Tamil Offensive Language Classification Via Transformer(BERT Model)

1. Project Goal

The primary objective of this project was to classify Tamil text into one of **six distinct offensive/non-offensive categories**. The task is a **multi-class text classification** problem, aimed at identifying the nature and target of offensive content in Tamil social media text.

2. Dataset Overview

The model was trained and evaluated using the **Tamil Offensive Language Detection Dataset**.

Training and Validation Data

Component	**File Path**	**Size (Samples)**	**Column Names Used**	**Purpose**
Training Set	tamil_offensive_full_train.csv	35,139	Text, Labels	Model weight optimization.
Validation Set	tamil_offensive_full_dev.csv	4,388	Text, Labels	Hyperparameter tuning and checkpointing (selecting the best epoch).
Final Test Set	tamil_offensive_full_test_with_labels.csv	4,392	Text, Labels	**Final, unbiased performance evaluation.**

Label Classes

The dataset uses **6 unique classes** for classification:

Numerical Label	String Label	Description
0	Not_offensive	Text is non-offensive.
1	Offensive_Targeted_Insult_Group	Offensive towards a group.
2	Offensive_Targeted_Insult_Individual	Offensive towards an individual.
3	Offensive_Targeted_Insult_Other	Offensive towards other specific entities.
4	Offensive_Untargetede	Offensive but without a clear target.
5	not-Tamil	Text is not in Tamil (used for data filtering/rejection).

3. Preprocessing and Encoding

Text Cleaning

The text preprocessing function cleaned the input by:

- Removing characters that are **not** letters, spaces, or Tamil/Indic unicode characters.
- Collapsing multiple spaces into a single space and stripping leading/trailing whitespace.

Tokenization and Encoding

- The **ai4bharat/indic-bert** pre-trained tokenizer was used.
- Text sequences were truncated or padded to a maximum length of **128 tokens**.
- The raw string labels were converted to numerical indices (0-5) using **LabelEncoder**, fitted exclusively on the training set to prevent data leakage.

4. Model Architecture: IndicBERT-LSTM-CNN

The classification model uses a **hybrid neural network** approach:

1. **IndicBERT (Frozen Feature Extractor):** The pre-trained **ai4bharat/indic-bert** model acts as the initial layer. It generates **contextualized embeddings** for the input sequence. Critically, its weights were **frozen (torch.no_grad())** during training to preserve its linguistic knowledge and speed up the training process. The output dimensionality is **768**.
2. **LSTM Layer:** The sequential output (last hidden state) from BERT is passed to an **LSTM layer** with a hidden_dim of **128**. The LSTM captures **long-range dependencies and sequential patterns** within the word embeddings.
3. **1D Convolutional Layer (CNN):** The output of the LSTM is processed by a **1D Convolutional layer** with 128 output channels and a kernel size of 3. This layer is designed to automatically learn **local, high-level features** (n-grams/phrase patterns) from the sequence.
4. **Global Average Pooling: Global Average Pooling** is applied across the sequence dimension of the CNN output to produce a fixed-size feature vector, summarizing the most important features learned by the CNN.
5. **Output Layer:** A final **Linear layer** maps the pooled feature vector (128 dimensions) to the **6 class logits** for classification.

5. Training and Configuration

Parameter	**Value**
Pre-trained Model	ai4bharat/indic-bert
Epochs	5
Batch Size	32
Learning Rate	2×10^{-5}
Loss Function	nn.CrossEntropyLoss()
Optimizer	torch.optim.Adam
Device	**CUDA** (GPU)

The model checkpointing strategy saved the best model based on the highest **Weighted F1-score** achieved on the **separate validation set** (tamil_offensive_full_dev.csv).

6. Training and Validation Results

The training progressed over 5 epochs, with the best model being saved after the final epoch.

Validation Performance (Weighted Average)

Epoch	Training Loss	Validation Loss	Accuracy	Precision	Recall	F1-score	Best Model?
1	0.9056	0.8784	0.7331	0.5679	0.7331	0.6253	✅
2	0.8621	0.8531	0.7416	0.5693	0.7416	0.6401	✅
3	0.8447	0.8434	0.7420	0.6530	0.7420	0.6407	✅
4	0.8338	0.8354	0.7429	0.6198	0.7429	**0.6438**	✅
5	0.8275	0.8325	0.7427	0.6358	0.7427	**0.6442**	✅

The model achieved its best weighted F1-score of **0.6442** at Epoch 5.

7. Final Test Set Evaluation

The model (state from Epoch 5) was evaluated on the held-out **tamil_offensive_full_test_with_labels.csv** to determine its final, expected performance.

Overall Macro Metrics

Metric	Value

Loss	0.8564
Accuracy	0.7375
Precision (Macro)	0.4613
Recall (Macro)	0.2236
F1-score (Macro)	**0.2273**

Detailed Classification Report (Test Set)

Class Label	**Precision**	**Recall**	**F1-score**	**Support**
Not_offensive	**0.74**	**1.00**	**0.85**	3190
Offensive_Targeted_Insult_Group	0.50	0.02	0.04	288
Offensive_Targeted_Insult_Individual	0.00	0.00	0.00	315
Offensive_Targeted_Insult_Other	0.00	0.00	0.00	71
Offensive_Untargetede	0.67	0.01	0.01	368
not-Tamil	0.86	0.32	0.47	160
Macro Avg	0.46	0.22	0.23	4392

Weighted Avg	0.66	0.74	0.64	4392

8. Conclusion and Future Work

Conclusion

The model achieved a **Weighted F1-score of 0.6442** on the validation set, which translates to a high **Weighted F1-score of 0.64** and **Accuracy of 73.75%** on the final test set.

However, the **Macro F1-score of 0.2273** and the detailed classification report highlight a severe **class imbalance issue** in the dataset and a failure of the model to effectively classify the minority "Offensive" categories. The model shows:

- **Excellent** performance on the majority class (Not_offensive, F1=0.85), often correctly predicting the large number of non-offensive texts.
- **Extremely poor** performance on all targeted and untargeted offensive classes (F1 near 0.00 to 0.04), indicating these minority classes are almost never correctly predicted. The high precision for some offensive classes (e.g., Offensive_Untargetede: 0.67) is misleading, as the recall (0.01) shows it only correctly identified a tiny fraction of total instances.

Recommendations for Future Work

1. **Address Class Imbalance:** Implement **re-sampling techniques** (e.g., SMOTE, oversampling minority classes) or utilize **cost-sensitive learning** by adjusting the weights of the nn.CrossEntropyLoss function to penalize misclassifications of minority classes.
2. **Fine-tuning BERT:** Instead of freezing the IndicBERT weights, **fine-tuning** them alongside the LSTM/CNN layers could capture more language-specific and task-relevant representations, significantly improving performance on specific classes.
3. **Hyperparameter Optimization:** Conduct a grid search or randomized search for optimal LSTM hidden_dim, CNN kernel size, learning rate, and batch size.
4. **Model Architecture:** Experiment with simpler architectures (e.g., just fine-tuned BERT) or more complex hybrid models, potentially including attention mechanisms (Self-Attention) after the LSTM.

Group Members and Contributions

Roll Number	Name	Contribution
2201CS51	Kankshitha Reddy	8.3%
2201CS52	Sasya Sri Harshitha	8.3%
2201CS53	Prachi Dhiraj Singh Thakur	8.3%
2201CS21	Srikar	8.3%
2201CS79	Karthik Reddy	8.3%
2201CS34	Kartheek Sai	8.3%
2201AI11	Uttej	8.3%
2201AI17	Sai Pratheek Reddy	8.3%
2201CS78	Vishwas	8.3%
2201CS75	Varthika	8.3%
2201CS37	Kavya Sree	8.3%
2201AI18	Kathyayani	8.3%