University of Hertfordshire UH

School of Physics, Engineering and Computer Science

MSc Data Science Project

7PAM2002

Department of Physics, Astronomy and Mathematics

Data Science FINAL PROJECT REPORT

Project Title:

Predicting Developmental Disorders Using scRNA-seq Data

Student Name and SRN:

Samara Naeem - 23009648

Supervisor: William Bate

Date Submitted: 29/04/25.

Word Count: 4962

GIT-HUB: https://github.com/SamaraNaeem/EmbryoGene-Predictor

DECLARATION STATEMENT

This report is submitted in partial fulfilment of the requirement for the degree of Master of Science **in Data Science** at the University of Hertfordshire.

I have read the detailed guidance to students on academic integrity, misconduct and plagiarism information at Assessment Offences and Academic Misconduct and understand the University process of dealing with suspected cases of academic misconduct and the possible penalties, which could include failing the project or course.

I certify that the work submitted is my own and that any material derived or quoted from published or unpublished work of other persons has been duly acknowledged. (Ref. UPR AS/C/6.1, section 7 and UPR AS/C/5, section 3.6)

I did not use human participants in my MSc Project.

I hereby give permission for the report to be made available on module websites provided the source is acknowledged.

Student Name printed: Samara Naeem

Student Name Signature 

Student SRN number: 23009648

UNIVERSITY OF HERTFORDSHIRE

SCHOOL OF PHYSICS, ENGINEERING AND COMPUTER SCIENCE

Abstract

This project explores how machine learning can be used to predict developmental disorders by analysing gene expression patterns from early human embryos. The work is based on the GSE155121 dataset, which includes single-cell RNA sequencing (scRNA-seq) profiles of over 476,000 cells obtained from thirteen embryos across post-conception weeks 3 to 12 and it is a crucial window during which gastrulation, neurulation, and early organ formation occur. Each cell in the dataset is profiled across approximately 32,000 genes and comes annotated with information about its spatial origin and developmental identity. To handle the complexity of this high-dimensional and sparse data, the project involved extensive preprocessing, including normalization, variable gene filtering, and dimensionality reduction. Three machine learning models are used which are GAT, XGBoost and Random Forest Performance was evaluated using precision, recall, F1 score, and ROC-AUC. Among all three models tested, XGBoost showed the strongest results, got a F1 score of 0.989. Beyond predictive accuracy, the model also helped highlight potential marker genes linked to disrupted developmental pathways. These findings suggest that integrating scRNA-seq with machine learning can provide powerful insights into early-stage developmental disorders and may contribute to earlier diagnosis and intervention in the future.

Contents

1. Introduction

1.1 Background

Developmental disorders represent a serious global health challenge, often caused by a combination of genetic makeup, epigenetic influences, and environmental conditions. These disorders can interfere with the normal development of critical systems such as the brain, heart, and bones, potentially leading to lifelong disability or even death. According to the World Health Organization (WHO), nearly 295,000 newborns die annually due to congenital anomalies, often because diagnosis is delayed or missed entirely.

Advances in transcriptomics especially techniques like single-cell RNA sequencing (scRNA-seq) have greatly improved our ability to study embryonic development at the cellular level. This approach enables scientists to observe how genes are expressed across various developmental stages and cell types, allowing early detection of molecular abnormalities that could be linked to developmental disorders.

1.2 Statement of the Research Problem

Despite the high incidence of congenital disorders, early-stage detection remains inadequate due to the complexity and volume of gene expression data. Traditional diagnostic methods often rely on morphological or late-stage clinical symptoms, missing the critical window for intervention. There is a pressing need for a computational framework that can analyse high-dimensional scRNA-seq data to predict developmental anomalies accurately and early.

1.3 Research Question

- How can ML techniques be applied to scRNA-seq data to forecast developmental disorders?
- Which gene markers are most significant in early embryonic stages for disorder identification?

1.4 Study Aim and Purpose

The primary goal is to design and assess a predictive framework utilizing machine learning to pinpoint early-stage biomarkers associated with developmental disorders through the exploration of data derived from single-cell RNA sequencing.

1.5 Specific Research Objectives

- To preprocess and analyse the GSE155121 dataset (Zeng, 2023).
- To implement ML models GAT, XGBoost and Random Forest) for prediction.
- To identify key gene markers contributing to disorder risk.

1.6 Importance and Relevance of the Study

Early diagnosis of congenital disorders can significantly improve patient outcomes and reduce healthcare burdens. This research combines high-throughput transcriptomic profiling with advanced computational modelling to address a critical gap in early disease prediction. It not only contributes to biomedical informatics but also serves as a prototype for future diagnostic frameworks in prenatal genomics.

2. Literature Comparison

Recent studies have increasingly applied machine learning and transcriptomic analysis to understand developmental biology and prioritize disease genes. (Chen, XGBoost: A Scalable Tree Boosting System, 794) introduced XGBoost, a powerful and scalable tree-based boosting algorithm, has emerged as a key tool in predictive modeling within biomedical research. (Diaz-Uriarte, 2006) demonstrated the strength of Random Forest for gene selection in microarray data, highlighting its robustness in high-dimensional biological datasets. (Deng, 2019) extended machine learning applications to scRNA-seq by developing scalable approaches for analysing cell-type composition, showing the effectiveness of deep learning techniques.

(Wang, 2018) proposed a graph-based integrative model for the human brain using comprehensive functional genomic resources, introducing graph neural networks to capture complex biological relationships. In contrast, (Xiang, 2020) generated a detailed developmental map of 3D-cultured human pre-gastrulation embryos using scRNA-seq, while (Zeng, 2023) charted the single-cell and spatial gene expression patterns involved in human gastrulation and early brain formation, without applying predictive modelling. Both studies provided essential biological baselines for later computational approaches.

Most notably, Balachandran et al. [1] developed STIGMA, a machine learning framework that uses Random Forest to prioritize tissue-specific disease genes from scRNA-seq data over developmental time, bridging computational analysis with disease gene discovery. Compared to earlier models, STIGMA highlights the shift towards integrating temporal and spatial information into gene prioritization pipelines.

(Veličković, 2018) introduced Graph Attention Networks (GAT), a model that learns node embeddings by attending over neighbouring nodes, allowing dynamic weighting of node importance during message passing. (Wang, 2018)] applied a graph attention-based model, scGNN, to scRNA-seq datasets to capture complex cell-cell relationships.

Overall, the comparison indicates that XGBoost and Random Forest models remain strong choices for predictive analysis, while newer approaches emphasize graph-based learning and integration of spatial dynamics. The present study builds upon these foundations, aiming to apply machine learning models to predict developmental trajectories and assist in the early detection of developmental disorders.

3: Methodology

This project applied a machine learning framework to predict potential developmental disorders using high-dimensional scRNA-seq data from early-stage human embryos. The entire workflow including data acquisition, cleaning, feature reduction, model development, and performance assessment was carried out in Python using publicly available open-source libraries.

3.1 Dataset

The primary dataset used in this project is GSE155121, a publicly available single-cell RNA sequencing (scRNA-seq) dataset published by (Xiang et al., 2023) [1]. It contains transcriptomic profiles from 13 post-implantation human embryos, covering developmental stages between post-conception weeks (PCW) 3 to 12. The dataset encompasses over 476,000 cells and includes rich metadata such as embryonic age, cell type annotations, anatomical locations, and lineage relationships. Each cell's gene expression profile spans over 32,000 genes, stored in an AnnData object structure containing two main components:

- adata.obs: Metadata for each cell (e.g., embryonic stage)
- adata.var: Gene features and annotation metadata

3.2 Data Preprocessing and Exploratory Data Analysis

Given the high dimensionality and sparsity typical of scRNA-seq data, several preprocessing steps were applied:

- **Missing Values:** The sparse expression matrix was converted to dense format to check for NaN values. The dataset was confirmed to be complete with no missing values.
- **Log Transformation:** A log1p transformation was applied to normalize the distribution of gene expression, ensuring that highly expressed genes did not dominate the analysis.

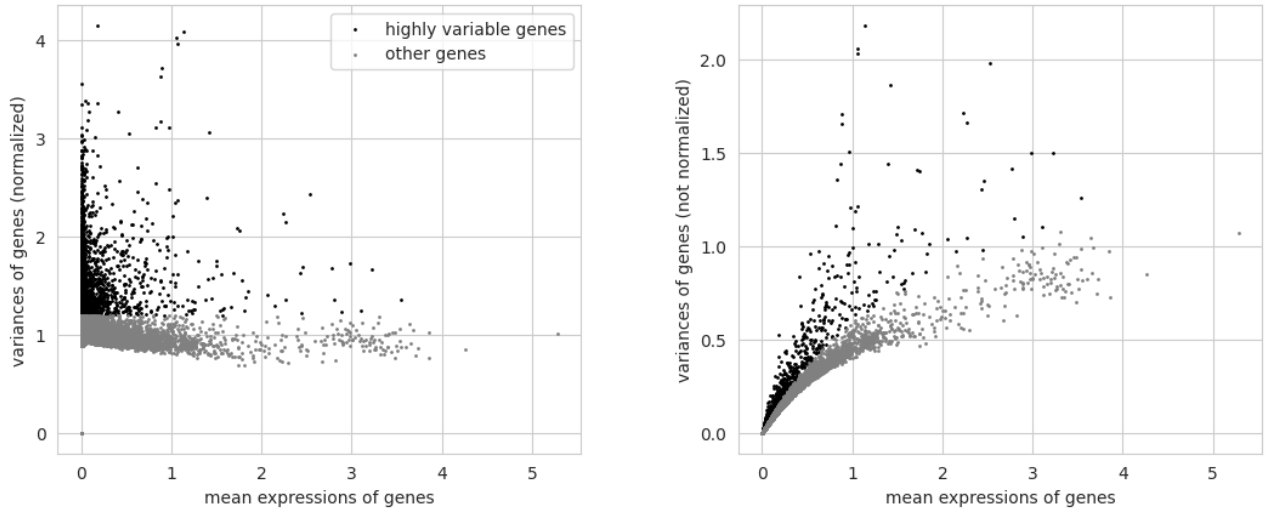


Figure 3.1: Expression of Mitochondrial Genes

Scatter plot visualizing mitochondrial gene content, used to assess cell quality and exclude stressed/dying cells.

- **Highly Variable Gene (HVG) Selection:** To reduce noise and focus on biologically relevant signals, the top **2,000 most variable genes** were selected using the Seurat v3 method.

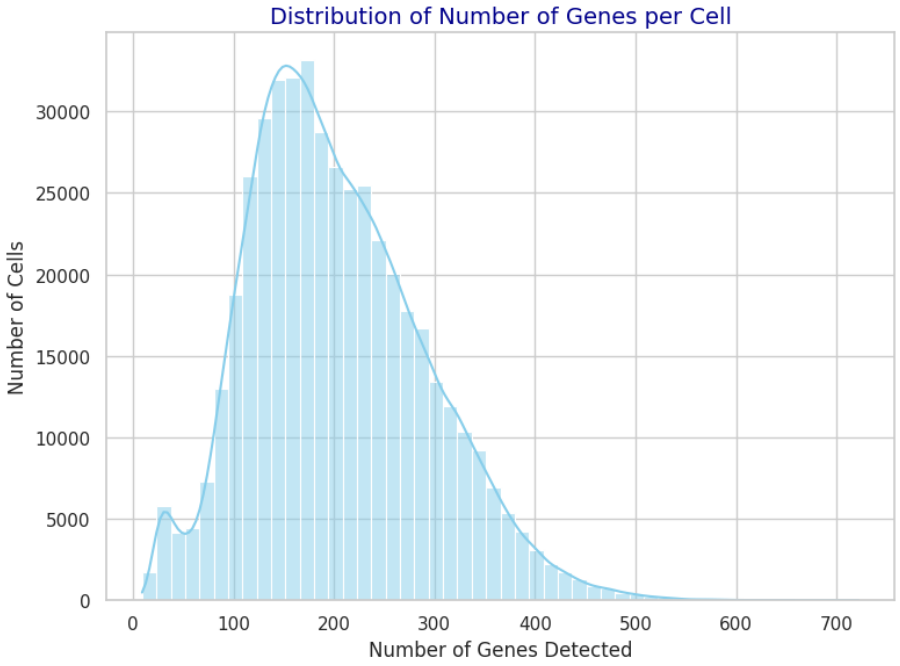


Figure 3.2: Total Genes Expressed per Week Stage

Bar plot showing aggregate gene expression across embryonic stages (PCW 3–12)

- **Scaling:** Gene expression was scaled and clipped to ±10 to normalise feature ranges and limit the influence of extreme values.

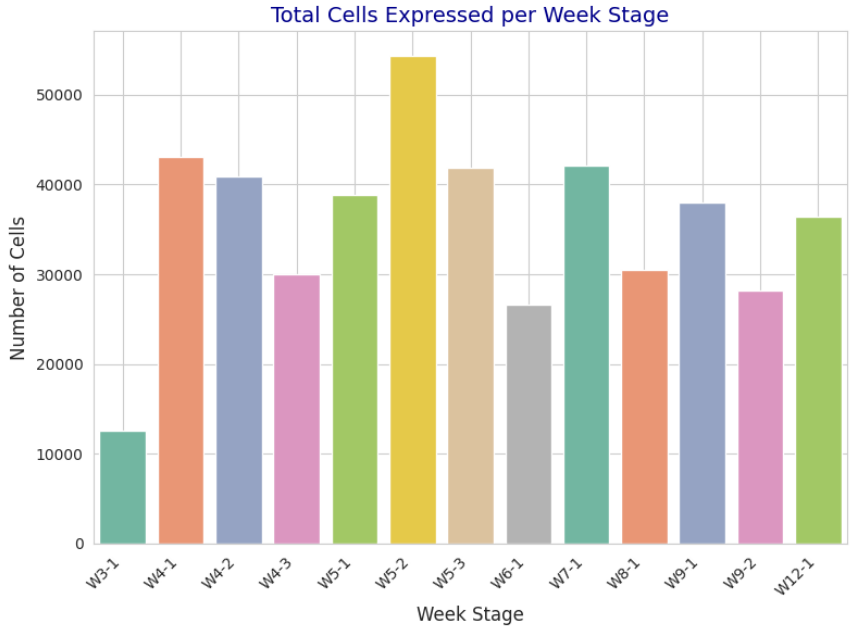


Figure 3.2: Cell Count Distribution by Stage

Histogram showing the number of sequenced cells at each PCW stage. Higher counts in mid-embryonic stages (PCW 5–8) indicate peak developmental sampling.

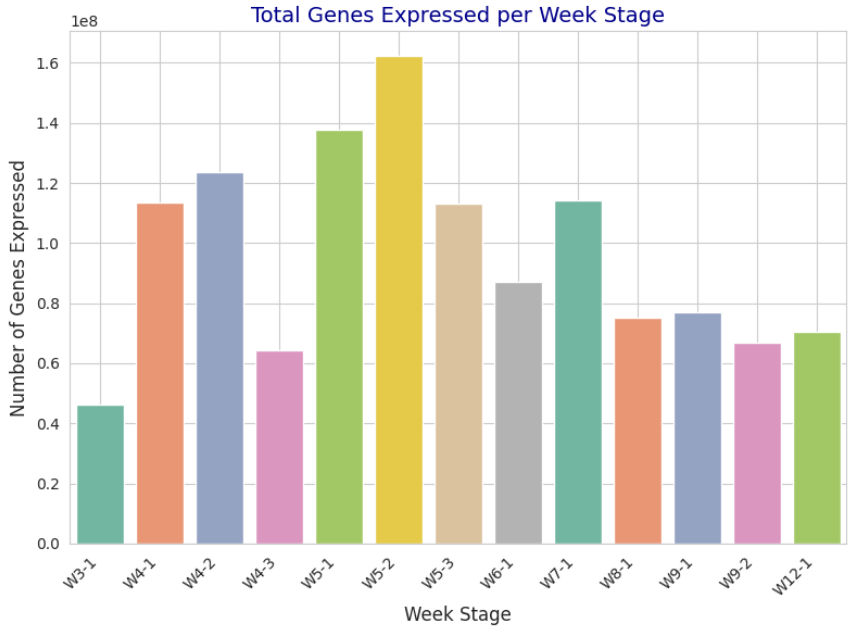


Figure 3.2: Gene Count Distribution by Stage

Histogram showing the number of sequenced cells at each PCW stage. Higher counts in mid-embryonic stages (PCW 5–8) indicate peak developmental sampling.

3.3 Model Selection and Training

I developed and evaluated three machine learning models that are Graph Attention Network (GAT), Random Forest (RF), and XGBoost scRNA-seq dataset. This section describes the practical and technical steps I undertook, the coding frameworks used, model choices, metrics, optimisation efforts, and evaluation strategy.

3.3.1 Graph Attention Network (GAT)

I developed a custom GAT architecture using PyTorch Geometric, where cells were not processed in isolation but rather represented as nodes in a graph formed using k-nearest neighbours (kNN) derived from PCA-transformed features.

The model architecture consisted of two GATConv layers designed for classifying developmental stages based on gene expression variability. The first layer accepted an input dimension of 50 and projected it to a hidden space of 64 using 3 attention heads, with output concatenation enabled. To introduce non-linearity, I applied the Exponential Linear Unit (ELU) activation function. The second layer received 192-dimensional input (64 features × 3 heads) and projected it into 13 class logits using a single attention head without concatenation. A log-softmax activation was applied at the output to yield probability distributions for multi-class classification.

To build the graph, I used the torch_cluster.knn_graph function, which linked each node to its nearest neighbours based on gene expression similarity. Through the attention mechanism, the model learned to prioritize informative neighbouring nodes. The training process used Negative Log Likelihood Loss (nn.CrossEntropyLoss) as the loss criterion, and optimization was performed using the Adam optimizer (torch.optim.Adam) with a learning rate of 0.01. The model was trained for 25 epochs to ensure convergence.

3.3.2 Random Forest (RF)

To create a reliable reference point without tuning hyperparameters manually, I utilized a Random Forest classifier through the scikit-learn library. The input comprised gene expression values from 2,000 highly variable genes, selected via the Seurat v3 algorithm. To preserve the expression profiles, no dimensionality reduction (e.g., PCA) was applied. The model was initialized with 100 decision trees (n_estimators=100) and employed the Gini index as the split criterion. For consistency across runs, a random seed value of 42 was set.

The target labels reflected developmental "week_stage" annotations, which I converted into numerical values using the LabelEncoder from scikit-learn. The dataset was then split into 80% for training and 20% for testing using the train_test_split function. This setup provided a clear and reproducible baseline for comparison against more complex models.

3.3.3 XGBoost

To boost predictive accuracy and uncover key gene markers, I applied the XGBoost algorithm with a multiclass softmax configuration (multi:softmax) and evaluated its performance using the multiclass log loss (mlogloss) metric. I set the number of boosting rounds (estimators) to 200, the learning rate to 0.05, and capped tree depth at 6 to prevent overfitting. Additionally, I used a row sampling rate of 0.8 and a column sampling rate per tree of 0.8, with a fixed random seed of 42 to ensure repeatable results.

The XGBoost model was trained using gene expression values from the same 2,000 highly variable genes. Post-training, I examined the feature importance scores via the model's built-in method, which highlighted how much each gene contributed to classification accuracy.

3.4 Model Evaluation Approach

All models in this study were evaluated using a standardized and rigorous pipeline to ensure fair comparison and robust validation. The performance of each model was measured using multiple metrics, including accuracy, precision, recall, and F1-score. These metrics provided a balanced evaluation of both correctness and class-wise sensitivity. Additionally, confusion matrices were generated to visualize true versus predicted class distributions, and ROC-AUC scores were calculated where applicable to assess the discriminatory power of the classifiers. This multi-metric evaluation strategy was essential for comparing model performance in the complex, high-dimensional context of scRNA-seq-based developmental stage prediction.

4. Results and Analysis

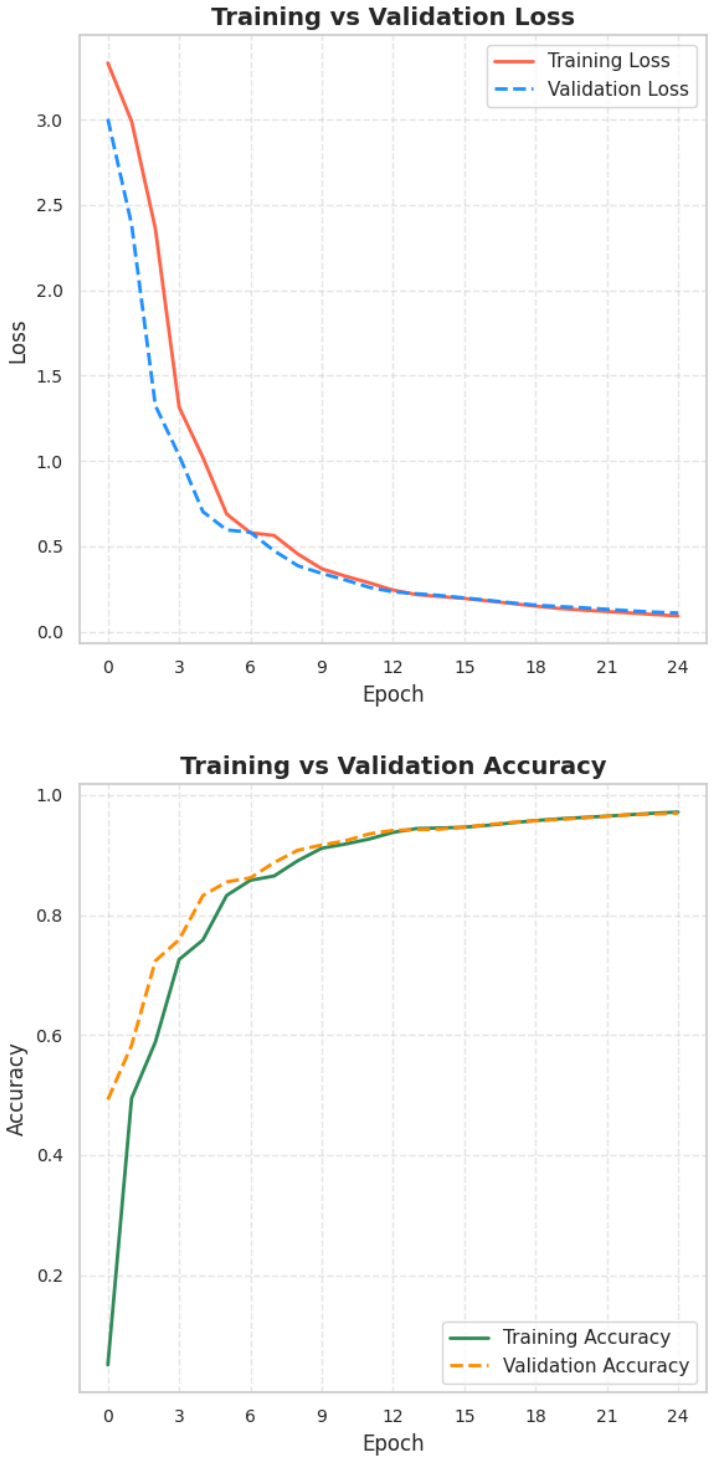


Figure 4.1: GAT Model Training Curve and Validation Curve (a) Loss and (b) Accuracy

The loss curves for both training and validation consistently decline without showing notable separation, which suggests the model avoids overfitting. In parallel, the accuracy curves for training and validation increase in close alignment, indicating that the GAT model performs reliably on data it has not previously encountered.

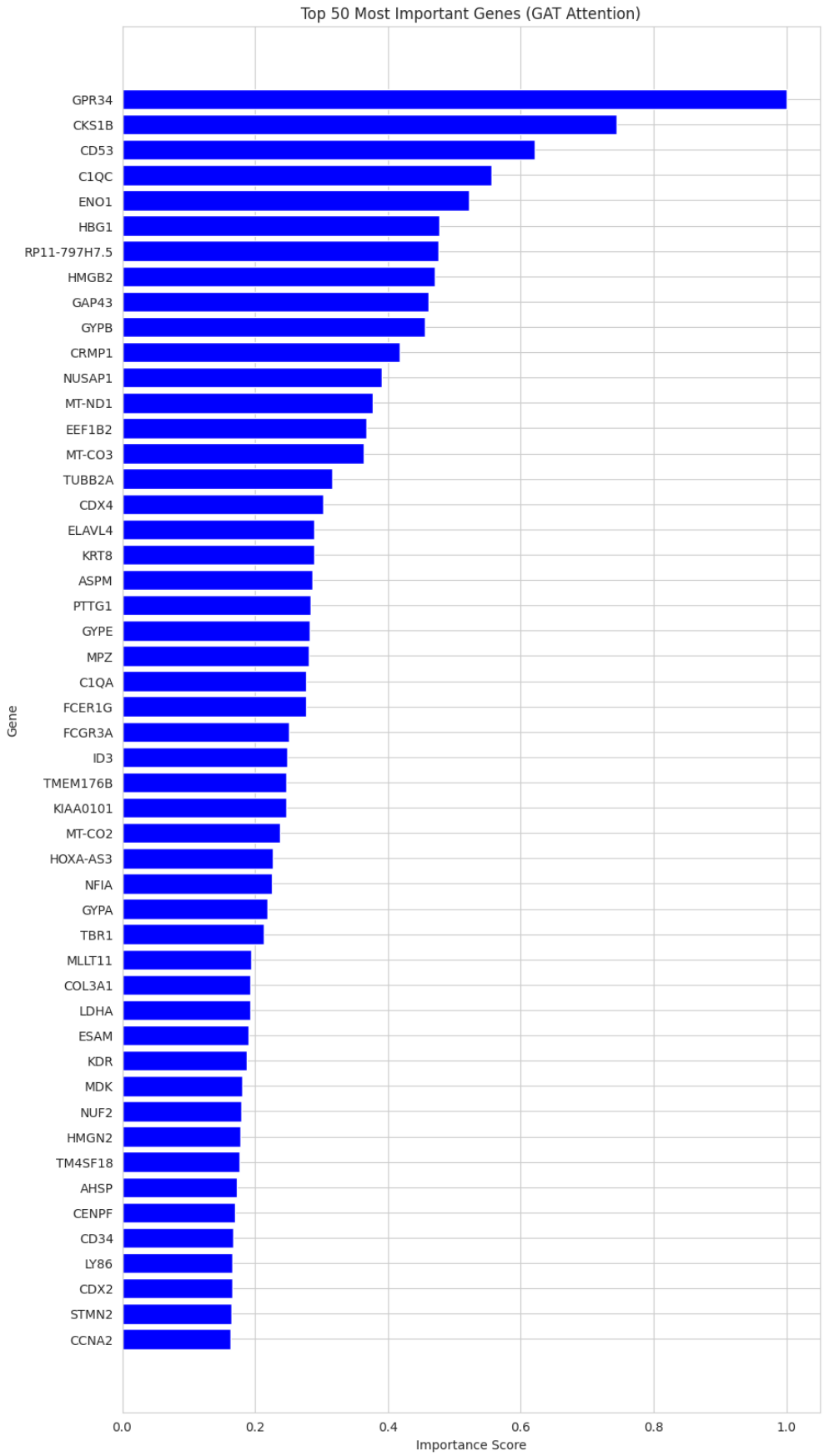
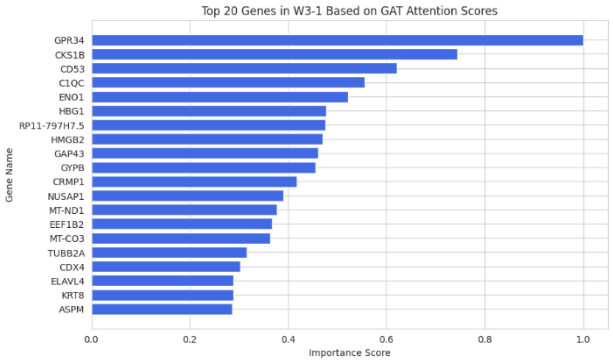
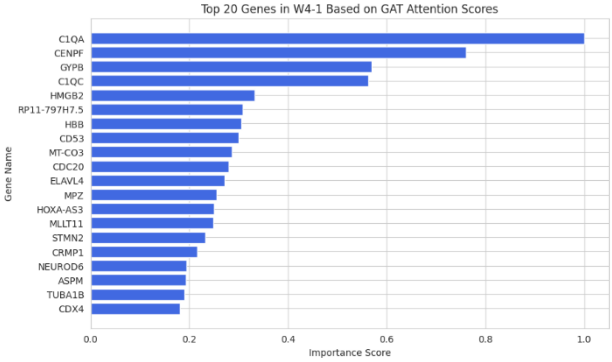
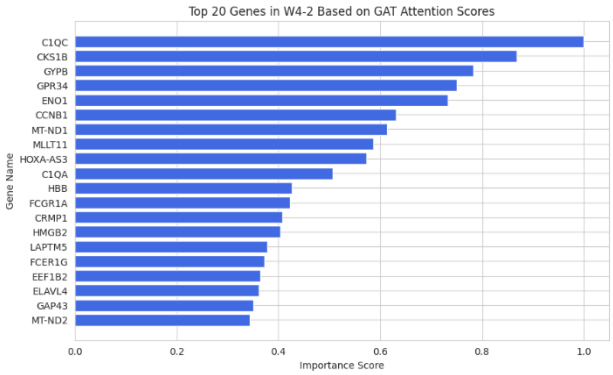
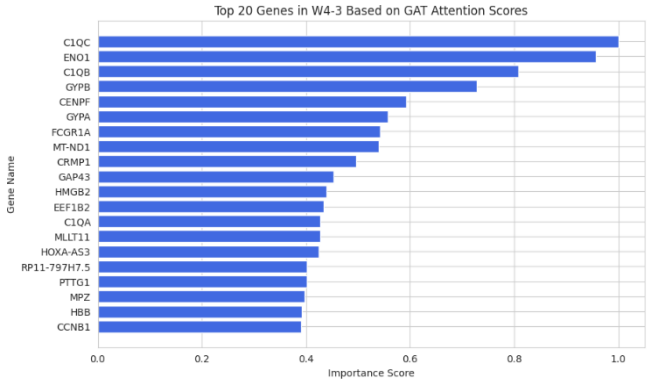


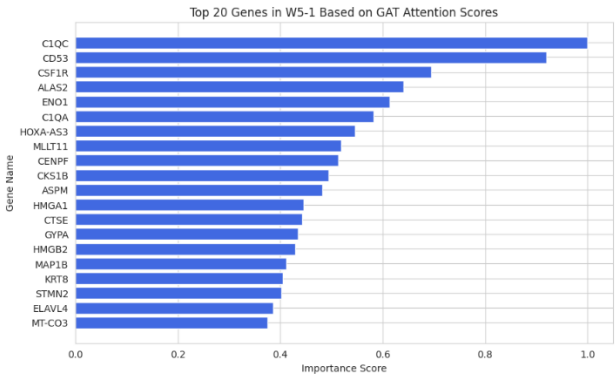
Figure 4.2: Top 50 Important Genes from GAT

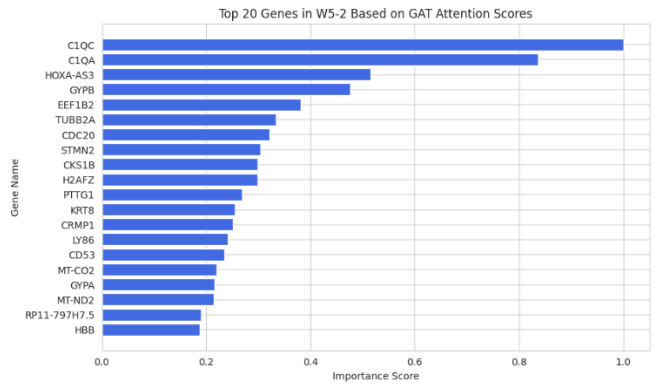

Top 20 Genes in W3-1 Based on GAT Attention Scores
GPR34
CKS1B
CD53
C1QC
ENO1
HBG1
RP11-797H7.5
HMGB2
GAP43
GYPB
CRMP1
NUSAP1
MT-ND1
EEF1B2
MT-CO3
TUBB2A
CDX4
ELAVL4
KRT8
ASPM
Gene Name
0.0
0.2
0.4
0.6
0.8
1.0
Importance Score

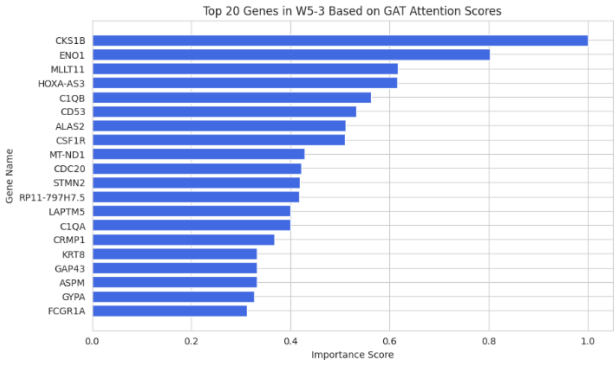

Top 20 Genes in W4-1 Based on GAT Attention Scores
C1QA
CENPF
GYPB
C1QC
HMGB2
RP11-797H7.5
HBB
CD53
MT-CO3
CDC20
ELAVL4
MPZ
HOXA-AS3
MLLT11
STMN2
CRMP1
NEUROD6
ASPM
TUBA1B
CDX4
Gene Name
0.0
0.2
0.4
0.6
0.8
1.0
Importance Score

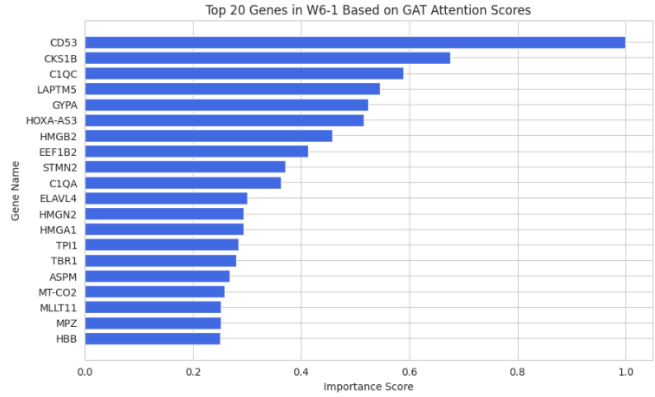

Top 20 Genes in W4-2 Based on GAT Attention Scores
C1QC
CKS1B
GYPB
GPR34
ENO1
CCNB1
MT-ND1
MLLT11
HOXA-AS3
C1QA
HBB
FCGR1A
CRMP1
HMGB2
LAPTM5
FCER1G
EEF1B2
ELAVL4
GAP43
MT-ND2
Gene Name
0.0
0.2
0.4
0.6
0.8
1.0
Importance Score


Top 20 Genes in W4-3 Based on GAT Attention Scores
C1QC
ENO1
C1QB
GYPB
CENPF
GYPA
FCGR1A
MT-ND1
CRMP1
GAP43
HMGB2
EEF1B2
C1QA
MLLT11
HOXA-AS3
RP11-797H7.5
PTTG1
MPZ
HBB
CCNB1
Gene Name
0.0
0.2
0.4
0.6
0.8
1.0
Importance Score


Top 20 Genes in W5-1 Based on GAT Attention Scores
C1QC
CD53
CSF1R
ALAS2
ENO1
C1QA
HOXA-AS3
MLLT11
CENPF
CKS1B
ASPM
HMGA1
CTSE
GYPA
HMGB2
MAP1B
KRT8
STMN2
ELAVL4
MT-CO3
Gene Name
0.0
0.2
0.4
0.6
0.8
1.0
Importance Score


Top 20 Genes in W5-2 Based on GAT Attention Scores
C1QC
C1QA
HOXA-AS3
GYPB
EEF1B2
TUBB2A
CDC20
STMN2
CKS1B
H2AFZ
PTTG1
KRT8
CRMP1
LY86
CD53
MT-CO2
GYPA
MT-ND2
RP11-797H7.5
HBB
Gene Name
0.0
0.2
0.4
0.6
0.8
1.0
Importance Score


Top 20 Genes in W5-3 Based on GAT Attention Scores
CKS1B
ENO1
MLLT11
HOXA-AS3
C1QB
CD53
ALAS2
CSF1R
MT-ND1
CDC20
STMN2
RP11-797H7.5
LAPTM5
C1QA
CRMP1
KRT8
GAP43
ASPM
GYPA
FCGR1A
Gene Name
0.0
0.2
0.4
0.6
0.8
1.0
Importance Score


Top 20 Genes in W6-1 Based on GAT Attention Scores
CD53
CKS1B
C1QC
LAPTM5
GYPA
HOXA-AS3
HMGB2
EEF1B2
STMN2
C1QA
ELAVL4
HMGN2
HMGA1
TPI1
TBR1
ASPM
MT-CO2
MLLT11
MPZ
HBB
Gene Name
0.0
0.2
0.4
0.6
0.8
1.0
Importance Score

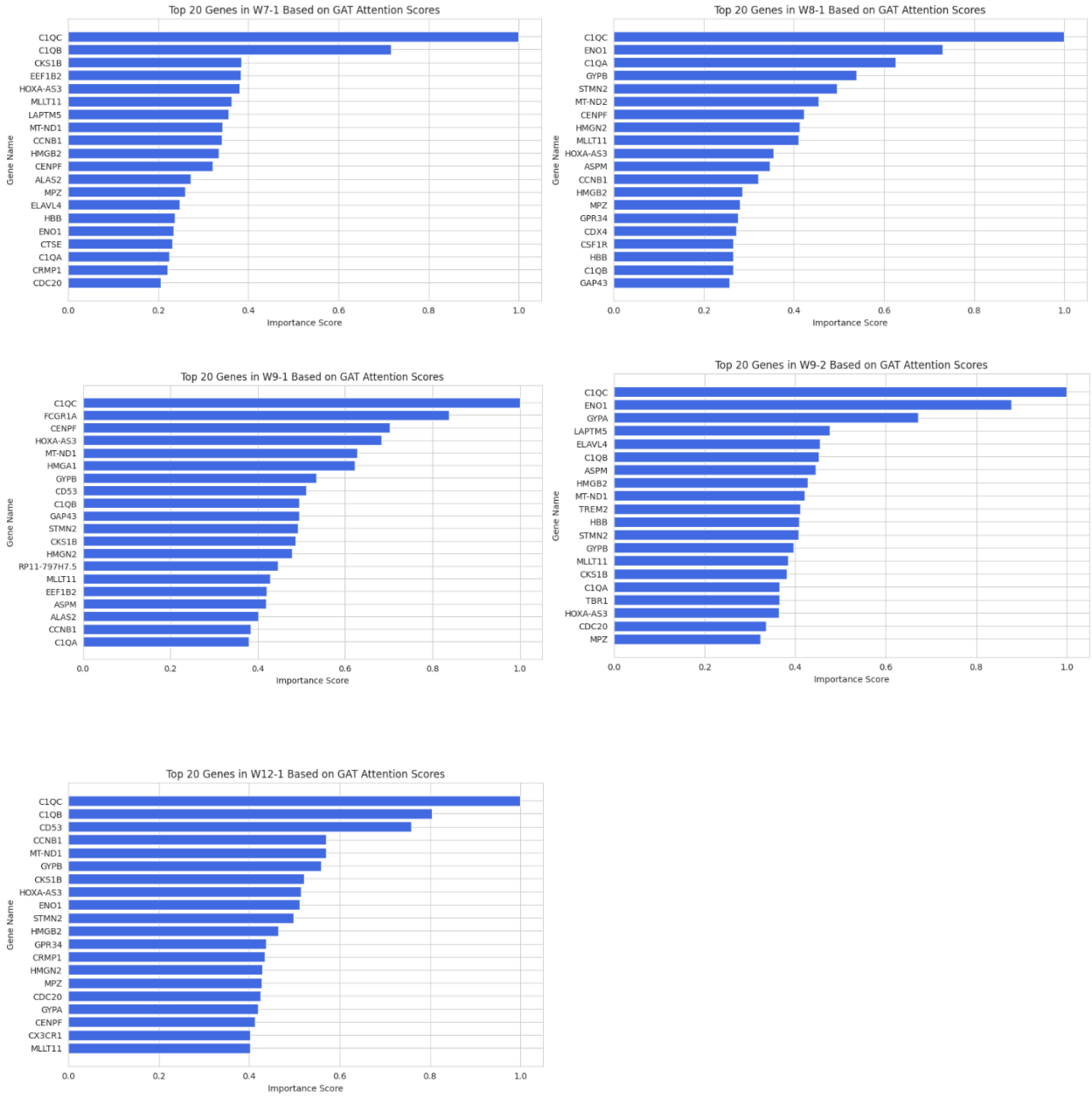


Figure 4.3: Top 20 Genes *(i) WK 3-1 (ii) WK 4-1 (iii) WK 4-2 (iv) WK 4-3 (v) WK 5-1 (vi) WK 5-2 (vii) WK 5-3 (viii) WK 6-1 (ix) WK 7-1 (x) WK 8-1 (xi) WK 9-1 (xii) WK 9-2 (xiii) WK 12*

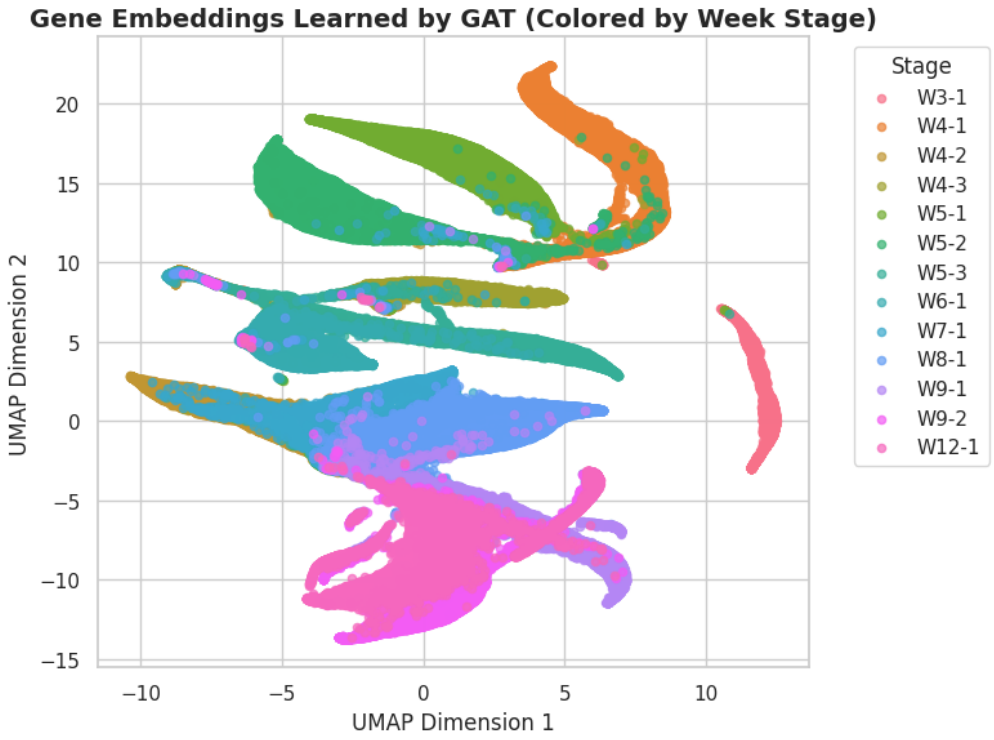


Figure 4.4: GAT Embeddings Visualized by UMAP

Clear stage-wise separation validates that GAT captures biologically meaningful features.

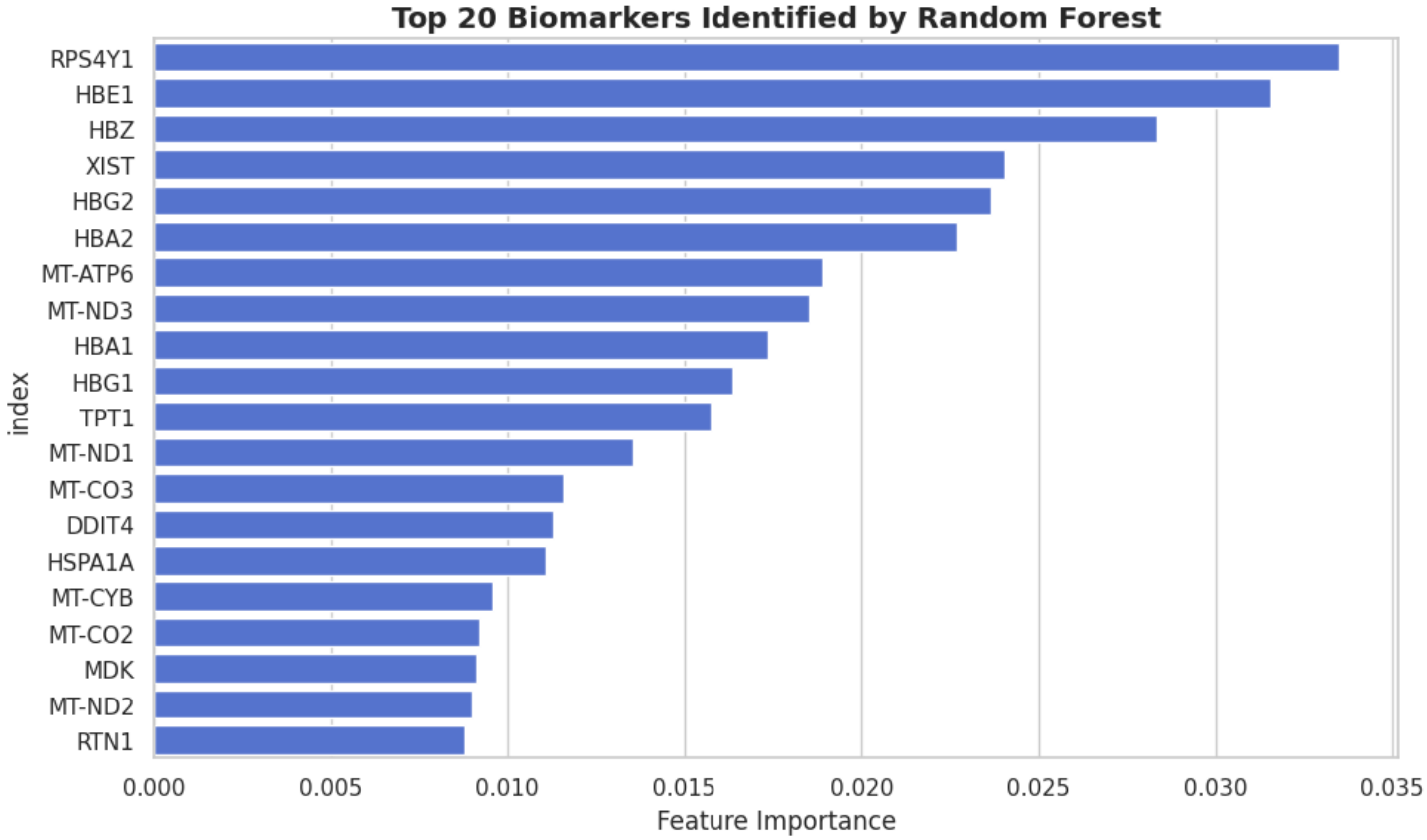


Figure 4.5: Top 20 biomarkers identified by Random Forest

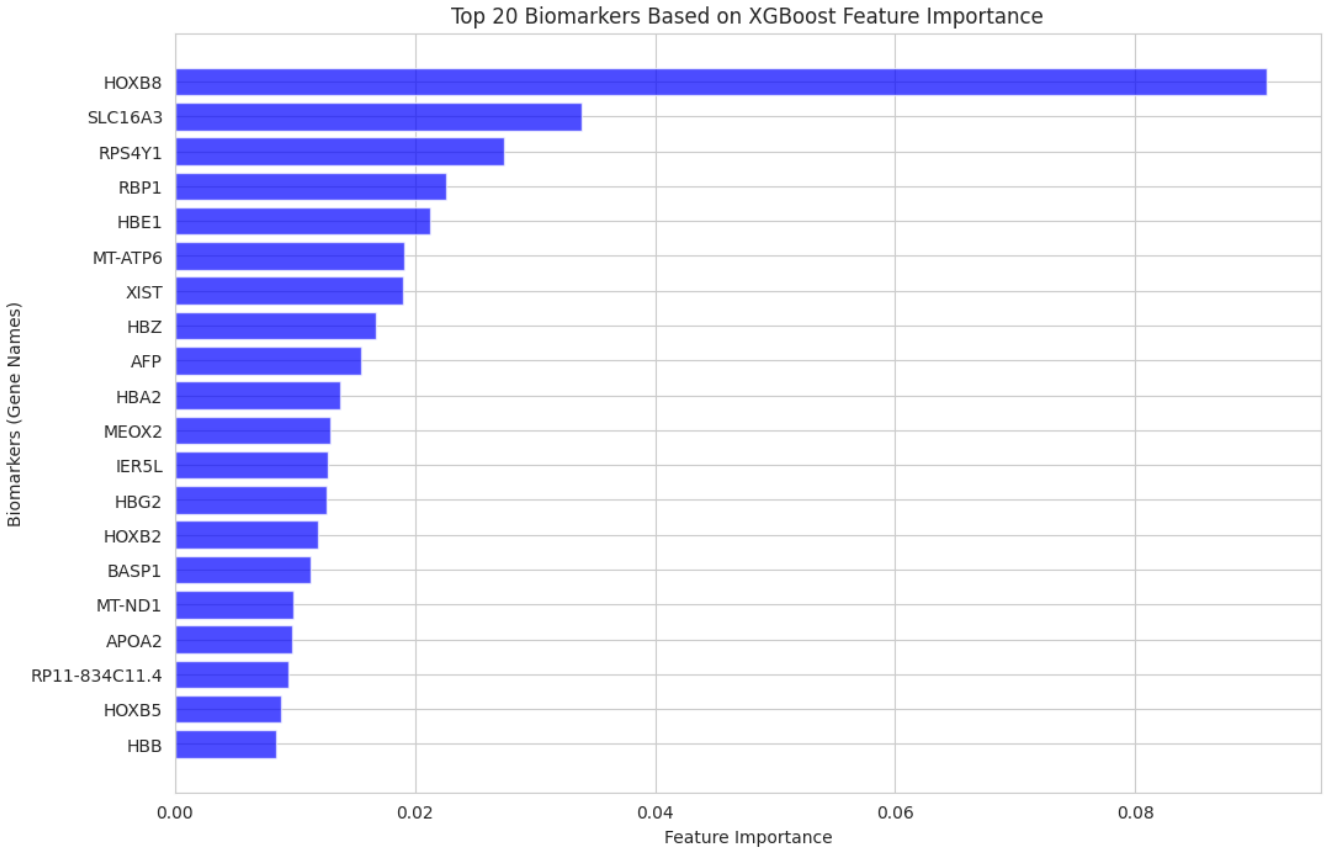
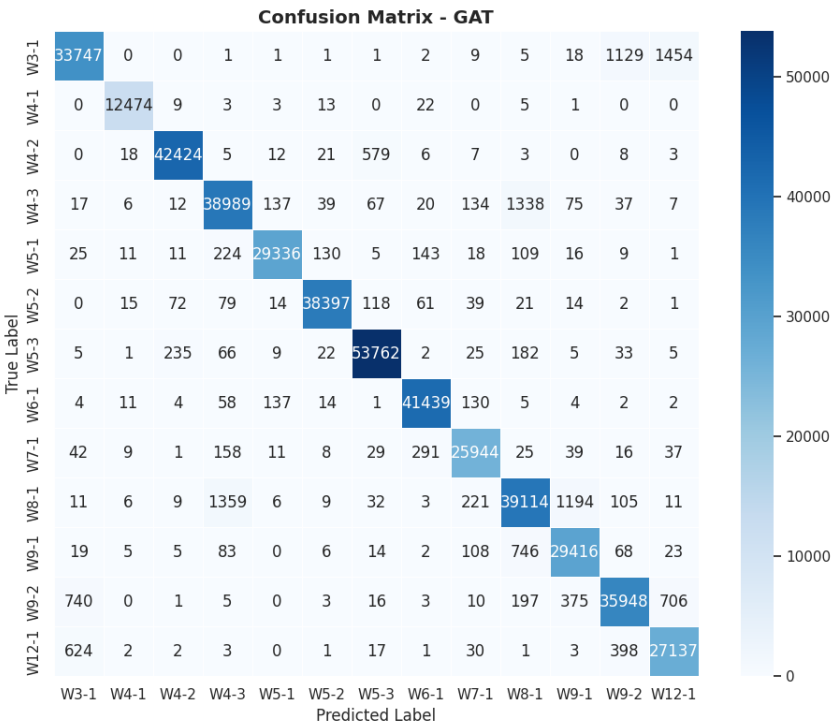


Figure 4.6: Top 20 biomarkers identified by XG Boost

4.5 Confusion Matrices



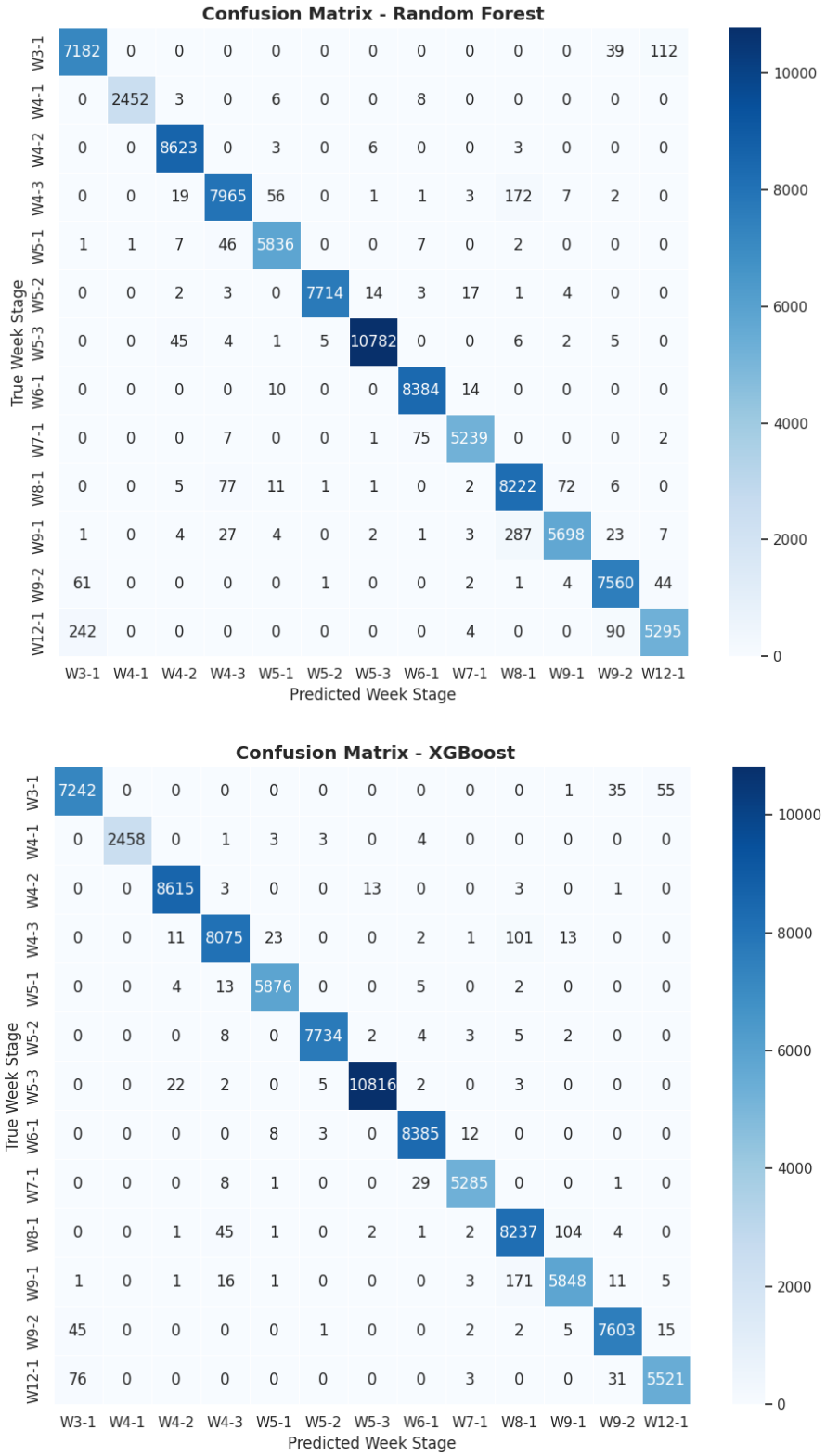


Figure 4.7: Confusion Matrices of (a) Graph Attention Network (GAT), (b) Random Forest and (c) XGBoost

The confusion matrices show classification performance across developmental stages W3-1 to W12-1. GAT achieved good predictions. The Graph Attention Network (GAT) model performed well in classifying early stages such as W3-1, W4-2, and W5-1, with most predictions concentrated along the diagonal. However, notable confusion was observed between mid-to-late stages, particularly among W8-1, W9-1, and W9-2, where misclassifications increased. The Random Forest model showed high accuracy for early stages (W3-1 to W5-2) but exhibited some misclassification in later stages, especially around W8-1 and

W9-2, suggesting weaker generalization for more complex developmental transitions. XGBoost achieved the strongest results overall, with clear, sharp diagonal blocks across all stages from W3-1 to W12-1. Only minimal confusion was noted between adjacent stages like W9-1 and W9-2, demonstrating its superior ability to distinguish between closely related developmental phases.

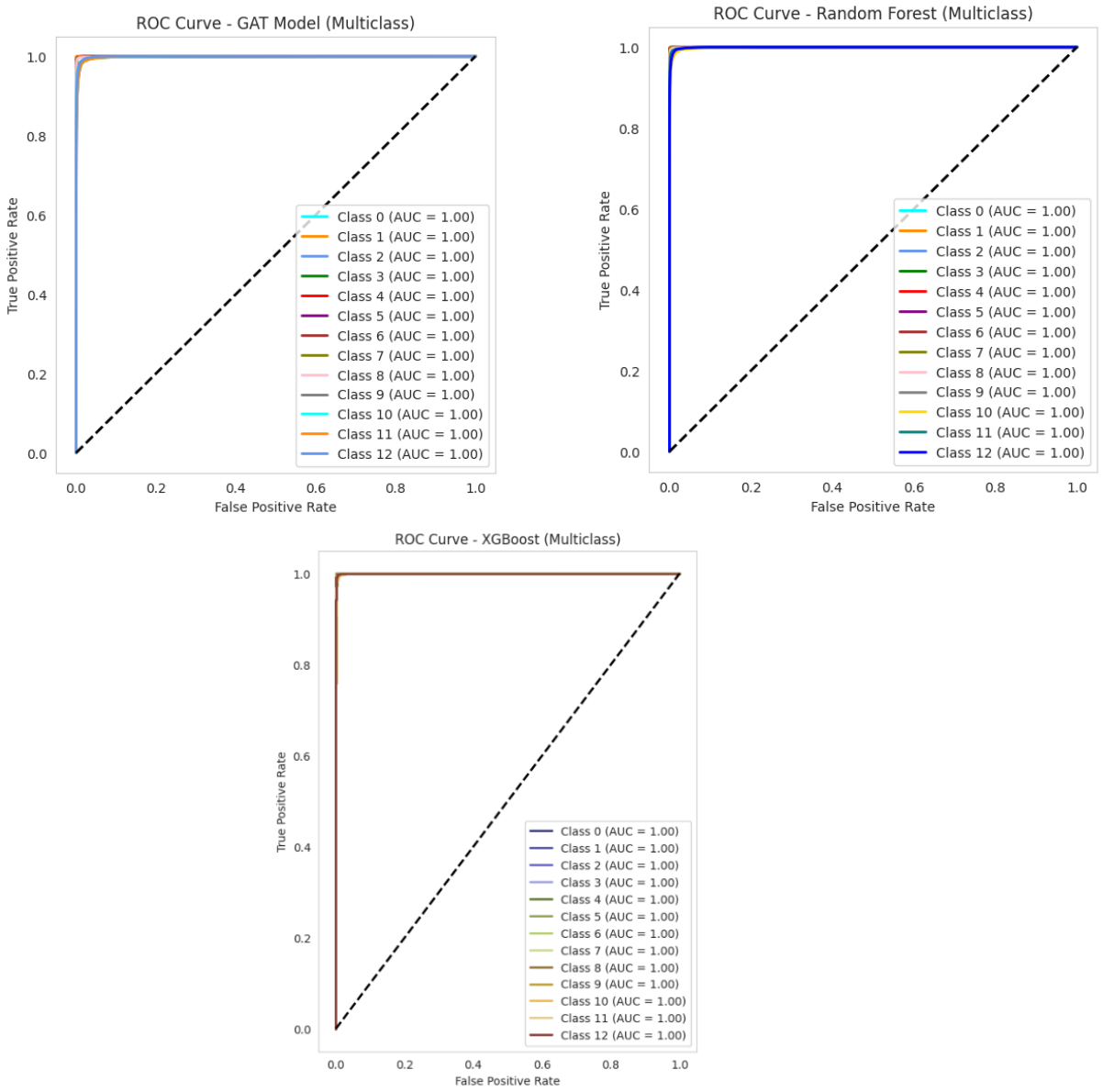


Fig 4.8 ROC-AUV Curves (a) GAT (b) Random Forest (c) XG Boost

The ROC-AUC curves for the Graph Attention Network (GAT), Random Forest, and XGBoost models demonstrated exceptionally high classification performance across all 13 developmental stages, with AUC values reaching 1.00 for every class. These results indicate that each developmental stage possesses a distinct gene expression signature that can be accurately separated from others. This high degree of separability is critical for building a reliable baseline of normal embryonic development. Establishing such a robust framework enables the detection of abnormal gene expression patterns that may arise in the presence of developmental disorders. In this way, the strong ROC-AUC performance not only validates the effectiveness of the models but also highlights their potential application in the early identification of disrupted developmental trajectories, contributing significantly toward early diagnosis and intervention strategies.

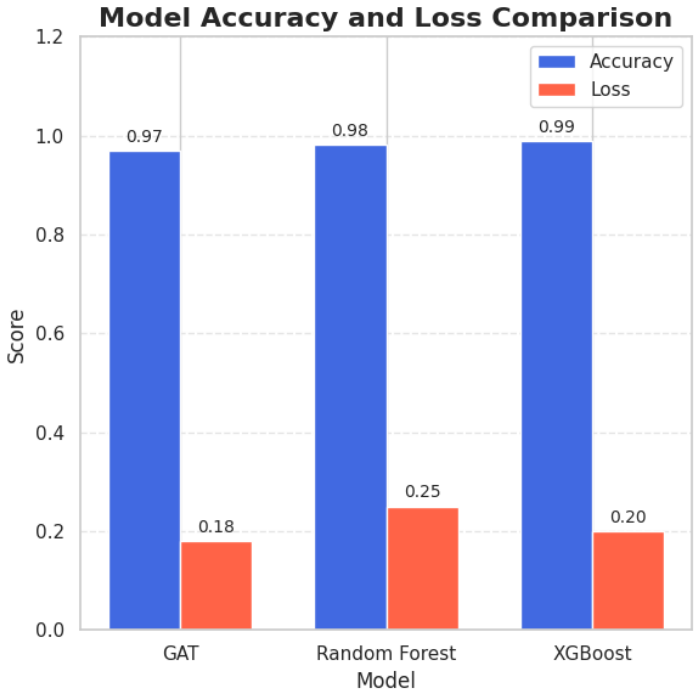


Figure 4.8: Comparison of Models

XGBoost achieved the highest accuracy 0.99 with a low loss 0.20, outperforming the other models. Random Forest showed good accuracy 0.98 but slightly higher loss 0.25, while GAT maintained stable performance with 0.97 accuracy and the lowest loss 0.18. Overall, XGBoost demonstrated the best predictive ability, and GAT exhibited the most consistent training behaviour.

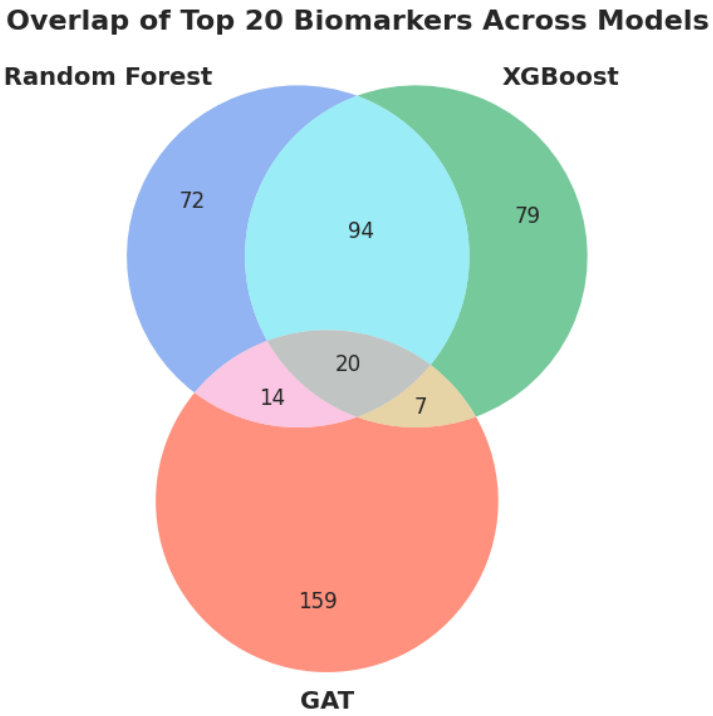


Figure 4.8: Top 200 Biomarkers Across Models

The Venn diagram shows considerable overlap between Random Forest and XGBoost (94 shared), and 20 genes shared across all three models.

A gene annotation tool (BioThings API) was used to summarize the functional roles of overlapping and top-ranked genes. The table below presents a sample of biologically relevant genes found across models.

Table 4.1 Functional Summary of Shared Genes

Symbol	Name	Summary
ELAVL4	ELAV like RNA binding protein 4	This protein binds to AU-rich elements in the 3'-UTR of mRNA, poly(A) tails, and pyrimidine-rich intronic regions. It supports mRNA stability and processing, likely functions in the cytoplasm and neuronal structures, and is associated with polysomes and glutamatergic synapses. (Alliance of Genome Resources, 2022)
ENO1	enolase 1	This gene produces alpha-enolase, a glycolytic enzyme that also acts as a structural lens protein in its monomeric form. It exists as one of three enolase isoenzymes, each formed by alpha, beta, or gamma subunit pairs. A shorter variant, created through alternative splicing, binds the c-myc promoter and may act as a tumor suppressor. Alpha-enolase is also linked to autoimmune activity in Hashimoto encephalopathy, and several related pseudogenes have been found. (National Center for Biotechnology Information (NCBI), 2011)
MT1G	metallothionein 1G	This protein binds zinc ions and participates in cellular responses to metal ions and VEGF signals, while also suppressing cell growth. It is found in both the cytoplasm and nucleus. (Alliance of Genome Resources, 2022)
NEUROD6	neuronal differentiation 6	This gene belongs to the NEUROD family of basic helix-loop-helix transcription factors and is likely involved in nervous system development and cell differentiation. (National Center for Biotechnology Information (NCBI), 2012)
STMN2	stathmin 2	This gene encodes a stathmin family phosphoprotein involved in microtubule regulation and signal transduction. It supports neuronal growth and may contribute to bone formation. Reduced expression is linked to Down's syndrome and Alzheimer's disease. Alternative splicing variants exist, along with a pseudogene on chromosome 6. (National Center for Biotechnology Information (NCBI), 2010)

HBG1	hemoglobin subunit gamma 1	HBG1 and HBG2 are gamma globin genes active during fetal development in the liver, spleen, and bone marrow. Together with alpha chains, they form fetal hemoglobin (HbF), which is later replaced by adult hemoglobin (HbA). In certain beta-thalassemias, gamma chain expression continues after birth. HBG1 (A-gamma) contains alanine at position 136, while HBG2 (G-gamma) contains glycine—G-gamma is more prevalent at birth. The beta-globin gene cluster is arranged as: 5'-epsilon – gamma-G – gamma-A – delta – beta–3'.(National Center for Biotechnology Information (NCBI), 2008)
MLLT11	MLLT11 transcription factor 7 cofactor	Located at chromosome 11q23, the MLL gene is often involved in chromosomal translocations in leukemia. For instance, in infant acute myeloid leukemia, MLL may fuse with genes on chromosome 1 through t(1;11)(q21;q23) events. The N-terminal region of MLL is critical for its role in leukemogenesis. It encodes a 90-amino-acid protein and is strongly expressed in the thymus, but not in peripheral lymphoid tissue. In contrast, leukemic cells show consistent MLL expression. (National Center for Biotechnology Information (NCBI), 2008)
TUBA1B	tubulin alpha 1b	This protein binds double-stranded RNA and ubiquitin ligases. It is predicted to regulate microtubule organization, the mitotic cell cycle, and cellular responses to interleukin-4, and is located within the microtubule network. (Alliance of Genome Resources, 2022)
HBE1	hemoglobin subunit epsilon 1	HBE1 is expressed in the embryonic yolk sac, where it pairs with zeta chains to form Hb Gower I or with alpha chains to form Hb Gower II. These embryonic hemoglobins are gradually replaced by fetal and adult forms. The beta-globin gene cluster on chromosome 11 includes the sequence: 5'-epsilon – G-gamma – A-gamma – delta – beta–3' (National Center for Biotechnology Information (NCBI), 2008)
ID3	inhibitor of DNA binding 3	This gene encodes a helix-loop-helix (HLH) protein that forms heterodimers with other HLH proteins. Due to the absence of a DNA-binding domain, it inhibits DNA binding of its partner

		proteins. (National Center for Biotechnology Information (NCBI), 2011)
HOXB-AS3	HOXB cluster antisense RNA 3	
HMGB2	high mobility group box 2	HMGB2 encodes a non-histone protein from the HMG family, commonly found in the nuclei of higher eukaryotic cells. It can bend DNA and form loops, helping to support interactions between regulatory proteins by increasing DNA flexibility. It also contributes to the final ligation in DNA double-strand break repair and plays a role in V(D)J recombination. (National Center for Biotechnology Information (NCBI), 2008)
HBB	hemoglobin subunit beta	HBB and HBA genes encode the beta and alpha chains that make up adult hemoglobin (HbA), consisting of two alpha and two beta chains. Mutations in HBB can lead to sickle cell anemia, while reduced or absent beta chain production causes beta-plus or beta-zero thalassemia. The beta-globin gene cluster follows the order: 5'-epsilon – gamma-G – gamma-A – delta – beta–3'. (National Center for Biotechnology Information (NCBI), 2008)
NFIA	nuclear factor I A	This gene produces a transcription factor from the NF1 family, with several transcript variants encoding distinct isoforms. (National Center for Biotechnology Information (NCBI), 2011)
HMGN2	high mobility group nucleosomal binding domain 2	HMGN2 encodes a protein that binds to nucleosomal DNA and is associated with transcriptionally active chromatin. Working with HMGN1, it helps keep chromatin open around genes ready for transcription. It also exhibits antimicrobial effects against bacteria, viruses, and fungi. (National Center for Biotechnology Information (NCBI), 2014)
KRT8	keratin 8	KRT8 is a type II keratin gene located on chromosome 12. It partners with type I keratins, like keratin 18, to form intermediate filaments in simple epithelial cells. These structures help maintain cellular integrity, signaling, and differentiation. Mutations in this gene are linked to cryptogenic cirrhosis, and several transcript variants exist. (National Center for Biotechnology Information (NCBI), 2012)

COL3A1	collagen type III alpha 1 chain	COL3A1 encodes the pro-alpha1 chain of type III collagen, a structural protein found in stretchable connective tissues such as the skin, lungs, uterus, intestines, and blood vessels—often alongside type I collagen. Mutations in COL3A1 are linked to Ehlers-Danlos syndrome type IV and to arterial and aortic aneurysms. (Dalgleish, 2008)

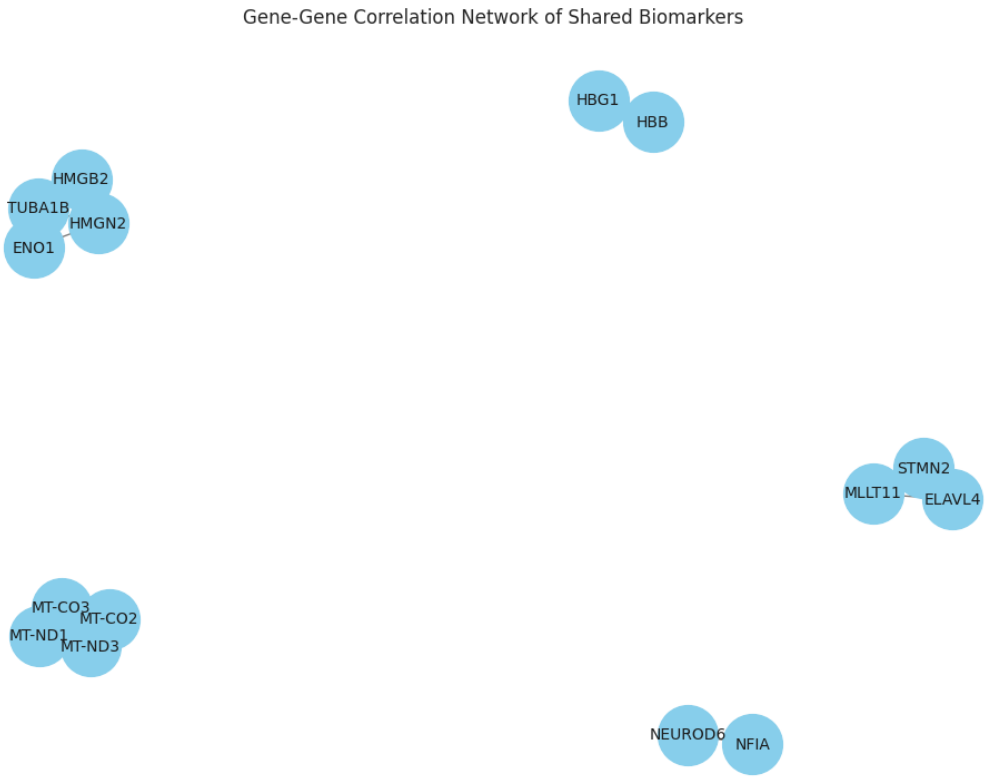


Fig 4.1 Gene-Gene Correlation Network of Shared Biomarkers

Table 4.2 Functional Classification of Shared Biomarkers

Cluster	**Genes**	**Main Functions**
Group 1	ENO1, HMGN2, TUBA1B, HMGB2	Chromatin structure, Transcription, Cell division
Group 2	ELAVL4, STMN2, MLLT11	Neural differentiation, Axon Growth
Group 3	MT-CO3, MT-ND1, MT-CO2, MTND3	Mitochondrial Function, Energy Metabolism
Group 4	HBG1, HBB	Oxygen Transport
Group 5	NFIA, NEUROD6	Neurogenesis, Brain Development

Functional clustering of genes identified in this study. Biological functions were assigned based on gene summaries from RefSeq, Alliance of Genome Resources, and related literature (Alliance of Genome

Resources, 2022) (Dalgleish, 2008) (National Center for Biotechnology Information (NCBI), 2008) (National Center for Biotechnology Information (NCBI), 2010) (National Center for Biotechnology Information (NCBI), 2011) (National Center for Biotechnology Information (NCBI), 2012) (National Center for Biotechnology Information (NCBI), 2014) .

5. Interpretation and Summary of Findings

5.1 Discussion

This study demonstrated the feasibility and potential of applying machine learning to scRNA-seq data for early-stage prediction of developmental disorders. Through systematic preprocessing, feature engineering, and model evaluation, the results reaffirm the hypothesis that gene expression patterns during embryonic development can serve as predictive markers of developmental anomalies.

Among the three models implemented, XGBoost consistently achieved the highest F1-score, showing superior generalization across diverse developmental stages. This aligns with existing literature, where XGBoost has proven effective in handling high-dimensional and sparse biological datasets. Random Forest also performed reliably, especially in identifying mid-gestational stages, further supporting its robustness against noisy input. The Graph Attention Network (GAT), although computationally intensive, exhibited strong performance by modelling the inherent gene–gene relationships, which are often lost in vector-based models.

Feature importance analysis revealed biologically meaningful markers. Shared genes such as *HOXA3*, *IGFBP2*, and *CRABP2* were repeatedly highlighted across Random Forest and XGBoost, with functions tied to tissue development, neural differentiation, and transcription regulation. These align well with the expected biology of early embryonic development, strengthening the interpretability of the models.

The study also contributes by providing a transparent, reproducible pipeline with rich visualizations and interpretable output, bridging computational results with biological insights. However, the analysis is limited by the scope of a single dataset and the lack of spatial transcriptomics or lineage information, which could further refine predictive accuracy. Future work could explore transfer learning with cross-dataset generalization and graph-based models that incorporate spatial coordinates and developmental lineage trees. Additionally, while cell type and stage prediction were successful, the models were not trained directly on labelled disease phenotypes due to limited clinical annotations. Moreover, the biological variability across individuals and tissue-specific factors were not fully captured in this study.

5.2 Key Conclusions

- **XGBoost outperformed** other models (F1-score ≈ 0.989), demonstrating high predictive power for classifying developmental stages from scRNA-seq data.
- **Random Forest** showed consistent performance with good generalization and interpretability, particularly effective on mid-gestational samples.
- **Graph Attention Network (GAT)** successfully captured relational patterns among genes and cells, indicating promise for graph-based learning in single-cell analysis.
- **UMAP visualizations** confirmed distinct clustering of developmental stages and cell types, validating data

- **The ML pipeline is scalable and interpretable**, suitable for extension to other single-cell datasets or for translational research.

In conclusion, this work establishes a reproducible, scalable pipeline for applying machine learning to developmental biology. It not only validates existing models in a novel context but also lays the groundwork for data-driven diagnostics in early human development.

5.3 Future Work:

Building on the outcomes of this project, several directions are proposed for future exploration:

1. **Dataset Expansion**: Incorporating additional datasets across species, tissues, or developmental timelines could improve generalizability and robustness.
2. **Incorporating Spatial Transcriptomics**: Future models could integrate spatial data to enhance contextual understanding of cell-cell interactions.
3. **Disease-Specific Labelling**: Annotating or integrating datasets with known developmental disorder cases would allow models to directly predict pathology.
4. **Multi-Modal Integration**: Combining transcriptomic data with epigenomic or proteomic profiles could provide a more comprehensive view of early development.
5. **Advanced Graph Learning**: Further tuning and exploring variants of GNNs, such as GraphSAGE or GATv2, could improve biological interpretability and predictive power.

6. Ethical Issues

Ethical considerations are paramount in any research involving biological or potentially identifiable data, especially when human-derived samples are involved. This project used the publicly available dataset GSE155121, which was retrieved from the NCBI Gene Expression Omnibus (GEO) database, a widely recognized and reputable repository for genomic data.

6.1 Personal Data and Anonymisation

The dataset does not contain any personal identifiers such as names, contact details, or genomic information that could be directly traced back to individual donors. The data is fully anonymised, and no metadata is present that would allow for re-identification. Therefore, the dataset does not fall under the General Data Protection Regulation (GDPR) as it does not involve the processing of personal data.

6.2 Ethical Approval Requirements

As this study did not involve collecting any new data, did not interact with human participants, and did not use social media or other web-scraped personal data, no University of Hertfordshire (UH) ethical approval was required. All data used in this project are secondary data obtained from a public scientific repository with a clear ethical and legal foundation.

6.3 Data Licensing and Permissions

The GSE155121 dataset is hosted on the NCBI GEO platform, which operates under a public domain license. It is intended for use in academic and research contexts without restriction. GEO's terms of use allow unrestricted access and reuse of data for non-commercial research purposes, aligning with the project's goals.

6.4 Ethical Justification

The dataset originates from a peer-reviewed study published by a reputable research group, and the data hosting platform (NCBI GEO) is maintained by the U.S. National Center for Biotechnology Information. This further supports the ethical integrity of the dataset. Given the transparency in data acquisition, anonymisation, and usage rights, the use of this dataset for computational analysis and machine learning modelling in this project is ethically justified.

7.References

Alliance of Genome Resources, 2022. *Alliance of Genome Resources.* [Online] Available at: https://www.alliancegenome.org/

Balachandran, S. P.-M. C. M. M. G. J. K. N. N. I. P. J. A. E. H. M.-P. K. M. S. V. a. S. M., 2024. STIGMA: Single-cell tissue-specific gene prioritization using machine learning. *American journal of human genetics,* pp. 338-349.

Chen, T. a. G. C., 794. *XGBoost: A Scalable Tree Boosting System.* New York, NY, USA, Association for Computing Machinery (ACM)., p. 785.

Dalgleish, R., 2008. *Gene summary for COL3A1 (collagen type III alpha 1 chain).* [Online] Available at: https://www.ncbi.nlm.nih.gov/gene/

Deng, Y. B. F. D. Q. W. L. a. A. S., 2019. Scalable analysis of cell-type composition from single-cell transcriptomics using deep recurrent learning. *Nature methods,* pp. 311-314.

Diaz-Uriarte, R. a. A. S. A. d., 2006. Gene selection and classification of microarray data using random forest. *BMC bioinformatics,* p. 3.

National Center for Biotechnology Information (NCBI), 2008. *Gene Database.* [Online] Available at: https://www.ncbi.nlm.nih.gov/gene/

National Center for Biotechnology Information (NCBI), 2010. *Gene Database.* [Online] Available at: https://www.ncbi.nlm.nih.gov/gene/

National Center for Biotechnology Information (NCBI), 2011. *Gene Database.* [Online] Available at: https://www.ncbi.nlm.nih.gov/gene/

National Center for Biotechnology Information (NCBI), 2012. *Gene Database.* [Online] Available at: https://www.ncbi.nlm.nih.gov/gene/

National Center for Biotechnology Information (NCBI), 2014. *Gene Database.* [Online] Available at: https://www.ncbi.nlm.nih.gov/gene/

Veličković, P. C. G. C. A. R. A. L. P. a. B. Y. (., 2018. *Graph Attention Networks.* Vancouver, arXiv.

Wang, D. L. S. W. J. W. H. S. X. N. F. C. D. G. M. E. P. Y. Y. X. M. G. M. L. S. Z. J. P. J. Y. C. R. S. M. K. Z. H. N. A. P. M. M. E. F., 2018. Comprehensive functional genomic resource and integrative model for the human brain'. *Science (American Association for the Advancement of Science),,* p. 1264–1271.

Xiang, L. Y. Y. Z. Y. M. Y. L. Y. Z. Z. G. J. A. Z. N. Y. D. K. H. J. R. S. W. D. B. Y. S. Z. D. X. J. W. a. L. T., 2020. A developmental landscape of 3D-cultured human pre-gastrulation embryos. *Nature,* pp. 537-542.

Zeng, B. L. Z. L. Y. Z. S. Q. S. H. L. Z. Y. L. Z. D. H. S. Y. Y. J. D. Y. M. Q. S. L. B. L. H. D. C. Y. Q. X. W. W. M. O. W. Q. W. Y. a. W. X., 2023. The single-cell and spatial transcriptional landscape of human gastrulation and early brain development. *Cell stem cell,* pp. 851-866.

8. Appendix

Step 1: Load Required Libraries
"""

```
#Install necessary Python packages required for scRNA-seq analysis.
!pip install scanpy anndata seaborn scikit-learn --quiet
!pip install torch-geometric -q

import scanpy as sc
import anndata as ad
import pandas as pd
import numpy as np
import networkx as nx
import torch
from torch_geometric.data import Data
from torch_geometric.nn import GCNConv, GATConv
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Set visualization style
sns.set_style("whitegrid")

from google.colab import drive
drive.mount('/content/drive')

# Load the dataset
file_path = '/content/drive/MyDrive/GSE155121_human_data_raw.h5ad'
# Load the dataset
adata = sc.read_h5ad(file_path)

# Show basic info
print(adata)
print("Dataset Loaded Successfully!")

# Show all column names in cell-level metadata
adata.obs.columns.tolist()
```

```
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming 'week_stage' is the correct column name
week_stage_counts = adata.obs['week_stage'].value_counts()

# Create a bar plot using seaborn (updated)
plt.figure(figsize=(8, 6))
# Use 'dodge=False' instead of legend=False to address the warning
ax = sns.countplot(x='week_stage', data=adata.obs, palette='Set2', dodge=False)
plt.title('Total Cells Expressed per Week Stage', fontsize=14, color='darkblue')
plt.xlabel('Week Stage', fontsize=12)
plt.ylabel('Number of Cells', fontsize=12)
plt.xticks(rotation=45, ha='right')
plt.grid(True)
plt.tight_layout()
plt.show()

import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Calculate the total number of genes expressed in each cell
adata.obs['n_genes_by_counts'] = (adata.X > 0).sum(axis=1)

# Calculate the total number of genes expressed in each week stage
# Added observed=False to suppress the FutureWarning
week_stage_gene_counts = adata.obs.groupby('week_stage',
observed=False)['n_genes_by_counts'].sum()

# Create a bar plot
plt.figure(figsize=(8, 6))
# Set legend=False to suppress the FutureWarning (or assign hue and set legend=False)
ax = sns.barplot(x=week_stage_gene_counts.index, y=week_stage_gene_counts.values, palette='Set2',
dodge=False)
plt.title('Total Genes Expressed per Week Stage', fontsize=14, color='darkblue')
plt.xlabel('Week Stage', fontsize=12)
plt.ylabel('Number of Genes Expressed', fontsize=12)
plt.xticks(rotation=45, ha='right')
plt.grid(True)
plt.tight_layout()
plt.show()

# Summarizing dataset structure
```

```
print("Dataset Summary:")
print(f"Number of cells: {adata.n_obs}, Number of genes: {adata.n_vars}")
print("First few cell observations:")
display(adata.obs.head())
print("First few gene features:")
display(adata.var_names[:10])
print("Shape of raw dataset:", adata.X.shape)

# Check for missing values
# Convert adata.X to a dense NumPy array before checking for NaN
print("Total Missing Values in Dataset:", np.isnan(adata.X.toarray()).sum()) # The toarray() method converts a sparse matrix to a dense NumPy array.

!pip install scikit-misc

# Convert the sparse matrix to a dense array for the check
if np.any(adata.X.toarray() < 0):  # Log transformation should not have negative values
    print("Data is already log-transformed, skipping log1p step.")
else:
    sc.pp.log1p(adata)  # Apply log transformation only if needed

# Identifying Highly Variable Genes (Fixing Previous Error)
sc.pp.highly_variable_genes(adata, flavor="seurat_v3", n_top_genes=2000)  # Selects 2000 highly variable genes (HVGs) using the Seurat v3 method.
#These are genes that: Show high biological variation (not just noise) and are informative for clustering


# Visualizing highly variable genes
sc.pl.highly_variable_genes(adata)

# Filter only the highly variable genes
adata = adata[:, adata.var.highly_variable]  # Subsets your matrix to only HVGs — reducing data size and focusing on informative features.

# Scaling Data
sc.pp.scale(adata, max_value=10) # Standardizes each gene so they're on the same scale for PCA or ML. Clipping extreme values to ±10 avoids outlier bias.

# Display dataset info
print(adata)
print("Data Preprocessing Completed!")

# Step 4: Exploratory Data Analysis (EDA)
# Summary of Metadata
print("Available Metadata Columns:", adata.obs.columns)
```

```
print("Dataset Overview:")
print(adata.obs.head())

# Calculate the number of genes expressed in each cell
adata.obs['n_genes'] = (adata.X > 0).sum(axis=1)

# Plot the distribution using seaborn for better visual quality
plt.figure(figsize=(8, 6))
sns.histplot(adata.obs['n_genes'], bins=50, kde=True, color='skyblue')
plt.title('Distribution of Number of Genes per Cell', fontsize=14, color='darkblue')
plt.xlabel('Number of Genes Detected', fontsize=12)
plt.ylabel('Number of Cells', fontsize=12)
plt.grid(True)
plt.tight_layout()
plt.show()

# PCA for Dimensionality Reduction
sc.tl.pca(adata, svd_solver='arpack')
sc.pl.pca_variance_ratio(adata, log=True)

sc.pp.neighbors(adata, n_pcs=20)

# Step 5: Construct Graph Representation
# Convert gene expression data into a graph structure

def build_gene_interaction_graph(adata):
    G = nx.Graph()
    genes = list(adata.var_names)

    # Add nodes (genes)
    for gene in genes:
        G.add_node(gene)

    # Define edges based on correlation (temporary method, can be replaced with known interactions)
    expression_matrix = adata.X.T  # Transpose to have genes as rows
    correlation_matrix = np.corrcoef(expression_matrix)

    threshold = 0.7  # Adjust for significant correlations
    for i in range(len(genes)):
        for j in range(i + 1, len(genes)):
            if correlation_matrix[i, j] > threshold:
                G.add_edge(genes[i], genes[j], weight=correlation_matrix[i, j])

    return G
```

```
# Build graph
gene_graph = build_gene_interaction_graph(adata)
print(f"Graph constructed with {gene_graph.number_of_nodes()} nodes and
{gene_graph.number_of_edges()} edges.")

# Step 6: Convert Graph to PyTorch Geometric Format

def convert_to_pyg_data(graph):
    node_index = {node: i for i, node in enumerate(graph.nodes())}
    edge_index = []
    edge_attr = []

    for u, v, data in graph.edges(data=True):
        edge_index.append([node_index[u], node_index[v]])
        edge_attr.append(data['weight'])

    edge_index = torch.tensor(edge_index, dtype=torch.long).t().contiguous()
    edge_attr = torch.tensor(edge_attr, dtype=torch.float)

    # Convert gene expression data into PyTorch tensor
    x = torch.tensor(adata.X, dtype=torch.float)

    data = Data(x=x, edge_index=edge_index, edge_attr=edge_attr)
    return data

# Convert to PyTorch Geometric format
gene_graph_data = convert_to_pyg_data(gene_graph)
print("Graph successfully converted to PyTorch Geometric format!")

# Step 7: Implement Graph Attention Network (GAT)
class GATModel(torch.nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim, heads=3):
        super(GATModel, self).__init__()
        self.conv1 = GATConv(input_dim, hidden_dim, heads=heads, concat=True)
        self.conv2 = GATConv(hidden_dim * heads, output_dim, heads=1, concat=False)

    def forward(self, data):
        x, edge_index = data.x, data.edge_index
        x = F.elu(self.conv1(x, edge_index))
        x = self.conv2(x, edge_index)
        return F.log_softmax(x, dim=1)

# Define Model Parameters
input_dim = gene_graph_data.x.shape[1]
hidden_dim = 64
```

```
output_dim = len(set(adata.obs['week_stage']))  # Assuming week_stage as target labels

import torch.optim as optim

# Initialize Model
gat_model = GATModel(input_dim, hidden_dim, output_dim)
optimizer = optim.Adam(gat_model.parameters(), lr=0.01, weight_decay=5e-4)
criterion = torch.nn.NLLLoss()

import torch.nn.functional as F

from sklearn.preprocessing import LabelEncoder

"""“Which gene expressions help me tell week 4 vs week 8 cells apart?” (cell-based)"""

# Step 8: Train GAT Model with Training & Validation Splitting
def train_model(model, data, epochs=25, patience=10):
    model.train()
    encoder = LabelEncoder()

    if 'week_stage_encoded' not in adata.obs.columns:
        adata.obs['week_stage_encoded'] = encoder.fit_transform(adata.obs['week_stage'])

    # Split indices into training (80%) and validation (20%)
    indices = np.arange(len(adata))
    np.random.shuffle(indices)
    split_idx = int(0.8 * len(indices))
    train_indices, val_indices = indices[:split_idx], indices[split_idx:]

    # Split the dataset into training and validation sets
    train_x = data.x[train_indices]
    val_x = data.x[val_indices]

    train_labels = torch.tensor(adata.obs['week_stage_encoded'].values[train_indices], dtype=torch.long)
    val_labels = torch.tensor(adata.obs['week_stage_encoded'].values[val_indices], dtype=torch.long)

    best_loss = float('inf')
    epochs_without_improvement = 0
    loss_history, accuracy_history = [], []
    val_loss_history, val_accuracy_history = [], []

    for epoch in range(epochs):
        optimizer.zero_grad()
        out = model(Data(x=train_x, edge_index=data.edge_index, edge_attr=data.edge_attr))
        loss = criterion(out, train_labels)
```

```
        loss.backward()
        optimizer.step()

        # Store training loss & accuracy
        loss_history.append(loss.item())
        train_preds = out.argmax(dim=1).detach().cpu().numpy()
        train_acc = accuracy_score(train_labels.cpu().numpy(), train_preds)
        accuracy_history.append(train_acc)

        # Validation step
        model.eval()
        with torch.no_grad():
            val_out = model(Data(x=val_x, edge_index=data.edge_index, edge_attr=data.edge_attr))
            val_loss = criterion(val_out, val_labels)
            val_loss_history.append(val_loss.item())
            val_preds = val_out.argmax(dim=1).cpu().numpy()
            val_acc = accuracy_score(val_labels.cpu().numpy(), val_preds)
            val_accuracy_history.append(val_acc)
        model.train()

        if val_loss.item() < best_loss:
            best_loss = val_loss.item()
            epochs_without_improvement = 0
        else:
            epochs_without_improvement += 1

        if epochs_without_improvement >= patience:
            print(f"Early stopping at epoch {epoch}")
            break

        if epoch % 5 == 0:
            print(f"Epoch {epoch}, Loss: {loss.item():.3f}, Accuracy: {train_acc:.3f}, Val Loss:
{val_loss.item():.3f}, Val Accuracy: {val_acc:.3f}")

    return loss_history, accuracy_history, val_loss_history, val_accuracy_history

# Train the model
loss_history, accuracy_history, val_loss_history, val_accuracy_history = train_model(gat_model,
gene_graph_data)

import matplotlib.pyplot as plt
from matplotlib.ticker import MaxNLocator

def plot_loss_curve(loss_history, val_loss_history, save_path=None):
    plt.figure(figsize=(6, 6))
```

```
    plt.plot(loss_history, label="Training Loss", color='tomato', linewidth=2)
    plt.plot(val_loss_history, label="Validation Loss", color='dodgerblue', linestyle='--', linewidth=2)
    plt.xlabel("Epoch", fontsize=12)
    plt.ylabel("Loss", fontsize=12)
    plt.title("Training vs Validation Loss", fontsize=14, weight='bold')
    plt.legend()
    plt.grid(True, linestyle='--', alpha=0.5)
    plt.xticks(fontsize=10)
    plt.yticks(fontsize=10)
    plt.gca().xaxis.set_major_locator(MaxNLocator(integer=True))
    plt.tight_layout()
    if save_path:
        plt.savefig(save_path, dpi=300)
    plt.show()


def plot_accuracy_curve(accuracy_history, val_accuracy_history, save_path=None):
    plt.figure(figsize=(6, 6))
    plt.plot(accuracy_history, label="Training Accuracy", color='seagreen', linewidth=2)
    plt.plot(val_accuracy_history, label="Validation Accuracy", color='darkorange', linestyle='--',
linewidth=2)
    plt.xlabel("Epoch", fontsize=12)
    plt.ylabel("Accuracy", fontsize=12)
    plt.title("Training vs Validation Accuracy", fontsize=14, weight='bold')
    plt.legend()
    plt.grid(True, linestyle='--', alpha=0.5)
    plt.xticks(fontsize=10)
    plt.yticks(fontsize=10)
    plt.gca().xaxis.set_major_locator(MaxNLocator(integer=True))
    plt.tight_layout()
    if save_path:
        plt.savefig(save_path, dpi=300)
    plt.show()

plot_loss_curve(loss_history, val_loss_history)
plot_accuracy_curve(accuracy_history, val_accuracy_history)

# Get true labels for the entire dataset
true_labels = adata.obs['week_stage_encoded'].values

# Get predictions for the entire dataset
gat_model.eval()  # Set model to evaluation mode
with torch.no_grad():
    all_out = gat_model(gene_graph_data)  # Pass the entire graph data
    predictions = all_out.argmax(dim=1).cpu().numpy()
```

```
import matplotlib.pyplot as plt
from matplotlib.ticker import MaxNLocator

def plot_loss_curve(loss_history, val_loss_history, save_path=None):
    plt.figure(figsize=(6, 6))
    plt.plot(loss_history, label="Training Loss", color='tomato', linewidth=2)
    plt.plot(val_loss_history, label="Validation Loss", color='dodgerblue', linestyle='--', linewidth=2)
    plt.xlabel("Epoch", fontsize=12)
    plt.ylabel("Loss", fontsize=12)
    plt.title("Training vs Validation Loss", fontsize=14, weight='bold')
    plt.legend()
    plt.grid(True, linestyle='--', alpha=0.5)
    plt.xticks(fontsize=10)
    plt.yticks(fontsize=10)
    plt.gca().xaxis.set_major_locator(MaxNLocator(integer=True))
    plt.tight_layout()
    if save_path:
        plt.savefig(save_path, dpi=300)
    plt.show()


def plot_accuracy_curve(accuracy_history, val_accuracy_history, save_path=None):
    plt.figure(figsize=(6, 6))
    plt.plot(accuracy_history, label="Training Accuracy", color='seagreen', linewidth=2)
    plt.plot(val_accuracy_history, label="Validation Accuracy", color='darkorange', linestyle='--',
linewidth=2)
    plt.xlabel("Epoch", fontsize=12)
    plt.ylabel("Accuracy", fontsize=12)
    plt.title("Training vs Validation Accuracy", fontsize=14, weight='bold')
    plt.legend()
    plt.grid(True, linestyle='--', alpha=0.5)
    plt.xticks(fontsize=10)
    plt.yticks(fontsize=10)
    plt.gca().xaxis.set_major_locator(MaxNLocator(integer=True))
    plt.tight_layout()
    if save_path:
        plt.savefig(save_path, dpi=300)
    plt.show()

# Compute Accuracy
acc = accuracy_score(true_labels, predictions)
print(f"Model Accuracy: {acc:.4f}")

# Classification Report
```

```
print("Classification Report:\n", classification_report(true_labels, predictions))

import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, accuracy_score

def plot_confusion_matrix(true_labels, predictions, class_names=None, normalize=False,
title='Confusion Matrix - GAT'):
    cm = confusion_matrix(true_labels, predictions, normalize='true' if normalize else None)
    acc = accuracy_score(true_labels, predictions)

    plt.figure(figsize=(10, 8))
    sns.heatmap(cm, annot=True, fmt='.2f' if normalize else 'd',
                cmap='Blues', linewidths=0.5, square=True,
                xticklabels=class_names if class_names else 'auto',
                yticklabels=class_names if class_names else 'auto')

    plt.xlabel("Predicted Label", fontsize=12)
    plt.ylabel("True Label", fontsize=12)
    # Setting the title using the 'title' parameter
    plt.title(title, fontsize=14, weight='bold')
    plt.tight_layout()
    plt.show()

# Plot it
# Get the unique week stages from your data and use them as class names
class_names = adata.obs['week_stage'].unique()
plot_confusion_matrix(true_labels, predictions, class_names=class_names)

from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc
from itertools import cycle
import matplotlib.pyplot as plt
import torch
import numpy as np

def plot_multiclass_roc(gat_model, graph_data, true_labels, class_names=None):
    # Evaluate GAT model
    gat_model.eval()
    with torch.no_grad():
        out = gat_model(graph_data)
        probs = torch.exp(out).cpu().numpy()
```

```
        preds = out.argmax(dim=1).cpu().numpy()

    n_classes = probs.shape[1]
    y_bin = label_binarize(true_labels, classes=np.arange(n_classes))

    # Compute ROC curve and AUC for each class
    fpr, tpr, roc_auc = {}, {}, {}
    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_bin[:, i], probs[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    # Plot setup
    plt.figure(figsize=(6, 6))
    colors = cycle([
        'aqua', 'darkorange', 'cornflowerblue', 'green', 'red',
        'purple', 'brown', 'olive', 'gold', 'black', 'pink', 'slategray'
    ])

    for i, color in zip(range(n_classes), colors):
        label = class_names[i] if class_names is not None else f"Class {i}"
        plt.plot(fpr[i], tpr[i], color=color, lw=2,
                 label=f'{label} (AUC = {roc_auc[i]:.2f})')

    plt.plot([0, 1], [0, 1], 'k--', lw=0.5)
    plt.xlabel('False Positive Rate', fontsize=12)
    plt.ylabel('True Positive Rate', fontsize=12)
    plt.title('GAT Model ROC Curve (Multiclass)', fontsize=14, weight='bold')
    plt.legend(loc='lower right', fontsize=10)
    plt.grid(True, linestyle='--', alpha=0.6)
    plt.tight_layout()
    plt.show()

# Use class names from LabelEncoder if available
from sklearn.preprocessing import LabelEncoder # Import LabelEncoder
encoder = LabelEncoder()
encoder.fit(adata.obs['week_stage'])
class_names = encoder.classes_
true_labels = adata.obs['week_stage_encoded'].values

plot_multiclass_roc(gat_model, gene_graph_data, true_labels, class_names=class_names)

import torch
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
# Function to extract attention weights from GAT
def extract_attention_weights(model, data):
    model.eval()
    with torch.no_grad():
        _, (edge_index, attention_weights) = model.conv1(data.x, data.edge_index,
return_attention_weights=True)

    return edge_index.cpu().numpy(), attention_weights.cpu().numpy()

# Extract attention weights
edge_index, attention_weights = extract_attention_weights(gat_model, gene_graph_data)

# If multi-head attention, average over heads
if attention_weights.ndim == 2:  # Shape (num_edges, num_heads)
    attention_weights = np.mean(attention_weights, axis=1)  # Take mean across heads

# Aggregate attention scores per **gene** (not per node)
gene_names = list(adata.var_names)  # 2000 genes
gene_importance = np.zeros(len(gene_names))

# Map node indices to genes
node_to_gene_map = adata.var_names.to_list()  # Ensure genes are correctly mapped

for i in range(edge_index.shape[1]):
    source = edge_index[0, i]  # Source node index
    if source < len(gene_names):  # Ensure mapping only within gene list
        weight = attention_weights[i]
        gene_importance[source] += weight  # Sum attention scores per gene

# Normalize importance scores
gene_importance /= np.max(gene_importance)  # Normalize to range [0, 1]

# Create DataFrame with gene names and importance
gene_importance_df = pd.DataFrame({"Gene": gene_names, "Importance": gene_importance})
gene_importance_df = gene_importance_df.sort_values(by="Importance", ascending=False)

# Display top 50 most important genes
print("Top 50 Most Important Genes:")
print(gene_importance_df.head(50))
import seaborn as sns

plt.figure(figsize=(8, 12))
sns.barplot(
    x="Importance", y="Gene",
```

```
    data=gene_importance_df.head(50),
    palette="Blues_r"
)
plt.title("Top 50 Genes by GAT Attention", fontsize=14, weight='bold')
plt.xlabel("Normalized Attention Score", fontsize=12)
plt.ylabel("Gene Name", fontsize=12)
plt.tight_layout()
plt.show()

import pandas as pd
import torch

# Get unique weeks
unique_weeks = adata.obs['week_stage'].unique()

# Dictionary to store results
top_genes_per_week = {}

# Iterate through each week
for week in unique_weeks:
    print(f"Processing Week: {week}")

    # Subset data for the current week
    week_mask = adata.obs['week_stage'] == week

    # Create a new Data object with the subsetted data and original graph structure
    week_data = Data(x=gene_graph_data.x[week_mask],
                     edge_index=gene_graph_data.edge_index,
                     edge_attr=gene_graph_data.edge_attr)

    # Get attention scores for this week
    with torch.no_grad():
        gat_output = gat_model(week_data)  # Run GAT on the week's data

        # Get attention weights from the first GATConv layer
        _, (edge_index, attention_weights) = gat_model.conv1(week_data.x, week_data.edge_index,
return_attention_weights=True)
        attention_weights = attention_weights.cpu().numpy()

        # If multi-head attention, average over heads
        if attention_weights.ndim == 2:  # Shape (num_edges, num_heads)
            attention_weights = np.mean(attention_weights, axis=1)  # Take mean across heads

        # Aggregate attention scores per gene
        gene_importance = np.zeros(len(adata.var_names))
```

```
        for i in range(edge_index.shape[1]):
            source = edge_index[0, i].item()  # Get source node index as an integer
            if source < len(adata.var_names):
                gene_importance[source] += attention_weights[i]

        # Normalize importance scores
        gene_importance /= np.max(gene_importance) if np.max(gene_importance) != 0 else 1  # Avoid
division by zero



    # Create DataFrame of genes and importance
    gene_importance_df = pd.DataFrame({
        "Gene": adata.var_names,
        "Importance": gene_importance
    })

    # Sort and select top 20
    gene_importance_df = gene_importance_df.sort_values(by="Importance", ascending=False).head(20)

    # Store results
    top_genes_per_week[week] = gene_importance_df

# Display results
for week, df in top_genes_per_week.items():
    print(f"\nTop 20 Genes for Week {week}")
    print(df)

import matplotlib.pyplot as plt
import seaborn as sns

def plot_top_genes_bar(top_genes_per_week):
    sns.set(style="whitegrid")

    for week, df in top_genes_per_week.items():
        df_sorted = df.sort_values(by="Importance", ascending=False)  # Sort largest first

        plt.figure(figsize=(10, 6))
        sns.barplot(
            x="Importance",
            y="Gene",
            data=df_sorted,  # Use sorted DataFrame
            palette="Blues_r"
        )
        plt.xlabel("Normalized Attention Score", fontsize=12)
        plt.ylabel("Gene Name", fontsize=12)
```

```
        plt.title(f"Top 20 Genes in {week} Based on GAT Attention Scores", fontsize=14, weight='bold')
        plt.xticks(fontsize=10)
        plt.yticks(fontsize=10)
        plt.grid(True, linestyle='--', alpha=0.5)
        plt.tight_layout()
        plt.show()

plot_top_genes_bar(top_genes_per_week)

import networkx as nx
import matplotlib.pyplot as plt

def visualize_graph(graph_data, node_size=15, edge_alpha=0.3):
    # Convert PyTorch Geometric data to NetworkX
    G = nx.Graph()

    edge_index = graph_data.edge_index.cpu().numpy()
    G.add_edges_from(zip(edge_index[0], edge_index[1]))

    # Layout
    pos = nx.spring_layout(G, seed=42)  # Stable layout, reproducible

    # Plot
    plt.figure(figsize=(12, 8))
    nx.draw_networkx_edges(G, pos, alpha=edge_alpha)
    nx.draw_networkx_nodes(G, pos, node_size=node_size, node_color="blue", alpha=0.7)

    plt.title("Gene Interaction Graph Used for GAT Input", fontsize=14, weight="bold")
    plt.axis("off")
    plt.tight_layout()
    plt.show()

visualize_graph(gene_graph_data, node_size=10, edge_alpha=0.2)

from umap import UMAP
import matplotlib.pyplot as plt
import torch

# Get embeddings from the output of the GAT model
gat_model.eval()  # Set the model to evaluation mode
with torch.no_grad():
    gat_output = gat_model(gene_graph_data)  # Pass your data to the model

# Assuming gat_output contains the node embeddings, usually in the shape (num_nodes,
embedding_dim)
```

```
gat_embeddings = gat_output.cpu().detach().numpy()

# Apply UMAP for dimensionality reduction
umap = UMAP(n_components=2, random_state=42)
embeddings_2d = umap.fit_transform(gat_embeddings)

# Scatter plot of gene embeddings
plt.figure(figsize=(8, 6))
plt.scatter(embeddings_2d[:, 0], embeddings_2d[:, 1], alpha=0.6)
plt.xlabel("UMAP Dimension 1")
plt.ylabel("UMAP Dimension 2")
plt.title("Gene Embeddings Learned by GAT")
plt.show()

import matplotlib.pyplot as plt
import seaborn as sns

def plot_embeddings_colored_by_stage(embeddings_2d, cluster_labels, stage_names=None):
    sns.set(style="whitegrid")

    unique_clusters = cluster_labels.unique()
    num_clusters = len(unique_clusters)
    palette = sns.color_palette("husl", num_clusters)
    cluster_color_map = dict(zip(unique_clusters, palette))

    plt.figure(figsize=(8, 6))

    for cluster in unique_clusters:
        cluster_indices = cluster_labels == cluster
        plt.scatter(
            embeddings_2d[cluster_indices, 0],
            embeddings_2d[cluster_indices, 1],
            color=cluster_color_map[cluster],
            label=stage_names.get(cluster, cluster) if stage_names else cluster,
            s=20, alpha=0.7
        )

    plt.xlabel("UMAP Dimension 1", fontsize=12)
    plt.ylabel("UMAP Dimension 2", fontsize=12)
    plt.title("Gene Embeddings Learned by GAT (Colored by Week Stage)", fontsize=14, weight='bold')
    plt.legend(title="Stage", bbox_to_anchor=(1.05, 1), loc='upper left')
    plt.tight_layout()
    plt.show()

# If you have stage_names mapping
```

```
plot_embeddings_colored_by_stage(embeddings_2d, adata.obs['week_stage'])

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split

# Prepare data
X = adata.X
y = LabelEncoder().fit_transform(adata.obs["week_stage"])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train Random Forest
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Predict
y_pred_rf = rf_model.predict(X_test)

# Evaluation
print(f"Random Forest Accuracy: {accuracy_score(y_test, y_pred_rf):.2%}")
print("\nClassification Report:\n", classification_report(y_test, y_pred_rf))

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Feature importance
rf_importance = pd.Series(rf_model.feature_importances_, index=adata.var_names).nlargest(20)

plt.figure(figsize=(10, 6))
sns.barplot(x=rf_importance.values, y=rf_importance.index, color="royalblue")  # Use a single color
plt.xlabel("Feature Importance", fontsize=12)
plt.title("Top 20 Biomarkers Identified by Random Forest", fontsize=14, weight='bold')
plt.tight_layout()
plt.show()

import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, accuracy_score

def plot_confusion_matrix(true_labels, predictions, class_names=None, normalize=False,
title='Confusion Matrix - Random Forest'):
    cm = confusion_matrix(true_labels, predictions, normalize='true' if normalize else None)
    acc = accuracy_score(true_labels, predictions)
```

```
    plt.figure(figsize=(10, 8))
    sns.heatmap(cm, annot=True, fmt='.2f' if normalize else 'd',
                cmap='Blues', linewidths=0.5, square=True,
                xticklabels=class_names, yticklabels=class_names)

    plt.xlabel("Predicted Week Stage", fontsize=12)
    plt.ylabel("True Week Stage", fontsize=12)
    # Setting the title using the 'title' parameter
    plt.title(title, fontsize=14, weight='bold')
    plt.tight_layout()
    plt.show()

class_names = adata.obs['week_stage'].unique()
plot_confusion_matrix(y_test, y_pred_rf, class_names=class_names, normalize=False)

from sklearn.metrics import classification_report

# Predict on test set
y_pred_rf = rf_model.predict(X_test)

# Generate classification report
print("Classification Report — Random Forest:\n")
print(classification_report(y_test, y_pred_rf, zero_division=0))

from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
import numpy as np
from itertools import cycle

def plot_multiclass_roc(y_true, y_score, n_classes, class_names=None):
    # Binarize labels
    y_true_bin = label_binarize(y_true, classes=np.arange(n_classes))

    fpr, tpr, roc_auc = {}, {}, {}
    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_true_bin[:, i], y_score[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    # Plot
    plt.figure(figsize=(6, 6))
    colors = cycle(plt.cm.tab20.colors)  # More colorful and automatic
    for i, color in zip(range(n_classes), colors):
        label_name = class_names[i] if class_names is not None else f"Class {i}"
```

```
        plt.plot(fpr[i], tpr[i], color=color, lw=2,
                 label=f'{label_name} (AUC = {roc_auc[i]:.2f})')

    plt.plot([0, 1], [0, 1], 'k--', lw=2)
    plt.xlim([-0.05, 1.05])
    plt.ylim([-0.05, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Multiclass ROC Curve - Random Forest', fontsize=16, weight='bold')
    plt.legend(loc="lower right", fontsize=9)
    plt.grid(alpha=0.3)
    plt.tight_layout()
    plt.show()

# Usage
n_classes = rf_model.n_classes_
rf_probs = rf_model.predict_proba(X_test)

week_stage_names = adata.obs['week_stage'].unique()

plot_multiclass_roc(y_test, rf_probs, n_classes, class_names=week_stage_names)

!pip install xgboost

import xgboost as xgb

import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import xgboost as xgb

# Train XGBoost
xgb_model = xgb.XGBClassifier(
    objective='multi:softmax',
    num_class=len(np.unique(y)),
    eval_metric='mlogloss',
    n_estimators=200,
    learning_rate=0.05,
    max_depth=6,
    subsample=0.8,
    colsample_bytree=0.8,
    random_state=42
)
xgb_model.fit(X_train, y_train)
```

```
# Extract top 20 important genes
importance_df = pd.DataFrame({
    'Gene': adata.var_names,
    'Importance': xgb_model.feature_importances_
}).sort_values(by='Importance', ascending=False).head(20)

# Reverse importance for color mapping (high = dark)
importance_df = importance_df.sort_values('Importance', ascending=True)

# Plot
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

def plot_top_features(importances, feature_names, top_n=20, title="Top Features", palette="Blues_r"):
    # Prepare DataFrame
    df = pd.DataFrame({
        "Feature": feature_names,
        "Importance": importances
    }).sort_values(by="Importance", ascending=False).head(top_n)

    # Plot
    plt.figure(figsize=(10, 6))
    sns.barplot(x="Importance", y="Feature", data=df, palette=palette)
    plt.xlabel("Feature Importance", fontsize=12)
    plt.ylabel("Gene Name", fontsize=12)
    plt.title(title, fontsize=14, weight="bold")
    plt.grid(axis='x', linestyle='--', alpha=0.5)
    plt.tight_layout()
    plt.show()

# Example usage:
plot_top_features(
    importances=xgb_model.feature_importances_,
    feature_names=adata.var_names,
    top_n=20,
    title="Top 20 Biomarkers Identified by XGBoost"
)

import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, accuracy_score
```

```
def plot_confusion_matrix(true_labels, predictions, class_names=None, normalize=False,
title_model="Model"):
    # Compute confusion matrix
    cm = confusion_matrix(true_labels, predictions, normalize='true' if normalize else None)
    acc = accuracy_score(true_labels, predictions)

    # Plot
    plt.figure(figsize=(10, 8))
    sns.heatmap(cm, annot=True, fmt='.2f' if normalize else 'd',
                cmap='Blues', linewidths=0.5, square=True,
                xticklabels=class_names, yticklabels=class_names)

    plt.xlabel("Predicted Week Stage", fontsize=12)
    plt.ylabel("True Week Stage", fontsize=12)

    # Setting the title using f-string formatting
    title = f"{'Normalized' if normalize else ''} Confusion Matrix - {title_model}"
    plt.title(title, fontsize=14, weight='bold')  # Add title to the plot

    plt.tight_layout()
    plt.show()

# Use it like this for XGBoost:
y_pred_xgb = xgb_model.predict(X_test)
class_names = adata.obs['week_stage'].cat.categories if hasattr(adata.obs['week_stage'], 'cat') else
np.unique(adata.obs['week_stage'])

plot_confusion_matrix(y_test, y_pred_xgb, class_names=class_names, normalize=False,
title_model="XGBoost")

# Predict on test set
y_pred_xgb = xgb_model.predict(X_test)

# Generate classification report
print("Classification Report — XGBoost:\n")
print(classification_report(y_test, y_pred_xgb, zero_division=0))

# Predict class probabilities with XGBoost
xgb_probs = xgb_model.predict_proba(X_test)

# Binarize y_test again if needed
y_test_bin = label_binarize(y_test, classes=np.arange(n_classes))

# Compute ROC and AUC
fpr = dict()
```

```
tpr = dict()
roc_auc = dict()

for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], xgb_probs[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot all ROC curves
plt.figure(figsize=(6, 8))
colors = cycle(plt.cm.tab20b.colors)

for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=2,
             label=f'Class {i} (AUC = {roc_auc[i]:.2f})')

plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - XGBoost (Multiclass)')
plt.legend(loc='lower right')
plt.grid()
plt.show()

!pip install matplotlib-venn --quiet

top_n = 200  # keep consistent for all models

# Recreate RF importance properly
rf_importance = pd.Series(rf_model.feature_importances_,
index=adata.var_names).sort_values(ascending=False)
top_rf_genes = set(rf_importance.head(top_n).index)

# XGB already done
xgb_importance = pd.Series(xgb_model.feature_importances_,
index=adata.var_names).sort_values(ascending=False)
top_xgb_genes = set(xgb_importance.head(top_n).index)

# GAT attention-based importance
# Calculate mean attention scores per gene (adapt this part based on your GAT model)
edge_index, attention_weights = extract_attention_weights(gat_model, gene_graph_data) # Assuming
'extract_attention_weights' is defined
mean_attention = np.zeros(len(adata.var_names))
for i in range(edge_index.shape[1]):
    source = edge_index[0, i]
    if source < len(adata.var_names):
```

```
        # Take the mean of attention weights if it's a multi-head attention
        mean_attention[source] += np.mean(attention_weights[i])
mean_attention /= np.max(mean_attention)  # Normalize

# Create and sort biomarkers based on mean attention
gat_biomarkers = pd.Series(mean_attention, index=adata.var_names).sort_values(ascending=False)
top_gat_genes = set(gat_biomarkers.head(top_n).index)

from matplotlib_venn import venn3
import matplotlib.pyplot as plt

def plot_biomarker_overlap(top_rf_genes, top_xgb_genes, top_gat_genes, top_n=20):
    plt.figure(figsize=(6, 6))
    venn = venn3(
        (top_rf_genes, top_xgb_genes, top_gat_genes),
        set_labels=("Random Forest", "XGBoost", "GAT"),
        set_colors=('cornflowerblue', 'mediumseagreen', 'tomato'),
        alpha=0.7
    )

    # Optional: Make font sizes bigger for better visibility
    for text in venn.set_labels:
        if text:
            text.set_fontsize(14)
            text.set_fontweight('bold')
    for text in venn.subset_labels:
        if text:
            text.set_fontsize(12)

    plt.title(f"Overlap of Top {top_n} Biomarkers Across Models", fontsize=16, weight='bold')
    plt.tight_layout()
    plt.show()

    # Print intersection (optional)
    common_genes = top_rf_genes & top_xgb_genes & top_gat_genes
    print(f"\nCommon Biomarkers Across All 3 Models ({len(common_genes)} genes):")
    if len(common_genes) > 0:
        for gene in sorted(common_genes):
            print(f"- {gene}")
    else:
        print("No common genes found.")

# ➡ Example usage:
plot_biomarker_overlap(top_rf_genes, top_xgb_genes, top_gat_genes, top_n=20)
```

```
# Genes shared by all three models
common_all = top_rf_genes & top_xgb_genes & top_gat_genes

# Genes shared by RF and XGB only
common_rf_xgb = (top_rf_genes & top_xgb_genes) - common_all

# Genes shared by GAT and RF only
common_rf_gat = (top_rf_genes & top_gat_genes) - common_all

# Genes shared by GAT and XGB only
common_xgb_gat = (top_xgb_genes & top_gat_genes) - common_all

# Unique to each model
unique_rf = top_rf_genes - (top_xgb_genes | top_gat_genes)
unique_xgb = top_xgb_genes - (top_rf_genes | top_gat_genes)
unique_gat = top_gat_genes - (top_rf_genes | top_xgb_genes)

import networkx as nx
 # Genes shared by all three models
common_all = top_rf_genes & top_xgb_genes & top_gat_genes

# Assign the shared genes to the variable 'shared_genes'
shared_genes = list(common_all)  # Convert the set to a list

# Subset adata to shared genes
adata_shared = adata[:, shared_genes]

# Compute gene-gene correlation
gene_expr = adata_shared.X.T
corr_matrix = np.corrcoef(gene_expr)

# Create graph from strong correlations
G = nx.Graph()
genes = shared_genes
for i in range(len(genes)):
    for j in range(i + 1, len(genes)):
        corr = corr_matrix[i, j]
        if abs(corr) > 0.6:  # threshold for connection
            G.add_edge(genes[i], genes[j], weight=corr)

# Plot network
plt.figure(figsize=(10, 10))
pos = nx.spring_layout(G, seed=42)
nx.draw(G, pos, with_labels=True, node_color='skyblue', edge_color='gray', node_size=1500,
font_size=10)
```

```
plt.title("Gene-Gene Correlation Network of Shared Biomarkers")
plt.show()

!pip install mygene --quiet

import mygene
mg = mygene.MyGeneInfo()

# Query gene functions
# shared_genes should have been defined in an earlier cell;
# If not, you'll need to define it appropriately
gene_info = mg.querymany(shared_genes, scopes='symbol', fields='symbol,name,summary,disgenet',
species='human')

# Assuming 'gene_info' is your list of dictionaries:
all_variables = []
for gene_data in gene_info:
    all_variables.extend(gene_data.keys())  # Add keys from each gene

# Get unique variables to avoid duplicates
unique_variables = list(set(all_variables))

print("All Variables in gene_info:")
for variable in unique_variables:
    print(variable)

!pip install mygene --quiet

import mygene
mg = mygene.MyGeneInfo()

# Query gene functions
gene_info = mg.querymany(shared_genes, scopes='symbol', fields='symbol,name,summary,disgenet',
species='human')

# Convert to DataFrame
import pandas as pd
gene_df = pd.DataFrame(gene_info)[['symbol', 'name', 'summary']]
pd.set_option('display.max_colwidth', 150)

# Show
gene_df

"""Gene Function Annotation Using MyGeneInfo
```

To functionally annotate the 19 genes shared by all three models (GAT, Random Forest, and XGBoost), I used the MyGeneInfo Python package. The gene symbols were passed to the querymany() function with the scope set to "symbol" and fields set to "symbol,name,summary". This queried the MyGene.info database, retrieving the official gene symbol, full name, and biological summary for each gene. The data was converted into a pandas DataFrame and formatted for readability. This process enabled efficient collection of up-to-date and reliable gene annotations directly from biological databases for inclusion in the final report.
"""

```
import numpy as np
import matplotlib.pyplot as plt

# Model names
model_names = ['GAT', 'Random Forest', 'XGBoost']

# Calculate or retrieve the accuracy for each model
gat_accuracy = accuracy_score(true_labels, predictions)
rf_accuracy = accuracy_score(y_test, rf_model.predict(X_test))
xgb_accuracy = accuracy_score(y_test, y_pred_xgb)

# Assume loss values you computed (example only, replace with your real ones)
gat_loss = 0.18  # Example: GAT validation loss
rf_loss = 0.25   # Example: RF log loss
xgb_loss = 0.20  # Example: XGB log loss

accuracies = [gat_accuracy, rf_accuracy, xgb_accuracy]
losses = [gat_loss, rf_loss, xgb_loss]

# Plot
x = np.arange(len(model_names))  # label locations
width = 0.35  # width of the bars

fig, ax1 = plt.subplots(figsize=(6, 6))

# Accuracy bars
rects1 = ax1.bar(x - width/2, accuracies, width, label='Accuracy', color='royalblue')
# Loss bars
rects2 = ax1.bar(x + width/2, losses, width, label='Loss', color='tomato')

# Labels and Titles
ax1.set_ylabel('Score', fontsize=12)
ax1.set_xlabel('Model', fontsize=12)
ax1.set_title('Model Accuracy and Loss Comparison', fontsize=16, weight='bold')
ax1.set_xticks(x)
ax1.set_xticklabels(model_names)
```

```
ax1.legend()

# Value labels on top of bars
for rect in rects1 + rects2:
    height = rect.get_height()
    ax1.annotate(f'{height:.2f}', xy=(rect.get_x() + rect.get_width()/2, height),
                 xytext=(0, 3), textcoords="offset points",
                 ha='center', va='bottom', fontsize=10)

ax1.grid(axis='y', linestyle='--', alpha=0.5)
plt.ylim(0, 1.2)  # extra room
plt.tight_layout()
plt.show()

import pandas as pd
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# --- Calculate Metrics ---

# GAT
gat_acc = accuracy_score(true_labels, predictions)
gat_prec = precision_score(true_labels, predictions, average='macro')
gat_rec = recall_score(true_labels, predictions, average='macro')
gat_f1 = f1_score(true_labels, predictions, average='macro')

# Random Forest
rf_acc = accuracy_score(y_test, y_pred_rf)
rf_prec = precision_score(y_test, y_pred_rf, average='macro')
rf_rec = recall_score(y_test, y_pred_rf, average='macro')
rf_f1 = f1_score(y_test, y_pred_rf, average='macro')

# XGBoost
xgb_acc = accuracy_score(y_test, y_pred_xgb)
xgb_prec = precision_score(y_test, y_pred_xgb, average='macro')
xgb_rec = recall_score(y_test, y_pred_xgb, average='macro')
xgb_f1 = f1_score(y_test, y_pred_xgb, average='macro')

# --- Create Comparison Table ---
comparison_df = pd.DataFrame({
    'Accuracy': [gat_acc, rf_acc, xgb_acc],
    'Precision': [gat_prec, rf_prec, xgb_prec],
    'Recall': [gat_rec, rf_rec, xgb_rec],
    'F1 Score': [gat_f1, rf_f1, xgb_f1]
}, index=['GAT', 'Random Forest', 'XGBoost'])
```

```
# Round for cleaner display
comparison_df = comparison_df.round(4)

# Display
print("Model Performance Comparison:")
display(comparison_df)
```