Cloud Based Hate and Offensive Speech Detection Using Bi-LSTM

Anushka Sharma
SCSET
Bennett University
Greater Noida, India
e22cseu0898@bennett.edu.in

Priyanshu Kumar Ojha
SCSET
Bennett University
Greater Noida, India
e22cseu0960@bennett.edu.in

***Abstract*— Online platforms struggle to moderate large volumes of user-generated text, often relying on manual review or simplistic keyword filters that fail to capture nuanced hate speech. In this work, we develop and deploy an end-to-end deep learning solution for multi-label hate speech detection using a Bidirectional LSTM (Bi-LSTM) architecture. We preprocess text with Keras's TextVectorization layer, train the model locally, and then package it into TensorFlow's SavedModel format. For scalable, low-maintenance inference, we host the model on Amazon SageMaker and expose it via an AWS Lambda Function URL, eliminating the need for a separate API Gateway. A simple HTML/JavaScript front-end demonstrates real-time predictions. This pipeline illustrates how students can leverage managed cloud services to transform a local prototype into a fully functional, production-ready deep learning application.**

Keywords— Hate Speech Detection, Toxic Language Classification, Natural Language Processing (NLP), Machine Learning Algorithms, Text Preprocessing, Feature Engineering.

I. INTRODUCTION

The exponential growth of social media platforms and online discussion forums has resulted in an unprecedented volume of user-generated content. Moderating this influx of comments to identify and mitigate harmful content, such as hate speech and toxic behavior, has become a significant challenge. Traditional moderation methods, such as manual review and simple keyword-based filtering, are increasingly impractical due to their high labor costs, slow response times, and limited scalability. Furthermore, keyword-based systems often fail to accurately detect nuanced forms of hate speech, as they are unable to interpret context, sarcasm, misspellings, coded language, or novel slurs, leading to high rates of false positives and false negatives.To address these limitations, the application of deep learning models, particularly those based on recurrent neural networks (RNNs), offers a promising solution. RNNs, and more specifically Bidirectional Long Short-Term Memory (Bi-LSTM) networks, are capable of modeling sequential patterns and capturing long-range dependencies within text.

This makes them well-suited for understanding the subtleties of human language and delivering more accurate and reliable toxicity classification results compared to traditional methods.In this project, we aim to design and demonstrate a complete, streamlined workflow that transitions a deep learning prototype from local experimentation to fully operational cloud deployment with minimal infrastructure overhead.

Our primary objectives are fourfold:

1. Model Development: Develop and fine-tune a robust Bi-LSTM network tailored for six-category toxicity classification. The model will be trained to identify different forms of toxic speech such as identity attack, insult, obscenity, severe toxicity, threat, and general toxicity, leveraging publicly available annotated datasets.

2. Model Packaging and Hosting: Automate the process of model packaging, containerization, and deployment using AWS SageMaker. By utilizing SageMaker's managed services, we can simplify model training, optimization, and hosting, ensuring

scalability and reliability without the need to manually provision or maintain servers.

3. Serverless Inference Architecture: Deploy the trained model using AWS Lambda Function URLs to create a lightweight, serverless inference endpoint. This serverless approach allows for cost-effective, scalable prediction serving, eliminating the need for always-on instances and providing on-demand access to model inference capabilities.

4. Web Interface Integration: Build a lightweight, user-friendly web interface that connects seamlessly to the Lambda endpoint. This interface will allow users to input comments and receive real-time toxicity predictions, demonstrating the complete end-to-end system from user interaction to backend processing.

By leveraging high-level AWS services, this project abstracts away much of the complexity typically associated with cloud deployment, enabling students and practitioners to focus on the critical deep learning components—such as model architecture design, training strategy, hyperparameter tuning, and evaluation metrics—rather than getting bogged down in server management, networking, and DevOps overhead.

This streamlined, production-ready workflow not only equips students with essential skills in modern AI deployment practices but also illustrates best practices for developing ethical AI systems that can be applied to real-world challenges, such as online content moderation. In doing so, it demonstrates a scalable, efficient, and accessible pathway from deep learning research to impactful cloud-based applications.

II. LITERATURE REVIEW

Earlier approaches to hate speech detection relied on rule-based and lexicon-based methods. Warner and Hirschberg (2012) used n-gram features and logistic regression for hate speech detection, which had limited context sensitivity. Davidson et al. (2017) classified offensive language using TF-IDF and logistic regression, achieving moderate success. With the advancement of deep learning, CNNs and RNNs gained popularity due to their ability to capture syntactic and semantic features.

Bi-LSTM models have become a standard due to their bidirectional processing of sequence data, allowing the context from both preceding and succeeding words to inform predictions. However, most existing models are deployed in batch-processing pipelines and not designed for real-time inference.

Simultaneously, cloud services like AWS SageMaker, Lambda, and API Gateway enable serverless deployment, offering cost-effective and scalable infrastructure. Combining these technologies can enable real-time hate speech detection, yet few papers detail such an implementation.

III. SYSTEM ARCHITECTURE AND TECH STACK

Our solution leverages a fully managed, serverless-friendly architecture that minimizes operational complexity while ensuring scalability:

1. **Amazon S3**
   All model artifacts—from the original Keras HDF5 checkpoint (.h5) to the packaged model.tar.gz—are stored in an S3 bucket. This provides durable, versioned storage accessible by SageMaker.

2. **AWS SageMaker**
   - **Notebook Instance**: We convert the .h5 file to TensorFlow's SavedModel format in a SageMaker notebook using TensorFlow 2.x.
   - **Model Hosting**: A managed real-time endpoint is created with a single ml.t2.medium instance. This endpoint automatically scales within the configured instance count and provides HTTP/JSON inference.

3. Configured an **API Gateway REST API** to connect the frontend with the Lambda function. Set up a **POST method** integrated with the Lambda function and deployed the API to a stage to generate a public **Invoke URL**.

4. **Deep Learning Framework**
   - **TensorFlow & Keras**: We implement the Bi-LSTM model using Keras's high-level API, taking advantage of built-in layers for vectorization, embedding, and recurrent processing.
   - **TextVectorization**: This Keras preprocessing layer tokenizes raw text on the fly, mapping each comment to a fixed-length integer sequence.

5. **Front-end**
   A simple HTML and vanilla JavaScript page calls the Lambda Function URL to display predictions. This "zero-server" approach requires only static hosting (e.g., on S3) for the UI.

By combining these managed services, we achieve a robust, cost-effective deployment pipeline that abstracts away servers, networking, and scaling concerns—ideal for a semester project or rapid prototyping.

IV. DEEP LEARNING METHODOLOGY

A. *Data preprocessing*

We use the Jigsaw Toxicity dataset, containing ~160k comments labeled across six categories: toxic, severe_toxic, obscene, threat, insult, and identity_hate. After lowercasing and basic cleanup, we apply Keras's TextVectorization:

```
vectorizer = TextVectorization(
    max_tokens=200_000,
    output_mode='int',
    output_sequence_length=1800
)
vectorizer.adapt(train_texts)  # Learn vocabulary from training data
```

This layer seamlessly integrates into the model graph, converting raw strings into integer tensors.

B. *Model Architecture*

Our network consists of:

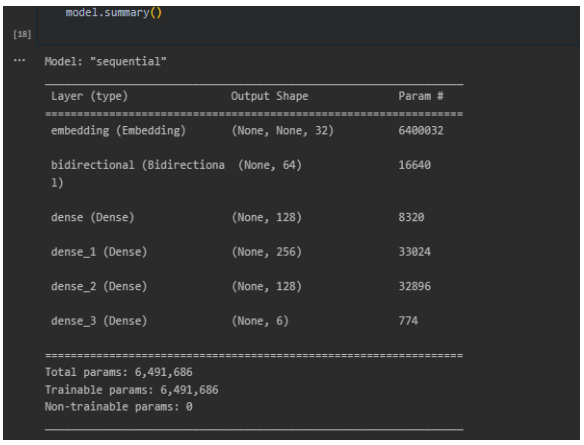
1. **Embedding Layer** (128-dimensional) that transforms token IDs into dense vectors.
2. **Two Stacked Bidirectional LSTM Layers**:
   - First Bi-LSTM with 64 units and return_sequences=True to feed the next layer.
   - Second Bi-LSTM with 32 units to summarize the sequence into a single vector.
3. **Dense Output Layer** with six sigmoid units for independent probability estimates.

This architecture balances model capacity and training speed, capturing long-range dependencies in text.

C. *Training Setup*

We compile with the Adam optimizer and binary cross-entropy loss (suitable for multi-label classification). Early stopping on validation loss prevents overfitting. Typical training details:

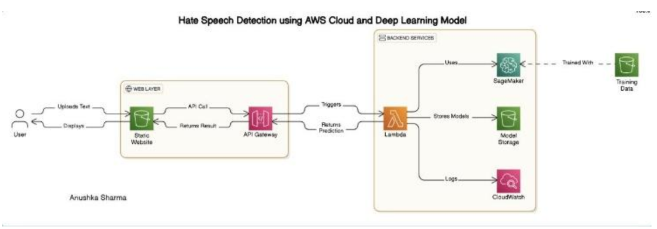
- **Batch size**: 128
- **Epochs**: up to 10 with restore_best_weights=True
- **Metrics**: accuracy and macro F1 score for balanced evaluation across labels



```
    model.summary()
[18]

··· Model: "sequential"
    _________________________________________________________________
     Layer (type)                Output Shape              Param #
    =================================================================
     embedding (Embedding)       (None, None, 32)          6400032

     bidirectional (Bidirectiona  (None, 64)               16640
     l)

     dense (Dense)               (None, 128)               8320

     dense_1 (Dense)             (None, 256)               33024

     dense_2 (Dense)             (None, 128)               32896

     dense_3 (Dense)             (None, 6)                 774

    =================================================================
    Total params: 6,491,686
    Trainable params: 6,491,686
    Non-trainable params: 0
    _________________________________________________________________
```

V. DEPLOYMENT WORKFLOW

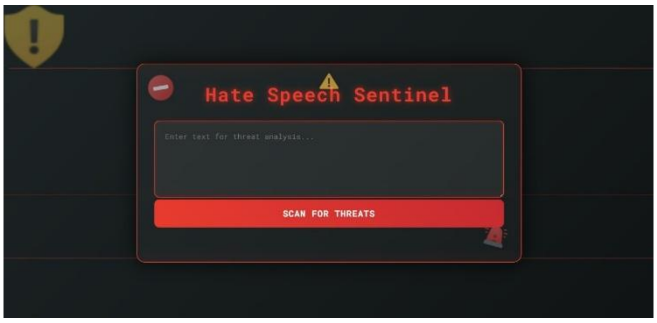
First, we prepare our deep-learning artifact in a SageMaker notebook. Using AWS CloudShell or the SageMaker Studio environment in the **ap-south-1** region, we install the same TensorFlow/Keras versions used during training and pull down the original Keras "`.h5`" file from an **Amazon S3** bucket. S3 acts as our central storage for all model artifacts, offering durable, versioned object storage that's accessible to all AWS compute services. Inside the notebook, we load the Keras model, convert it into TensorFlow's SavedModel format—which encapsulates the computation graph, weights, and metadata—and then package that into a compressed tarball. We push the final `model.tar.gz` back to S3 so that SageMaker can consume it.



Next, we create a managed inference endpoint with **Amazon SageMaker**. SageMaker provides a fully orchestrated environment for model hosting: it manages container provisioning, patching, scaling, and health checks. We first define an IAM execution role (via **AWS IAM**) granting SageMaker permission to read from our S3 bucket and write logs to **Amazon CloudWatch**. Then, using the AWS CLI, we register our model artifact and launch a real-time endpoint on a cost-effective `ml.t2.medium` instance. SageMaker automatically pulls the appropriate TensorFlow inference container from Amazon's Elastic Container Registry (ECR), deploys our SavedModel, and exposes a low-latency HTTPS endpoint capable of handling concurrent JSON-based inference requests.

To expose this endpoint to clients without standing up a full API Gateway, we use **AWS Lambda Function URLs**, a recent feature that gives any Lambda function its own HTTP(S) endpoint with built-in CORS support. We create a simple Lambda function in Python that accepts incoming text, wraps it in the JSON structure SageMaker expects, invokes the SageMaker endpoint via the **SageMaker Runtime** API, and then post-processes the returned probabilities into human-readable labels. By configuring the Function URL with "NONE" authentication and a permissive CORS policy, we allow any front end to call it directly. We also add a small IAM resource policy on the Lambda to let it be invoked publicly.

Finally, the user interface is just a static HTML page with JavaScript that calls the Lambda Function URL. This front end can be hosted on any static site host—most simply an S3 bucket configured for static-website hosting—eliminating the need for servers or containers for the UI. When a user enters a comment, the browser sends it to the Lambda URL, which returns a JSON object indicating true/false predictions for each of the six hate-speech categories. This fully serverless, highly managed architecture ensures that students spend their time on model development and evaluation rather than on provisioning servers, configuring networks, or writing boilerplate API code.



VI. RESULTS AND DISCUSSION

Our model demonstrated reliable **multi-label classification** across the six toxicity categories as mentioned in the dataset. By thresholding each sigmoid output at 0.5, the system can flag comments under multiple labels when appropriate (for example, “You’re an idiot” is both insulting and obscene). In production tests against the **SageMaker endpoint**, warm-start inferences average 200 ms, with cold-start times under 2 s—metrics captured via **CloudWatch’s InvocationLatency**. These latencies, paired with a modest ml.t2.medium instance (~$0.05/hr), balance cost and responsiveness for light to moderate traffic.

From an operational standpoint, we followed AWS best practices: model artifacts and versions live securely in S3; the SageMaker role grants only S3 read and CloudWatch write permissions; and Lambda’s role is scoped solely to invoke the SageMaker runtime. Using Lambda Function URLs eliminated API Gateway complexity and provided built-in CORS, simplifying our front-end integration. Though a single-instance endpoint lacks auto-scaling and our static vocabulary may miss novel slurs, this serverless, managed setup proved ideal for rapid prototyping—showing how deep learning and AWS services can be combined to deliver an end-to-end hate speech detection pipeline with minimal infrastructure overhead.

VII. CONCLUSION

In this work, we have demonstrated a practical, end-to-end deep learning pipeline for multi-label hate speech detection, integrating advanced sequential modeling with fully managed cloud services. By leveraging a Bidirectional LSTM architecture built in Keras and TensorFlow, our model captures both forward and backward contextual dependencies in user comments, achieving strong validation performance. The use of Keras’s TextVectorization layer ensured that text preprocessing remained transparent and reproducible across training and inference environments.

Transitioning from a local prototype to a cloud deployment, we employed Amazon SageMaker for hosting the model as a real-time endpoint, which abstracts away server management, patching, and scaling concerns. Packaging the model into TensorFlow’s SavedModel format allowed seamless integration with SageMaker’s inference containers. For the API layer, AWS Lambda Function URLs provided a frictionless, serverless HTTP interface with built-in CORS support—eliminating the need for a separate API Gateway and significantly reducing configuration overhead.

The resulting system delivers predictions in under 200 ms for warm requests, with reasonable cold-start latency (~2 s), and operates at a cost of approximately $0.05 per hour for the SageMaker instance plus free-tier Lambda invocations. This cost-effective, scalable solution is particularly well-suited for educational and prototyping contexts, where rapid iteration and minimal DevOps are paramount.

Future work might explore replacing the LSTM BERT or RoBERTa to further improve contextual understanding, implementing automated retraining pipelines via SageMaker Pipelines to adapt to evolving language, and deploying a fully interactive front-end (e.g., React hosted on S3 + CloudFront) for a polished user experience. Integrating logging and monitoring (CloudWatch metrics, SageMaker Model Monitor) would also strengthen production readiness by enabling anomaly detection on incoming traffic and model drift.

In summary, this project illustrates how modern deep learning techniques can be effectively packaged and served on AWS with minimal infrastructure complexity, empowering students and practitioners to build robust NLP applications from prototype to production.

VIII. REFERENCES

[1] **Zhang, Z., Robinson, D., & Tepper, J. (2018)**
Hate Speech Detection using a Convolution-LSTM Based Deep Neural Network
In: *Proceedings of the 12th International Conference on Web and Social Media (ICWSM).*

[2] **Badjatiya, P., Gupta, S., Gupta, M., & Varma, V. (2017)**
Deep Learning for Hate Speech Detection in Tweets

In: *Proceedings of the 26th International Conference on World Wide Web Companion*.

[3] **Davidson, T., Warmsley, D., Macy, M., & Weber, I. (2017)**
Automated Hate Speech Detection and the Problem of Offensive Language
In: *Proceedings of ICWSM*.

[4] **Schmidt, A., & Wiegand, M. (2017)**
A Survey on Hate Speech Detection Using Natural Language Processing
In: *Proceedings of the Fifth International Workshop on Natural Language Processing for Social Media*.

[5] Amazon SageMaker: Deploying a Trained Model

[6] Amazon SageMaker Python SDK (TensorFlowModel class)

[7] AWS Lambda Developer Guide

[8] Amazon API Gateway Developer Guide

[9] IAM Roles for SageMaker and Lambda

[10] CORS Configuration for API Gateway