# NLP Classification for Cyber Harm Detection: Analysis on Existing Similar Projects

September 4, 2025

## 1 Analysis of Existing Projects

The referenced projects focus on using NLP to classify harmful content (e.g., hate speech, spam, fraud, phishing) in texts like tweets, emails, URLs, and job postings. Given your group's preference for fine-tuning pre-trained models, we highlight projects leveraging LLMs, while noting traditional ML for context:

- **Data Sources:** Projects utilize labeled datasets from Kaggle, Hugging Face, or custom sources (e.g., Twitter data [2], SMS spam [10]). Imbalanced classes (e.g., 6% hate speech [2]) are common, addressed via oversampling (SMOTE [7]) or focal loss [1]).
- **Techniques:** Advanced projects fine-tune pre-trained LLMs like BERT [4, 5, 6] or Flan-T5 [9] for superior context understanding, especially in few-shot scenarios. Earlier projects use traditional ML (Naive Bayes, SVM, Random Forest) with TF-IDF [10, 3], but LLMs outperform in nuanced tasks like detecting slang or deceptive language.
- **Workflow:** Standard pipeline includes data cleaning, tokenization, embedding generation (e.g., BERT embeddings [4]), fine-tuning, evaluation, and deployment (e.g., Flask [7], Heroku [2]).
- **Evaluation:** Emphasis on recall/F1 for rare harms (e.g., 62% recall [2], 98% accuracy with BERT [10]). Visualizations include word clouds and confusion matrices.
- **Challenges Addressed:** Handling slang (e.g., hate speech nuances [2]), class imbalance, and multimodal features (e.g., URLs + text [5]).
- **Scale and Innovation:** Fine-tuned LLMs like Spam-T5 [9] excel in few-shot settings, while simpler projects [3] use sentiment for quick pilots.

These projects, often capstone/academic efforts, start with Jupyter notebooks and scale to apps, using GitHub for collaboration.

## 2 Recommended Approach

A five-step pipeline centered on fine-tuning pre-trained LLMs, inspired by projects like [4, 9, 5], with traditional ML baselines for initial validation.

### 2.1 Data Collection and Preparation

- **Source:** Use Hugging Face datasets (e.g., "hate_speech_offensive" for hate, "imanoop7/phishing_u for scams) to gather 5,000–50,000 samples across fraud, scams, and hate speech.
- **Preprocessing:** Tokenize with BERT tokenizer [5], remove stopwords, lemmatize (spaCy/NLTK). Use stratified splits (70% train, 15% validation, 15% test) to handle imbalance.
- **Tools:** Pandas for loading, Hugging Face Datasets for access.

### 2.2 Feature Extraction

- **LLM Embeddings:** Use pre-trained BERT ("bert-base-uncased") via Hugging Face Transformers to generate contextual embeddings, capturing nuances like sarcasm or scam patterns [4, 6].
- **Baselines:** For comparison, extract TF-IDF features [3].
- **Extras:** Add sentiment scores (VADER [2]) or n-grams for fraud indicators.

### 2.3 Model Selection and Fine-Tuning

- **Primary Approach:** Fine-tune BERT or Flan-T5 using Hugging Face Trainer API [5, 9]. Add a classification head for multi-label (normal, hate, fraud, scam) or binary tasks. Train for 3–5 epochs, learning rate 2e-5.
- **Few-Shot:** Fine-tune on 100–500 samples per harm to simulate low-data scenarios [9].
- **Baselines:** Train Logistic Regression/SVM on TF-IDF for quick validation [10].
- **Tools:** PyTorch/Hugging Face for LLMs, scikit-learn for baselines. Use Google Colab GPU.
- **Imbalance:** Apply focal loss [1] or class weights.

### 2.4 Evaluation

- **Metrics:** Prioritize recall/F1 (target 95%+ F1 [10]) for rare harms. Include ROC-AUC.
- **Methods:** 5-fold cross-validation, confusion matrices, word clouds for top harm words.
- **Error Analysis:** Check biases (e.g., misclassifying cultural slang [2]).

### 2.5 Deployment

- **App:** Build a Flask/Streamlit interface for text input and predictions [2, 7].
- **Hosting:** Use Heroku/AWS for scalability.
- **Ethics:** Address biases (e.g., over-flagging minority languages).

## 3 Challenges and Mitigations

- **Class Imbalance:** Use focal loss [1] or SMOTE [7].
- **Compute:** Leverage Google Colab's free GPU for LLM fine-tuning.
- **Nuances:** Fine-tuned LLMs handle slang/deceptive language better than traditional ML [9].

## References


[1] Osika, A. (2025). NLP to Identify Toxic or Abusive Language. https://github.com/andiosika/NLP-to-identify-toxic-or-abusive-language-for-online-conversat

[2] Kung, S. (2025). Twitter Hate Speech Detection. https://github.com/sidneykung/twitter_hate_speech_detection

[3] Mubashir, R. (2025). Fraud App Detection Using Sentiment Analysis. https://github.com/RaoMubashir760/Fraud-App-Detection-using-sentiment-analysis-By-Rao-Mubashir

[4] Mohammadi, N. (2025). BERT Mail Classification. https://github.com/Nargesmohammadi/Bert-Mail-Classification

[5] Maurya, A. (2025). Fine-Tuning BERT for Phishing URL Detection. *Towards AI*. https://pub.towardsai.net/fine-tuning-bert-for-phishing-url-detection-a-beginners-guide-619fad27db41

[6] Gull, A. (2025). Detect AI-Generated Phishing Emails with BERT. https://github.com/AsimGull/Data-Science-Projects/tree/c751d862cc31535d4e0d3077fb1febd0388e117a

[7] Ervenderr. (2025). Fraud Detection in Job Postings. https://github.com/ervenderr/Fraud-Detection-in-Job-Postings-using-NLP-and-Machine-Learning

[8] Justmephoenix. (2025). Phishing Detection with NLP. https://github.com/Justmephoenix/PHISHING-DETECTION-WITH-NLP

[9] Labonne, M., & Moran, S. (2023). Spam-T5: Benchmarking LLMs for Email Spam Detection. https://github.com/jpmorganchase/llm-email-spam-detection

[10] Aniass. (2025). Spam Detection inMappings Messages. https://github.com/aniass/Spam-detection