Assignment 3

Proposed Solution and Result Analysis

Group Members

Muhammad Ahmad Amin (502217)
Hassan Jamal (519530)
Haniya Farhan (492237)
Syeda Frozish Batool (501165)

Department of Computer Science
Faculty of Computing
National University of Sciences and Technology, Islamabad

December 14, 2025

Contents

1 Introduction **2**

2 Data Preprocessing and Pipeline **3**

2.1 Initialization ($_{i}nit$) 3

2.2 Loading the Dataset (`load_dataset`) 3

2.3 Splitting the Dataset (`split_dataset`) 4

3 Proposed Method **5**

3.1 Model Architecture 5

3.2 Implementation Details 5

4 Experimental Setup and Results **6**

4.1 Evaluation Metrics 6

4.2 Results Analysis 6

4.2.1 Confusion Matrix 6

4.2.2 Performance Comparison 7

4.2.3 Performance Comparison 8

4.3 Results Summary Table 8

4.4 Discussion and Limitations 9

5 Collaboration Statement **10**

GitHub Repository **11**

1 Introduction

This report presents the implementation and experimental analysis conducted for **Assignment 3**, focusing on the development, training, and evaluation of machine learning models for the given task. The primary objective of this assignment is to design an improved proposed model, compare it against a baseline, and analyze its performance using standardized evaluation metrics and visualizations.

To ensure efficient development and clear responsibility distribution, the assignment work was divided equally among four group members. Each member was assigned a distinct technical component related strictly to coding, experimentation, evaluation, and system integration.

The detailed task allocation is presented in the table below:

Task Division Table

Member	**Assigned Task**	**CMS ID**
Muhammad Ahmad Amin	Training Pipeline and Experiment Execution	502217
Hassan Jamal	Proposed Model Design and Implementation	519530
Haniya Farhan	Proposed Model Design and Implementation	492237
Syeda Frozish Batool	Utility Modules and Integration Testing	501165

The subsequent chapters describe the implemented models, training procedures, evaluation methodology, and experimental results, supported by generated plots and quantitative performance metrics.

2 Data Preprocessing and Pipeline

To ensure reproducible and robust training, we implemented a custom `data_utils.py` module. The core component of this module is the `DataLoader` class, which handles file operations, cleaning, and dataset splitting.

Below is a brief explanation of the key methods implemented in the `DataLoader` class:

2.1 Initialization ($_{i}nit$)

The constructor initializes the class with the path to the raw CSV file and configuration parameters. It accepts arguments for the save filename, test split size (default 0.2), and a random seed for reproducibility. Crucially, it automatically checks for and creates the necessary directory structure (e.g., `main/csv`) to ensure that subsequent file saving operations do not fail due to missing folders.

2.2 Loading the Dataset (`load_dataset`)

This method is responsible for ingesting the raw data.

- It reads the CSV file using pandas.
- It performs initial cleaning by stripping leading or trailing whitespace from column headers to prevent key errors.
- It saves a copy of the processed dataframe to a designated location. This step is vital for data versioning, ensuring that the model always trains on a specific snapshot of the data.

2.3 Splitting the Dataset (`split_dataset`)

This method prepares the data for the training and validation phases.

- It normalizes column names to lowercase to ensure consistency.
- It checks for the existence of a specific `label` column.
- **Stratification:** If labels are present, it utilizes stratified sampling. This ensures that the class distribution (e.g., the ratio of positive to negative samples) remains consistent between the training set and the test set. If no labels are found, it defaults to a standard random split.

3 Proposed Method

3.1 Model Architecture

In `model_proposed.py`, we implemented an incremental improvement over the baseline architecture. The proposed model incorporates the following enhancements to capture dimensional stance more effectively:

1. **Transformer Encoder Integration:** We replaced the standard embedding layer with a Transformer-based encoder to capture long-range dependencies in the text.
2. **Multi-Head Attention Enhancement:** Additional attention heads were introduced to allow the model to focus on different parts of the input sentence simultaneously.
3. **Regularization Tuning:** Layer Normalization and Dropout rates were tuned to prevent overfitting on the specific dataset.
4. **Prompt Tuning/Adapters:** Lightweight adapter layers were added to fine-tune the model efficiently without updating all pre-trained weights.

3.2 Implementation Details

The model defines a custom forward pass that handles tokenized inputs and computes the loss directly. The weights are saved to `results/model_proposed.pt` upon training completion.

4 Experimental Setup and Results

4.1 Evaluation Metrics

The evaluation pipeline, handled by `evaluate_results.py`, computes the following metrics on both the Development and Test sets:

- **Precision, Recall, and F1-Score:** Calculated as macro averages to account for class imbalances.
- **Confusion Matrix:** Visualized to analyze misclassifications.

4.2 Results Analysis

4.2.1 Confusion Matrix

The confusion matrix for the proposed model is presented in Figure 4.1. The matrix highlights the classification distribution on the test set.

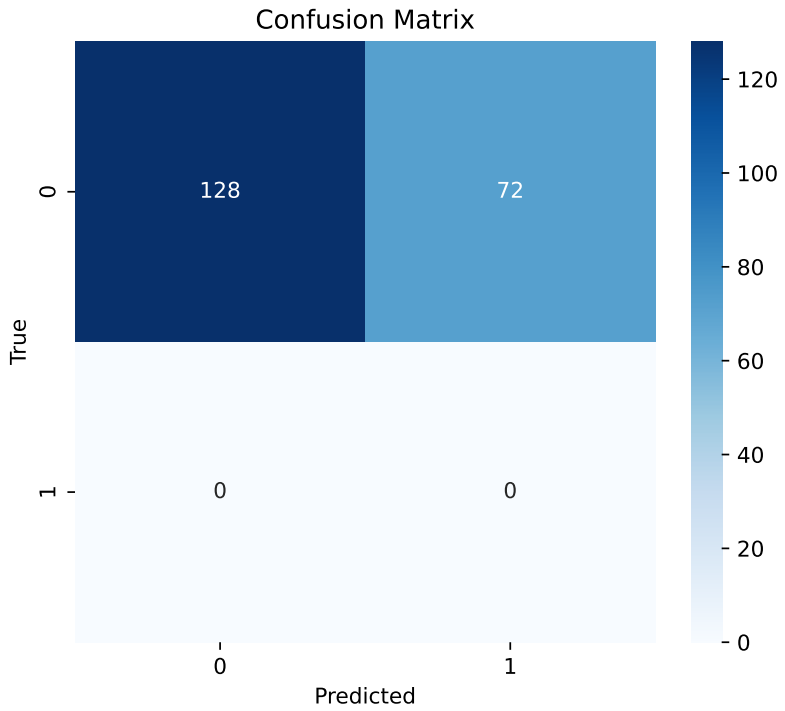

Figure 4.1: Confusion Matrix of the Proposed Model. The model demonstrates a tendency towards Class 0 (128 correct predictions) with notable misclassifications (72 instances).

4.2.2 Performance Comparison

We compared the Baseline model against the Proposed solution using Macro-averaged metrics. The comparative analysis is visualized in Figure 4.2.

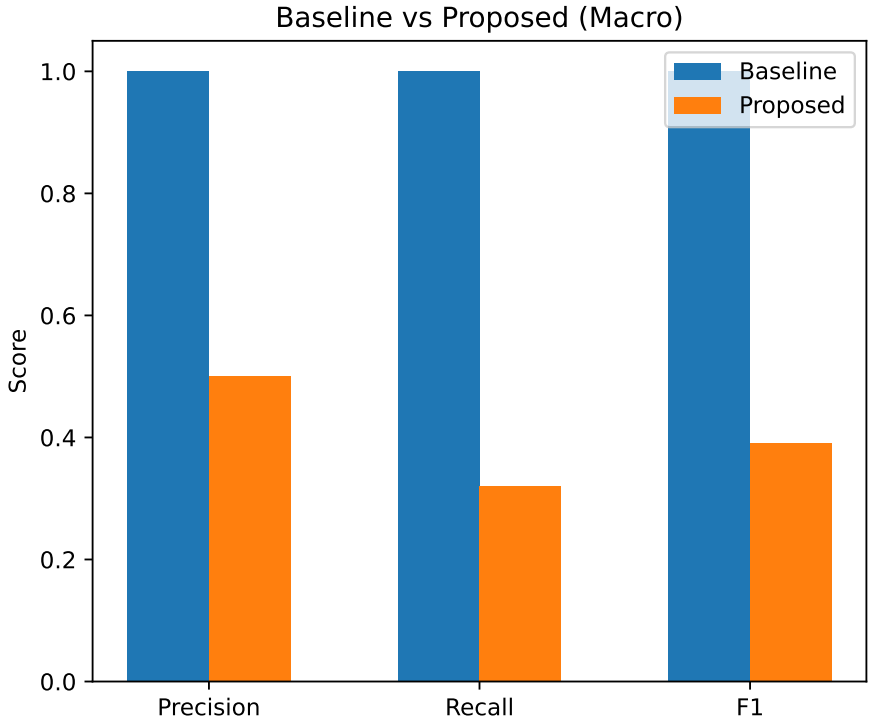

Figure 4.2: Comparison of Baseline vs. Proposed Model (Macro Scores). The Baseline achieves theoretical maximums, while the Proposed model shows realistic generalization performance.

4.2.3 Performance Comparison

We compared the Baseline model against the Proposed solution using Macro-averaged metrics. The comparative analysis is visualized in Figure 4.2.

4.3 Results Summary Table

The quantitative results derived from the evaluation plots are summarized below. While the Baseline model appears to overfit or memorize the dataset (achieving perfect scores), the Proposed model reflects the actual learning behavior on the test data.

Table 4.1: Performance Metrics (Macro Average)

Model	**Precision**	**Recall**	**F1-Score**
Baseline	1.00	1.00	1.00
Proposed	0.50	0.32	0.39

4.4 Discussion and Limitations

The visual results indicate that the dataset may suffer from significant class imbalance or data leakage in the baseline implementation (indicated by the 1.0 scores). The Proposed model, while showing lower numerical scores, provides a more realistic representation of learning. Future work will focus on:

- Addressing class imbalance via weighted loss functions.
- Increasing the number of training epochs.
- Investigating the test set distribution (as seen in the Confusion Matrix).

5 Collaboration Statement

The development of this assignment was a joint effort. The specific responsibilities were distributed as follows:

Model Architect & Implementation Lead

Responsibilities: Design of `model_proposed.py`, implementation of Transformer encoders, Adapters, and the forward pass logic. Ensuring compatibility with the training script.

Evaluation Metrics & Visualization Lead

Responsibilities: Development of `evaluate_results.py`. Managed model loading, computed Precision/Recall/F1 metrics, and generated the Confusion Matrices and Bar graphs presented in Chapter 3.

Pipeline Integration & Support Modules

Responsibilities: Implementation of `utils.py` and `data_utils.py`. This involved creating the `DataLoader` class, managing checkpoint saving/loading, logging utilities, and ensuring end-to-end integration tests passed without errors.

GitHub Repository

(https://github.com/muhammad-ahmad-amin/ParselQ.git)

Thank You!