National College of Ireland

Comparative Study of Transformer Models for Text Classification in Healthcare

MSc Research Project
Data Analytics

Parth N. Gosavi
Student ID: x23223235

School of Computing
National College of Ireland

Supervisor: Dr. Abdul Qayum

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Parth N. Gosavi
Student ID:	X23223235
Programme:	Data Analytics
Year:	2024
Module:	MSc Research Project
Supervisor:	Dr. Abdul Qayum
Submission Due Date:	12/12/2024
Project Title:	Comparative Study of Transformer Models for Text Classification in Healthcare
Word Count:	1043

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:	
Date:	12th December 2024

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:	

Comparative Study of Transformer Models for Text Classification in Healthcare

Parth Nandkishor Gosavi
x23223235

1 Hardware and Software Requirements

1.1 Hardware Requirements

In this project, the following hardware was used:

- *System:* MacBook Pro M2 for preliminary preprocessing and development
- *GPU Environment:* Kaggle's NVIDIA Tesla P100 GPU for model training and evaluation, ensuring efficient computation for resource-intensive transformer models.
- *RAM:* 16 GB, this will help in preprocessing and manipulating the data efficiently.
- **Storage:** 50 GB storage capacity for PubMed dataset, pre-trained weights, and output models.

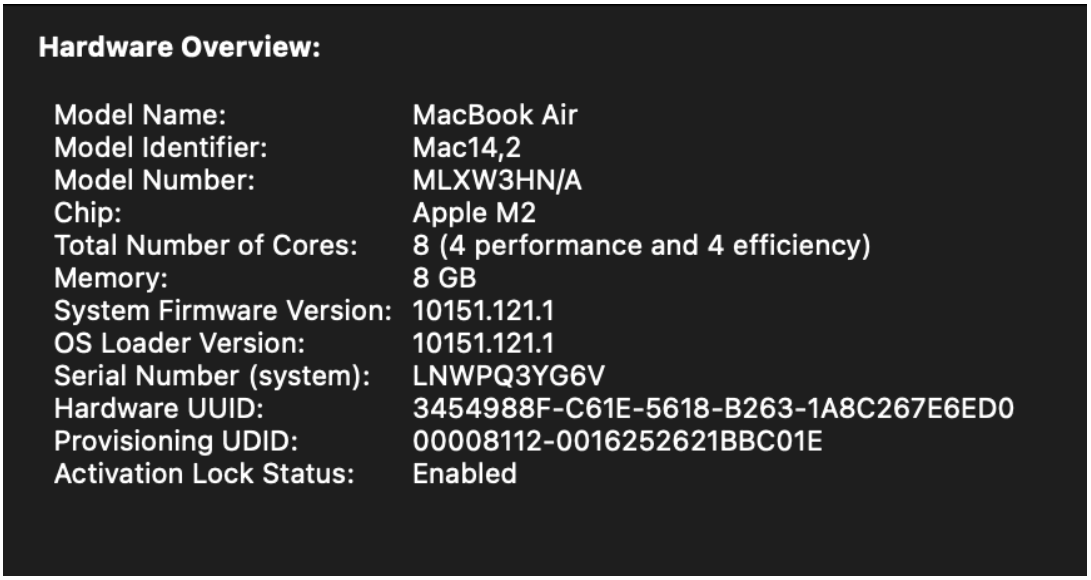

Figure 1: Hardware used

1.2 Software Requirements

The following software tools and libraries were used in this project.

- *Operating System:* Development is on macOS while the model is trained on a Linux-based GPU environment with Kaggle.
- *Programming Language:* Python 3.8 for all coding problems.
- *Development Environment:*
  - Jupyter Notebook for local development/debugging.
  - Kaggle Notebook for GPU-based training and execution.
- **Python Libraries:**
  - Hugging Face Transformers: For building and training BERT, RoBERTa, DistilBERT and XLNet.
  - PyTorch: The main deep learning framework used.
  - Pandas and NumPy: For manipulating and preprocessing datasets.
  - Scikit-Learn: For testing on performance metrics such as accuracy and F1-score.
  - Matplotlib and Seaborn: For creating plots.
  - NLTK: For text preprocessing (e.g., tokenization, stopword removal).

Figure 2: Code Environment

2 Dataset Description

In this use case, we will use the PubMed Multi-Label Text Classification Dataset, Contains titles and abstracts of scientific articles from PubMed. The dataset consists of Medical Subject Headings (MeSH), hierarchical labels that are organized in a tree-like structure to represent the traditional medical concepts that can be tagged to each of given

paper. multi-label classification is a common challenge in healthcare text and can lead to errors in different stages of the processing pipeline. analysis and can be helpful in assessing how transformer models perform complex tasks `https://www.kaggle.com/code/mohamedaref000/pubmedt5/input?select=PubMed+Multi+Label+Text+Classification+Dataset.csv`

2.1 Key Features

- **Titles and Abstracts:** Each of them includes a title and an abstract intro devoting details about the research work.
- **MeSH Labels:** Hierarchy of medical knowledge is covered under multi-label annotations data. For example:
  - **Root Label:** Ear
  - **Subcategory:** Ear Diseases
  - **Further Subcategory:** Hearing Disorders

2.2 Challenges

- **Multi-Label Structure:** The classification task is further complicated by the fact that articles can have multiple labels.
- **Label Sparsity:** Some labels occur infrequently, requiring careful preprocessing to reduce dimensionality while preserving information.

3 Data pre-processing

To prepare the PubMed dataset for model training, several preprocessing steps were performed to address missing values, label sparsity, and data inconsistencies. The following points outline the preprocessing pipeline:

3.1 Handle Missing Values

- Missing entries in the `Title` and `abstractText` columns were filled with empty strings to ensure data integrity.
- Records with missing MeSH Major labels were removed since they are critical for the classification task.

```
# Step 1: Handle Missing Values
# Fill missing text fields with an empty string and drop rows with missing␣
 ↪labels
data['Title'] = data['Title'].fillna('')
data['abstractText'] = data['abstractText'].fillna('')
data.dropna(subset=['meshMajor'], inplace=True)
```

Figure 3: Handling Missing Values

3.2 Use MultiLabelBinarizer to Convert Labels into Binary Format

- The multi-label annotations were converted into a binary format using `MultiLabelBinarizer`.
- This transformation allowed the model to treat each label as a separate binary classification problem.

```
# Use MultiLabelBinarizer to convert labels into binary format
mlb = MultiLabelBinarizer()
labels = mlb.fit_transform(data['meshMajor'])
```

Figure 4: Converting Labels into Binary Format

3.3 Text Cleaning Function

A custom text cleaning function was applied to remove noise from the text:

- Converted text to lowercase.
- Removed punctuation and special characters.
- Removed stopwords using `NLTK`.

```
# Step 3: Text Cleaning Function
def clean_text(text):
    text = text.lower()  # Convert text to lowercase
    text = re.sub(r'\b\w{1,2}\b', '', text)  # Remove short words
    text = re.sub(r'[^\w\s]', '', text)  # Remove punctuation
    text = ' '.join([word for word in text.split() if word not in stop_words])
↪# Remove stopwords
    return text
```

Figure 5: Text Cleaning Function

3.4 Combine `Title` and `abstractText` for Model Input

- The `Title` and `abstractText` columns were concatenated to create a single input field for the models.
- This step provided richer context for classification.

3.5 Split the Data

- The dataset was split into training and testing sets using an 80:20 ratio to evaluate model performance effectively.

```
# Apply text cleaning to 'Title' and 'abstractText'
data['Title'] = data['Title'].apply(clean_text)
data['abstractText'] = data['abstractText'].apply(clean_text)
```

```
# Step 4: Combine 'Title' and 'abstractText' for Model Input
data['text'] = data['Title'] + ' ' + data['abstractText']
```

Figure 6: Combining Title and abstractText for Model Input

```
# Step 5: Split the Data
X = data['text'].values
y = labels

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

Figure 7: spliting the Data

3.6 Save Processed Data for Model Training

- To break the cyclic nature of the data and reduce the training time, the raw processed data was saved locally.

4 Evaluation

This project evaluation involved three steps: defining the models and tokenizers to be used; training each model (and, in some cases, their corresponding tokenizer); and visualizing the results in order to compare performance across each metric.

This section introduces the transformer models used in this study by describing their definitions, training processes as well as visualizations of their performances.

4.1 Defining Models and Tokenizers

The transformer models that were evaluated in this work are BERT, RoBERTa, DistilBERT, XLNet. Along with each model was its respective tokenizer, which tokenized the text input in a suitable manner for that specific architecture. We used the Hugging Face Transformers library to load the models and the tokenizers.

Tokenization Process:

- **BERT and RoBERTa**: Using the WordPiece algorithm for text, splitting into common subword units for more generalization and word context.
- **DistilBERT**: Used a light version of BERT's tokenizer for efficiency and speed.
- **XLNet**: SentencePiece tokenization employed, a vocabulary-dependent method allowing for more inclusive input tokenization.

```
# Step 6: Save Processed Data for Model Training
np.save('X_train.npy', X_train)
np.save('X_test.npy', X_test)
np.save('y_train.npy', y_train)
np.save('y_test.npy', y_test)

print("Data preprocessing complete. Processed data saved for model training.")
```

```
Data preprocessing complete. Processed data saved for model training.
```

Figure 8: Save Processed Data for Model Training

```
# Define the models and their tokenizers
models = {
    'BERT': (BertForSequenceClassification.from_pretrained('bert-base-uncased',␣
 ↪num_labels=y_train.shape[1]),
             BertTokenizer.from_pretrained('bert-base-uncased')),
    'RoBERTa': (RobertaForSequenceClassification.
 ↪from_pretrained('roberta-base', num_labels=y_train.shape[1]),
                RobertaTokenizer.from_pretrained('roberta-base')),
    'DistilBERT': (DistilBertForSequenceClassification.
 ↪from_pretrained('distilbert-base-uncased', num_labels=y_train.shape[1]),
                   DistilBertTokenizer.
 ↪from_pretrained('distilbert-base-uncased')),
    'XLNet': (XLNetForSequenceClassification.
 ↪from_pretrained('xlnet-base-cased', num_labels=y_train.shape[1]),
              XLNetTokenizer.from_pretrained('xlnet-base-cased'))
}
```

Figure 9: Defining Models and Tokenizers

4.2 Train Each Model and Collect Results

All models were fine-tuned on PubMed dataset with same training set up for fair comparison. The parameters used during the training process were as follows:

- **Optimizer**: AdamW with a learning rate of 2×10^{-5}.
- **Batch Size**: 16.
- **Epochs**: 3.
- **Loss Function**: Binary cross-entropy for multi-label classification tasks.

Training Loop: Both models were trained and tested on the same splits of training and testing data. Both performance metrics (accuracy, F1-score) and epoch-wise training time were recorded to evaluate the efficiency and efficacy of the model.

4.3 Visualization

Various figures were generated to visualize these results and compare the performance of the models based on different metrics.

```python
# Train each model and collect results
results = {}
for model_name, (model, tokenizer) in models.items():
    print(f"\nTraining {model_name}...")
    train_loader = preprocess_data(X_train, y_train, tokenizer)
    test_loader = preprocess_data(X_test, y_test, tokenizer)
    results[model_name] = train_model(model, tokenizer, train_loader,␣
 ↪test_loader)
```

Figure 10: Train Each Model and Collect Results

```
Training BERT…

/opt/conda/lib/python3.10/site-packages/transformers/optimization.py:591:
FutureWarning: This implementation of AdamW is deprecated and will be removed in
a future version. Use the PyTorch implementation torch.optim.AdamW instead, or
set `no_deprecation_warning=True` to disable this warning
  warnings.warn(

Epoch 1 | Loss: 0.3537 | Accuracy: 0.8643 | F1: 0.8291 | Time: 190.01s
Epoch 2 | Loss: 0.2975 | Accuracy: 0.8668 | F1: 0.8365 | Time: 191.64s
Epoch 3 | Loss: 0.2890 | Accuracy: 0.8680 | F1: 0.8410 | Time: 191.54s

Training RoBERTa…

/opt/conda/lib/python3.10/site-packages/transformers/optimization.py:591:
FutureWarning: This implementation of AdamW is deprecated and will be removed in
a future version. Use the PyTorch implementation torch.optim.AdamW instead, or
set `no_deprecation_warning=True` to disable this warning
  warnings.warn(

Epoch 1 | Loss: 0.3420 | Accuracy: 0.8654 | F1: 0.8305 | Time: 193.98s
Epoch 2 | Loss: 0.2957 | Accuracy: 0.8664 | F1: 0.8356 | Time: 193.20s
Epoch 3 | Loss: 0.2879 | Accuracy: 0.8694 | F1: 0.8414 | Time: 193.54s

Training DistilBERT…
```

Figure 11: Result-1

4.3.1 Model Performance Metrics (Figure 13)

For each model, a line chart was drawn, plotting the accuracy, F1-score and loss of the model over the training epochs. This visualisation reflects the consistency and learning patterns seen throughout fine-tuning.

4.3.2 Model Comparison Radar Chart (Figure 14)

A radar chart was drawn in order to obtain a multi-dimensional comparison of the models. The metrics shown were accuracy, F1-score, loss and training time, indicating the trade-offs between model's performance and the computational costs.

4.3.3 Comparison Visualization(Figure 15)

To visualize training times for each model, we made use of a bar chart. The chart highlights the computational efficiency of each model and how performance metrics trade-

```
/opt/conda/lib/python3.10/site-packages/transformers/optimization.py:591:
FutureWarning: This implementation of AdamW is deprecated and will be removed in
a future version. Use the PyTorch implementation torch.optim.AdamW instead, or
set `no_deprecation_warning=True` to disable this warning
  warnings.warn(

Epoch 1 | Loss: 0.3572 | Accuracy: 0.8654 | F1: 0.8305 | Time: 99.77s
Epoch 3 | Loss: 0.2862 | Accuracy: 0.8711 | F1: 0.8442 | Time: 99.82s

Training XLNet…

/opt/conda/lib/python3.10/site-packages/transformers/optimization.py:591:
FutureWarning: This implementation of AdamW is deprecated and will be removed in
a future version. Use the PyTorch implementation torch.optim.AdamW instead, or
set `no_deprecation_warning=True` to disable this warning
  warnings.warn(

Epoch 1 | Loss: 0.3283 | Accuracy: 0.8656 | F1: 0.8313 | Time: 248.80s
Epoch 2 | Loss: 0.2947 | Accuracy: 0.8677 | F1: 0.8339 | Time: 248.83s
Epoch 3 | Loss: 0.2861 | Accuracy: 0.8705 | F1: 0.8401 | Time: 249.24s
```

Figure 12: Result-2

off with training time.

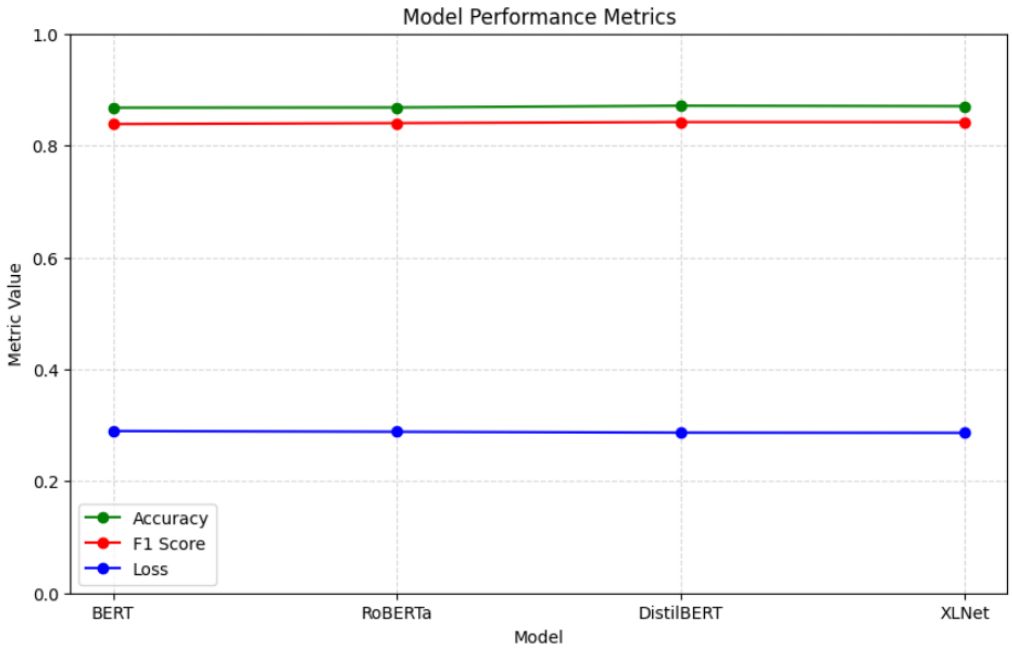

Figure 13: Model Performance Metrics

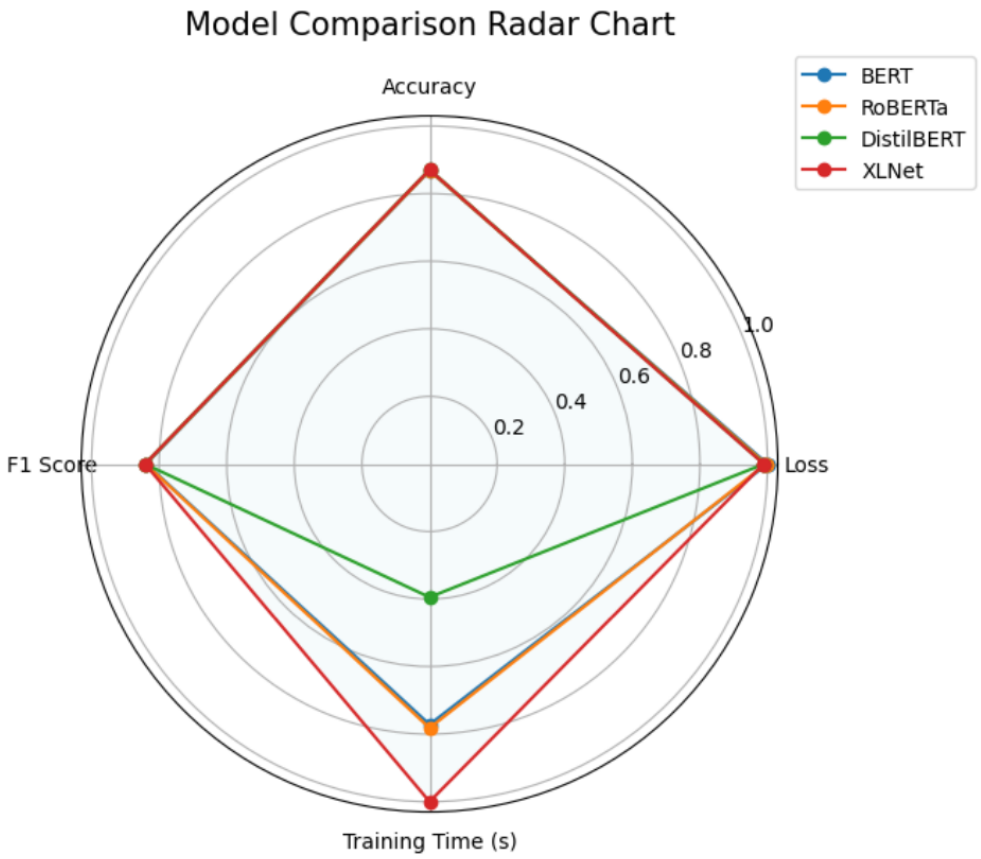

Figure 14: radar Chart

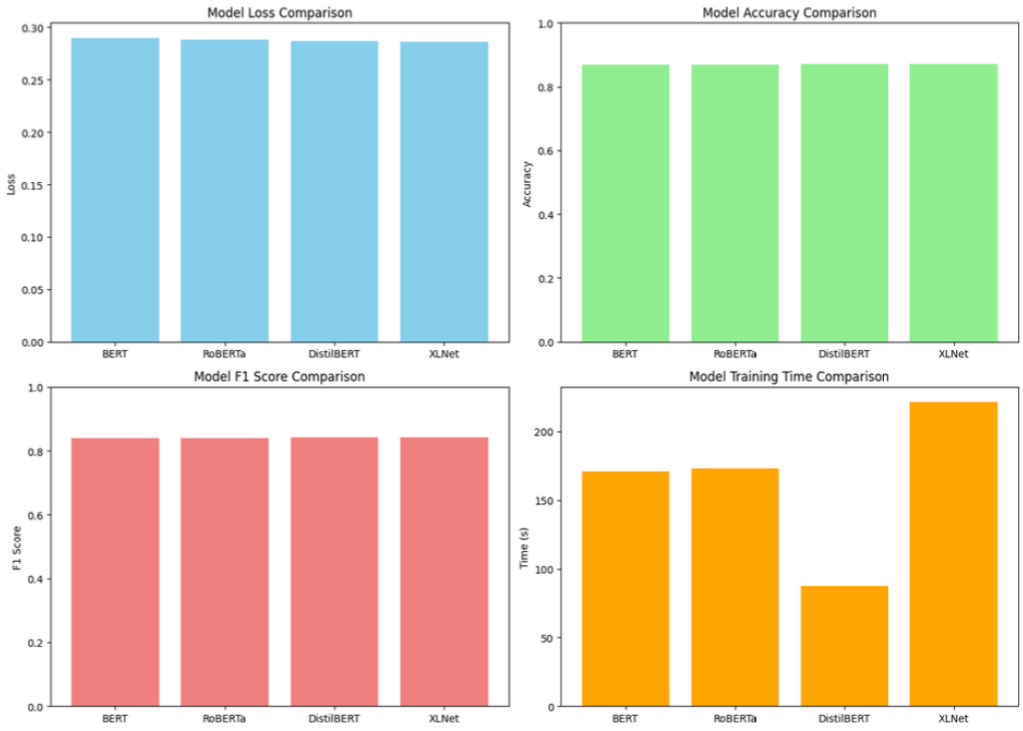

Figure 15: Comparison Visualization