AWS Based Hate Speech Detection using BERT

1st Anushka Sharma
SCSET
Bennett University
Greater Noida, India
e22cseu00898@bennett.edu.in

2nd Priyanshu Kumar Ojha
SCSET
Bennett University
Greater Noida, India
e22cseu0960@bennett.edu.in

3rd Saurabh Jadhao
SCSET
Bennett University
Greater Noida, India
e22cseu0931@bennett.edu.in

***Abstract*—Online platforms are constantly dealing with a flood of user-generated content, and spotting hate speech—especially when it's subtle or disguised—is a tough job. Most sites still rely on manual moderation or simple keyword filters, but those methods often miss the bigger picture, especially when context matters. To tackle this, we built a complete deep learning system that can catch multiple types of hate speech using a Bidirectional Encoder Representations from Transformers model. We used Keras's TextVectorization layer to handle the basics—like cleaning up the text, breaking it into tokens, and converting it into numbers. After training the model locally with labeled data, we saved it using TensorFlow's SavedModel format. For deployment, we chose Amazon SageMaker because it's scalable and handles the heavy lifting. Then, we connected the model to an AWS Lambda function and used the built-in Function URL to serve predictions—this way, we didn't need a separate API Gateway, which kept things simpler. Finally, we built a lightweight front end using HTML and JavaScript where users can type in text and see predictions instantly. This whole setup shows how students can take an idea from local testing to a working, cloud-based AI tool using deep learning and AWS services.**

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. In this era of social computing, the interaction between individuals becomes more striking, especially through social media platforms and chat forums.[4] 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.Hate and aggression through social networks should be rationed and regulated by policy makers and should be alsocountered by harnessing the power of artificial intelligence and machine learning algorithms toautomate the detection of hate speech in social media.[6]

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.However, nowadays transformer models such as BERT, RoBERTa etc are also coming into play for such challengers. 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 BERT 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. LITERTURE REVIEW

The detection of hate speech on online platforms has become increasingly difficult due to the large increasing number of user-generated content. Researchers have found out a range of machine learning (ML) and deep learning (DL) models and techniques over the time to solve this problem effectively.

Traditional ML algorithms have been the basic methods in early hate and offensive speech detection efforts. Earlier these were the only way to come to any solution regarding the problem. For example, **Putri et al. (2020)** worked on comparative analysis of different classifiers such as Naïve Bayes, Multi-Layer Perceptron, AdaBoost, Decision Tree, and Support Vector Machine (SVM) on 4,002 Indonesian tweets dataset.[9] The Naïve Bayes algorithm achieved the recall of 93.2% that has been highest among all and an accuracy of 71.2%, indicating its effectiveness in this context. **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.

There have been advances in Deep Learning to introduce models like Long Short-Term Memory (LSTM) networks and Convolutional Neural Networks (CNN) for hate and offensive speech detection. **Malik et al. (2022)**[8] performed a large-scale comparison between deep learning and shallow Machine Learning models across various datasets. Their results demonstrated and proved that DL models, particularly LSTM and CNN, generalized better across a;; domains and achieved higher accuracy.

The studies for specific languages have also been conducted. In the context of Afaan Oromo, a CNN-based model utilizing word embeddings achieved an F1-score of 78.3%. However, challenges were still faced in accurately identifying tweets containing both racism and sexism.
Transformer-based models such as BERT have recently gained widespread attention for their remarkable ability to grasp the context of language. These models are particularly useful in detecting subtle and complex hate speech but often require large datasets and considerable computing resources to train effectively.

Whilst traditional machine learning techniques can still perform well when combined with careful feature selection, research generally shows that deep learning methods, especially those built on Transformer architectures, provide better accuracy and a deeper understanding of context. This makes them a preferred and majorly used choice for handling the intricacies involved in hate speech detection.

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
o Notebook Instance: We convert the .h5 file to TensorFlow's SavedModel format in a SageMaker notebook using TensorFlow 2.x.
o 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
o **TensorFlow & Keras:** We implement the BERT model using Keras's high-level API, taking advantage of built-in layers for vectorization, embedding, and recurrent processing.

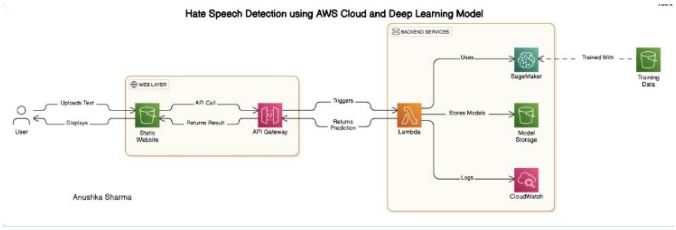

Fig. 1. AWS Architecture

o **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. METHODOLOGY

This section elaborates the methodology used to design, develop, and evaluate a robust system for the automatic detection of hate speech and toxic language using the BERT (Bidirectional Encoder Representations from Transformers) architecture. The study follows a comprehensive experimental pipeline consisting of data understanding, preprocessing, model formulation, training, and evaluation.

A. Dataset Description

The research is conducted using the Jigsaw Toxic Comment Classification Challenge dataset, a benchmark dataset provided by Kaggle and developed in collaboration with Google. This dataset comprises 159,571 user-generated comments collected from Wikipedia talk pages. Each comment is annotated across six binary toxicity labels:

1) 1. Toxic
2) 2. Severe Toxic
3) 3. Obscene
4) 4. Threat
5) 5. Insult
6) 6. Identity Hate

TABLE I
DISTRIBUTION OF TOXICITY LABELS

Label	Count	Percentage (%)
toxic	15,294	9.58
severe_toxic	1,595	1.00
obscene	8,449	5.29
threat	478	0.30
insult	7,877	4.94
identity_hate	1,405	0.88

Since each comment can have multiple labels simultaneously (e.g., a comment can be both "toxic" and "obscene"), this classification problem is formulated as a multi-label classification task. The dataset is notably imbalanced, with a large proportion of comments falling into the "non-toxic" category and significantly fewer instances labeled as "threat" or "identity hate". This imbalance poses a significant challenge for traditional learning models and necessitates the use of advanced learning techniques and loss handling strategies.

B. Data preprocessing

Preprocessing plays a crucial role in preparing the raw text data for input into transformer-based architectures. However, unlike classical NLP pipelines, transformer models like BERT require minimal text preprocessing since they are trained on raw language data.

The preprocessing steps undertaken include:

Retention of casing and punctuation: The model variant used is bert-base-uncased, which expects lowercased input but retains punctuation to capture sentiment and emphasis.

Removal of null or empty values: Ensuring data integrity before tokenization.

Tokenization using BERT's WordPiece tokenizer: The text is tokenized into sub-word units, which allows for efficient handling of rare and out-of-vocabulary words.

```
def clean_text(text):
    text = str(text)
    text = re.sub(r'\s+', ' ', text).strip()
    return text
```

Padding and truncation: Each comment is padded or truncated to a fixed length (typically 128 or 256 tokens) to ensure uniform input size across training batches.

Attention masks generation: Attention masks are created to distinguish between actual content and padding during model processing.

These steps result in structured inputs that the BERT model can process effectively while preserving the semantic and syntactic richness of the original comments.

C. Model Architecture

BERT is conceptually simple and empirically powerful. It obtains new state-of-the-art results on eleven natural language processing tasks.[7] The core of the model is built upon BERT (Bidirectional Encoder Representations from Transformers), a transformer-based language model developed by Google. Specifically, the bert-base-uncased variant is used, which comprises:

12 Transformer encoder layers

768-dimensional hidden state

12 attention heads

To adapt BERT for the downstream task of multi-label text classification, a custom classification head is appended to the model:

- A Dropout Layer (with dropout probability of 0.3) is added to prevent overfitting.
- A fully connected Dense Layer with six output units, each representing one of the toxicity labels, is used.
- Sigmoid activation is employed to output independent probabilities for each label, making it suitable for multi-label classification.
  ‘
- Unlike softmax (which assumes mutually exclusive classes), sigmoid allows for simultaneous activation of multiple labels.

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

D. Training Setup

The model is fine-tuned on the labeled dataset using supervised learning. The training strategy includes the following components:

- **Loss Function:** The Binary Cross-Entropy Loss is used, which calculates the error independently for each class and is well-suited for multi-label classification problems.
- **Optimizer:** The AdamW optimizer is employed with weight decay regularization, a commonly used variant of Adam that improves generalization in transformer models.
- **Learning Rate and Batch Size:** Learning rates are set within the range of 2e-5 to 5e-5, with batch sizes of 16 or 32 depending on the hardware constraints.
- **Epochs:** Training is typically run for 3–5 epochs. Early stopping is used to halt training if validation performance degrades, preventing overfitting.
- **Handling Class Imbalance:** Though not explicitly re-sampled, the model benefits from BERT's strong contextual learning. In some configurations, class weighting or focal loss can be added to increase attention to minority classes.

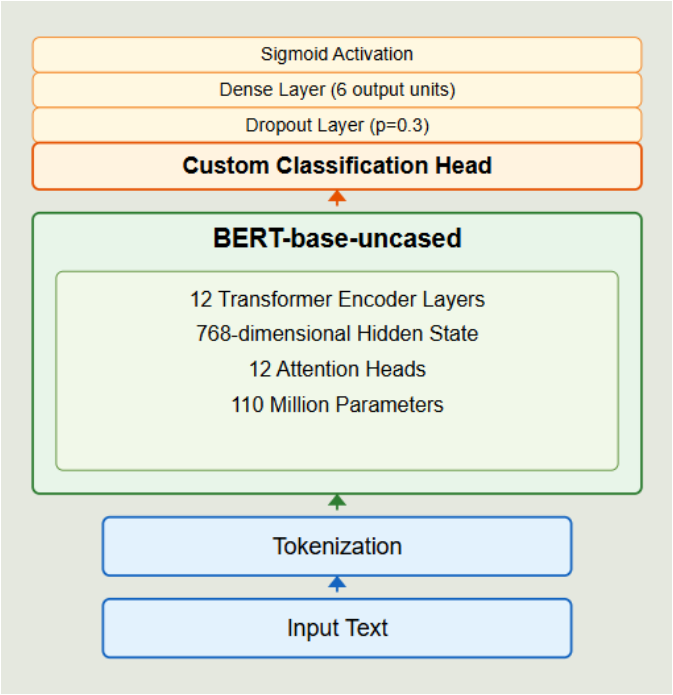

Fig. 2. BERT model overview

To account for disconnections in the Google Colab environment, model checkpointing is employed. This ensures that training progress can be saved periodically and resumed from the last best state if interrupted.

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

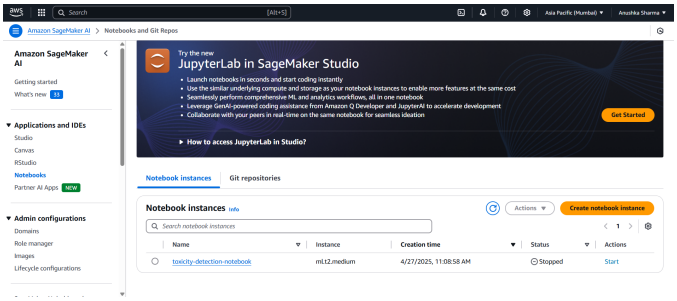

Fig. 3. Snippet from Instance notebook

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.

Fig. 4. Lambda overview

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.

E. Evaluation Strategy

Given the multi-label nature of the task and the class imbalance present in the dataset, the following evaluation metrics are used to comprehensively assess model performance:

- **ROC AUC (Receiver Operating Characteristic - Area Under Curve):**
  Computed per class and averaged to measure the model's ability to distinguish between classes regardless of threshold.
- **Precision:**
  The proportion of correctly predicted positive instances out of all positive predictions, defined as:
  $$\text{Precision} = \frac{TP}{TP + FP}$$
- **Recall:**
  The proportion of actual positive instances correctly predicted by the model, defined as:
  $$\text{Recall} = \frac{TP}{TP + FN}$$
- **F1-Score:**
  The harmonic mean of precision and recall, offering a balanced performance measure:
  $$F_1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$
- **Hamming Loss:**
  The fraction of incorrectly predicted labels to the total number of labels, calculated as:
  $$\text{Hamming Loss} = \frac{1}{N \times L} \sum_{i=1}^{N} \sum_{j=1}^{L} \mathbf{1}\left[y_{ij} \neq \hat{y}_{ij}\right]$$
  where N is the number of samples, L is the number of labels, y_{ij} is the true label, and $\hat{y}_{ij}$ is the predicted label.
- **Subset Accuracy:**
  Measures the exact match between predicted and actual label sets for each sample:
  $$\text{Subset Accuracy} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}\left[\mathbf{y}_i = \hat{\mathbf{y}}_i\right]$$

Cross-validation is used in some configurations to test model stability, particularly across underrepresented classes.

V. RESULTS AND DISCUSSION

Our analysis begins with a clear observation of **class imbalance** within the dataset, which is a common challenge in multi-label toxicity detection tasks. The distribution of toxicity categories is heavily skewed, while going through the dataset, it became obvious that some labels are more common than others. Most of the comments were marked as toxic, insult, or obscene, while categories like threat, severe toxic, and identity

hate barely showed up in comparison. This sort of imbalance isn't surprising—it's actually quite common in datasets like this—but it does create a challenge. The model ends up being better at spotting the common labels and doesn't do so well with the rare ones.

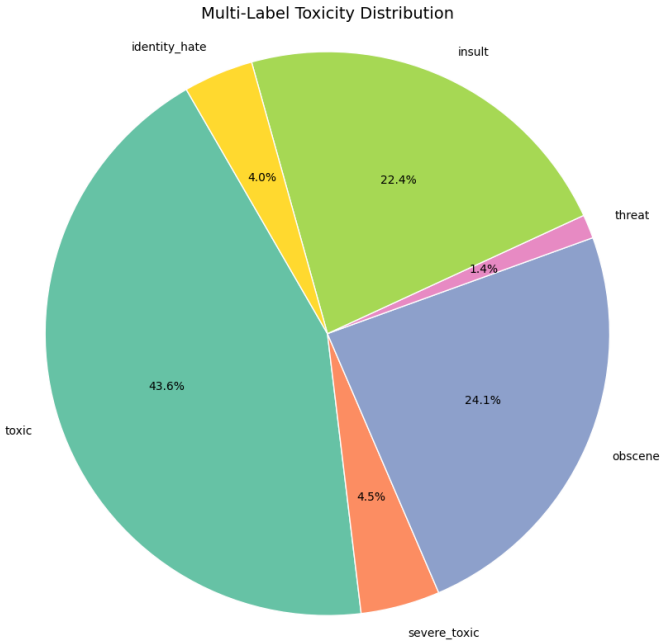

Fig. 5. Pie chart showing class imbalance

We also noticed something interesting with sentence lengths. Shorter comments tended to have fewer toxic labels. That could mean they're more direct—either toxic or not—and don't usually mix multiple types of toxic behavior in one go. Longer comments, on the other hand, seem more likely to include more than one label, possibly making it easier for the model to detect layered toxicity.

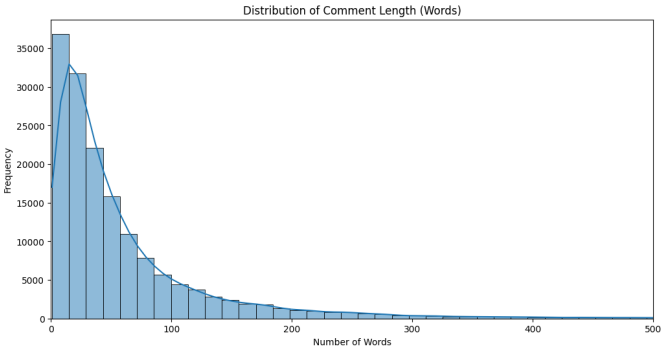

Fig. 6. Sentence length- frequency graph

To get a better idea of how the labels connect, we created a correlation matrix. It turned out that obscene and insult had a strong link (around 0.74), which makes sense because they often go hand in hand. Toxic also correlated pretty well with both of them. We found some moderate links too, like between severe toxic and obscene, and identity hate with insult. But when we looked at threat, it didn't really have strong connections with the others. That might mean threatening comments are usually standalone and don't show up with other toxic traits. The threat label shows very low correlations (0.12 to 0.16) with other categories, implying that threatening comments tend to be more independent in nature.

These findings underscore the complexity of the multi-label problem and highlight the importance of designing models that capture inter-label dependencies.

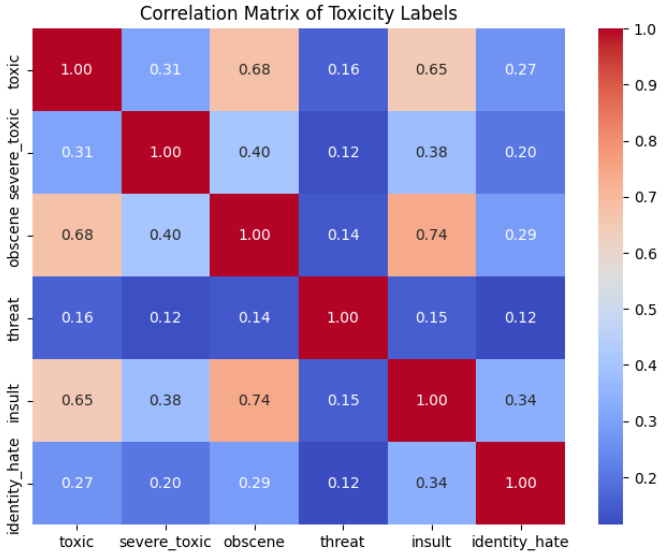

Fig. 7. Correlation heatmap among classes

Our model's training progression, visualized through the ROC curve and epoch-wise performance, shows steady improvement. After several epochs, it started balancing out—performance got better without signs of overfitting, which was good to see.

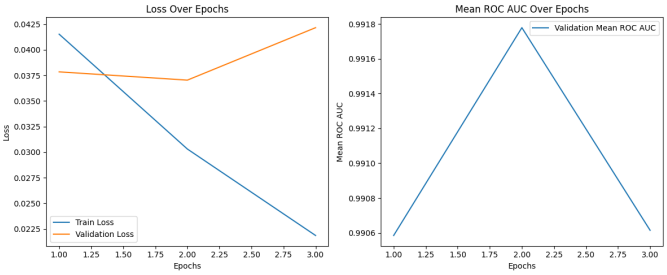

Fig. 8. Loss over epochs & ROC Curve

When we evaluated the model, the subset accuracy was about 91%, which means it predicted all the correct labels for

a comment most of the time. Precision, recall, and F1 score (micro-averaged) were in the 0.79–0.80 range, which is solid. We also saw a very low Hamming loss, meaning the model didn't make many wrong guesses per label.

That said, the macro-averaged F1 was lower—about 0.65—because it gives equal weight to all classes, including the rare ones the model struggled with.

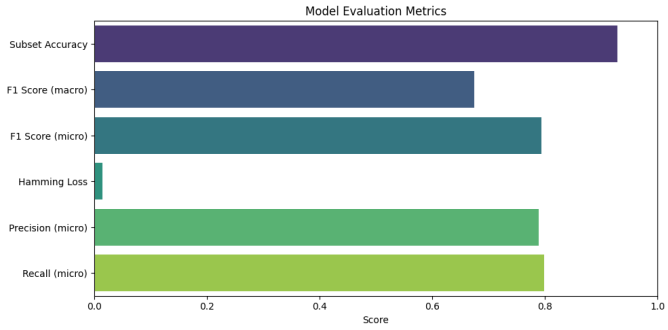

Fig. 9. Evaluation metrics

Per-class evaluation clarifies this pattern: dominant labels such as toxic, obscene, and insult show strong precision, recall, and F1 scores above 0.80, whereas rarer classes like severe_toxic, threat, and identity_hate lag behind with scores around 0.50–0.60. This disparity aligns with the observed class imbalance and highlights the ongoing challenge of reliably detecting less common but critical toxic behaviors.

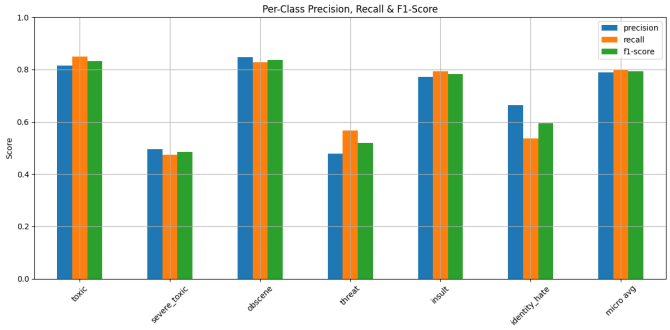

Fig. 10. Class wise evaluatioon

In conclusion, while the model performs robustly for prevalent toxicity types and benefits from capturing label correlations, further improvements are necessary to enhance its work for edge cases. To improve this, we could look into data augmentation, oversampling, or changing the loss function to better handle minority classes. Also, since some labels show strong connections, future work could focus on learning those inter-label patterns more deeply.

VI. CONCLUSION

As more and more people share their thoughts online, we've also seen a worrying rise in toxic behavior—stuff like hate speech, threats, and personal attacks. While platforms try their best to keep things clean and respectful, picking out harmful content quickly and accurately is still a major hurdle. That's mostly because the way people express toxicity isn't always straightforward. It can be sarcastic, subtle, or buried in context. That's what pushed us to work on a system that could detect various forms of toxicity more intelligently and handle large volumes of content at once.

We decided to use BERT in our work, which is a transformer-based model been widely praised for how well it understands the context in the language. Compared to older methods that depend heavily on specific keywords or hand-written rules, BERT dives deeper into what's being said and how. This makes it more reliable for spotting hateful or harmful comments that don't always use obvious trigger words.

Our experiments were based on the Jigsaw dataset, which includes six labels: toxic, severe toxic, obscene, threat, insult, and identity hate. Pretty early on, we ran into a familiar problem—some labels appeared a lot more than others. Toxic, obscene, and insult were everywhere, while things like threat and identity hate were way less common. That kind of imbalance can throw off even the best models, making them good at detecting the usual stuff but bad at catching rare but serious cases.

To understand what we were working with, we took a close look at things like how often different labels showed up together, how long the comments were, and how similar the categories were statistically. We found that insults and obscene language often went hand-in-hand, while threat-related comments tended to stand alone. That kind of insight was useful—it reminded us that not all toxic comments follow the same pattern.

Once we had that background, we fine-tuned BERT to handle multiple labels per comment. The model was trained with a sigmoid output and binary cross-entropy loss, which worked well for this type of task. Training went smoothly, and after a few rounds, the model started to perform consistently across both training and validation data, without signs of overfitting.

When it came to results, the model did pretty well. We got about 91% subset accuracy, which means that for most comments, it nailed the full set of toxic labels. On top of that, we saw micro-averaged precision, recall, and F1 scores between 0.79 and 0.80, which shows it was balanced in detecting true positives and avoiding false ones. The Hamming loss was low too, which was another good sign—it didn't mess up much on individual labels.

But, as expected, the macro F1 score told a different story—sitting at around 0.65. That score treats each label equally, so the rare ones weighed it down. When we checked per-class performance, the usual suspects like toxic, insult, and obscene performed well, while the rare ones—threat, severe toxic, and identity hate—lagged behind. The lack of training examples for those labels clearly made things harder for the model.

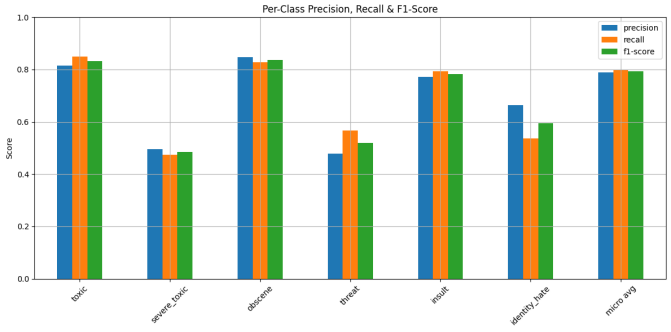

Fig. 11. Class wise evaluatioon

To sum it all up, BERT proved to be a solid choice for multi-label toxicity classification. It handled common types of toxic speech well and made sense of complex, overlapping labels. Still, the imbalance in the dataset held it back when it came to less frequent but equally important categories. Going forward, adding techniques like smart sampling, label-specific tuning, or data augmentation could help level the playing field. Also, digging deeper into how these labels relate to each other might open doors for more refined models in the future.

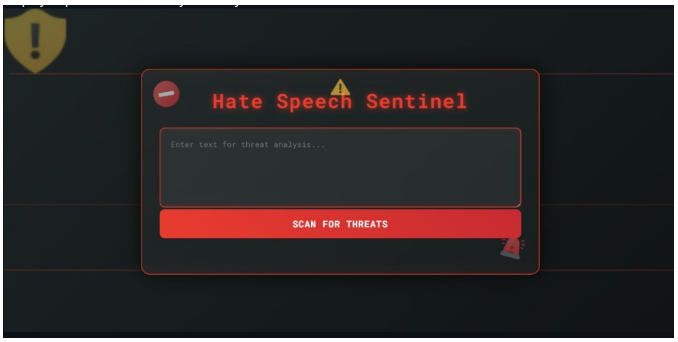

Fig. 12. Snippet from website

VII. FUTURE WORK

Building on the findings and limitations identified in this research, several promising directions can be pursued to enhance model effectiveness, ethical robustness, and real-world applicability:

1) **Advanced Techniques to Mitigate Class Imbalance**
Future models can benefit from approaches like:

- *Focal Loss* or *Class-balanced Loss* to emphasize learning on minority classes.
- *Multi-label Synthetic Oversampling* (e.g., ML-SMOTE) to artificially generate new training samples for underrepresented labels.
- *Data reweighting* or *dynamic sampling strategies* during training to ensure balanced exposure across labels.

2) **Utilizing More Powerful or Specialized Models**
Exploring newer transformer variants such as:

- *RoBERTa*, *DeBERTa*, and *XLNet*, which offer improvements in robustness and generalization over vanilla BERT.
- *Multitask Learning Frameworks* that jointly learn auxiliary tasks like sentiment detection or offensive intent classification to improve context capture.

3) **Incorporating Label Dependencies**
Since toxicity types are often correlated, modeling these relationships directly may enhance multi-label prediction. Possible methods include:

- *Classifier Chains* where predictions for one label inform another.
- *Graph Neural Networks (GNNs)* that represent label co-occurrence as graphs.
- *Attention-based Multi-label Models* that dynamically adjust weights based on label interactions.

4) **Data Augmentation for Linguistic Diversity**
Introducing syntactic and semantic variation in toxic content using:

- *Back-translation* for paraphrased toxicity.
- *Paraphrase generation models* (e.g., T5, Pegasus).
- *Adversarial data generation* to make models robust against evasion tactics.

5) **Scalable Deployment for Moderation**
In practical scenarios:

- Develop *real-time toxicity detection systems* using streaming inference with reduced latency.
- Include *human-in-the-loop systems* for validation of edge cases.
- Incorporate *feedback-based retraining pipelines* to adapt to emerging trends and new forms of toxic language.

VIII. ACKNOWLEDGEMENT

The authors acknowledge the support of the Department of Computer Science at Bennett University for providing the resources and research guidance necessary for the project.

Appreciation is also extended to open source resources and different research enthusiasts for their works in his field.

REFERENCES

[1] Zhang, Z., Robinson, D., Tepper, J. (2018), "Hate Speech Detection using a Convolution-LSTM Based Deep Neural Network" 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

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

[4] M. S. Jahan and M. Oussalah, "A systematic review of hate speech automatic detection using natural language processing," *Neurocomputing*, vol. 546, pp. 126232, 2023. [Online].

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

[6] Al-Hassan, Areej Al-Dossari, Hmood. (2019). DETECTION OF HATE SPEECH IN SOCIAL NETWORKS: A SURVEY ON MULTILINGUAL CORPUS. 83-100. 10.5121/csit.2019.90208.

[7] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), Minneapolis, MN, USA, Jun. 2019, pp. 4171–4186, doi: 10.18653/v1/N19-1423.

[8] J. S. Malik, H. Qiao, G. Pang, and A. van den Hengel, "Deep learning for hate speech detection: A comparative study," arXiv preprint arXiv:2202.09517, 2022. [Online].

[9] A. R. Putri, E. Sari, and F. H. Putra, "Comparative analysis of machine learning classifiers for hate speech detection in Indonesian tweets," 2020 International Conference on Computer Science, Information Technology, and Electrical Engineering (ICOMITEE), Malang, Indonesia, 2020, pp. 99–104

[10] Amazon SageMaker: Deploying a Trained Model

[11] Amazon SageMaker Python SDK (TensorFlowModel class)

[12] AWS Lambda Developer Guide

[13] Amazon API Gateway Guide

[14] IAM Roles for SageMaker and Lambda