AIL721: Deep Learning
Assignment 3
Q1. CNN-LSTM for Multi-Category Classification

SHREYAS SHIMPI
Entry Number: 2024AIB2291

April 7, 2025

0.1 CNN-LSTM Module

CNN Layer: The CNN layers convolve over the embedding matrix to detect local patterns in the text. Different kernel sizes are used to capture n-gram features.

LSTM Layer: The CNN outputs are then fed into an LSTM layer, which models the temporal dynamics and long-term dependencies of the sequence. This combination allows the model to exploit both local and sequential information.

0.2 Self-Attention Mechanism

After processing with LSTM, a self-attention layer computes attention scores over the sequence. This mechanism allows the network to focus on key tokens that contribute most significantly to the classification decision. The attention output is then passed to the final dense layer for classification.

1 Hyperparameters

The following hyperparameters were used in the model configuration:

- **max_len:** 500
- **embedding_dim:** 200
- **num_filters:** 150
- **lstm_hidden_dim:** 150
- **num_classes:** 5
- **dropout:** 0.5
- **batch_size:** 32
- **num_epochs:** 70
- **learning_rate:** 1e-3

2 Model Implementation

The following pseudocode summarizes the implementation of the proposed model:

Listing 1: Pseudocode for CNN-LSTM with Self-Attention and Dynamic Meta-Embedding

```
# Define input embeddings from multiple sources
input_embeddings = dynamic_meta_embedding(input_data)

# Feature extraction using CNN
cnn_features = CNN(input_embeddings)

# Sequence modeling with LSTM
lstm_output = LSTM(cnn_features)

# Apply self-attention to LSTM outputs
attention_scores = compute_attention(lstm_output)
attention_output = weighted_sum(lstm_output, attention_scores)

# Classification layer
logits = Dense(attention_output)

# Loss and optimizer setup
loss = cross_entropy(logits, labels)
optimizer = AdamOptimizer(loss)

# Training loop
for epoch in range(num_epochs):
    train_step()
```

3 Experiments and Results

3.1 Performance Metrics

Models were trained for 70 epochs. Performance was evaluated using test loss and micro F1 score. Table 1 summarizes the results obtained from various model variants.

Table 1: Test Performance of Various Model Variants

Model Variant	Test Loss	Test Metric	Attention	Meta Embedding
OHE	1.4407	F1: 0.4340	No	No
Embedding from Data	0.1743	F1: 0.9510	No	No
Meta-Embedding	0.1369	F1: 0.9673	No	Yes
Embedding + Self-Attention	0.2329	F1: 0.9442	Yes	No
Meta-Embedding + Self-Attention	0.1522	F1: 0.9592	Yes	Yes
GloVe Embedding	0.1335	F1: 0.9701	Yes	No
FastText Embedding	0.1146	F1: 0.9714	Yes	No
Word2Vec Embedding	0.1367	F1: 0.9660	Yes	No

3.2 Graphical Results

Below are the training loss and F1 score curves for different model variants. Each figure includes both loss and F1 score evolution over epochs.

Model Variant 1: OHE Embeddings

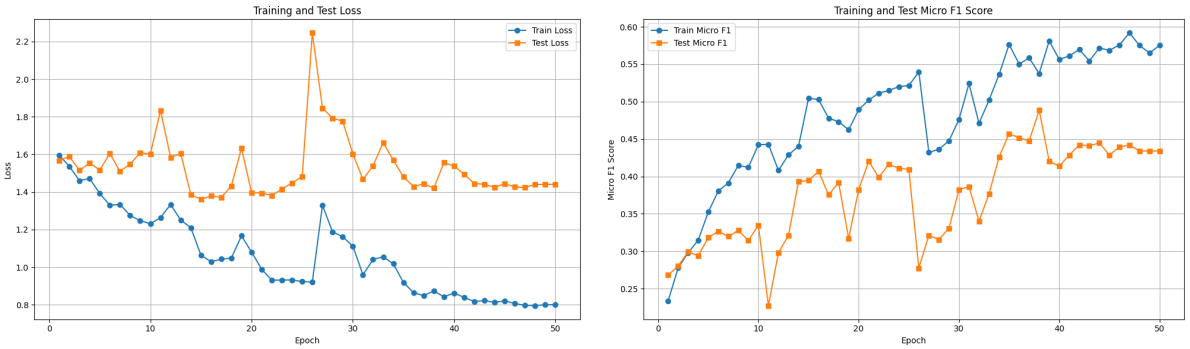


Figure 1: Loss and F1 Score for CNN-LSTM with OHE Embeddings

Model Variant 2: Embedding from Data

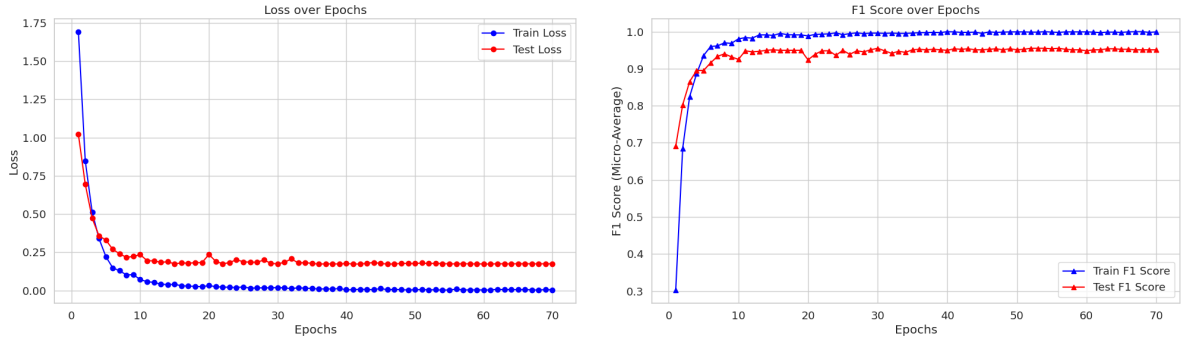


Figure 2: Loss and F1 Score for CNN-LSTM with Data Embedding

Model Variant 3: Meta-Embedding

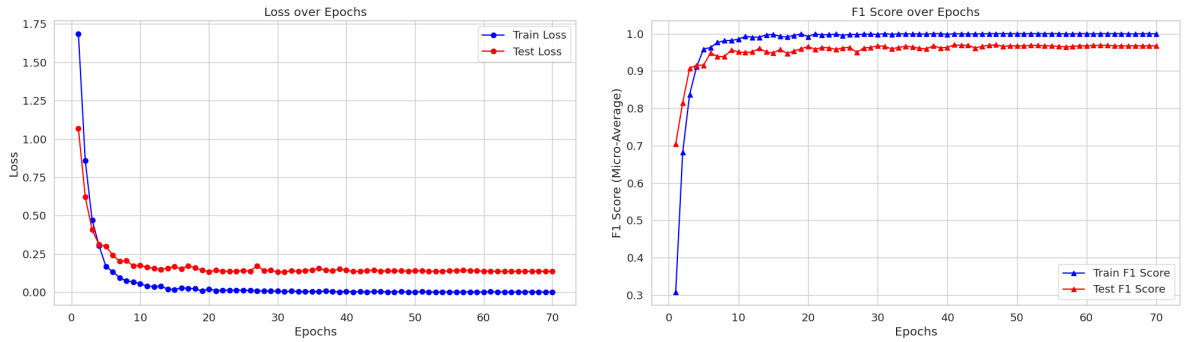


Figure 3: Loss and F1 Score for CNN-LSTM with Meta-Embedding

Model Variant 4: Embedding from Data + Self-Attention

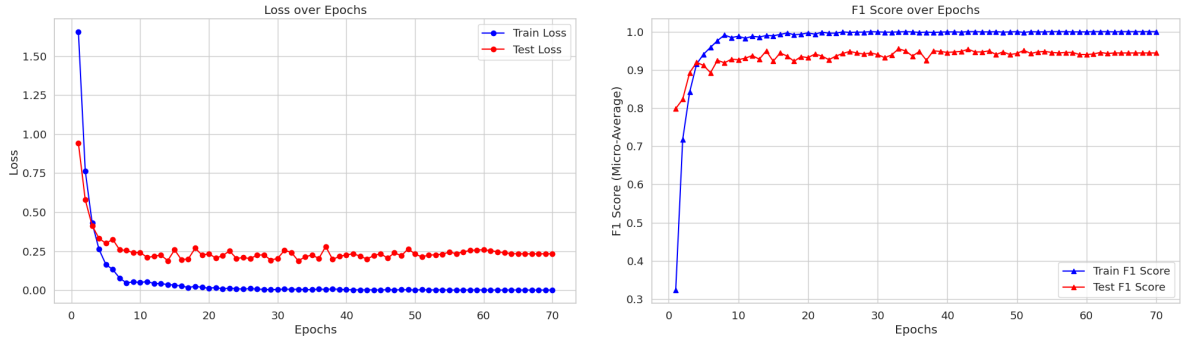


Figure 4: Loss and F1 Score for CNN-LSTM with Data Embedding + Self-Attention

Model Variant 5: Meta-Embedding + Self-Attention

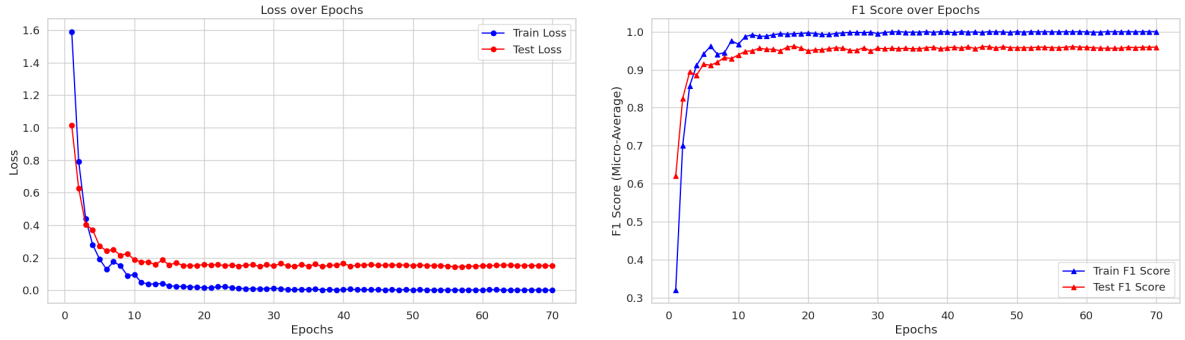


Figure 5: Loss and F1 Score for CNN-LSTM with Meta-Embedding + Self-Attention

Model Variant 6: GloVe Embedding

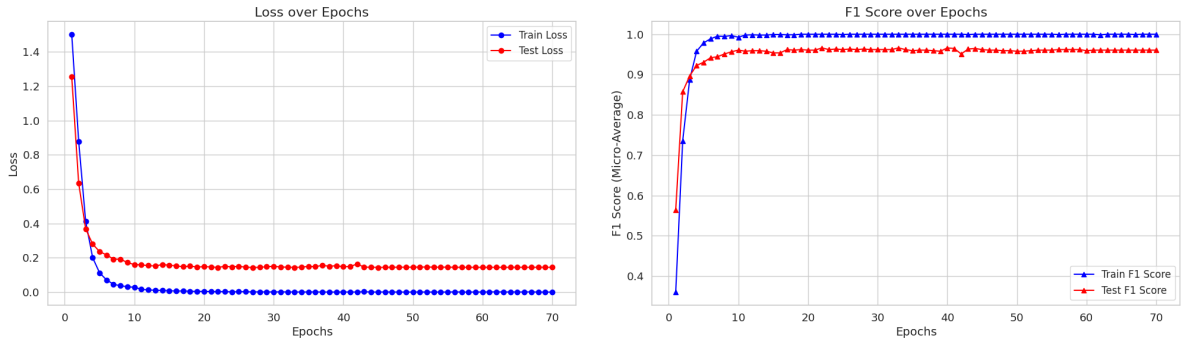


Figure 6: Loss and F1 Score for CNN-LSTM with GloVe Embedding

Model Variant 7: FastText Embedding

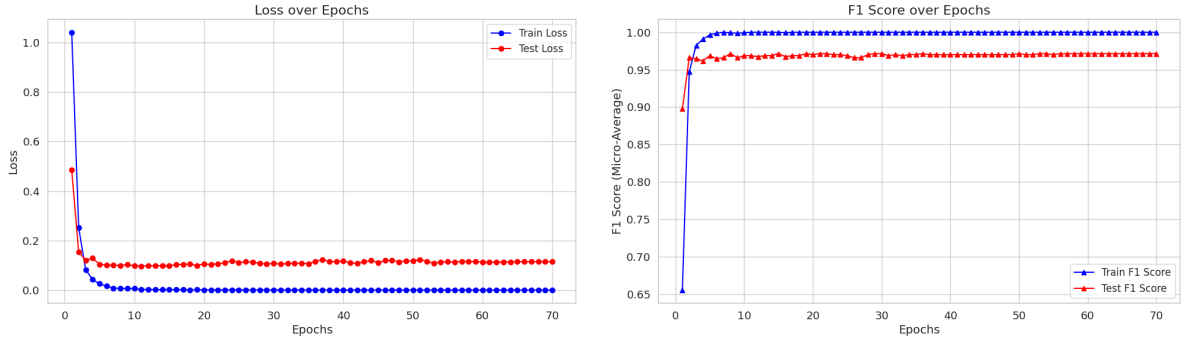


Figure 7: Loss and F1 Score for CNN-LSTM with FastText Embedding

Model Variant 8: Word2Vec Embedding

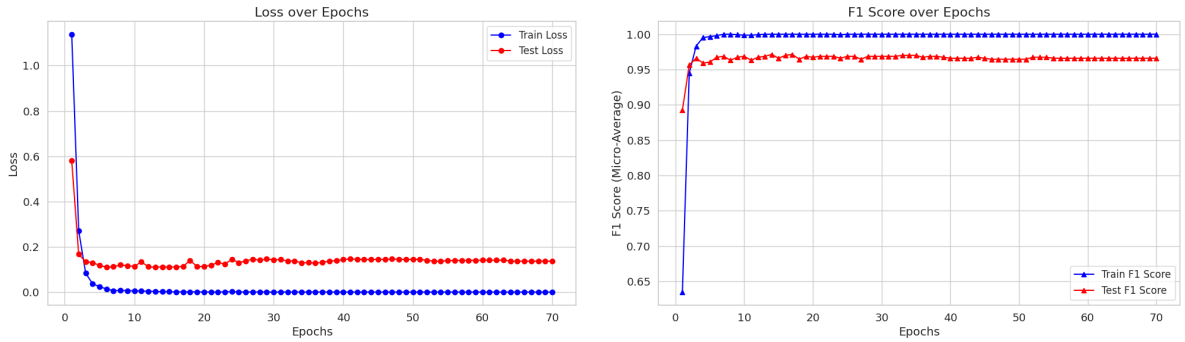


Figure 8: Loss and F1 Score for CNN-LSTM with Word2Vec Embedding

4 Conclusion

We developed a CNN-LSTM model enhanced with a self-attention mechanism for multi-category classification, and we evaluated its performance using embeddings trained solely on data & various pretrained embeddings: GloVe, Word2Vec, and FastText. Our experiments demonstrated that pretrained embeddings significantly boost performance compared to embeddings trained solely on the available data. Among the embeddings tested, FastText achieved the highest micro-average F1 score of **0.9714**, likely due to its ability to handle subword information and out-of-vocabulary words effectively.

Additionally, the incorporation of self-attention helped the model to focus on the most relevant tokens in the sequence, further refining its output representation and improving classification accuracy. These results underscore the importance of selecting robust embedding techniques and integrating attention mechanisms in neural architectures for text classification.

Q2. Transformer-based Text Classifier

5 Model Architecture

5.1 Architecture Overview

The classifier is built on an **encoder-only transformer** architecture. The main components are:

- **Embedding Layer:** Maps input tokens to a dense vector space of dimension $d_{\text{model}} = 128$. The embeddings are scaled by $\sqrt{d_{\text{model}}}$ to ensure stable gradients.
- **Learnable Positional Encoding:** A learnable parameter matrix is added to the token embeddings to incorporate sequence order information. This component can be enabled or disabled to study its effect.
- **Transformer Encoder Blocks:** Each block includes:
  - A multi-head self-attention mechanism with 4 attention heads.
  - A position-wise feed-forward network.
  - Residual connections and layer normalization.

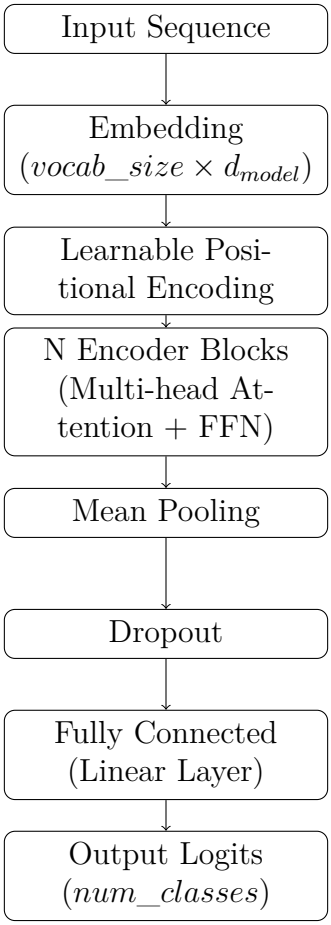
  Multiple encoder blocks (from 1 to 6) are stacked to learn progressively abstract representations.
- **Global Context Extraction:** Mean pooling is applied over the sequence dimension to obtain a global representation.
- **Output Layer:** A linear layer projects the global context to one of the five output classes.

5.2 Rationale for Architectural Choices

- **Transformer Architecture:** Self-attention effectively captures long-range dependencies, making it suitable for understanding complex, full-length news articles.
- **Multiple Encoder Blocks:** Stacking layers can help the model learn richer and more abstract representations. However, adding too many layers might lead to overfitting or training instability.
- **Learnable Positional Encoding:** This allows the model to adapt positional signals rather than relying on fixed sinusoidal patterns, potentially capturing task-specific positional features. However, in our experiments, positional encoding consistently decreased the F1 score. One possible explanation is that for this task, the sequence lengths are relatively fixed and the model may already capture sufficient positional information through its architecture. Additionally, the extra parameters introduced by learnable positional encoding might add noise or increase the risk of overfitting, particularly in deeper models.
- **Label Smoothing:** With a smoothing factor of 0.1, the loss function reduces overconfidence in predictions, which improves generalization.
- **Data Augmentation:** Techniques like random deletion are employed during training to enhance model robustness by exposing it to diverse sentence structures.

5.3 Architecture Diagram

Below is a concise diagram of the model architecture:



6 Hyperparameter Settings

The following hyperparameters were used in the experiments:

- $\mathbf{d_{model}} = \mathbf{128}$
- **Attention Heads** $= \mathbf{4}$
- **Dropout** $= \mathbf{0.5}$
- **Learning Rate** $= \mathbf{0.001}$
- **Batch Size** $= \mathbf{32}$
- **Max Sequence Length** $= \mathbf{500}$
- **Label Smoothing** $= \mathbf{0.1}$
- **Data Augmentation Probability** $= \mathbf{0.1}$

7 Experimental Evaluation

7.1 Test Set Performance

A series of experiments were conducted by varying:

1. The number of encoder blocks (1, 2, 3, 4, 5, and 6).
2. The usage of learnable positional encoding (enabled vs. disabled).

Table 2 summarizes the micro-average F1 scores achieved on the test set for each configuration.

Table 2: Micro-average F1 scores for different configurations of encoder blocks and positional encoding.

Configuration	Positional Encoding	Micro-F1 Score (ON TEST DATA)
Encoders_1	True	0.9374
Encoders_1	False	**0.9469**
Encoders_2	True	0.9388
Encoders_2	False	0.9388
Encoders_3	True	0.9442
Encoders_3	False	0.9184
Encoders_4	True	0.9333
Encoders_4	False	0.9265
Encoders_5	True	0.9401
Encoders_5	False	0.9197
Encoders_6	True	0.8952
Encoders_6	False	0.9333

7.2 Graphical Analysis

Figure 9 provides a visual representation of how the number of encoder blocks and the inclusion of positional encoding affect the micro-average F1 score.

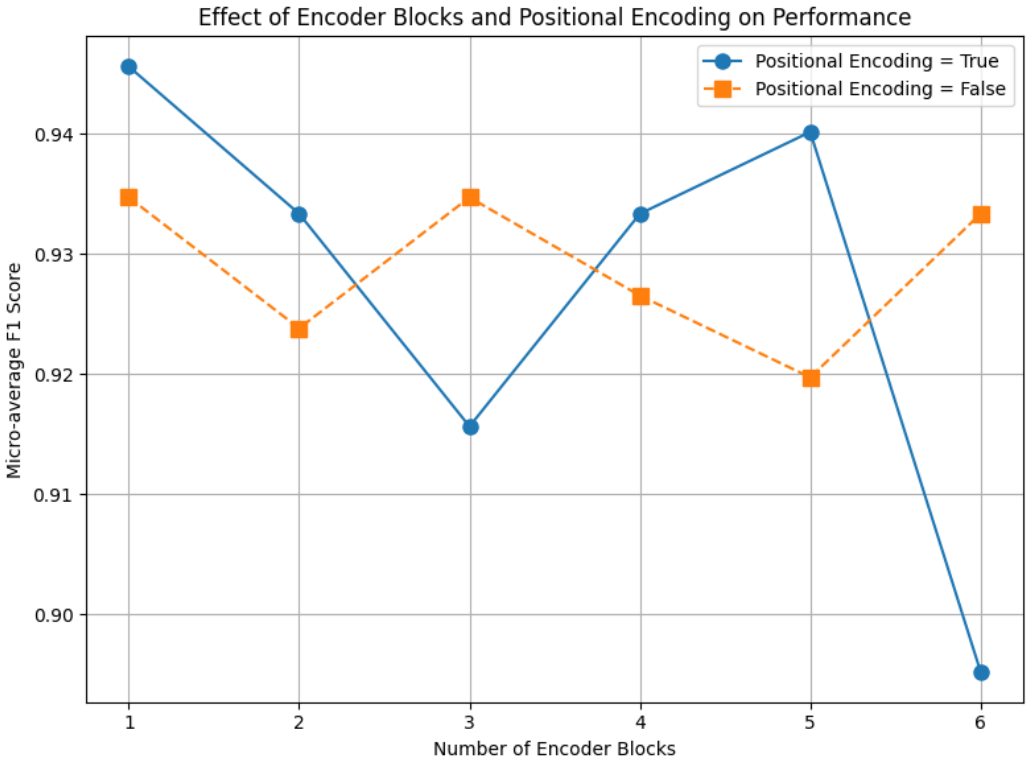


Figure 9: Effect of the Number of Encoder Blocks and Positional Encoding on Micro-average F1 Score.

7.3 Detailed Observations

From the experimental results and graphical analysis, several observations can be made:

- **Best Performing Configuration:** The highest micro-average F1 score of **0.9469** was

achieved with a single encoder block and without positional encoding. This suggests that for this task, a simpler model with minimal depth is sufficient.

- **Effect of Encoder Blocks:**
  - With only 1 encoder block, the performance is already very high, indicating that adding more layers might not be necessary.
  - As the number of encoder blocks increases (up to 6), the performance tends to vary and in some cases degrades. For example, the performance drops to 0.8952 with 6 blocks when positional encoding is used.
- **Role of Positional Encoding:**
  - In some configurations (e.g., 1 and 3 encoder blocks), enabling positional encoding leads to slightly lower performance compared to when it is disabled.
  - The extra parameters from learnable positional encoding might introduce additional noise or lead to overfitting, especially in deeper models, which can explain the observed decrease in F1 score.
  - The task's characteristics—such as relatively fixed sequence lengths—may already be well-handled by the model architecture, reducing the need for explicit positional cues.
- **Trade-off Consideration:** While positional encoding is theoretically beneficial for capturing sequence order, its impact in this experimental setting suggests that for certain configurations or datasets, it may introduce unnecessary complexity or interfere with the model's ability to generalize.

8 Conclusion

The experiments conducted provide valuable insights into the architecture of transformer-based text classifiers:

- A single encoder block without positional encoding yielded the best performance with a micro-average F1 score of **0.9469**. This indicates that for the given dataset and task, a simpler architecture can be highly effective.
- Increasing the number of encoder blocks did not consistently lead to better performance. In fact, deeper models (e.g., with 6 blocks) sometimes suffered from decreased performance, possibly due to overfitting or training instabilities.
- The inclusion of learnable positional encoding, while intended to help capture sequence order, appears to decrease performance in several configurations. This may be due to the additional noise introduced by the extra parameters, or the possibility that the model is already capturing sufficient positional information through its inherent structure. Furthermore, for tasks with fixed sequence lengths, the benefit of explicit positional encoding may be minimal.

In addition, the choice to employ an encoder-only model for the classification task is motivated by several key considerations. Encoder-only architectures are particularly effective in capturing the contextual relationships within the input data through self-attention mechanisms. This structure focuses solely on transforming the input sequence into a robust, fixed-length representation, which is ideally suited for classification problems. Unlike encoder-decoder architectures designed for sequence-to-sequence tasks, the encoder-only model reduces computational complexity while efficiently extracting and emphasizing the most relevant features necessary for accurate classification. This streamlined approach contributes to improved performance, especially when the task involves fixed-length sequences where subtle inter-token relationships are critical.