Fan, “ScCATCH: Automatic Annotation on Cell Types of Clusters from Single-Cell RNA Sequencing Data,” *IScience*, vol. 23, no. 3, p. 100 882, Mar. 2020, Accessed: 2024-06. DOI: 10.1016/j.isci.2020.100882. [Online]. Available: https://doi.org/10.1016/j.isci.2020.100882.

[8] X. Zhang, Y. Lan, J. Xu, *et al.*, “CellMarker: A Manually Curated Resource of Cell Markers in Human and Mouse,” *Nucleic Acids Research*, vol. 47, no. D1, pp. D721–D728, Oct. 2018, Accessed: 2024-06. DOI: 10.1093/nar/gky900. [Online]. Available: https://doi.org/10.1093/nar/gky900.

[9] Y. Cao, J. Chen, Q. Wei, *et al.*, “SCSA: A Cell Type Annotation Tool for Single-Cell RNA-Seq Data,” *Frontiers in Genetics*, vol. 11, May 2020, Accessed: 2024-06. DOI: 10.3389/fgene.2020.00490. [Online]. Available: https://doi.org/10.3389/fgene.2020.00490.

[10] J. Zhang, Q. Liu, H. Hu, *et al.*, “SCINA: Semi-Supervised Analysis of Single Cells in Silico,” *Genes*, vol. 10, no. 7, p. 531, Jul. 2019, Accessed: 2024-06. DOI: 10.3390/genes10070531. [Online]. Available: https://doi.org/10.3390/genes10070531.

[11] A. W. Zhang, C. O’Flanagan, E. A. Chavez, *et al.*, “Probabilistic Cell-Type Assignment of Single-Cell RNA-Seq for Tumor Microenvironment Profiling,” *Nature Methods*, vol. 16, no. 10, pp. 1007–1015, Sep. 2019, Accessed: 2024-06. DOI: 10.1038/s41592-019-0529-1. [Online]. Available: https://doi.org/10.1038/s41592-019-0529-1.

[12] V. Y. Kiselev, K. Kirschner, M. T. Schaub, *et al.*, “Scmap: Projection of Single-Cell RNA-Seq Data across Data Sets,” *Nature Methods*, vol. 15, no. 5, pp. 359–362, Apr. 2018, Accessed: 2024-06. DOI: 10.1038/nmeth.4644. [Online]. Available: https://doi.org/10.1038/nmeth.4644.

[13] Y. Tan and P. Cahan, “SingleCellNet: A Computational Tool to Classify Single Cell RNA-Seq Data across Platforms and across Species,” *Cell Systems*, vol. 9, no. 2, 207–213.e2, Aug. 2019, Accessed: 2024-06. DOI: 10.1016/j.cels.2019.06.004. [Online]. Available: https://doi.org/10.1016/j.cels.2019.06.004.

[14] J. K. de Kanter, P. Lijnzaad, T. Candelli, T. Margaritis, and F. C. Holstege, “CHETAH: A Selective, Hierarchical Cell Type Identification Method for Single-Cell RNA Sequencing,” *Nucleic Acids Research*, Jun. 2019, Accessed: 2024-06. DOI: 10.1093/nar/gkz543. [Online]. Available: https://doi.org/10.1093/nar/gkz543.

[15] F. Wagner and I. Yanai, “Moana: A Robust and Scalable Cell Type Classification Framework for Single-Cell RNA-Seq Data,” Oct. 2018, Accessed: 2024-06. DOI: 10.1101/456129. [Online]. Available: https://doi.org/10.1101/456129.

[16] T. Johnson, T. Wang, Z. Huang, *et al.*, “LAmbDA: label ambiguous domain adaptation dataset integration reduces batch effects and improves subtype detection,” *Bioinformatics*, vol. 35, no. 22, pp. 4696–4706, 2019, Accessed: 2024-06. DOI: 10.1093/bioinformatics/btz295. [Online]. Available: https://doi.org/10.1093/bioinformatics/btz295.

[17] P. Xie, M. Gao, C. Wang, *et al.*, “SuperCT: a supervised-learning framework for enhanced characterization of single-cell transcriptomic profiles,” *Nucleic Acids Research*, vol. 47, no. 8, e48–e48, 2019, Accessed: 2024-06. DOI: 10.1093/nar/gkz116. [Online]. Available: https://doi.org/10.1093/nar/gkz116.

[18] F. Ma and M. Pellegrini, “Actinn: Automated identification of cell types in single cell rna sequencing,” *Bioinformatics*, 2019, Accessed: 2024-06. DOI: 10.1093/bioinformatics/btz592. [Online]. Available: https://doi.org/10.1093/bioinformatics/btz592.

[19] J. C. Kimmel and D. R. Kelley, “scNym: Semi-supervised Adversarial Neural Networks for Single Cell Classification,” 2020, Accessed: 2024-06. DOI: 10.1101/2020.06.04.132324. [Online]. Available: https://doi.org/10.1101/2020.06.04.132324.

[20] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, “Attention is all you need,” in *Advances in Neural Information Processing Systems*, Accessed: 2024-06, vol. 30, 2017.

[21] S. Raschka, *Understanding Encoder and Decoder LLMs*, https://magazine.sebastianraschka.com/p/understanding-encoder-and-decoder, Accessed: 2024-06, 2023.

[22] *The annotated transformer*, http://nlp.seas.harvard.edu/2018/04/03/attention.html, Accessed: 2024-06, Apr. 2018.

[23] H. Cui, C. Wang, H. Maan, *et al.*, “Scgpt: Toward building a foundation model for single-cell multi-omics using generative ai,” *Nature Methods*, Feb. 2024, Accessed: 2024-06. DOI: 10.1038/s41592-024-02201-0. (visited on 06/08/2024).

[24] K. Kamali, *Galaxy training: Deep learning (part 1) - feedforward neural networks (fnn)*, https://training.galaxyproject.org/training-material/topics/statistics/tutorials/FNN/tutorial.html, Accessed: 2024-06, 2024.

[25] W. Wang, G. Chakraborty, and B. Chakraborty, “Predicting the risk of chronic kidney disease (ckd) using machine learning algorithm,” *Applied Sciences*, vol. 11, p. 202, Dec. 2020, Accessed: 2024-06. DOI: 10.3390/app11010202.

[26] H. M. Kang, M. Subramaniam, S. Targ, *et al.*, “Multiplexed droplet single-cell RNA-sequencing using natural genetic variation,” *Nat Biotechnol.*, vol. 36, no. 1, p. 89, 2018. DOI: 10.1038/nbt.4042.

[27] T. Stuart, A. Butler, P. Hoffman, *et al.*, “Comprehensive integration of single-cell data,” *Cell*, 2019. DOI: 10.1016/j.cell.2019.05.031.

[28] A. M. Ranzoni, A. Tangherloni, I. Berest, *et al.*, “Integrative Single-cell RNA-Seq and ATAC-Seq Analysis of Human Developmental Hematopoiesis,” *Cell Stem Cell*, 2020.

Appendix A - Materials and Methods

The code used in this thesis can be found at the following github link: `https://github.com/andreafabbricatore/transformers_annotation`

Moreover, below one can find the references for the various datasets used.

A.1 PBMC Data

PBMCs from eight patients with systemic lupus erythematosus were collected and processed using the $10\times$ Chromium Genomics platform [26]. The dataset is composed of a control group (6573 cells) and an interferon-β stimulated group (7466 cells). We considered the 8 distinct cell-types identified by the authors following a standard workflow [26]. The count matrices were downloaded from Seurat's tutorial *"Integrating stimulated vs. control PBMC datasets to learn cell-type specific responses"* (`https://satijalab.org/seurat/v3.0/immune_alignment.html`).

A.2 PIC Data

PIC datasets were generated independently using four different platforms: CEL-Seq (1004 cells), CEL-Seq2 (2285 cells), Fluidigm C1 (638 cells), and Smart-Seq2 (2394 cells). For our tests, we considered the 13 different cell-types across the datasets identified in [27] by applying PCA on the scaled integrated data matrix. The count matrices were downloaded from Seurat's tutorial *"Integration and Label Transfer"* (`https:`

//satijalab.org/seurat/v3.0/integration.html).

A.3 Foetal Data

The HFC dataset contains 15 independent scRNA-Seq experiments of human fetal liver and bone marrow [28] samples, sequenced using the SmartSeq2 protocol. We grouped some cell subtypes to analyze the major cell types, namely: B cells, Endothelial cells, Erythrocytes, Granulocytes, Hematopoietic Stem Cells / Multipotent Progenitors (HSC/MPPs), Lympho-Myeloid Progenitors (LMPs), Megakaryocyte-Erythroid-Mast Progenitors (MEMPs), Mast cells, Megakaryocytes, Monocytes, Natural Killer (NK) cells, Unspecified, and Plasmacytoid Dendritic Cells (pDCs).

A.4 TNBC Data

Dataset still not public, obtained with permission from the author.