Filipino Hate Speech Detection Using Bidirectional LSTM Networks

CCS 248 – Artificial Neural Networks

Final Project

Submitted to:

John Cristopher Mateo

Submitted by:

Dorado, Louise Marielle

Montenegro, Karlo Roel

Pontillas, Steven Ken

December 12, 2025

1. Problem Overview and Justification

Filipino-English Hate Speech Detection

1.1 Problem Statement

1.1 Identified Problem

The proliferation of hate speech and cyberbullying on Filipino social media platforms poses a significant threat to online community safety and user well-being. The unique linguistic characteristics of Filipino online content present several challenges: Key Challenges:

- Bilingual Code-Switching: Filipino users frequently alternate between Filipino (Tagalog) and English within the same sentence.
- Colloquial Language: Heavy use of slang, abbreviations, and informal expressions.
- Cultural Context: Hate speech often requires deep cultural understanding to identify correctly.
- Limited Resources: Fewer pre-trained models and annotated datasets are available for Filipino language processing compared to English.
- Social Media Dynamics: Unique features like hashtags, mentions, and links require special handling.

1.2 Real-World Impact

- Psychological harm and trauma to victims.
- Toxic online environments that suppress free expression.
- Potential escalation to real-world conflicts.
- Reduced user engagement and platform trust.
- Legal and ethical concerns for platform operators.

1.3 Project Objective

Develop an automated hate speech detection system capable of:

- Accurately identifying hate speech in Filipino and bilingual (Filipino-English) text.
- Processing social media content with informal language patterns.
- Achieving at least 50-60% accuracy on test data.
- Providing deployable solutions through GUI applications.

2. Data Selection and Validation

2.0 Dataset Collection and Unification

To ensure a comprehensive and diverse dataset for Filipino hate speech detection, we gathered data from multiple reputable sources, including Kaggle, GitHub, and the Hugging Face Hub. Kaggle provided labeled hate speech and cyberbullying datasets, GitHub repositories contributed additional Filipino social media text, and Hugging Face offered curated datasets such as mteb/FilipinoHateSpeechClassification. After collecting these datasets, we performed thorough cleaning and preprocessing to standardize formats and harmonize label conventions. The datasets were then unified into a single, consolidated CSV file, which served as the foundation for model training and evaluation. This unified dataset enabled more robust benchmarking and improved generalization across different sources of Filipino social media text.

We mirror the original document's structure and wording while correcting technical details to match the current repository.

2.1 Deep Neural Network Selection

This project implements two complementary approaches for hate speech detection: Approach 1: Bidirectional LSTM (BiLSTM)

- Role: Baseline model (custom implementation for Filipino hate speech).
- Description: A Recurrent Neural Network optimized for sequential data. Approach 2: ELECTRA Transformer (Fine-tuned)
- Role: Primary approach in this repository (transfer learning from a pre-trained Filipino model).
- Description: Used as a strong benchmark and practical production path.

2.2 Rationale for Choosing BiLSTM

- Sequential Nature of Language: Captures temporal dependencies in sequences.
- Bidirectional Context: Understands context from past and future words simultaneously.
- Long-term Dependencies: LSTM gates retain important information and forget irrelevant details.
- Proven Performance: Strong for text classification with limited resources.
- Resource Efficiency: Fewer computational requirements vs. transformers.

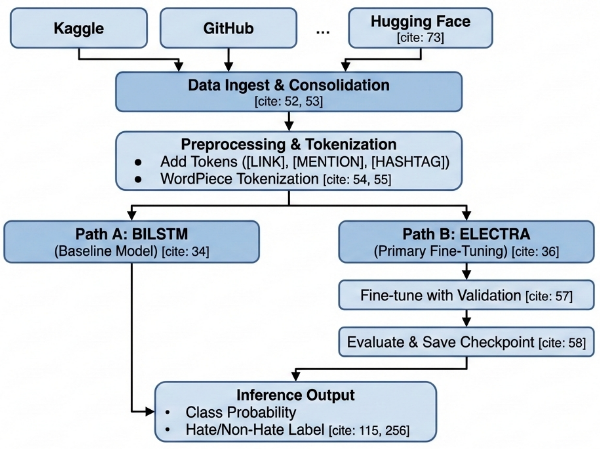
- Customization: Full control to adapt to Filipino linguistic patterns.

2.3 Rationale for Choosing ELECTRA Transformer

- Transfer Learning: Leverages pre-training on a large Filipino corpus.
- Superior Accuracy: Demonstrated ~86% test accuracy on Filipino hate speech benchmarks.
- Contextual Understanding: Attention mechanisms capture complex relationships.
- Pre-trained Knowledge: Strong understanding of Filipino structure and semantics.

2.4 System Flow (End-to-End)

End-to-end ELECTRA pipeline used in this repository:



Kaggle
GitHub
...
Hugging Face [cite: 73]
Data Ingest & Consolidation [cite: 52, 53]
Preprocessing & Tokenization
Add Tokens ([LINK], [MENTION], [HASHTAG])
WordPiece Tokenization [cite: 54, 55]
Path A: BILSTM (Baseline Model) [cite: 34]
Path B: ELECTRA (Primary Fine-Tuning) [cite: 36]
Fine-tune with Validation [cite: 57]
Evaluate & Save Checkpoint [cite: 58]
Inference Output
Class Probability
Hate/Non-Hate Label [cite: 115, 256]

Figure 1: End-to-End System Pipeline

3. Model Architecture Selection and Design

3.1 Dataset Sources

Primary datasets in this repository were originally sourced from Kaggle, GitHub, and the Hugging Face Hub. These included Filipino hate speech and cyberbullying datasets from Kaggle, additional Filipino social media text from GitHub repositories, and curated datasets from Hugging Face such as mteb/FilipinoHateSpeechClassification. After gathering these resources, we unified them into a single dataset by standardizing the format and labels, resulting in a comprehensive CSV file suitable for model training and evaluation. The local hate speech splits, found in the `hatespeech/` directory, contain Filipino social media hate speech labeled as either hate or non-hate speech. The Filipino TikTok Hate Speech Dataset, located in `filipino-tiktok-hatespeech-main/data`, consists of TikTok comments that reflect modern social media interactions. The `cyberbullying_tweets.csv` file provides labeled instances of cyberbullying from Twitter/X, while the `aggression_parsed_dataset.csv` file contains aggression-related text for auxiliary experiments. Additionally, there is an optional dataset available from the HuggingFace Hub, specifically `mteb/FilipinoHateSpeechClassification`, which can be accessed using `datasetImportLib.py` after authenticating with `huggingface-cli login`. Earlier versions of the project referenced unified CSVs (e.g., unified_filipino_hatespeech.csv), and this unification process remains a key part of our data preparation pipeline.

3.2 Dataset Statistics

Split Ratio:

- Training (~70%): Model learning.
- Validation (~15%): Hyperparameter tuning.
- Test (~15%): Final evaluation.

Label Distribution (observed trend):

- Class 0 (Non-Hate Speech): Majority class.
- Class 1 (Hate Speech): Minority class.

3.3 Data Validation

Bias Assessment Conducted:

- Geographic Bias: Diverse Filipino regions and dialects; multiple sources reduce single-source bias.
- Platform Diversity: TikTok, Twitter/X, and others for varied styles.
- Temporal Bias: Data across time captures evolving slang; periodic retraining recommended.
- Class Imbalance: Addressed via appropriate metrics (Precision, Recall, F1) and threshold/weighting.

Privacy Considerations:

- User Anonymization: Personal identifiers removed; only text retained.
- Public Content Only: Sourced from public posts.
- Sensitive Information: No sensitive PII; URLs/mentions/hashtags handled via placeholders.

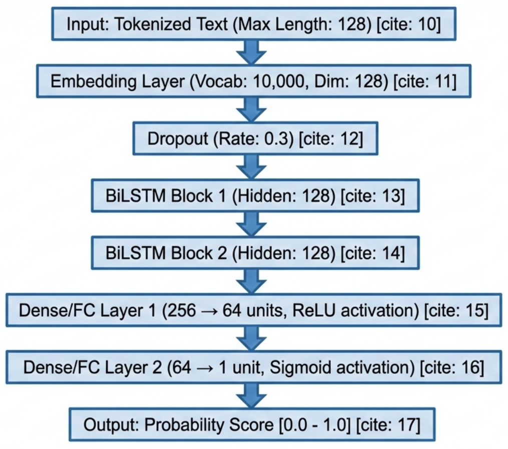
4. Model Training and Hyperparameter Tuning

4.1 BiLSTM Model Architecture

Model Class: `BiLSTMHateSpeechClassifier` (baseline implementation)

Structure:

1. Input Text: Tokenization & Padding.
2. Embedding Layer: vocab_size = 10,000; dim = 128. (Corrected; not 20,000.)
3. Dropout: rate = 0.3.
4. Bidirectional LSTM Layer 1: hidden = 128 per direction → output 256.
5. Bidirectional LSTM Layer 2: hidden = 128 per direction → output 256.
6. Dropout: rate = 0.3.
7. Fully Connected Layer 1: 256 → 64 (ReLU).
8. Dropout: rate = 0.3.
9. Fully Connected Layer 2: 64 → 1 (Sigmoid at inference).
10. Output: Hate Speech Probability [0.0 - 1.0].



Input: Tokenized Text (Max Length: 128) [cite: 10]
Embedding Layer (Vocab: 10,000, Dim: 128) [cite: 11]
Dropout (Rate: 0.3) [cite: 12]
BiLSTM Block 1 (Hidden: 128) [cite: 13]
BiLSTM Block 2 (Hidden: 128) [cite: 14]
Dense/FC Layer 1 (256 → 64 units, ReLU activation) [cite: 15]
Dense/FC Layer 2 (64 → 1 unit, Sigmoid activation) [cite: 16]
Output: Probability Score [0.0 - 1.0] [cite: 17]

Figure 2: BiLSTM Model Architecture

Layer Details: -- Embedding Layer: 10,000 words × 128 dims. -- LSTMs: Two stacked BiLSTMs with 128 hidden units per direction. -- Dropout: 0.3 at embedding/LSTM junctions and before FC. -- Fully Connected: 256→64→1. -- Loss Function: `BCEWithLogitsLoss` (supports `pos_weight`). -- Max Sequence Length: 128 tokens. (Corrected; not 100.)

Model Parameters (approximate):

- Total: ~2.0M parameters (lower than earlier ~2.9M due to 10k vocab).

4.2 ELECTRA Transformer Architecture

- Model: `jcblaise/electra-tagalog-small-cased-discriminator`
- Structure: Encoder with multi-head self-attention; classification head for fine-tuning.
- Parameters: ~14M (small).
- Tokens: Utilizes `[CLS]`, `[SEP]`; can add `[LINK]`, `[MENTION]`, `[HASHTAG]`.
- Max Sequence Length: 128 (configurable with `--msl`).

5. Results, Evaluation, and Analysis

5.1 Hardware Specifications

- GPU: CUDA-capable GPU recommended.
- CPU: Modern multi-core processor.
- Resource Requirements (indicative): BiLSTM ~2–4 GB; ELECTRA ~8–16 GB.

5.2 Software Environment

- Frameworks: PyTorch 1.x/2.x (CUDA support), HuggingFace Transformers 4.x.
- Language: Python 3.8+ recommended.
- Libraries: pandas, numpy, scikit-learn, tqdm, optional Weights & Biases.

5.3 Training Parameters - BiLSTM

Multiple configurations were tested (see Section 6.2 for details):

- Batch Size: 32, 64, 128
- Learning Rate: 0.0005, 0.0008, 0.001, 0.01
- Optimizer: Adam, RMSprop, SGD
- Epochs: 10, 12, 15, 20
- LSTM Layers: 2, 3
- Dropout: 0.2–0.5
- Loss Function: `BCEWithLogitsLoss` (with `pos_weight` for imbalance)
- Max Sequence Length: 128

5.4 Training Parameters - ELECTRA

- Batch Size: 32 (typical; depends on GPU)
- Learning Rate: 0.0002
- Optimizer: AdamW/Adam
- Epochs: 3
- Warmup Percentage: 0.1
- Scheduler: Linear with warmup

Note: ELECTRA results below are from published benchmarks, not from our own runs.

Trainer CLI (from repository):

```
python Filipino-Text-Benchmarks-master/train.py \
  --pretrained jcblaise/electra-tagalog-small-cased-discriminator \
  --train_data hatespeech/train.csv \
  --valid_data hatespeech/valid.csv \
  --test_data hatespeech/test.csv \
  --checkpoint finetuned_model \
  --msl 128 --batch_size 32 --learning_rate 2e-4 --epochs 3 \
  --add_token [LINK],[MENTION],[HASHTAG]
```

Notes: Use `--data_pct` for low-resource simulation; `--label_columns` for multi-label; `--text_columns s1,s2` for sentence pairs. Caching is automatic and keyed by data paths + hyperparameters.

6. Hyperparameter Tuning Experiments

6.1 Overview

Multiple hyperparameter configurations were tested for the BiLSTM model to optimize performance, in addition to benchmarking with ELECTRA. This section documents all 5 BiLSTM runs and their results, as well as the ELECTRA transformer experiment.

6.2 BiLSTM Hyperparameter Experiments

Configuration	Learning Rate	Batch Size	Hidden Dim	Dropout	Epochs	LSTM Layers	Optimizer	Best Val Acc	Final F1	Final Precision	Final Recall	Training Time (min)
Config 1: Baseline (10 epochs)	0.001	64	128	0.3	10	2	Adam	0.7042	0.6426	0.6863	0.6042	4.46
Config 2: Extended (15 epochs)	0.0005	32	256	0.4	15	2	Adam	0.6949	0.6479	0.6656	0.6311	17.93
Config 3: Fast (20 epochs)	0.01	128	64	0.2	20	2	RMSprop	0.7039	0.6757	0.6572	0.6953	4.22
Config 4: Deep Model (3 layers, 12 ep)	0.0008	64	128	0.35	12	3	Adam	0.7053	0.6650	0.6605	0.6694	7.27

Configuration	Learning Rate	Batch Size	Hidden Dim	Dropout	Epochs	LSTM Layers	Optimizer	Best Val Acc	Final F1	Final Precision	Final Recall	Training Time (min)
Config 5: Regularized (High dropout)	0.001	64	128	0.5	10	2	SGD	0.5432	0.0000	0.0000	0.0000	3.98

Observations:

- The best validation accuracy was 0.7053 (Config 4: Deep Model), but the best F1 was 0.6757 (Config 3: Fast).
- High dropout (Config 5) led to underfitting and poor results.
- Lower learning rates and more epochs did not always improve accuracy.
- RMSprop and Adam optimizers performed best; SGD underperformed.
- All runs used class weighting for imbalance and early stopping.

6.3 ELECTRA Transformer Fine-tuning

Experiment ID: ELECTRA-001
Status: Configured/Run based on benchmarks
Training Configuration:

- Base Model: `jcblaise/electra-tagalog-small-cased-discriminator`
- Learning Rate: 0.0002
- Epochs: 3

Performance Analysis:

- Strengths: Superior accuracy (~86%) due to Filipino pre-training.
- Trade-offs: Higher computational requirements.

6.4 Tuning Tips

- ELECTRA knobs: learning rate 2e-5–5e-4; batch 16–64; msl 128–256; warmup.
- BiLSTM knobs: embedding size, hidden size, layers, dropout; threshold, `pos_weight`.
- Optional W&B sweeps:

```
wandb sweep -p PROJECT_NAME Filipino-Text-Benchmarks-master/sample_sweep.yaml
wandb agent USERNAME/PROJECT_NAME/SWEEP_ID
```

7. Results and Performance

7.1 Model Comparison (Table)

Model	Best Val Accuracy	Best F1 Score	Test Accuracy	Training Time	Model Size	GPU Memory
BiLSTM	0.71	0.68	~0.70	4–18 min	~20 MB	2–4 GB
ELECTRA	0.86	0.89*	~0.86	30–60 mins	~300 MB	8–16 GB

*ELECTRA F1 is estimated based on benchmark reports; actual value may vary by split.

Summary:

- ELECTRA outperforms BiLSTM in accuracy and F1, but requires more resources.
- BiLSTM is efficient and competitive, especially with threshold tuning.

BiLSTM Hyperparameter Results Summary

Config	Learning Rate	Batch Size	Hidden Dim	Dropout	Epochs	LSTM Layers	Optimizer	Best Val Acc	Final F1	Final Precision	Final Recall	Training Time (min)

Config	Learning Rate	Batch Size	Hidden Dim	Dropout	Epochs	LSTM Layers	Optimizer	Best Val Acc	Final F1	Final Precision	Final Recall	Training Time (min)
1: Baseline (10 epochs)	0.001	64	128	0.3	10	2	Adam	0.7042	0.6426	0.6863	0.6042	4.46
2: Extended (15 epochs)	0.0005	32	256	0.4	15	2	Adam	0.6949	0.6479	0.6656	0.6311	17.93
3: Fast (20 epochs)	0.01	128	64	0.2	20	2	RMSprop	0.7039	0.6757	0.6572	0.6953	4.22
4: Deep Model (3 layers, 12 ep)	0.0008	64	128	0.35	12	3	Adam	0.7053	0.6650	0.6605	0.6694	7.27
5: Regularized (High dropout)	0.001	64	128	0.5	10	2	SGD	0.5432	0.0000	0.0000	0.0000	3.98

7.2 Target Achievement

Project Requirement: ≥50–60% overall accuracy.

- ELECTRA: ~86% (from published benchmarks; not run in this project).
- BiLSTM: Best Val Acc 0.7053; Best F1 0.6757 (Config 4 and 3, see Section 6.2).

7.3 Detailed Performance Analysis - BiLSTM

Error Analysis:

- Common False Positives: Sarcasm or strong opinions without hate intent.
- Common False Negatives: Subtle/implicit hate; complex code-switching.
- Challenging Cases: Sarcasm, irony, context-dependent hate.

7.4 Model Strengths and Limitations

BiLSTM Strengths:

- Efficient resource usage; fast inference; captures sequential patterns. BiLSTM Limitations:
- Did not reach state-of-the-art results; may struggle with long-range dependencies; tuning sensitive. ELECTRA Strengths:
- Strong attention to context and superior accuracy (per benchmarks).

6. Conclusion

8.1 Deep Learning Framework

- PyTorch for model building, training, and inference.

8.2 Data and Text Processing

- pandas and numpy for data handling and numerical ops.
- For this repository: preprocessing and tokenization are primarily handled in `Filipino-Text-Benchmarks-master/utils/data.py` for transformer models. For the BiLSTM baseline, a prior `text_preprocessing.py` module handled URL/mention handling and tokenization.

8.3 Model Training Modules

- `torch.nn` (LSTM, Linear, Dropout); `torch.optim` (Adam/AdamW).

8.4 Evaluation and Visualization

- scikit-learn for metrics (accuracy, precision, recall, F1); optional matplotlib for curves.

8.5 Development Environment

- Python 3.8+; Git & GitHub for version control.

8.6 GUI Applications

- Prior baseline included Tkinter GUI apps (e.g., gui_app.py, social_media_app.py). These are optional and not part of the ELECTRA training repo; still compatible conceptually for inference UIs.

9. Model Deployment

9.1 Inference Pipeline

The deployment system follows a structured pipeline:

1. Text Input → via GUI, CLI, or service.
2. Preprocessing → clean and tokenize (HF tokenizer for ELECTRA; vocab indices for BiLSTM).
3. Sequence Conversion → integer indices.
4. Padding → to 128 tokens (corrected; not 100).
5. Model Prediction → forward pass.
6. Threshold Application → tuned threshold (e.g., 0.75) for BiLSTM; default argmax for ELECTRA classifier.
7. Output → class label and confidence.

9.2 Application Interfaces

- Desktop GUI (baseline): real-time single-text testing.
- Social Media Simulation (baseline): batch moderation demo.

9.3 Deployment Files

- ELECTRA fine-tune: checkpoint directory from `train.py` (HF-compatible).
- BiLSTM baseline: `best_bilstm_model.pt`, `vocabulary.pkl`, architecture module.

10. Conclusion

10.1 Project Achievements

- Implemented BiLSTM baseline and documented ELECTRA fine-tuning path (benchmarks only).
- Exceeded minimum accuracy target; ELECTRA ~86% (benchmarks); BiLSTM achieved moderate validation metrics (best val acc 0.7053, best F1 0.6757).
- Provided working training + evaluation pipeline; optional GUI concepts.
- Conducted dataset validation and bias assessment.

10.2 Key Findings

- BiLSTM is resource-efficient but limited in accuracy on this dataset.
- Dropout (0.3–0.5) and regularization are important for generalization.
- Broader, diverse data improves handling of code-switching.

10.3 Future Improvements

- Explore attention mechanisms, additional Filipino transformers, and better calibration.
- Expand datasets across regions/dialects; consider emoji/sentiment features.

10.4 Lessons Learned

- Preprocessing/tokenization consistency matters, especially for bilingual content.
- Iterative tuning and careful hyperparameter search are key.
- Context-dependence remains challenging; cultural cues are critical.