Filipino Hate Speech Detection — Revised Technical Documentation

Mirrored from “ann docu.pdf” with corrections + system flow

Project Team

2025-12-13

Filipino Hate Speech Detection

Revised Technical Documentation — Corrected Details & System Flow

Date: 2025-12-13

1 1. Problem Statement

1.1 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 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 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 2. Solution Approach

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

2.1 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 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 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 2.4 System Flow (End-to-End)

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

1. Data ingest (CSV or HuggingFace) → consolidate splits
2. Preprocess and add social tokens (`[LINK]`, `[MENTION]`, `[HASHTAG]` when applicable)
3. Tokenize via WordPiece and cache TensorDatasets (MD5-keyed)
4. Initialize pretrained model and tokenizer
5. Fine-tune with validation; track best checkpoint
6. Evaluate on test; export weights
7. Deploy for inference (CLI, scripts, or service)

ASCII flow:

```
[CSV / HF Datasets]
  │
```

```
[Preprocess + Add Tokens]
  |
 [Tokenize (WordPiece)]
  |
   [Cache Datasets]
  |
[Fine-Tune ELECTRA + Eval]
  |
 [Best Checkpoint Saved]
  |
    [Test + Report]
```

3 3. Dataset Information

3.1 3.1 Dataset Sources

Primary datasets in this repository: 1. Local Hate Speech Splits - Directory: hatespeech/ (train.csv, valid.csv, test.csv) - Content: Filipino social media hate speech (binary labels `text,label`). 2. Filipino TikTok Hate Speech Dataset - Directory: filipino-tiktok-hatespeech-main/data (train.csv, valid.csv, test.csv) - Platform: TikTok comments; modern social media interactions. 3. Cyberbullying Tweets Dataset - File: cyberbullying_tweets.csv - Platform: Twitter/X; labeled cyberbullying instances. 4. Aggression Dataset - File: aggression_parsed_dataset.csv - Content: Aggression-related text for auxiliary experiments. 5. Optional HuggingFace Dataset - Source: mteb/FilipinoHateSpeechClassification via datasetImportLib.py - Note: Requires `huggingface-cli login`.

Note: Earlier versions referenced unified CSVs (e.g., unified_filipino_hatespeech.csv). In this repository, we use the datasets above.

3.2 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 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 4. Neural Network Architecture

4.1 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].

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 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 5. Training Configuration

5.1 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 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 5.3 Training Parameters - BiLSTM

- Batch Size: 64
- Learning Rate: 0.001
- Optimizer: Adam ($\beta 1$=0.9, $\beta 2$=0.999)
- Weight Decay: 0.0
- Epochs: 10
- Loss Function: `BCEWithLogitsLoss` (recommend `pos_weight` for imbalance)
- Threshold: tuned; 0.75 reduced false positives in analysis.
- Max Sequence Length: 128 (corrected).

5.4 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

Trainer CLI (this 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 6. Hyperparameter Tuning Experiments

6.1 6.1 Overview

Two main architectures were considered to compare performance and resource needs.

6.2 6.2 Experiment 1: BiLSTM Baseline Configuration

Experiment ID: BiLSTM-001 Status: Completed - Baseline Training Configuration: - Embedding Dimension: 128 - Hidden Dimension: 128 - Layers: 2 - Dropout: 0.3 - Batch Size: 64 - Learning Rate: 0.001

Observations: - Strengths: Efficient training; good sequential modeling. - Optimizer Choice: Adam worked well without manual scheduling.

6.3 6.3 Experiment 2: 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 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 7. Results and Performance

7.1 7.1 Model Comparison (qualitative)

- Test Accuracy: ELECTRA ~86% on benchmarks; BiLSTM depends on split and threshold tuning.
- Training Time: BiLSTM ~1–2 hours; ELECTRA ~30–60 mins (hardware dependent).
- Model Size: BiLSTM tens of MB; ELECTRA ~300+ MB.
- GPU Memory: BiLSTM ~2–4 GB; ELECTRA ~8–16 GB.

7.2 7.2 Target Achievement

Project Requirement: ≥50–60% overall accuracy. - ELECTRA: ~86% (Target met with margin). - BiLSTM: Val Acc ~0.843; F1 ~0.885 in a prior run (threshold 0.75 reduced false positives).

7.3 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 7.4 Model Strengths and Limitations

BiLSTM Strengths: - Efficient resource usage; fast inference; captures sequential patterns. BiLSTM Limitations: - May struggle with very long-range dependencies; tuning sensitive. ELECTRA Strengths: - Strong attention to context and superior accuracy.

8 8. Tools and Technologies

8.1 8.1 Deep Learning Framework

- PyTorch for model building, training, and inference.

8.2 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 8.3 Model Training Modules

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

8.4 8.4 Evaluation and Visualization

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

8.5 8.5 Development Environment

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

8.6 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 9. Model Deployment

9.1 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 9.2 Application Interfaces

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

9.3 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 10. Conclusion

10.1 10.1 Project Achievements

- Implemented BiLSTM baseline and ELECTRA fine-tuning path.
- Exceeded minimum accuracy target; ELECTRA ~86% on benchmark; BiLSTM achieved strong validation metrics with threshold tuning.
- Provided working training + evaluation pipeline; optional GUI concepts.
- Conducted dataset validation and bias assessment.

10.2 10.2 Key Findings

- BiLSTM effective for sequential classification under resource constraints.
- Dropout (0.3) important for generalization.
- Broader, diverse data improves handling of code-switching.

10.3 10.3 Future Improvements

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

10.4 10.4 Lessons Learned

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

11 11. Practical Notes

- HuggingFace auth: `huggingface-cli login` before fetching Hub datasets.
- Boolean CLI args use `true/false` strings in `train.py`.
- Special tokens must match training-time preprocessing (e.g., `[LINK]`).

12 12. Reproducibility

- Set seeds; pin versions where feasible.
- Keep data splits and cache keys stable between runs.
- Record CLI args and environment (logs or W&B).