DEEP LEARNING

An Explainable Deep Learning Model to Identify Critical Gene Sets for discriminating different statuses of patients with Chronic Lymphocytic Leukemia

Costanzo Gabriel

Intro

Chronic Lymphocytic Leukemia (CLL)

- **CLL is the most common leukaemia of adults in Western countries and is an incurable progressive malignancy characterized by the clonal e**

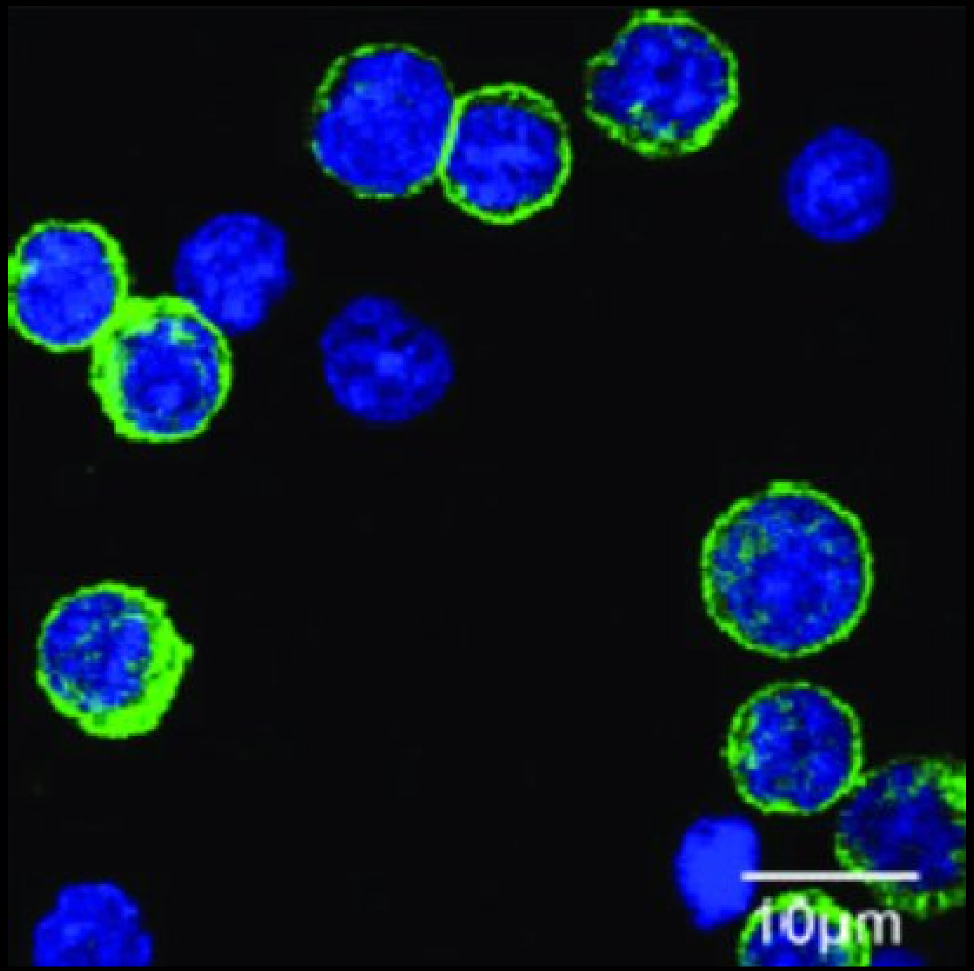
- **CD49d expression in CLL has been found to have an aggressive clinical course, shorter time to first treatment, and poorer prognosis.**

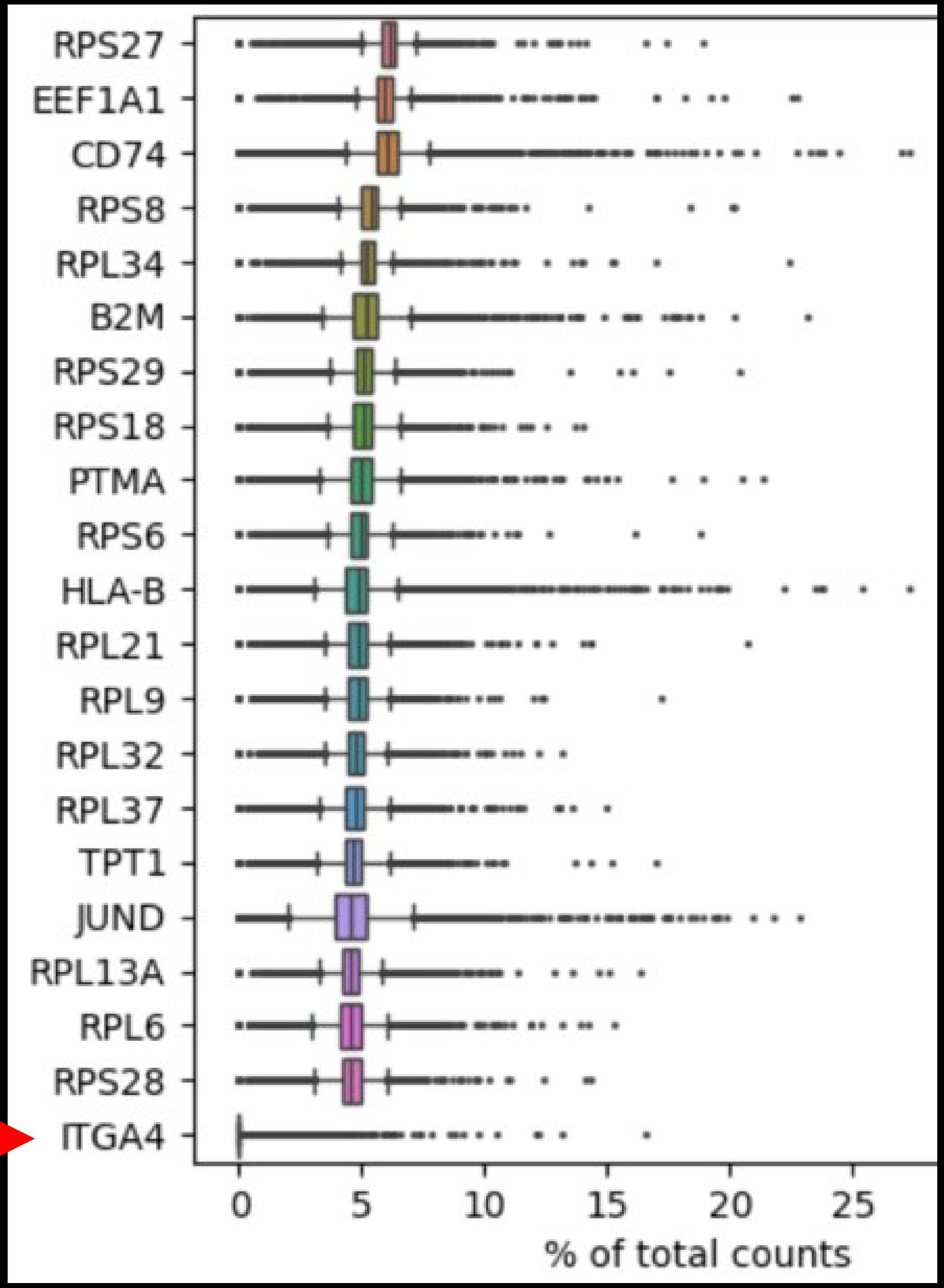
https://www.biorxiv.org/content/10.1101/2023.12.27.573410v1.full.pdf

ttps://www.ncbi.nlm.nih.gov/pmc/articles/PMC8276061/

https://www.thieme-connect.com/products/ejournals/pdf/10.1055/s-0042-1748828.p

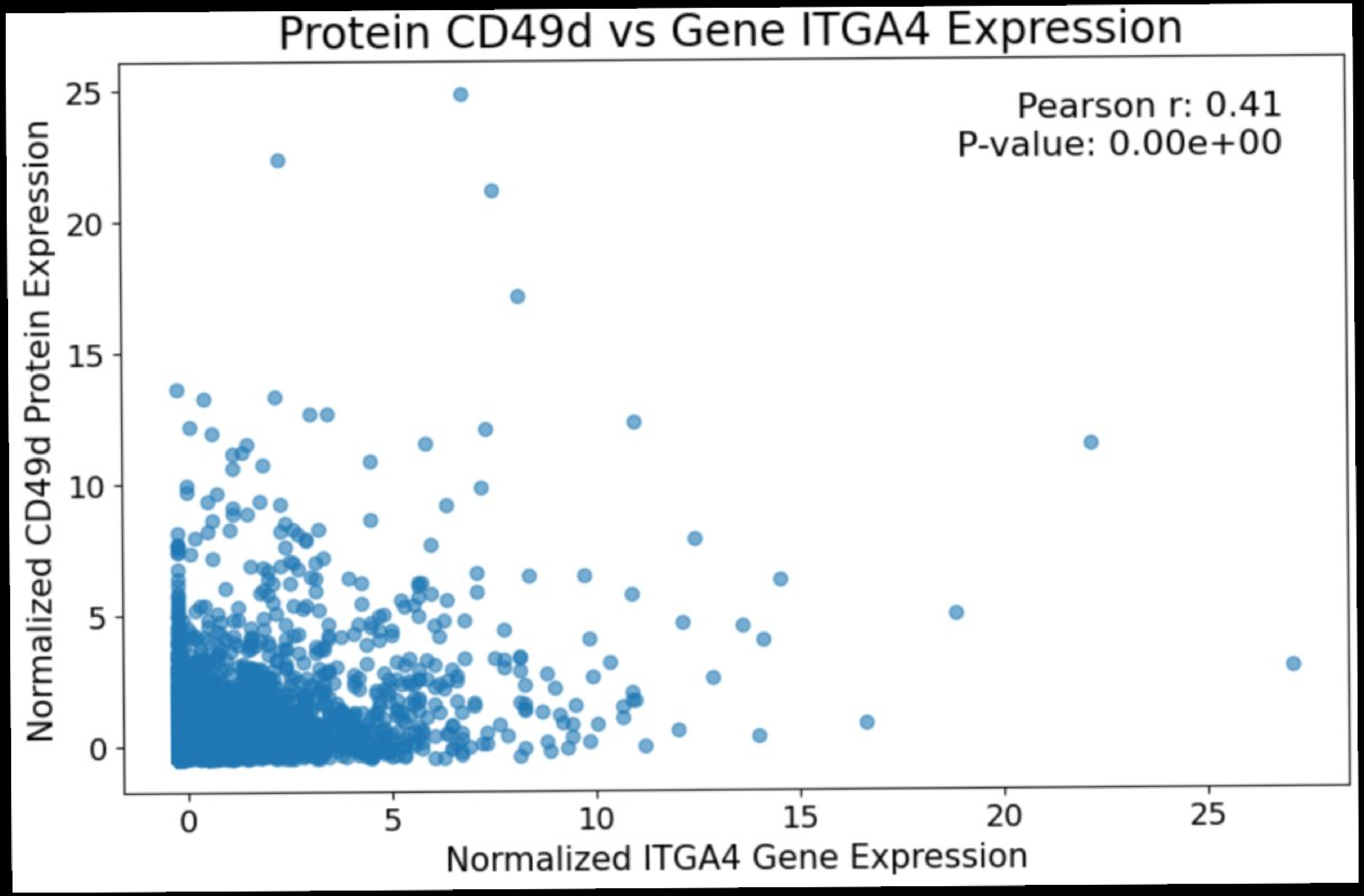
Problem





Easy to detect the CD49d protein but not its gene

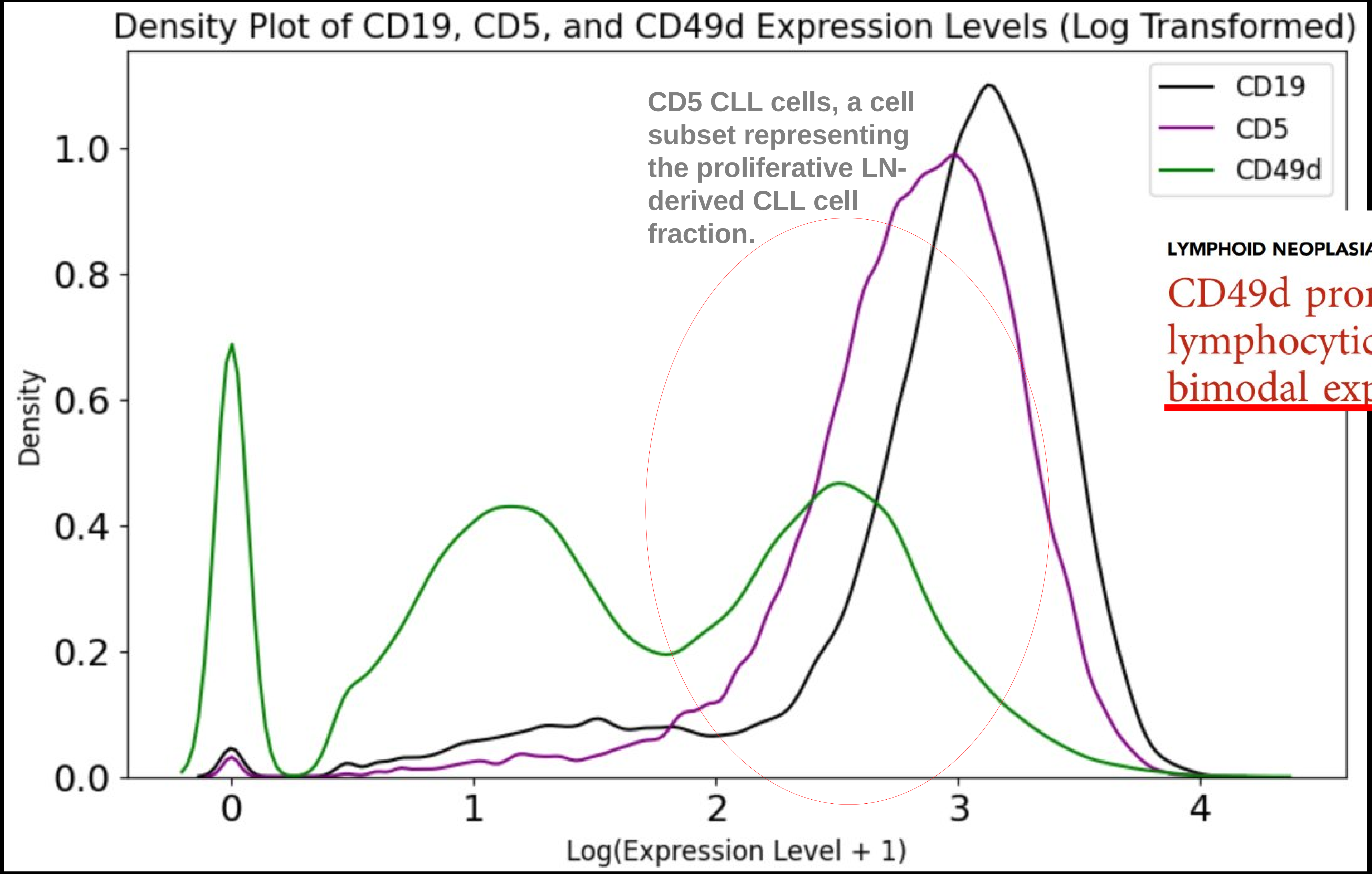
Problem



The absence of a clear correlation can indicate several biological phenomena, such as post-transcriptional regulation, protein degradation, or differences in translation efficiency that cause the protein levels to not directly reflect mRNA levels.

Data

Expression of the markers CD49d, CD19 and CD5



LYMPHOID NEOPLASIA

CD49d promotes disease progression in chronic lymphocytic leukemia: new insights from CD49d bimodal expression

https://www.frontiersin.org/journals/oncology/articles/10.3389/fonc.2020.592205/full

https://www.iris.unict.it/retrieve/fdfeb74d-01ce-4ad4-b0a4-49086c5aef9d/CD49d.pdf

https://www.biorxiv.org/content/10.1101/2023.12.27.573410v1

https://www.ncbi.nlm.nih.gov/pmc/articles/P

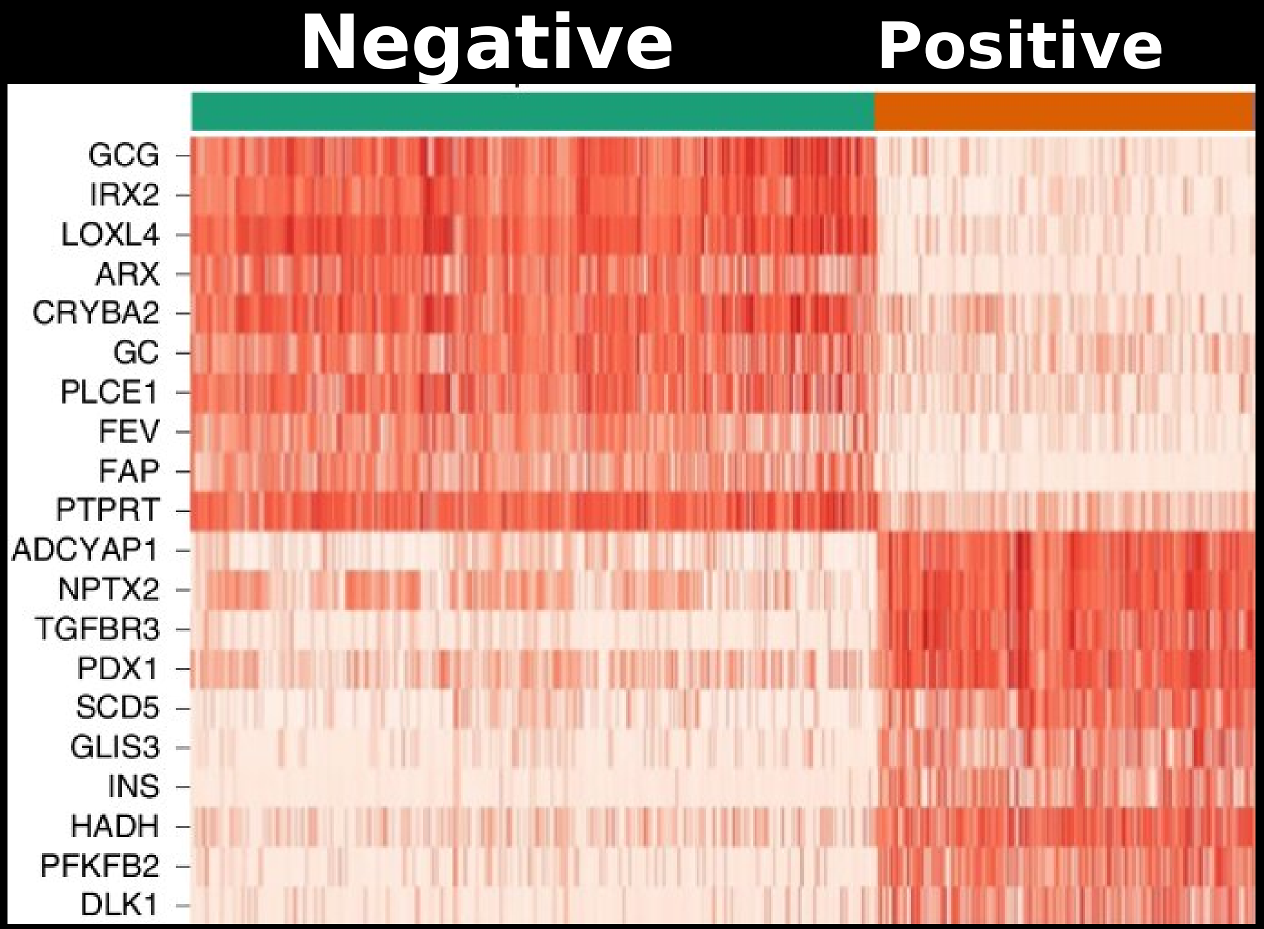
Project Goal and data

- ✓ **Use a deep learning model to find a set of genes (signature) whose expression level is related to the CD49d expression level and disease prognosis.**

Data
scRNA from 10 patients with CLL (17,218 cells)

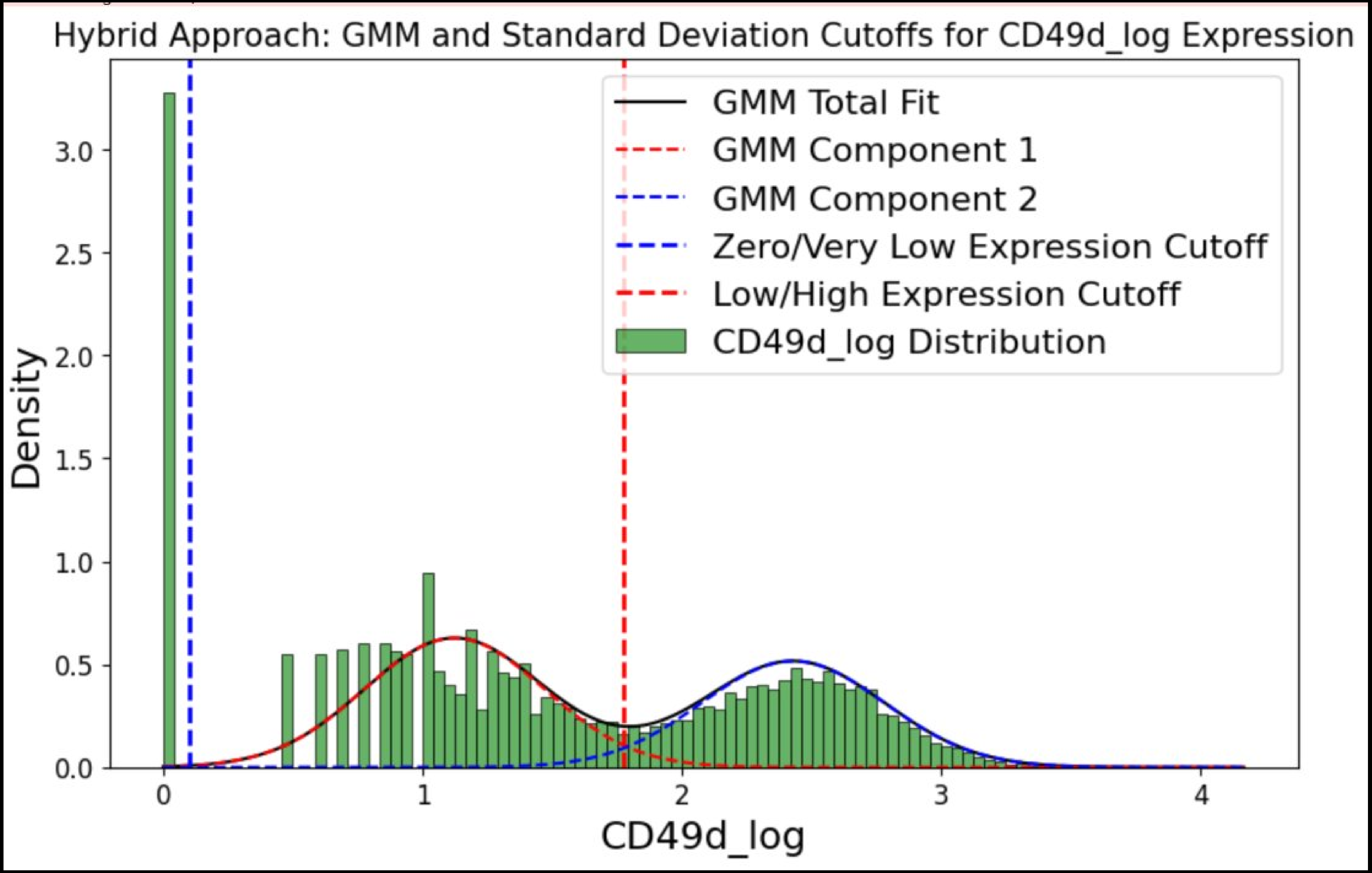
$\longrightarrow$ ScBERT Model $\longrightarrow$

CD49d B cells



Data processing

Three statuses for CD49d marker in B cells



Counts of cells by CD49d_labe

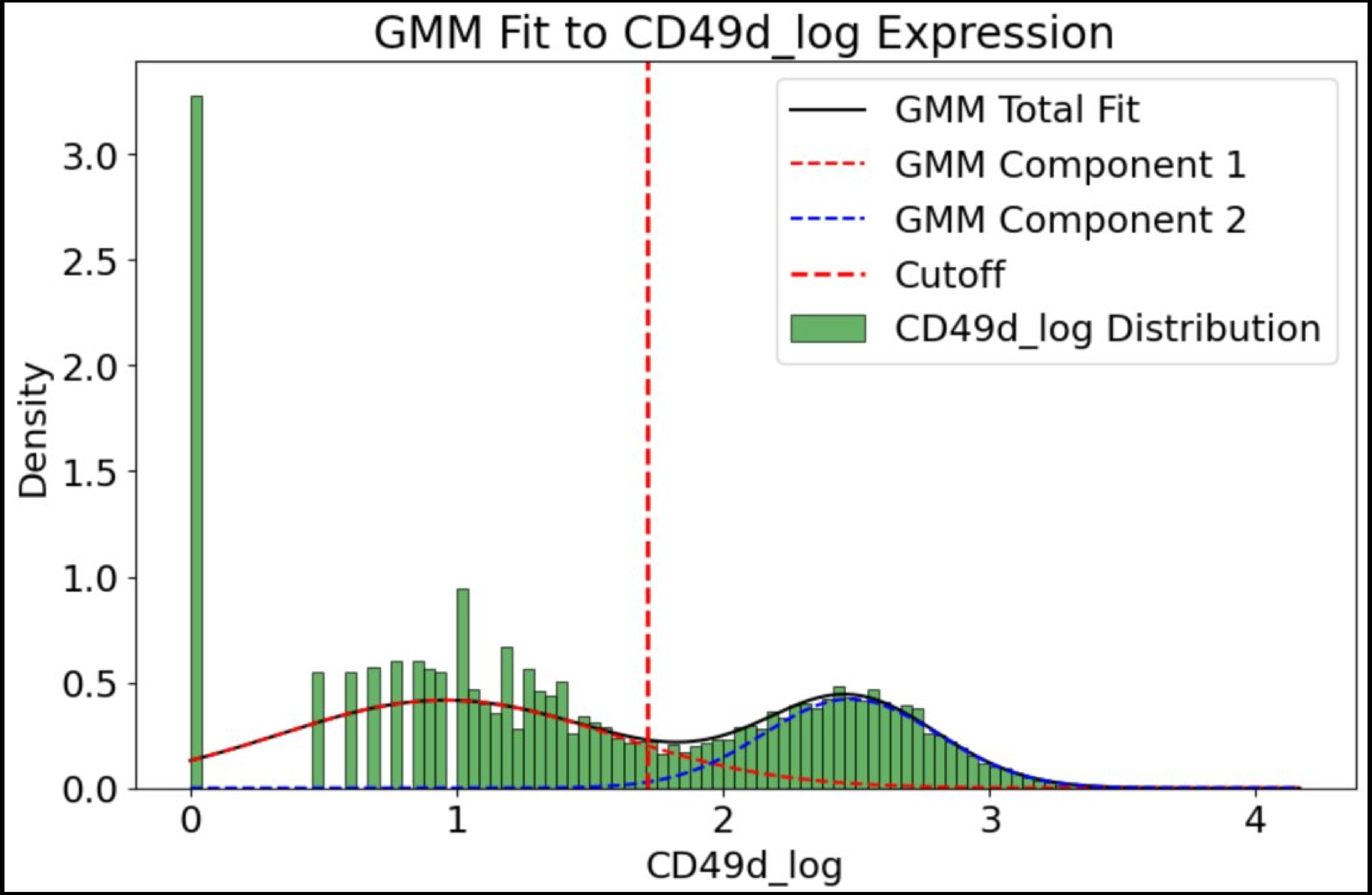
High	8038
Low	7160
zero	2020

Resampling to avoid sample imbalance which affect the model performance (classification)

High	8038
Low	8038
zero	8038

Data processing

Select CD49d positive and negative cells.



Counts of cells by CD49d_labe

Positive	7837
Nevative	5929

Resampling to avoid sample imbalance which affect the model performance (classification)

Positive	7837
Nevative	7837

Data processing

Synthetic Minority Over-sampling Technique (SMOTE)

Select a Minority Class Sample X_i

Identify Nearest Neighbors: Find the k-nearest neighbors of X_i within the same minority class.

Interpolation Process: Generate new synthetic samples by creating points along the line segment between X_i and Xneighbor. This is done by linear interpolation.

Linear Interpolation: The new synthetic data point Xnew is created by adding a random proportion of the difference between X_i and Xneighbor. Mathematically, this can be expressed as:

$$x_{new} = x_i + \alpha \times (x_{neighbor} - x_i)$$

where α is a random number between 0 and 1.

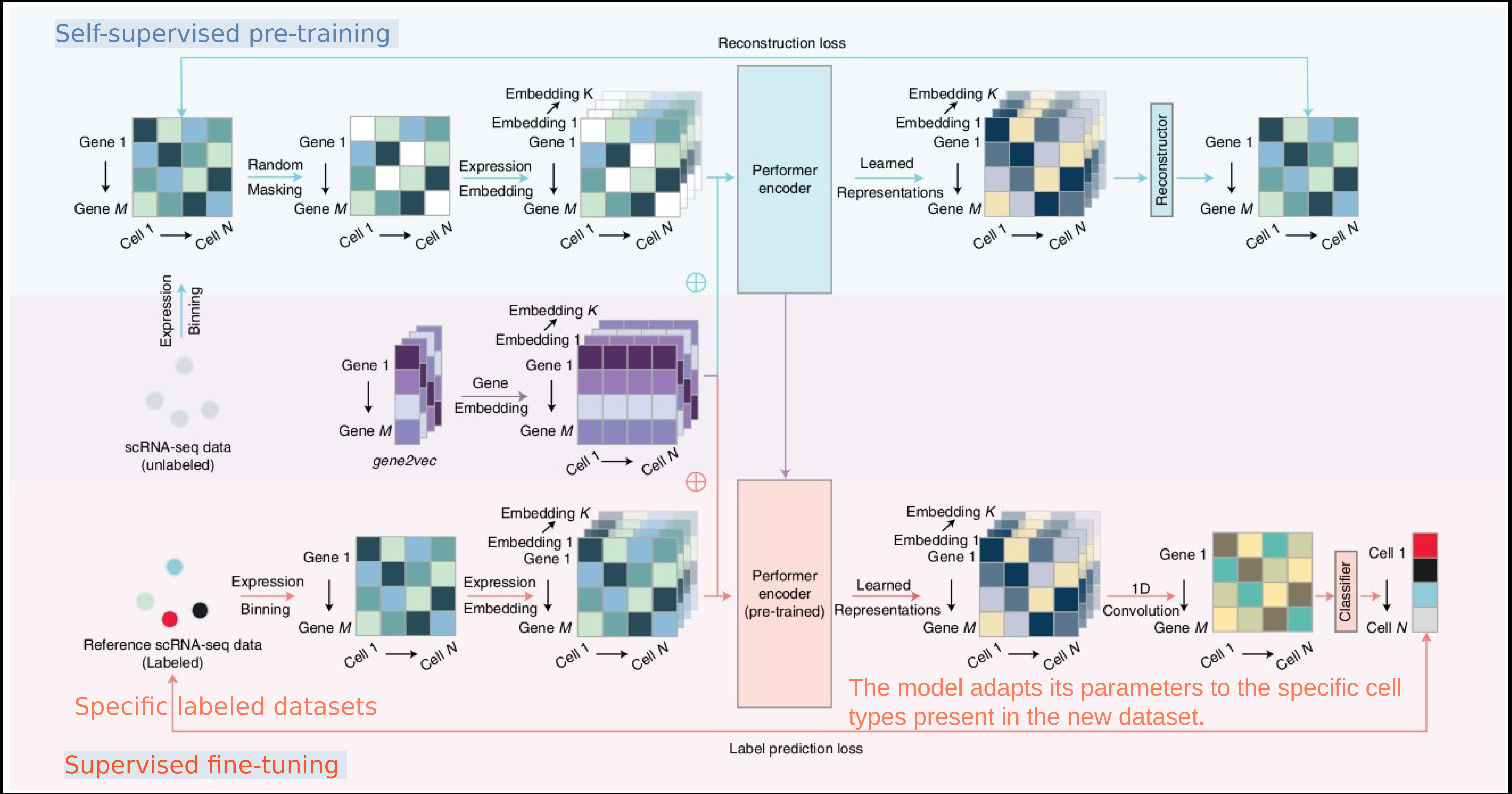
The term "line segment" refers to the straight line connectingX_i and Xneighbor in the multidimensional feature space. By choosing α randomly between 0 and 1, the new sample can be anywhere along this line segment.

The model

Single-cell Bidirectional Encoder Representations from Transformers (scBERT)

Run the code in the ORFEO cluster

scBERT model



Self-supervised pre-training

Specific labeled datasets

The model adapts its parameters to the specific cell types present in the new dataset.

Supervised fine-tuning

scBERT model

INPUT DATA

Bin Gene Expression Data

Determine Range: Identify the range of expression values in your data (e.g., minimum and maximum read counts).

Define Bins: Divide the range into a set number of bins. For example, if you have expression values ranging from 0 to 1000, you might divide this range into 10 bins of width 100 each.

Assign Values to Bins: Map each gene's expression value to the corresponding bin. For example:

Expression value 50 might fall into bin 1 (0-99).

Expression value 150 might fall into bin 2 (100-199).

Expression value 350 might fall into bin 4 (300-399).

Encoding Binned Values: Assign a unique identifier or embedding to each bin, which can be used as input for further processing in the model.

scBERT model

Self-Supervised Pretraining

Objective: To learn general patterns and relationships in the gene expression data without using labeled data.

Input: The model is fed with large-scale, unlabelled scRNA-seq datasets collected from integrated 209 human single-cell datasets comprising 74 tissues with 1,126,580 cells obtained from different experimental sources via various platforms.

Process:

Random Masking: Some parts of the input data (gene expression levels) are randomly masked.

Reconstruction Task: The model tries to predict the masked parts based on the unmasked data.

Loss Function: The reconstruction loss is calculated using a cross-entropy loss function to measure how well the model predicts the masked values.

Output:

Learned Representations: The model learns robust and general embeddings that capture the underlying structure of the gene expression data. These embeddings encode information about gene-gene interactions and are useful for various downstream tasks.

scBERT model

Reconstruction Loss

The non-zero gene expression is randomly masked and then reconstructed the original inputs by model predictions using the remaining genes. The model tries to minimize the difference between the original input and the reconstructed output, refining the embeddings to capture essential features

$$L_{\text{Rec}} = -\sum_{i=1}^{M} \sum_{j=1}^{N} y_{i,j} \log(p_{i,j})$$

where M is the number of samples and N is the number of classes or outputs. $y_{i,j}$ are the true values and $p_{i,j}$ are the predicted probabilities. The loss is computed as the negative log-likelihood, summed over all samples i and classes j.

scBERT model

gene2vec

Layers and Components:

Input Layer:

Context Genes: One-hot encoded vectors representing the context (neighboring) genes.

Example: If the vocabulary size is V (total number of unique genes), each context gene is a V-dimensional vector with a single 1.

Hidden Layer:

Projection Layer: A weight matrix W of size V×N where N is the embedding dimension.

Averaging: The input vectors are averaged to form a single context vector.

where C is the number of context genes.

$$\text{Hidden\_Output} = \frac{1}{C}\sum_{i=1}^{C} W \cdot \text{Context\_Gene}_i$$

scBERT model

gene2vec

Output Layer:
Weight Matrix: Another weight matrix W′ of size N×V.
Softmax Activation: Produces a probability distribution over the vocabulary.

$$\text{Output} = \text{Softmax}(W' \cdot \text{Hidden\_Output})$$

Loss Function:

Negative Log-Likelihood: The objective is to maximize the probability of predicting the target gene given the context genes.

$$L = -\log P(\text{Target\_Gene}|\text{Context\_Genes})$$

Training Process:
Minimization: The model minimizes the negative log-likelihood loss by adjusting weights W and W′ using gradient descent.

scBERT model

Supervised Fine-Tuning

Objective: To adapt the pre-trained model to a specific task, such as cell type annotation, using labeled data.

Process:

Input: The pre-trained embeddings are fine-tuned using labeled scRNA-seq data.

1D Convolution: A 1D convolutional layer extracts features from the gene embeddings.

Classifier: A classification head (multi-layer neural network) is added to predict cell types.

Loss Function: The label prediction loss is calculated using a cross-entropy loss function to measure how well the model predicts the correct cell types.

Output:
Cell Type Predictions: The fine-tuned model outputs the probabilities of each cell belonging to predefined cell types.

scBERT model

1D Convolution

Input:

Combined embeddings from the pre-trained model (gene embeddings + expression embeddings).

Input shape: 3D tensor [batch_size, sequence_length, embedding_dim].

Purpose:

Extracts local features from the input data by applying filters (kernels).

Useful for capturing spatial or temporal patterns.

Operation:

Applies a sliding window (kernel) across the input to produce a feature map.

Each kernel learns to detect a specific feature.

Structure:

Contains filters (kernels) with learnable weights.

Each filter produces one output channel (feature map).

Output: 3D tensor with the number of channels determined by the number of filters.

scBERT model

1D Convolution

Fully Connected (Dense) Layer:

Input: Flattened 1D tensor (from previous layers).

Purpose:

Integrates features learned from previous layers to perform classification.

Each neuron is connected to every neuron in the previous layer.

Operation:

Multiplies the input by a weight matrix and adds a bias vector.

Often used at the end of the network for making predictions.

Structure:

Dense connections with learnable weights and biases.

Output: 1D tensor representing the final predictions.

scBERT model

PERFORMER

scBERT model

“The heart of transformers”: Self-attention mechanism

For each input element, three vectors are computed: the key (K), query (Q), and value (V) vectors.

These are linear transformations of the input embedding:

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V$$

where X is the matrix of input embeddings, and W_Q , W_K, and W_V are the learned weight matrices for the queries, keys, and values, respectively.

scBERT model

Self-attention mechanism

Query (Q) Vectors: determines how much attention should be given to the other elements (genes).

Key (K) Vectors: represents each element in the sequence that can be attended to.

Value (V) Vectors: contains the actual information of the elements. It is what gets aggregated to form the final output for the current element.

The attention mechanism uses the attention scores (derived from queries and keys) to weigh the value vectors and compute the final representation for each element.

scBERT model

Self-attention process

1- Compute Attention Scores: $\text{score}(Q_i, K_j) = Q_i \cdot K_j$

2- Scale the Scores: $\text{attention\_score}(Q_i, K_j) = \frac{Q_i \cdot K_j}{\sqrt{d_k}}$ stabilize the gradients during training

3- Softmax Normalization: $\alpha_{ij} = \text{softmax}\left(\frac{Q_i \cdot K_j}{\sqrt{d_k}}\right)$

4- Weighted Sum of Values: $\text{output}_i = \sum_j \alpha_{ij} V_j$

scBERT model

Multi-Head Attention

Multiple Heads: Instead of performing a single self-attention operation, multi-head attention involves several attention mechanisms running in parallel (one for each sample or cell). Each head has its own set of Q, K, and V weight matrices.

Concatenation and Linear Transformation:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \text{head}_2, \ldots, \text{head}_h)W_O$$

scBERT model

Performer instead of Transformers

Scalability: Transformer= $O(n)^2$ challenging to scale for very long sequences.

Performer is an efficient version of Transformer that uses a linear approximation for the self-attention mechanism, significantly reducing the computational complexity. This makes it possible to handle much longer sequences.

scBERT model

Via positive Orthogonal Random features approach

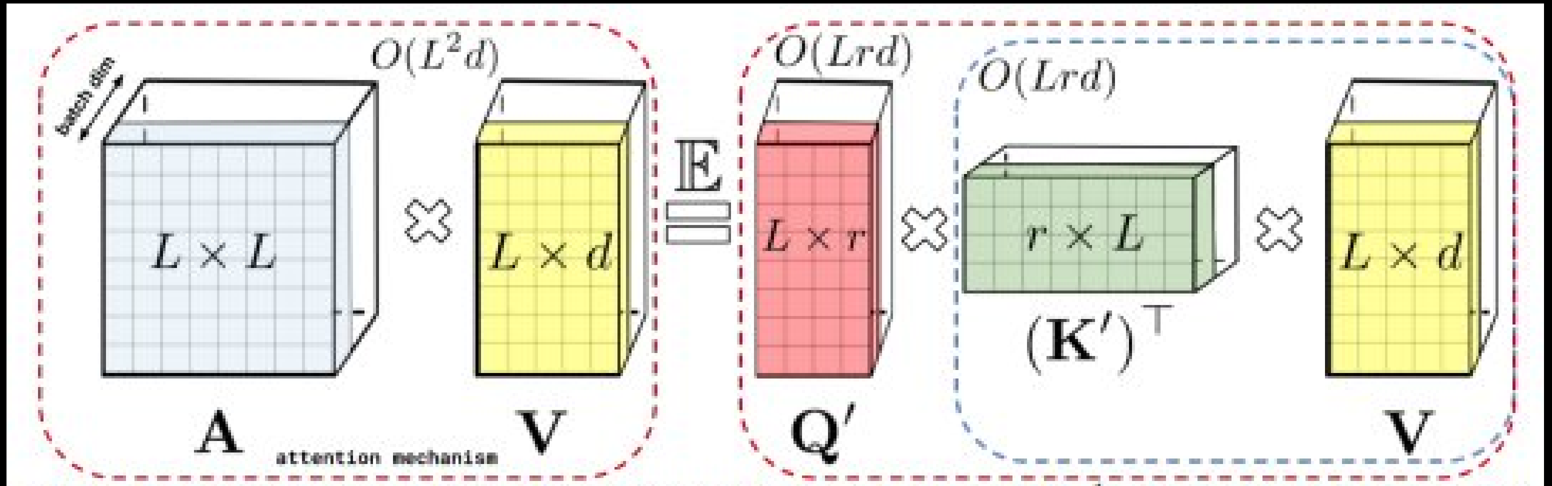


Figure 1: Approximation of the regular attention mechanism **AV** (before $\mathbf{D}^{-1}$-renormalization) via (random) feature maps. Dashed-blocks indicate order of computation with corresponding time complexities attached.

Regular Attention Mechanism:

A (Attention Scores): This is a L×L matrix where L is the sequence length. Calculating this matrix involves the dot product of the query and key matrices, leading to a time complexity of $O(L^2d)$.

V (Value Matrix): This matrix has dimensions L×d, where d is the dimensionality of the embeddings. The multiplication of the attention scores with the value matrix is also $O(L^2d)$.

scBERT model

Via positive Orthogonal Random features approach

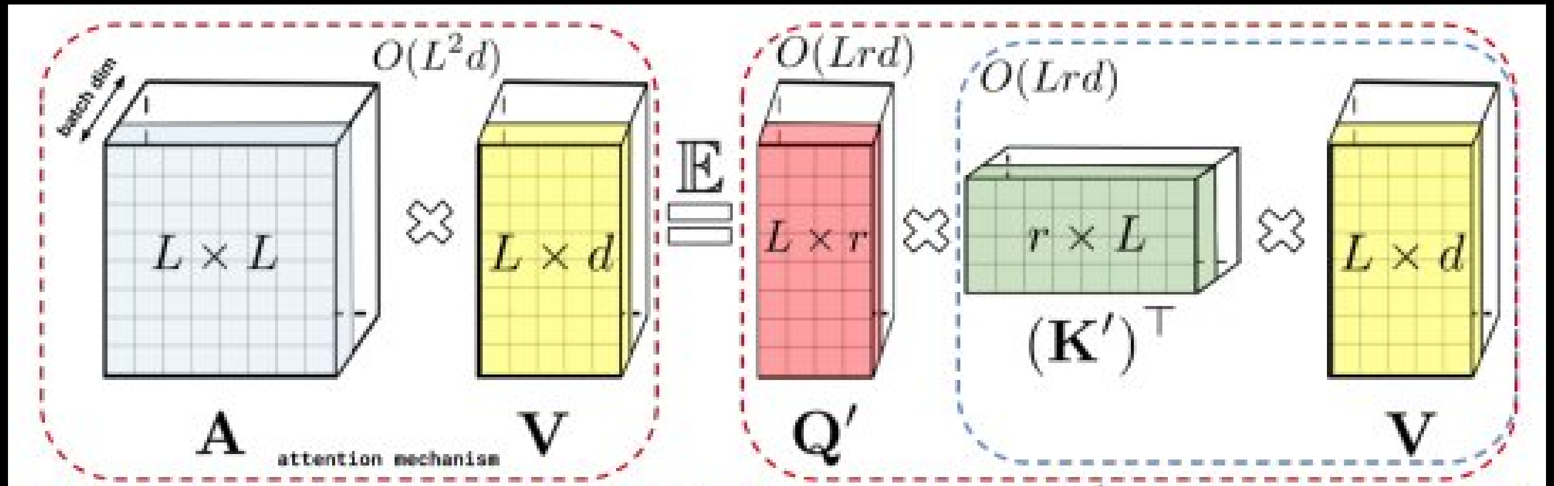


Figure 1: Approximation of the regular attention mechanism $\mathbf{AV}$ (before $\mathbf{D}^{-1}$-renormalization) via (random) feature maps. Dashed-blocks indicate order of computation with corresponding time complexities attached.

Approximation with Random Feature Maps:

Q' (Transformed Queries) and K' (Transformed Keys):

These transformations reduce the computational complexity by mapping the queries and keys into a lower-dimensional space r, where r is typically much smaller than L.

The time complexity for computing Q' and K' is O(Lrd), where r is the reduced dimensionality.

scBERT model

Via positive Orthogonal Random features approach

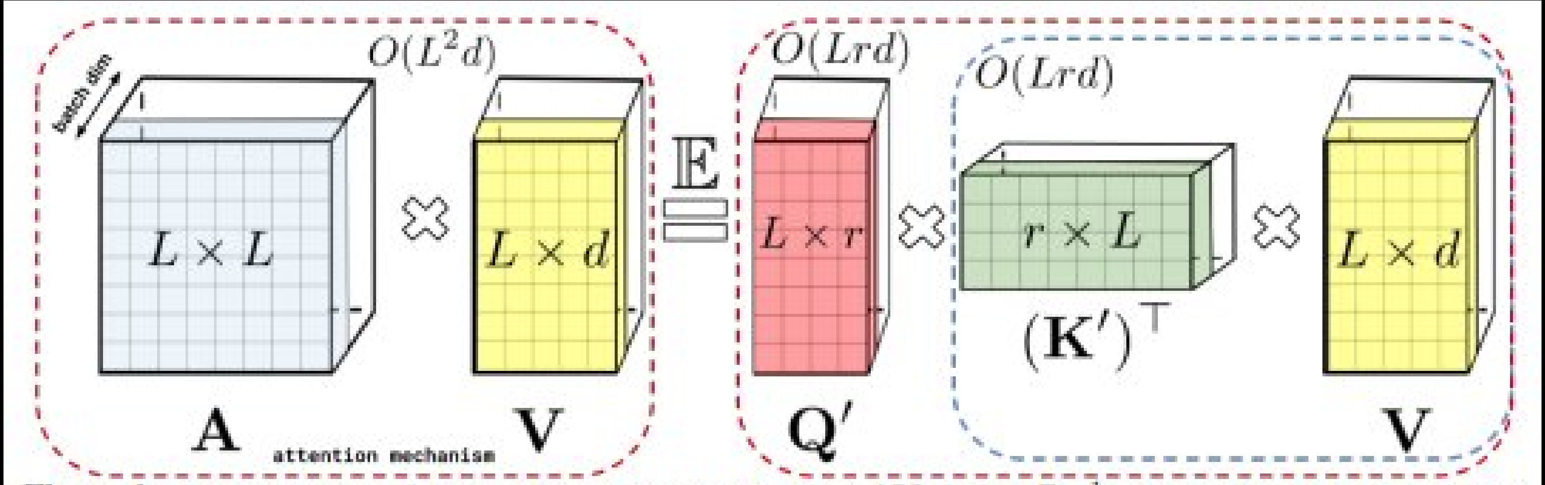


Figure 1: Approximation of the regular attention mechanism **AV** (before $\mathbf{D}^{-1}$-renormalization) via (random) feature maps. Dashed-blocks indicate order of computation with corresponding time complexities attached.

Step 1: Compute Q′ and K′ with complexity O(Lrd).

Step 2: Compute the product of Q′ and $K'^{\top}$, resulting in a matrix of size L×L but done efficiently with complexity O(Lrd).

Step 3: Multiply this intermediate result with V, maintaining the complexity O(Lrd).

scBERT model

PerformerLM model

Embedding Layers: Transform input tokens and positional information into dense vectors.

Dropout Layer: Helps in regularization randomly setting a fraction of input units to 0.

Performer Block: Core layers that perform self-attention and feedforward transformations, structured to handle long input sequences efficiently.

Projection Updater: additional mechanism for refining or updating internal representations.

Normalization and Output Layers: Final layers that normalize the output and produce the final predictions.

Each block contributes to the model's ability to process input sequences, capture long-range dependencies, and produce meaningful predictions for tasks like cell type annotation in scRNA-seq data.

scBERT model

The GELU (Gaussian Error Linear Unit)

$$\mathrm{GELU}(x) = x \cdot \Phi(x)$$

Where Φ(x) is the cumulative distribution function of the standard normal distribution.
An approximation of GELU is:

$$\mathrm{GELU}(x) \approx x \cdot \frac{1}{2}\left(1 + \tanh\left(\sqrt{\frac{2}{\pi}}\left(x + 0.044715x^3\right)\right)\right)$$

scBERT model

The GELU (Gaussian Error Linear Unit)

Key Points about GELU:

Smooth Activation: It combines aspects of the ReLU (Rectified Linear Unit) and the sigmoid function, providing a smooth transition for values around zero.

Probabilistic Interpretation: The use of the normal CDF gives GELU a probabilistic interpretation. It can be thought of as scaling the input by the probability that the input is positive given a standard normal distribution.

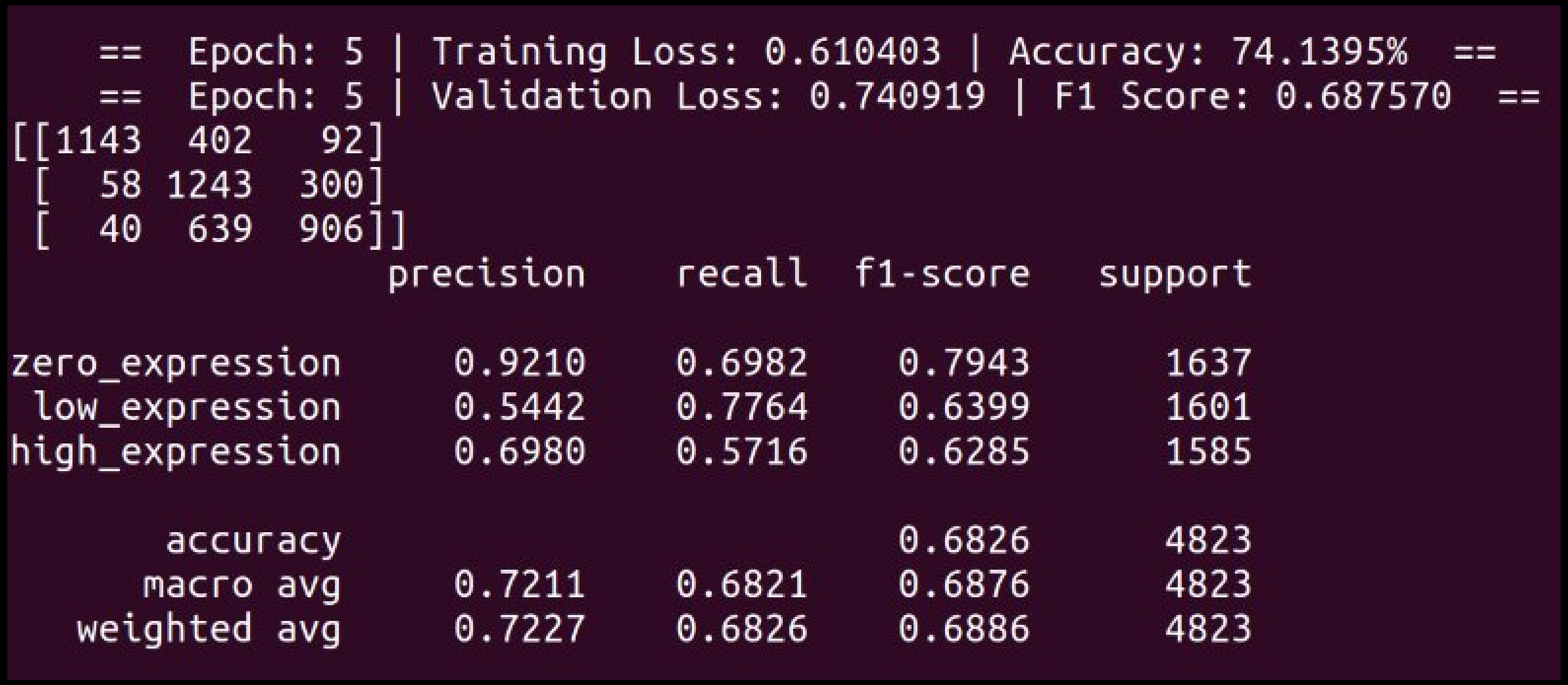
Approximation: While the exact form of GELU uses the CDF of the normal distribution, the tanh approximation is often used for computational efficiency.

Usage in Transformers: GELU has gained popularity in recent years, especially in transformer models like BERT. It has been found to perform better than ReLU and other traditional activation functions in these contexts.

Benefits: GELU activation tends to retain more information about the input's magnitude compared to ReLU, which simply zeroes out negative inputs. This can lead to improved learning dynamics in neural networks.

Model Evaluation

Three statuses for CD49d in B cells

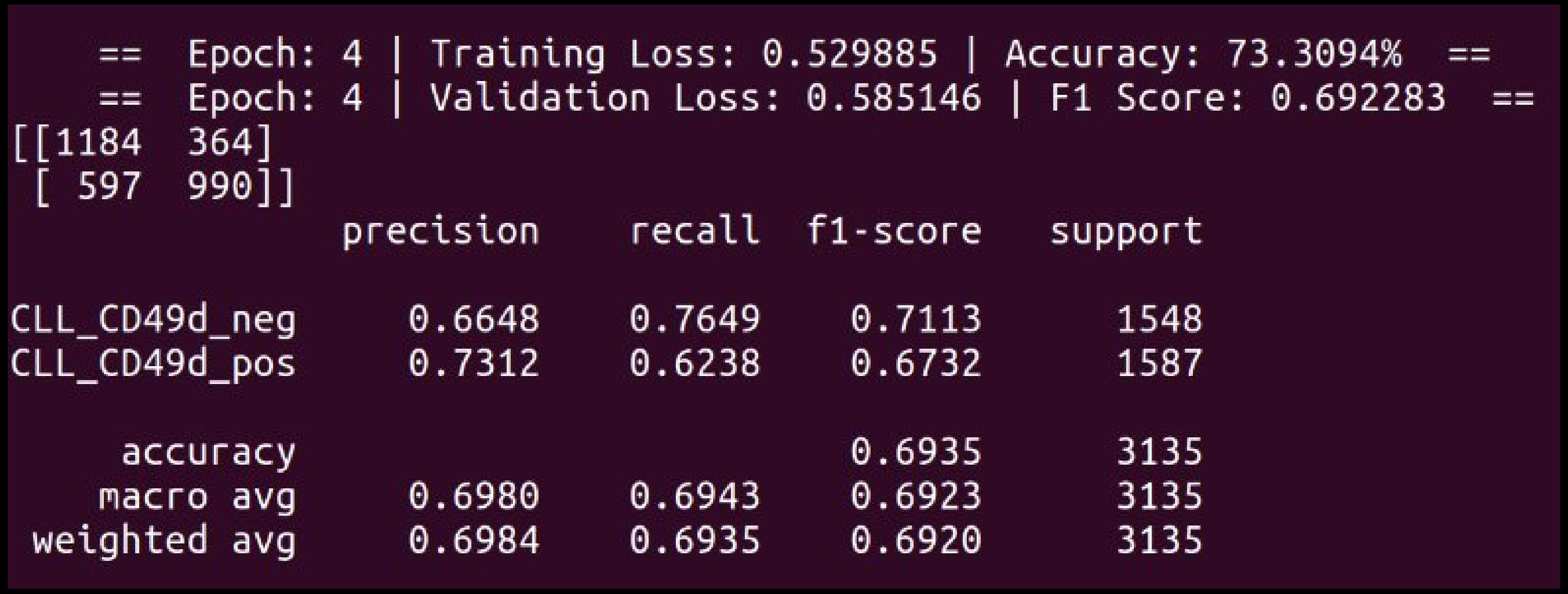


```
    ==  Epoch: 5 | Training Loss: 0.610403 | Accuracy: 74.1395%  ==
    ==  Epoch: 5 | Validation Loss: 0.740919 | F1 Score: 0.687570  ==
[[1143  402   92]
 [  58 1243  300]
 [  40  639  906]]
                 precision    recall  f1-score   support

zero_expression     0.9210    0.6982    0.7943      1637
 low_expression     0.5442    0.7764    0.6399      1601
high_expression     0.6980    0.5716    0.6285      1585

       accuracy                         0.6826      4823
      macro avg     0.7211    0.6821    0.6876      4823
   weighted avg     0.7227    0.6826    0.6886      4823
```

CD49d positive and negative cells.



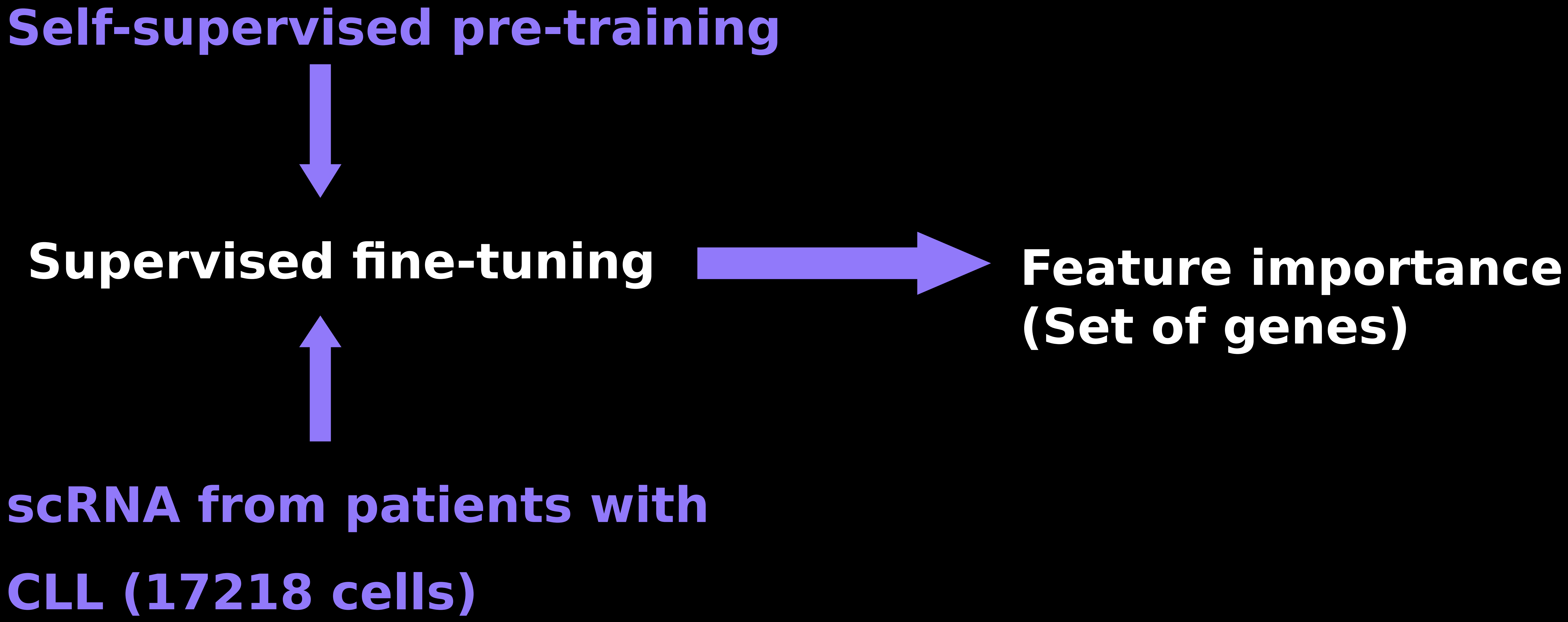
```
    ==  Epoch: 4 | Training Loss: 0.529885 | Accuracy: 73.3094%  ==
    ==  Epoch: 4 | Validation Loss: 0.585146 | F1 Score: 0.692283  ==
[[1184  364]
 [ 597  990]]
               precision    recall  f1-score   support

CLL_CD49d_neg     0.6648    0.7649    0.7113      1548
CLL_CD49d_pos     0.7312    0.6238    0.6732      1587

     accuracy                         0.6935      3135
    macro avg     0.6980    0.6943    0.6923      3135
 weighted avg     0.6984    0.6935    0.6920      3135
```

The model shows reasonable but varying performance across classes, with an overall accuracy of 74% and an F1 score of 0.69, suggesting room for improvement.

Attention weight extraction



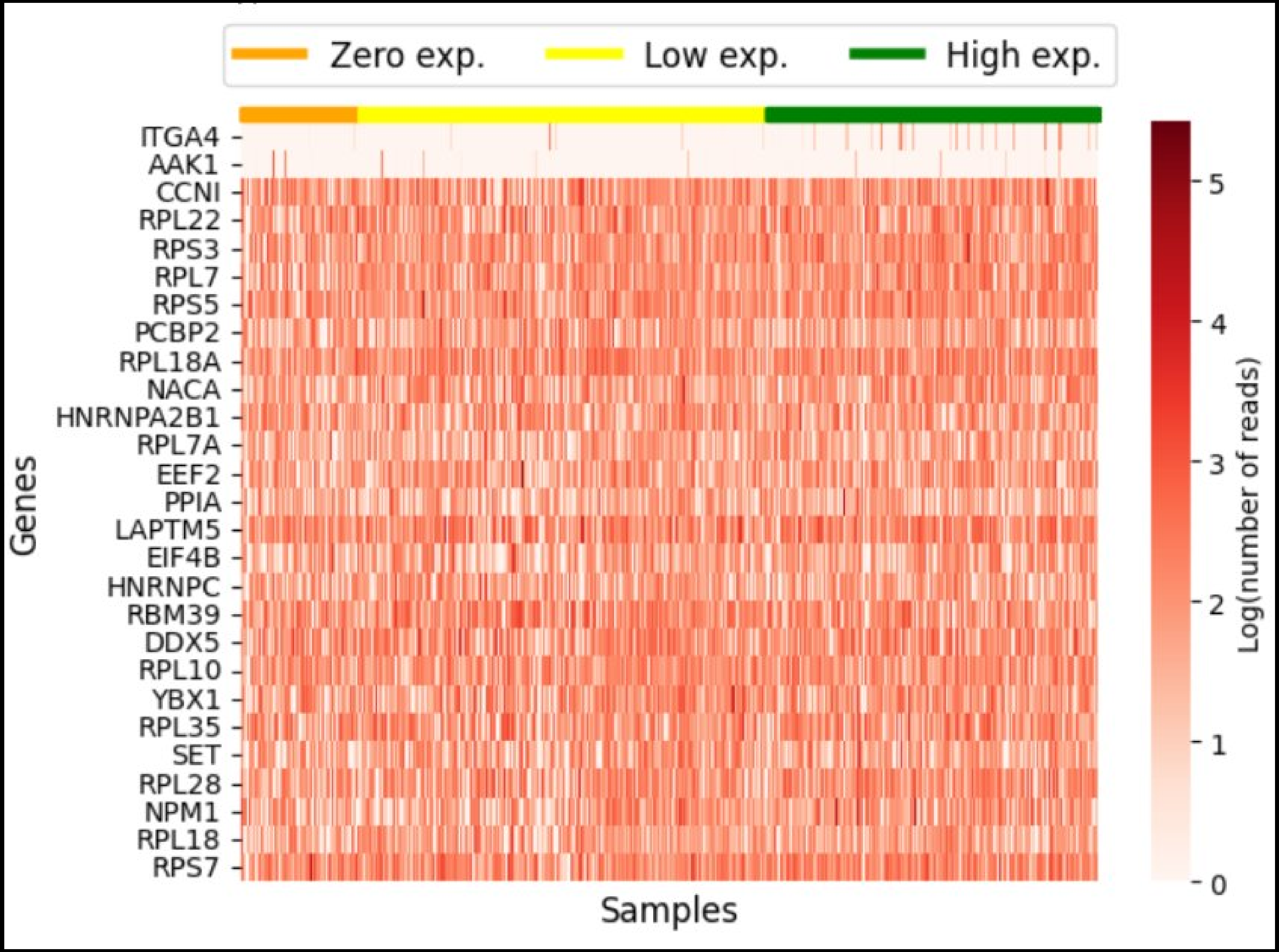
Feature selection

1- Sum the attention weights over all samples in each group.

2- Order in descendant order.

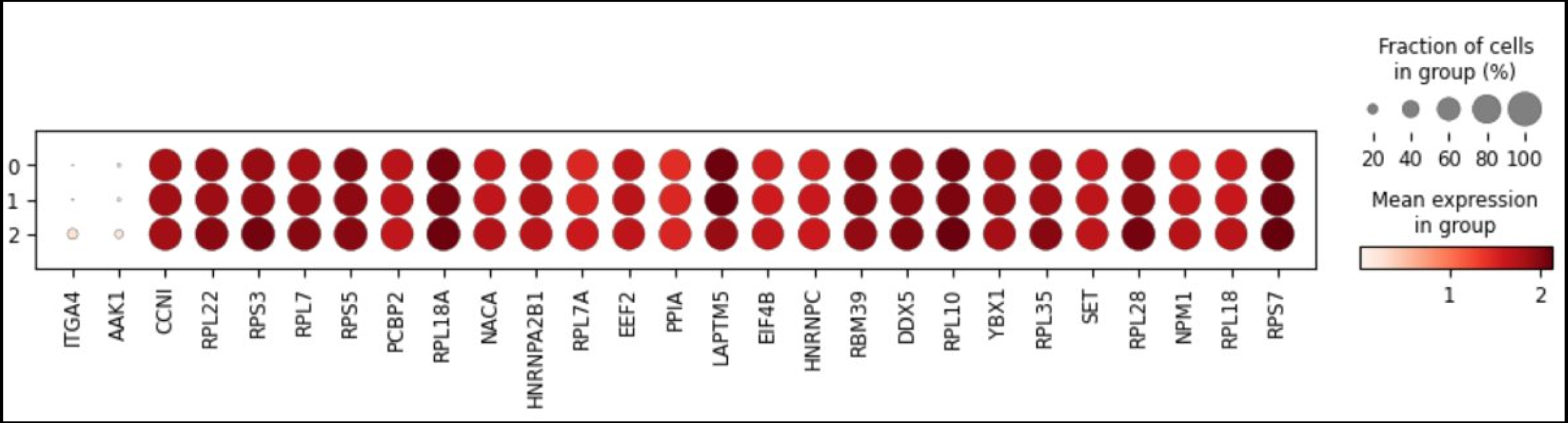
3- Select the most important genes for each group.

Results



Zero exp.
Low exp.
High exp.
ITGA4
AAK1
CCNI
RPL22
RPS3
RPL7
RPS5
PCBP2
RPL18A
NACA
HNRNPA2B1
RPL7A
EEF2
PPIA
LAPTM5
EIF4B
HNRNPC
RBM39
DDX5
RPL10
YBX1
RPL35
SET
RPL28
NPM1
RPL18
RPS7
Genes
Samples
Log(number of reads)
0
1
2
3
4
5

Results



Fraction of cells in group (%)
20
40
60
80
100
Mean expression in group
1
2
0
1
2
ITGA4
AAK1
CCNI
RPL22
RPS3
RPL7
RPS5
PCBP2
RPL18A
NACA
HNRNPA2B1
RPL7A
EEF2
PPIA
LAPTM5
EIF4B
HNRNPC
RBM39
DDX5
RPL10
YBX1
RPL35
SET
RPL28
NPM1
RPL18
RPS7

Future steps

- Try different values of dropout and learning rate to avoid over-fitting during the training

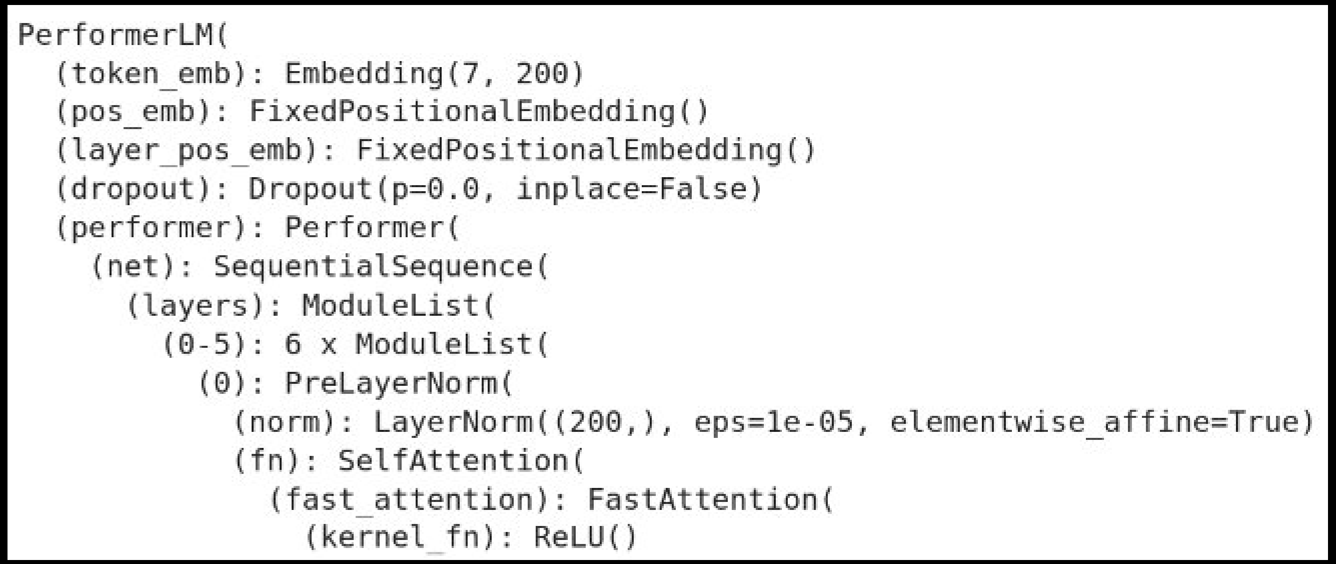
- Apply the model to a easier to discriminate group of cells as T cell vs B cells.

- Run another classification ML model as Random Forest with the same dataset to compare the performances.

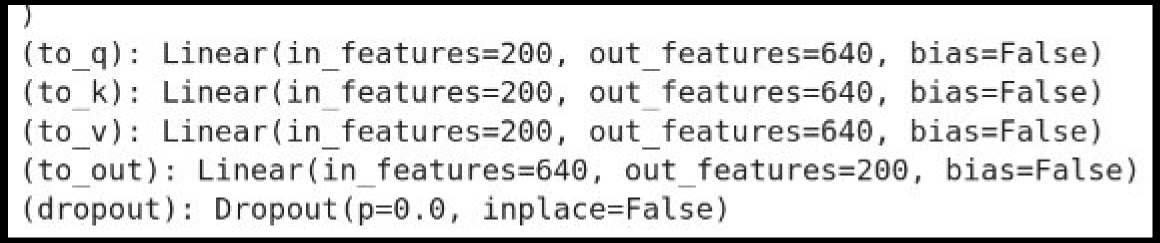
- Run the model with NO oversampled dataset. Random oversampling might lead to overfitting, as the model might learn to memorize the duplicated samples rather than generalize from the data.

Many thanks

scBERT model



```
PerformerLM(
  (token_emb): Embedding(7, 200)
  (pos_emb): FixedPositionalEmbedding()
  (layer_pos_emb): FixedPositionalEmbedding()
  (dropout): Dropout(p=0.0, inplace=False)
  (performer): Performer(
    (net): SequentialSequence(
      (layers): ModuleList(
        (0-5): 6 x ModuleList(
          (0): PreLayerNorm(
            (norm): LayerNorm((200,), eps=1e-05, elementwise_affine=True)
            (fn): SelfAttention(
              (fast_attention): FastAttention(
                (kernel_fn): ReLU()
```



```
)
(to_q): Linear(in_features=200, out_features=640, bias=False)
(to_k): Linear(in_features=200, out_features=640, bias=False)
(to_v): Linear(in_features=200, out_features=640, bias=False)
(to_out): Linear(in_features=640, out_features=200, bias=False)
(dropout): Dropout(p=0.0, inplace=False)
```

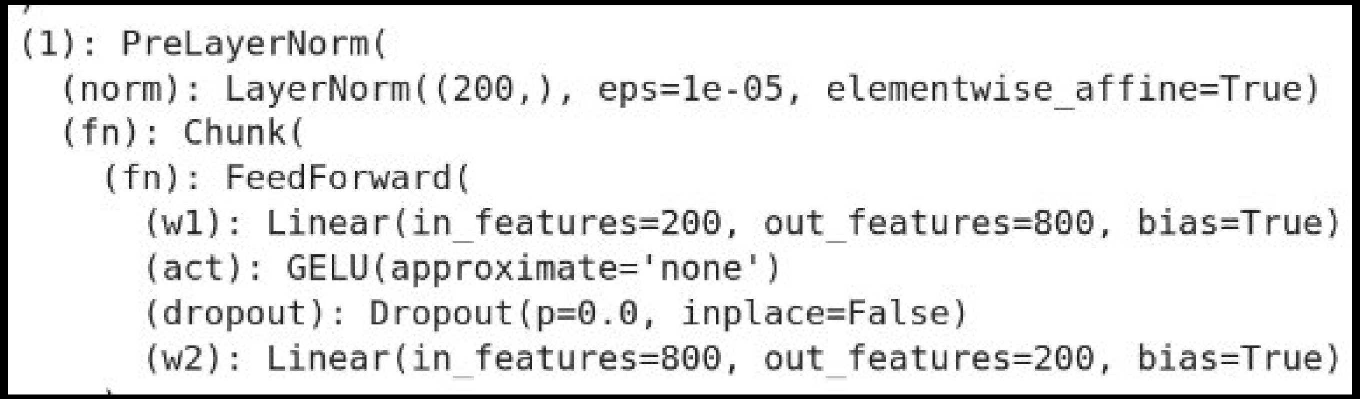
Embedding Layers:
token_emb: Embedding(7, 200): This is an embedding layer that converts the input tokens (with a vocabulary size of 7) into 200-dimensional vectors.
pos_emb: FixedPositionalEmbedding(): This layer adds positional information to the tokens, helping the model to understand the order of the tokens.

Dropout Layer:
dropout: This layer helps prevent overfitting by randomly setting a fraction of input units to 0 at each update during training time.

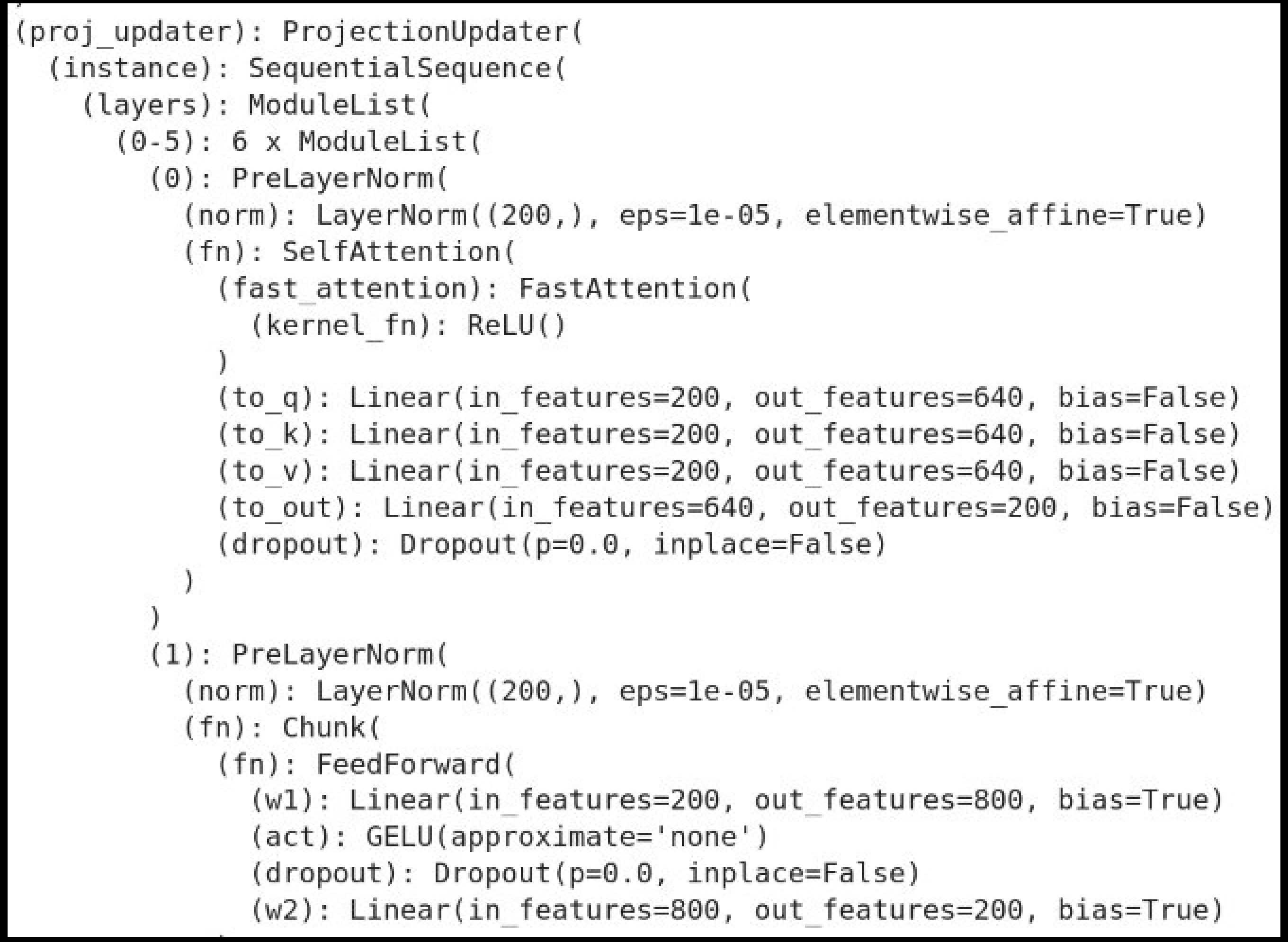
Performer Block:
performer: This block contains the core Performer layers.
net: SequentialSequence with ModuleList layers: This sequential container contains six layers (indexed 0-5), each consisting of:

PreLayerNorm: Applies layer normalization before the main function, which helps stabilize and accelerate the training process.
norm: LayerNorm((200,), eps=1e-05, elementwise_affine=True): Normalizes the input to have mean zero and variance one.
fn:
SelfAttention: The attention mechanism that allows the model to focus on different parts of the input sequence.
fast_attention: FastAttention: A more efficient attention mechanism.
kernel_fn: ReLU(): Activation function used within the attention mechanism.
to_q, to_k, to_v: Linear transformations to compute query, key, and value matrices from the input.
to_out: Linear transformation to produce the output from the attention mechanism.
dropout: Dropout layer within the attention mechanism.

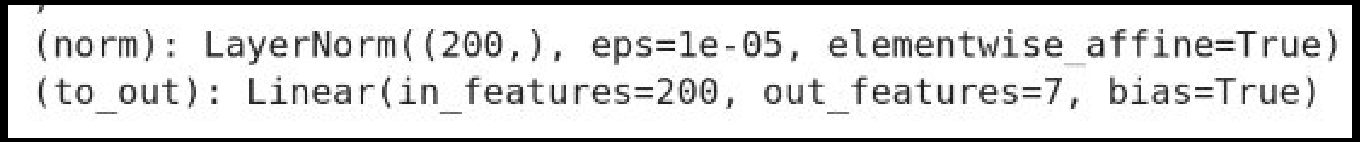
scBERT model



```
(1): PreLayerNorm(
  (norm): LayerNorm((200,), eps=1e-05, elementwise_affine=True)
  (fn): Chunk(
    (fn): FeedForward(
      (w1): Linear(in_features=200, out_features=800, bias=True)
      (act): GELU(approximate='none')
      (dropout): Dropout(p=0.0, inplace=False)
      (w2): Linear(in_features=800, out_features=200, bias=True)
```



```
(proj_updater): ProjectionUpdater(
  (instance): SequentialSequence(
    (layers): ModuleList(
      (0-5): 6 x ModuleList(
        (0): PreLayerNorm(
          (norm): LayerNorm((200,), eps=1e-05, elementwise_affine=True)
          (fn): SelfAttention(
            (fast_attention): FastAttention(
              (kernel_fn): ReLU()
            )
            (to_q): Linear(in_features=200, out_features=640, bias=False)
            (to_k): Linear(in_features=200, out_features=640, bias=False)
            (to_v): Linear(in_features=200, out_features=640, bias=False)
            (to_out): Linear(in_features=640, out_features=200, bias=False)
            (dropout): Dropout(p=0.0, inplace=False)
          )
        )
        (1): PreLayerNorm(
          (norm): LayerNorm((200,), eps=1e-05, elementwise_affine=True)
          (fn): Chunk(
            (fn): FeedForward(
              (w1): Linear(in_features=200, out_features=800, bias=True)
              (act): GELU(approximate='none')
              (dropout): Dropout(p=0.0, inplace=False)
              (w2): Linear(in_features=800, out_features=200, bias=True)
```



```
(norm): LayerNorm((200,), eps=1e-05, elementwise_affine=True)
(to_out): Linear(in_features=200, out_features=7, bias=True)
```

Chunk: Groups operations together for more efficient processing.
FeedForward: A two-layer feedforward neural network.

w1: Linear layer from input dimension 200 to hidden dimension 800. It performs a matrix multiplication between the input and a weight matrix of size 200x800, followed by adding a bias vector of size 800.

act: GELU(): Activation function that performs a smooth approximation to the Rectified Linear Unit (ReLU). It zeroes out negative inputs in a smooth and continuous manner, which can help with gradient flow and model performance.

Dropout layer.

w2: Similar to w1, this layer performs a matrix multiplication between the input (now 800-dimensional) and a weight matrix of size 800x200, followed by adding a bias vector of size 200.

Projection Updater:
proj_updater: Another sequential block that repeats the structure of the main Performer block, likely used for updating projections or embeddings.
Normalization and Output:

norm: LayerNorm((200,), eps=1e-05, elementwise_affine=True): Final layer normalization.

to_out: Linear(in_features=200, out_features=7, bias=True): Linear transformation to produce the final output, mapping from the model's internal representation (dimension 200) to the output classes (dimension 7).