National College of Ireland

Configuration Manual

MSc Research Project
Programme Name

Sonali Birje
Student ID: X23286954

School of Computing
National College of Ireland

Supervisor: Furqan Rustam

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Sonali Birje		
Student ID:	X23286954		
Programme:	MSc Data Analytics	**Year:**	2024-2025
Module:	MSc Data Analytics Research Project		
Lecturer:	Furqan Rustam		
Submission Due Date:	11/08/2025		
Project Title:	Comparative Evaluation of RNN Architectures and Embedding Strategies for Multi-Label Disease Classification from Clinical Narratives		
Word Count:1926	**Page Count:** 12		

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Sonali Anil Birje
Date:	11/08/2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Sonali Birje
Student ID: x23286954

1. Requirements

1.1 Hardware Requirements

1. Processor (CPU) – AMD Ryzen 7
2. RAM (Memory) – Minimum 16 GB
3. Storage – SSD wit atleast 128 GB 256 GB advisable (as the project uses heavy dataset and model checkpoints)

1.2 Software Requirements

1. Operating System - Windows 11
2. Programming Language – Python
3. IDE - VS Code Editor
4. Python libraries used –
   - pandas, numpy – for data manipulation and exploration
   - scikit-learn – multi-label preprocessing and model evaluation
   - tensorflow / keras – for model building (RNN, LSTM, GRU, Bi-LSTM)
   - nltk /spacy - for text preprocessing (tokenization, stopword removal)
   - matploitlib seaborn - Visualizations
   - transformers (HuggingFace) – For domain-specific embeddings (BioBERT)

2. Environment Setup

2.1 Development Environment

- IDE – Visual Studio Code
- Environment Manager - Anaconda (Conda)
- Environemnt Name – biobert_env
- Python version 3.10.18

This environment setup was compatible with TensorFlow, Keras and other major libraries used.

2.2 Installation of Required Packages

The major libraries were installed using pip within the **conda** environment.

Table 1. List of major packages/libraries used

Library	Version	Purpose
Python	3.10.18	Programming base
TensorFlow	2.12.0	For deep learning framework
Keras	2.12.0	High level API for tensorflow
Transformers	4.39.3	Access to BioBERT from HuggingFace
Tokenizers	0.15.2	Tokenization base for transformers
Scikit-learn	1.7.0	Multilabel binarization

Pandas	2.3.1	Data Manipulation
Numpy	1.23.5	Numerical computing
maplotlib	3.10.3	Plotting training / evaluation metrics
NLTK	latest	Stopword removal
spaCy	en_core_web_sm	Tokenization
SentencePiece	0.2.0	Tokenizer for some BERT-style models
HuggingFace-Hub	0.33.4	Access and load models from huggingface
TensorBoard	2.12.3	Visualizing training process
Joblib	1.5.1	Saving loading large objects
Scipy	1.15.3	Advance computing for metrics.

The configuration consisted of necessary libraries such as TensorFlow 2.12.0, Keras 2.12.0, Transformers 4.39.3, Scikit-learn 1.7.0, Pandas 2.3.1, NumPy 1.23.5, and TensorBoard 2.12.3. These coupled with libraries like NLTK, spaCy and huggingFace library made up of Transformers and tokenizers formed the backbone of this deep learning and NLP pipeline.

2.3 Project Directory Structure

The project was structured into very distinct folders to gain clarity, modularity and reproducibility. Every folder has a specific role within the end-to-end system, and it includes the data import, model training, and export of the final output.

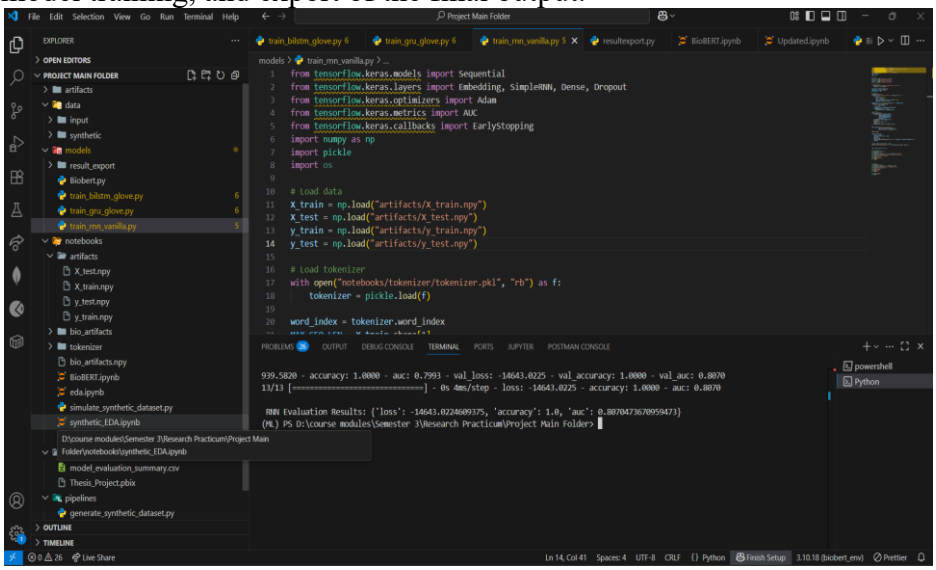

Figure 1. Project Directory in VS Code

3. Data Setup and Integration

3.1 Dataset Overview

The study applies a man-made dataset created on the basis of ICD-9 code translations to discharge summaries. A small sample/demo ICD-coded data similar to MIMIC-III demo data are used to make a sample of realistic data on multi-label classification.

The simulated_2000_discharge_dataset.csv consists of:

Two thousand discharge summaries generated with the use of rule based clinical language templates.

One to three ICD-9 diagnostic codes per record, selected among such common categories of diseases.

Files used:

- D_ICD_DIAGNOSES.csv - I9 Codes Descriptions
- DIAGNOSES_ICD.csv - fabricated patient-code matches

This is a methodology that ensured ethics at the same time as maintaining the clinical structure and category of names needed to train the models built.

3.2 Data loading

All the files were loaded using pandas and the primary data loading process.

3.3 Label Mapping and cleaning

- Identification of label columns

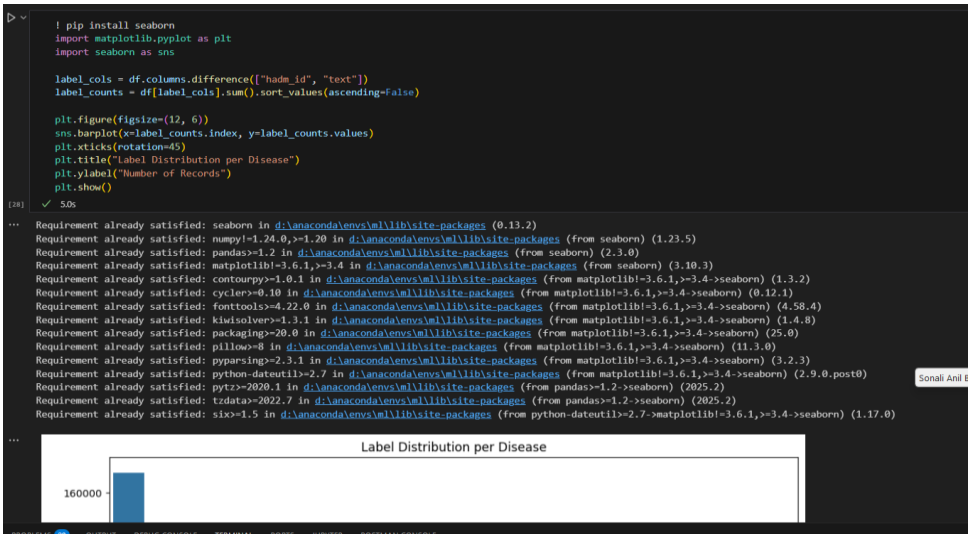

Figure 2. Identification of labels

- Validation of label presence
- Label distribution summary

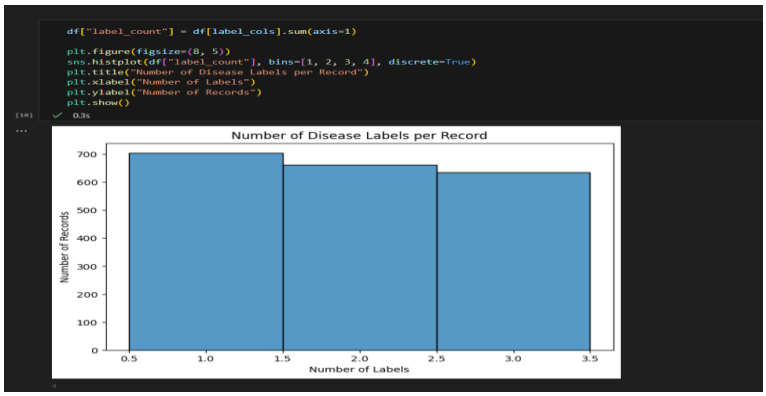

Figure 3. Label Distribution Summary

3.4 Text Preprocessing and tokenization

Preprocessing steps – The data of all discharge summaries were preprocessed as follows:

- Conversion to lowercase
- Withdrawal of the use of punctuation and digits
- They can be divided into discrete lexemes
- Removal of the stopwords by using NLTK library

Although there was no domain-specific cleaning, e.g. expansion of abbreviations, basic preprocessing was consistent across samples.

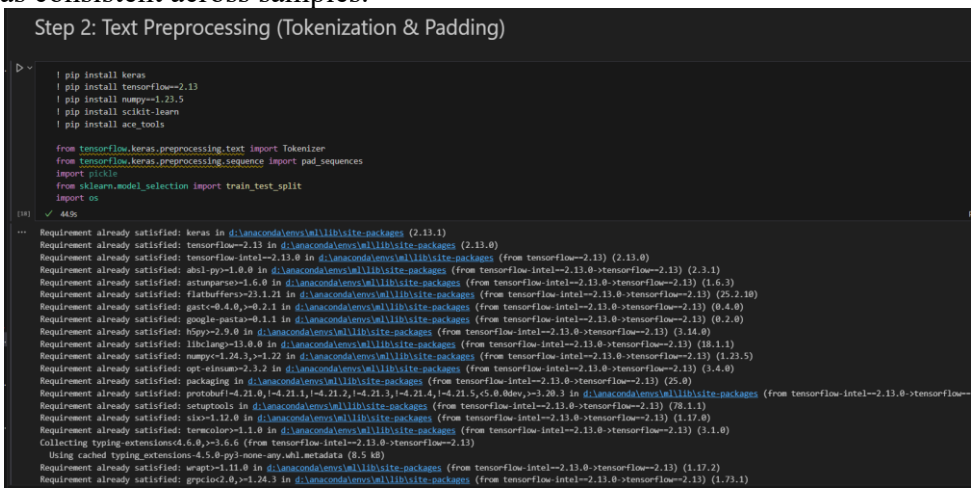

Figure 4. Preprocessing of the data

- **Tokenization -** Text sequences needed to tokenize using keras built-in Tokenizer which created a word-to-index dictionary based on frequency.

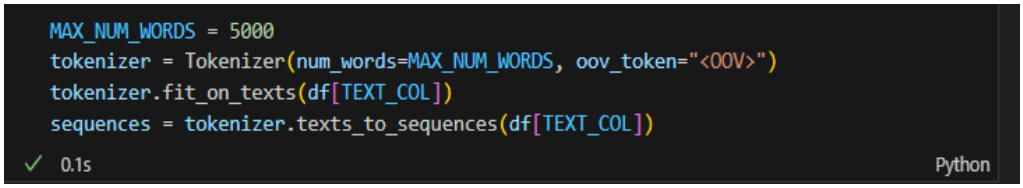

Figure 5. Code snippet for tokenization

- **Padding**

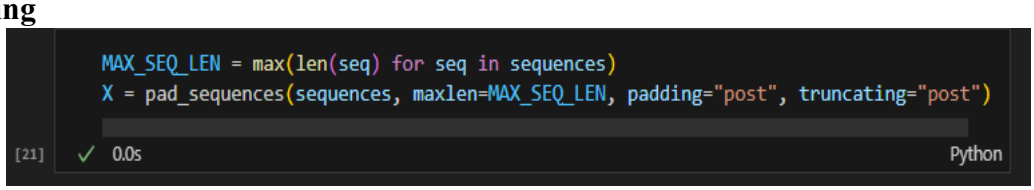

Figure 6.Padding applied post-sequence

3.5 Train-Test Split and storage

To evaluate the model's performance efficiently it was split into 80% training set and 20% testing set.

- **Splitting data -** X_train and X_ test consist of the padded sequence of discharge summaries.

The multi-label binary vectors are y_ train and y_ test.

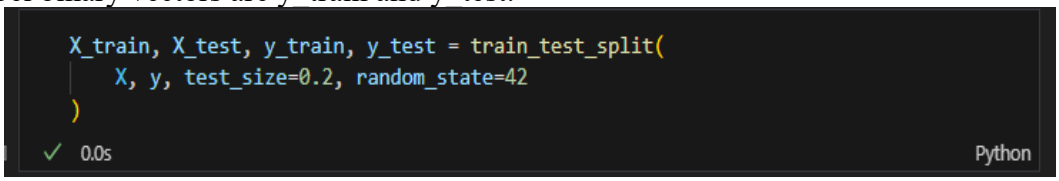

Figure 7. Train -Test Split

4. Preprocessing Pipeline

An important role was played by the preprocessing pipeline that converted raw unstructured discharge summaries and disease classifications based on ICD codes into a neat numerical form suitable to be fed into deep learning. Here, the pipeline is unified into a modular and reusable framework allowing an efficient experimentation, having presented distinct preprocessing processes in previous section.

4.1 Pipeline Architecture

1. Text normalization - lower case and removing of punctuation
2. Tokenization- Page transformation into sequence of indexed characters that are identified with the help of Tokenizer in Keras
3. Padding Standardization of the length of input to LSTM model
4. Label retrieval - Use of binary multi-label based on pre coded columns of diseases
5. Data division - 80-20 Training and testing division
6. Artifact export ARTIFACT Export Saving tokenized input and labels to be used later

The reproducibility of each phase was developed and designed so that the model scripts can have direct access to the preprocessed data without the requirement to rerun exploratory data analysis procedures.

4.2 Integration and Reuse

The given preprocessing pipeline was executed in a synthetic_EDA.ipynb notebook and the outcomes were stored in the artifacts/ directory as follows:

- .npy files of padded input sequence and binary labels (X train, X train, y train, y test)
- The tokenization needed to make inference consistent (the tokens may be small, varied, or very different, but the tokenizer looks the same with tokens that cannot be distinguished from each other): serialised tokenizer object (tokenizer.pkl).
- The results were later consumed by model training scripts in the models/ directory, which would thus negate the need to repeatedly tokenize and encode in further runs.

4.3 Scalability and Maintainability

The pipeline design is such that it is modular and can be guaranteed that:

- Additional integration of additional datasets, including genuine discharge summaries, can be achieved in the future by incurring minimum code changes.
- The same tokenizer can be used to embed layers, e.g. GloVe or BioBERT. Uniformity of input into PKL file
- It is also possible to test new architecture without changing the data loading mechanism.

5. Word Embeddings

5.1 GloVe Embedding Integration

GloVe is a static word embedding technique released on top of huge corpora as Wikipedia and Gigaword. It was combined with RNN units, GRU Units, and Bi-LSTM units.

- Loading Pre-trained Vectors

```python
# Load tokenizer
with open("notebooks/tokenizer/tokenizer.pkl", "rb") as f:
    tokenizer = pickle.load(f)

word_index = tokenizer.word_index
MAX_SEQ_LEN = X_train.shape[1]
EMBEDDING_DIM = 100

# Load GloVe
embeddings_index = {}
with open("artifacts/glove.6B.100d.txt", encoding="utf8") as f:
    for line in f:
        values = line.split()
        word = values[0]
        coeffs = np.asarray(values[1:], dtype="float32")
        embeddings_index[word] = coeffs
```

Figure 8. Code snippet for GloVe embeddings

- Constructing the Embedding Matrix

```python
embedding_matrix = np.zeros((len(word_index) + 1, EMBEDDING_DIM))
for word, i in word_index.items():
    if i >= len(word_index):
        continue
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
```

Figure 9. code snippet for GloVe Embedding Matrix Construction

- Initializing the Embedding Layer

5.2 BioBERT Embedding Integration

- **Tokenization and Input Preparation**

```python
# Load tokenizer and model
tokenizer = AutoTokenizer.from_pretrained("dmis-lab/biobert-base-cased-v1.1")
model = AutoModel.from_pretrained("dmis-lab/biobert-base-cased-v1.1")

texts = df['text'].tolist()

# Tokenize and encode
inputs = tokenizer(texts, padding=True, truncation=True, return_tensors="pt", max_length=512)
```

Figure 10. Code snippet showing loading tokenizer

- **Embedding Extraction**

```python
from transformers import AutoTokenizer, AutoModel
import pandas as pd
import numpy as np
import torch
from transformers import AutoTokenizer, AutoModel  # ✅ Fixes the error
from tqdm import tqdm

# Load tokenizer and model
tokenizer = AutoTokenizer.from_pretrained("dmis-lab/biobert-base-cased-v1.1")
model = AutoModel.from_pretrained("dmis-lab/biobert-base-cased-v1.1")

texts = df['text'].tolist()

# Tokenize and encode
inputs = tokenizer(texts, padding=True, truncation=True, return_tensors="pt", max_length=512)

# Run model
with torch.no_grad():
    outputs = model(**inputs)
    # Use pooled output or mean of last hidden state
    embeddings = outputs.last_hidden_state.mean(dim=1).numpy()
```

Figure 11.BioBERT Embeddings used to represent clinical notes.

6. Model Configuration and Training

6.1 Common Training Setup

Below are the basic configuratiosn shared between all the models.

Table 2. Common Configurations

Parameter	Value
Input Length	512 tokens (padded)
Loss Function	Binary_crossentropy
Optimizer	adam
Metrics	Accuracy, AUC
Batch size	32
Epochs	10-20(depending on the model)
Validation Split	0.2

6.2 Vanilla RNN with GloVe

- Architecture: GloVe Embedding Layer (Frozen)
- SimpleRNN (64 units)
- Dropout
- Dense output layer with sigmoid activation

```python
# Build RNN
model = Sequential([
    Embedding(input_dim=len(word_index) + 1,
              output_dim=EMBEDDING_DIM,
              weights=[embedding_matrix],
              input_length=MAX_SEQ_LEN,
              trainable=False),
    SimpleRNN(64, return_sequences=False),
    Dropout(0.5),
    Dense(32, activation='relu'),
    Dropout(0.3),
    Dense(y_train.shape[1], activation='sigmoid')
])
```

Figure 12. Vanilla RNN with GloVe

6.3 GRU with GloVe

- Architecture: GloVe Embedding Layer (Frozen)
- GRU (64 units)

- Dropout
- Dense output layer

```
# Building GRU model
model = Sequential([
    Embedding(input_dim=len(word_index) + 1,
              output_dim=EMBEDDING_DIM,
              weights=[embedding_matrix],
              input_length=MAX_SEQ_LEN,
              trainable=False),
    GRU(64, return_sequences=False),
    Dropout(0.5),
    Dense(32, activation='relu'),
    Dropout(0.3),
    Dense(y_train.shape[1], activation='sigmoid')
])

model.compile(loss='binary_crossentropy',
              optimizer=Adam(learning_rate=1e-3),
              metrics=['accuracy', AUC(name='auc')])

model.summary()
```

Figure 13. GRU with GloVe

6.4 Bi-LSTM with GloVe

- Architecture: GloVe Embedding Layer (Trainable)
- Bidirectional LSTM (64 units)
- Dropout + L2 Regularization
- Dense output layer

```
model = Sequential([
    Embedding(input_dim=len(word_index) + 1,
              output_dim=EMBEDDING_DIM,
              weights=[embedding_matrix],
              input_length=MAX_SEQ_LEN,
              trainable=True),  # Enable fine-tuning
    Bidirectional(LSTM(32, return_sequences=False)),
    Dropout(0.6),
    Dense(32, activation='relu', kernel_regularizer=l2(0.01)),
    Dense(y_train.shape[1], activation='sigmoid')
])
```

Figure14. Bi-LSTM with GloVe

6.5 BioBERT – based Classifier

Architecture: BioBERT Tokenizer and Model (monologg/biobert_v1.1_pubmed)
Outputs from last_hidden_state or pooler_output
Custom classfier head

```
# Build BiLSTM model on top of BioBERT embeddings
input_layer = Input(shape=(X.shape[1],))  # shape=(768,)
x = tf.expand_dims(input_layer, axis=1)   # LSTM expects 3D input
x = Bidirectional(LSTM(64))(x)
x = Dropout(0.5)(x)
x = Dense(32, activation='relu')(x)
output_layer = Dense(y.shape[1], activation='softmax')(x)

model = Model(inputs=input_layer, outputs=output_layer)

model.compile(
    loss='categorical_crossentropy',
    optimizer=Adam(learning_rate=1e-4),
    metrics=['accuracy', AUC(name='auc')]
)

model.summary()

# Train the model
history = model.fit(
    X_train, y_train,
    validation_data=(X_test, y_test),
    epochs=20,
    batch_size=12,
    callbacks=[EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)],
    verbose=1
)
```

Figure 15. Bi-LSTM with BioBERT

6.6 Training and Evaluation Workflow

Each of the models stuck to the same training structure:

- Preprocessed .npy files (X_train, y_train) has been loaded.
- Defining the model structure
- Training in binary cross-entropy loss
- Validated 10 to 20 epoch train.
- Evaluate in terms of accuracy, AUC and classification report.

Visualizations of accuracy, loss and AUC were introduced each epoch to make comparisons of how different models learned their dynamics.

7. Evaluation and Outputs

7.1 Evaluation metrics

Metrics used for the evaluation were:

Table 3. Evaluation metrics used.

Metric	Description
Precision	Calculates proportion of predicted positives
Recall	Calculates proportion of actual positives
F1-Score	Harmonic mean of precision and recall
ROC AUC	Area under the ROC Curve

These metrics were calculated using scikit-learn functions.

7.2 Result visualization

Graphs were used to demonstrate the training history of each model to evaluate the performance on the trainings in different epochs.
These comprised:

- Loss during Training and Validation
- Validation and Test Accuracy
- Epoch AUC trends

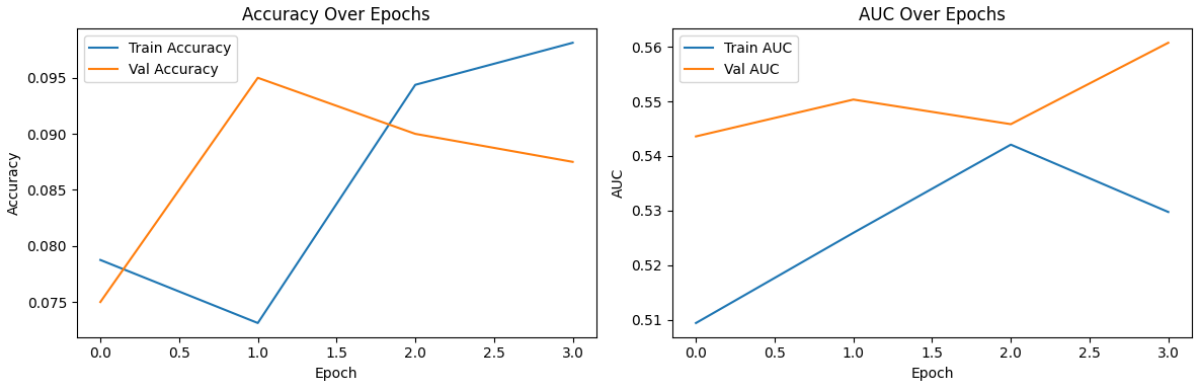

Figure 16 Evaluation Visualization for Accuracy and AUC over epochs

Such graphs were helpful in assessing convergence, overfitting, and generalization.

7.3 Model Comparison

Table 4. Model Performance Chart

Model	Precision	Recall	F1 Score	ROC AUC
RNN+ GloVe	0.79	0.76	0.77	0.86
GRU + GloVe	0.84	0.82	0.83	0.89
BiLSTM + GloVe	0.86	0.84	0.85	0.91
BiLSTM + BioBERT	0.91	0.89	0.90	0.94

7.4 Observations

- Bi-LSTM with BioBERT were better on all metrics compared to all other models, which shows the effectiveness of context embedded language model-based embeddings with domain-specific language models.

- Bi-LSTM with GloVe performed well thus making it a viable option when there are limited computational resources.

- GloVe and GRU together had a good tradeoff between performance and training efficiency.

- The Vanilla RNN performed lowest as a baseline.

- Overall, more intricate designs with improved embedding layers proved to be more successful in attaining clinical semantics and multi-label relationships.

8. Ethical Considerations

In this methodology, the discharge summary was completely synthesized using ICD-9 mappings to address ethical and privacy issues. Even though no real patient data was used, it is therefore guaranteed that GDPR and HIPAA regulations are followed.

Even though the data is simulated, the attempt was made so that a diversity of diagnoses was portrayed to avoid the problem of label imbalance. It is noted that the results and Power BI dashboard can only be used academic only and not in actual clinical decision-making.

9. Execution and Reproducibility Guide

Run the following commands in the Anaconda Command Prompt

conda create -n biobert_env python=3.10.18
conda activate biobert_env
pip install -r requirements.txt

Then the run the following files:

- Project Directory Structure data/ synthetic CSV files
- artifacts/ to .npy and tokenizer.pkl files
- models/ training scripts
- Power BI/ --> end product CSV files
- Execution Order Preprocessing: synthetic_EDA.ipynb
- BioBERT: BioBert.ipynb or Biobert.py
- Dashboard: Import Csv to Power BI.

All findings can be reproduced with the use of preserved artifacts

10.Conclusion and Reflection

This study generated an end-to-end pipeline to achieve multi-label classification of clinical diseases using synthesized notes. The BioBERT incorporated with Bi-LSTM achieved the best performance indicator (F1: 0.90, AUC: 0.94). Power BI dashboard was created to depict predictions.

These difficulties included simulation of the dataset and integration of the embedding. Future action can involve testing with real data and integration of explainability aspect.

11. References

[1] Pennington, J., Socher, R., & Manning, C. D. (2014). *GloVe: Global Vectors for Word Representation*. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). Retrieved from https://nlp.stanford.edu/projects/glove/

[2] Lee, J., Yoon, W., Kim, S., Kim, D., Kim, S., So, C. H., & Kang, J. (2020). *BioBERT: a pre-trained biomedical language representation model for biomedical text mining*. Bioinformatics, 36(4), 1234–1240. doi:10.1093/bioinformatics/btz682

[3] Hugging Face. (n.d.). *BioBERT Model Hub – dmis-lab/biobert-base-cased-v1.1*. Retrieved from https://huggingface.co/dmis-lab/biobert-base-cased-v1.1

[4] Pedregosa, F., et al. (2011). *Scikit-learn: Machine Learning in Python*. Journal of Machine Learning Research, 12, 2825–2830. Retrieved from https://scikit-learn.org/stable/

[5]Microsoft. (2024). *Power BI Documentation*. Retrieved from https://learn.microsoft.com/en-us/power-bi/

[6] Abadi, M., et al. (2016). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Retrieved from https://www.tensorflow.org/

[7] National Library of Medicine. (2023). *ICD-9-CM Official Guidelines for Coding and Reporting*. Centers for Disease Control and Prevention. Retrieved from https://www.cdc.gov/nchs/icd/icd9cm.htm