DSTI School of Engineering

Detection of toxic comments

Deep Learning Course Project

Stéphanie Agbodjogbe

Nathan De Blecker

Karl Do Santos Zounon

Hortis Doumakpe

Laura Lacombe

23/11/2025

Introduction

1. Context and Problem Statement: The Need for Automated Moderation

The objective of this project is to develop an automatic classification model capable of identifying and categorizing different types of toxicity found in online comments. This is a multi-label classification problem in which a single comment can simultaneously belong to several categories, obscenity, insult, hate speech, threats, personal attacks, and more. The challenge therefore goes far beyond a simple binary toxic / non-toxic distinction: it requires capturing the nuances, degrees, and facets of toxicity to improve the effectiveness of content moderation.

This work builds on the Kaggle competition **"Jigsaw Toxic Comment Classification Challenge"**, which provides a high-quality annotated dataset and a standardized evaluation framework for systematically comparing model performance.

Why is this project essential?

Several reasons underline both the relevance and the importance of this work:

A major societal challenge.

The proliferation of hate speech, harassment, and toxic content on digital platforms undermines public discourse and can have severe psychological consequences for exposed individuals. According to the ADL, nearly 40% of American internet users report having experienced online harassment. Given the ever-growing volume of content posted every minute, human moderation alone is no longer sufficient: partial automation has become indispensable.

A stimulating technical challenge.

From an NLP perspective, toxicity detection represents a particularly rich case study:

- Multi-label classification: categories are not mutually exclusive and may be strongly correlated.
- Complex contextual understanding: irony, sarcasm, coded or implicit language make the task difficult for naïve models.
- **Severe class imbalance**: most comments are non-toxic, requiring appropriate strategies to avoid biased models.
- **Direct industrial impact.**
  The solutions developed here are directly applicable across the digital ecosystem: social media platforms, forums, online news outlets, collaborative communities, and more—all of which face the challenge of maintaining safe, constructive discussion spaces.

2. Literature Review and Existing Approaches

Automatic toxicity detection is a dynamic research field positioned at the intersection of NLP, linguistics, and the social sciences. The methods used have evolved significantly over the past decade.

a) Traditional Approaches: Manual Features and Classical Classifiers

Before the rise of deep learning, approaches primarily relied on:

- **Bag-of-Words and TF-IDF representations**, which count word frequencies while ignoring context and semantics;
- **N-grams**, enabling limited modeling of local context;
- **Standard classifiers** such as logistic regression, SVMs, or random forests.

These methods are simple and interpretable but insufficient for capturing the subtleties of natural language.

b) The Contribution of Word Embeddings

A major breakthrough came with **word embeddings** (Word2Vec, GloVe, FastText), which represent words in continuous vector spaces, enabling semantic generalization. These representations served as the foundation for more complex architectures capable of capturing finer linguistic nuances.

c) Deep Learning and the Transformer Revolution

Deep learning models have gradually reshaped NLP:

- **RNNs and LSTMs** initially enabled the modeling of text sequences and long-range dependencies.
- The **Transformer architecture**, introduced by Vaswani et al. (2017), then revolutionized the field with its global attention mechanism, surpassing RNN-based models in performance and efficiency.

Pre-trained language models such as **BERT** established a new paradigm: pre-training on massive general corpora, then fine-tuning on specific tasks. Improved or more efficient variants—RoBERTa, DeBERTa, DistilBERT—quickly emerged, significantly advancing the state of the art in toxicity detection.

d) The Jigsaw Dataset: A Reference Standard

The dataset provided by Jigsaw contains over 150,000 Wikipedia comments annotated across six toxicity categories. Its size and annotation quality make it a recognized benchmark, enabling rigorous evaluation and direct comparison with the wider data science community.

e) Current Challenges and Directions for Improvement

Despite recent advances, several challenges remain:

- **Bias in training data**: models may reproduce or even amplify societal biases present in the dataset, leading to problematic errors, particularly for identity-related terms.
- **Adversarial robustness**: modified spelling (“id!ot”), invisible characters, or paraphrasing can be used to evade detection systems.
- **Interpretability**: techniques such as LIME and SHAP are increasingly necessary to provide moderators with reliable, transparent explanations for model decisions.

This project therefore lies at the forefront of applied NLP research. By leveraging the Jigsaw benchmark dataset and exploring state-of-the-art architectures such as pre-trained Transformers, our goal is not only to build a high-performing classifier but also to deepen the understanding of the practical challenges of automated moderation. The remainder of this report details our methodology, from exploratory data analysis and text preprocessing to model implementation, training, comparative evaluation, and finally the discussion of results and future perspectives.

Methods

1. General Approach

The overall strategy follows a classical but robust deep learning workflow for multi-label text classification. The objective is to build an efficient and reproducible system capable of detecting several toxicity categories simultaneously at the comment level. The task is framed as ***supervised learning***, combining:

- modern NLP architectures (pre-trained Transformers),
- a carefully designed **preprocessing pipeline** for noisy user-generated text,
- and a set of **evaluation and rebalancing techniques** tailored to highly imbalanced labels.

The methodology can be summarized as:

1. **Exploratory data analysis** to understand the Jigsaw dataset and identify key challenges (severe imbalance, noisy text, multi-label correlation).
2. **Text cleaning and normalization** to reduce noise while preserving semantics relevant to toxicity.
3. **Train/validation split** using a semi-stratified approach based on a binary "any toxicity" signal.
4. **Baseline model** using TF-IDF features and logistic regression for comparison.
5. **Fine-tuning Transformer models** (DistilBERT, then BERT-base) for multi-label classification.
6. **Handling class imbalance** via a combination of positive class weighting and oversampling through a WeightedRandomSampler.
7. **Threshold tuning** per label to convert sigmoid probabilities into binary predictions.
8. **Deployment** of the final BERT model on Hugging Face, with a lightweight Python inference script and an interactive Gradio application.

This approach emphasizes both **scientific rigor** (reproducible experiments, explicit baselines, properly defined metrics) and **practical usability** (deployable model and simple API for real-world integration).

2. Data description

The Jigsaw Toxic Comment Classification dataset contains **159,571** user-generated comments and **8 columns**. Each row corresponds to a Wikipedia comment annotated for different types of toxicity.

- *id*: unique identifier of the comment (string), no missing values.
- *comment_text*: raw textual content of the comment (string), no missing values.

The remaining six columns are **binary label indicators** (int64, values 0/1):

Table 1 : Definitions of labels

toxic	*severe_toxic*	*obscene*	*threat*	*insult*	*identity_hate*
generic toxicity	highly toxic content	vulgar or offensive language	explicit threats	insulting or derogatory language	hateful speech targeting a group or identity

Each comment can belong to **multiple categories at once**, which makes this a **multi-label classification problem** rather than multi-class. No missing values were found in either text or labels, so the dataset is immediately suitable for supervised learning without imputation.

3. Exploratory Analysis

a) Label Distribution and Class Imbalance

The first step was to examine the distribution of labels and the proportion of toxic vs non-toxic comments.

- Around **90.4**% of comments have **no toxicity label at all**.

Table 2 : Representation of labels in the sample in percentage

toxic	*severe_toxic*	*obscene*	*threat*	*insult*	*identity_hate*
9.6	1.0	5.3	0.3	4.9	0.9

In other words, a naïve classifier predicting "**non-toxic**" **for every comment** would already achieve a very high **accuracy** but be nearly useless in practice. This confirms the need to rely on : metrics that are **robust to imbalance** (macro F1, per-label F1, average precision) and **rebalancing strategies** (class weights, oversampling, and threshold tuning).

These choices are reflected both in the model training procedure and in the evaluation protocol described later.

b) Comment Length

We also explored the distribution of comment length, both in characters and words. Most comments are relatively short, but a non-negligible number of very long comments exist. Density plots and box plots did not reveal any clear correlation between length and toxicity: short and long comments can be either toxic or non-toxic.

As a result, we decided not to encode length explicitly as a feature. Instead, length is handled implicitly by the Transformer tokenizer through truncation and padding.

4. Text Cleaning and Normalization

Because the dataset consists of noisy, user-generated text, a dedicated cleaning function *clean_text_series* was implemented and applied to the *comment_text* column. The goal was to remove obvious noise while preserving lexical and semantic signals that may correlate with toxicity.

The main transformations are:

1. **Unicode Normalization (NFKC)**
   All characters are normalized to a canonical, compatible form. This avoids treating different representations of the same symbol (accented characters, ligatures, special punctuation) as distinct tokens.
2. **Replacement of Structured Entities**
   - URLs are replaced by *[URL]*.
   - IPv4 addresses by *[IP]*.
   - E-mail addresses by *[EMAIL]*.

   This reduces sparsity due to arbitrary links or addresses while keeping the information that such entities were present (often relevant for spammy or abusive content).
3. **Case Normalization**
   All text is converted to lowercase. This reduces vocabulary size and is consistent with uncased pre-trained models such as *bert-base-uncased*.
4. **Whitespace and Line Break Cleanup**
   - Line breaks (*\n*, *\r*) are replaced by spaces.
   - Multiple spaces collapse into one.

   This ensures consistent tokenization and makes the text more regular without altering its semantics.
5. **Character Repetition Reduction**
   To handle expressive, informal writing:
   - sequences of the same letter repeated more than twice (e.g. "*fuuuuuck*") are reduced to *two* occurrences (→ "*fuuck*"),
   - very long sequences of punctuation (e.g. "*??????*") are clipped to at most *three* characters.

   This keeps a trace of emphasis and tone (important toxicity detection) while avoiding explosions of rare tokens.

A typical example illustrates the effect of this pipeline:

Original:
"Explanation\nWhy the edits made under my username Hardcore Metallica Fan were reverted? ... 89.205.38.27"
Cleaned:
"explanation why the edits made under my username hardcore metallica fan were reverted? ... [IP]"

The cleaned text is stored in a new column *clean_comment*, which is then used for all subsequent modeling steps. No empty comments were created by this process.

5. Train/Validation Split

To evaluate models fairly, the dataset was split into:

- **Training set**: 143,613 comments (**90**%),
- **Validation set**: 15,958 comments (**10**%).

Because full multi-label stratification is non-trivial, we used a semi-stratified approach based on a binary indicator *detect_toxic* that is equal to 1 if at least one of the six labels is positive. The split is stratified on this variable so that the proportion of "any toxic" vs "completely clean" comments remain similar in train and validation.

This does not guarantee perfect balance for the rare labels (e.g. *threat*), but the label distributions remain very close between training and validation sets, which is sufficient for the present project. For future work, multi-label stratification or cross-validation could be considered.

The cleaned and split datasets persisted as:

- *train_clean_full.csv:* full cleaned dataset,
- *train_clean.csv:* training subset,
- *valid_clean.csv:* validation subset.

6. Baseline Model: TF-IDF + Logistic Regression

Before moving to deep learning, we implemented a baseline using:

- **TF-IDF** features on the cleaned comments (*clean_comment*),
- a **one-vs-rest logistic regression** classifier for each label.

The text is vectorized with standard word-level TF-IDF, with a moderate vocabulary size to keep training fast and avoid overfitting. A separate logistic regression model is trained for each of the six labels, using the same features but different targets.

This baseline has several roles:

- It provides a simple, interpretable reference.
- It allows us to quantify the benefits of pre-trained Transformers in a fair way.
- It is very cheap to train and can be used as a fallback model if needed.

The performance of this baseline is reported in the Results section.

7. Transformer Models and Training Procedure

We then moved to Transformer-based models, using the Hugging Face transformers library:

- **DistilBERT** (*distilbert-base-uncased*)
- **BERT-base** (*bert-base-uncased*), which is ultimately selected as the final model.

Both models are fine-tuned on the cleaned text for multi-label classification. The architecture is standard:

- a pre-trained encoder (DistilBERT or BERT),
- followed by a linear classification head with **six outputs** (one per label),
- and a **sigmoid** activation to obtain per-label probabilities.

a) Tokenization and Input Formatting

For Transformer models, we bypass the manual text cleaning and work directly on comment_text or clean_comment using the corresponding tokenizer:

- The tokenizer handles **subword tokenization** (WordPiece)
- inputs are **padded** and **truncated** to a fixed maximum sequence length:
  - *MAX_LEN* = 256 for *DistilBERT*,
  - *MAX_LEN* = 128 for *BERT-base* in the final experiment.

Inputs are encoded as:

- token IDs (*input_ids*),
- attention masks (*attention_mask*),
- and label tensors with shape (*batch_size*, 6).

b) DistilBERT Setup

For DistilBERT:

Table 3 : DistilBERT setup

MODEL_NAME	MAX_LEN	BATCH_SIZE	EPOCHS
"*distilbert-base-uncased*"	*256*	*32*	*10*

Training is performed using the Hugging Face *Trainer*, with an evaluation on the validation set at the end of each epoch. The model uses the AdamW optimizer with default settings. This experiment serves as a first deep learning benchmark and a way to validate the overall pipeline.

c) BERT-base Setup (Final Model)

For the final model, we use BERT-base (bert-base-uncased) with a slightly shorter maximum length to reduce memory usage:

Table 4 : BERT-base setup

MODEL_NAME	MAX_LEN	BATCH_SIZE	EPOCHS
"*bert-base-uncased*"	*128*	*16*	*4*

The training is again orchestrated by a custom subclass of *Trainer* called *WeightedBCETrainer*, which overrides the loss computation to incorporate class-specific positive weights (see next subsection). The *TrainingArguments* specify:

- *eval_strategy* = "*epoch*" and *save_strategy* = "*epoch*",
- *load_best_model_at_end* = *True*,
- *metric_for_best_model* = "*f1_macro*",
- logging every 200 steps.

Thus, checkpoints are saved at the end of each epoch, and the best model according to macro F1 on the validation set is reloaded at the end.

The final BERT model is also saved locally in a folder called *bert-base-uncased_epoch4*. In parallel, the model is pushed to a public Hugging Face repository (*NathanDB/toxic-bert-dsti*) to simplify deployment, sharing and used in the Gradio App and test.py script.

8. Handling Class Imbalance

Given the extreme imbalance of some labels, we combined two complementary strategies: Positive class weighting in the loss function and Oversampling of minority labels at the sample level.

a) Class Weights in the Loss

Class frequencies on the training set were aggregated to compute, for each label, the number of positive and negative examples. A vector of positive class weights *pos_weight* was then defined as:

$$\text{pos\_weight}_\ell = \sqrt{\frac{\text{negatives}_\ell}{\text{positives}_\ell + \epsilon}}$$

with a small smoothing term ϵ for numerical stability. For the *threat* label, an additional mild boost was applied (×1.5) to compensate for its extremely low frequency.

Table 5: Weight results

toxic	*severe_toxic*	*obscene*	*threat*	*insult*	*identity_hate*
3.07	9.89	4.22	27.18	4.39	10.57

These weights are passed to *BCEWithLogitsLoss* through the *pos_weight* argument inside the custom *WeightedBCETrainer*. This increases the penalty for false negatives on rare labels, encouraging the model to detect them more often.

b) Sample-Level Oversampling

To further mitigate imbalance, we also use a *WeightedRandomSampler* at the sample level. The idea is to oversample comments that contain rare labels, while not completely discarding non-toxic comments.

We first compute normalized label weights (inverse frequency) such that rare labels receive higher weights:

Table 6: Higher weights

toxic	*severe_toxic*	*obscene*	*threat*	*insult*	*identity_hate*
0.0177	0.1675	0.0319	0.5577	0.0343	0.1910

For each training comment, its sample weight is computed as the sum of the weights of its positive labels. Comments with no positive label receive the mean weight, so they still appear in the training batches.

These sample weights are then passed to *WeightedRandomSampler*, which is used by the *DataLoader* during training. As a result, batches contain a more balanced mixture of comments, and minority labels are seen more frequently by the model.

9. Threshold Tuning and Evaluation Protocol

The model outputs sigmoid probabilities between 0 and 1 for each label. The simplest strategy to obtain binary predictions is to use a global threshold of 0.5. However, because labels have very different base rates, this can be suboptimal.

We therefore tune one threshold per label on the validation set, with the objective of maximizing per-label F1 score. The optimization procedure is simple but effective:

- For each label separately, we sweep a set of candidate thresholds in [0.05, 0.95] and compute the F1 score on the validation set.
- We pick the threshold that maximizes F1 for that label.
- The resulting vector of thresholds is then used at inference time.

The final thresholds used in the inference script (test.py) are approximately:

Table 7 : final thresholds

toxic	*severe_toxic*	*obscene*	*threat*	*insult*	*identity_hate*
0.90	0.25	0.90	0.10	0.40	0.15

For reporting model performance, we rely on : micro F1 (global balance of precision/recall across all predictions), macro F1 (unweighted mean of F1 scores over the six labels), per-label F1 to assess which categories remain difficult, and occasionally Hamming loss, subset accuracy, and macro average precision for the baseline and DistilBERT models.

10. Deployment: Hugging Face Model, Inference Script, and Gradio App

Ensuring reproducibility, accessibility, and maintainability was a central requirement of the project. To meet these objectives, a deployment workflow was designed that enables seamless model distribution, consistent execution across environments, and long-term traceability.

a) Hugging Face model hosting

The fine-tuned Transformer model, due to its size could not be stored efficiently within in a standard Git repository. To address this constraint, the model and its associated tokenizer were uploaded to the Hugging Face hub under this namespace:

model = AutoModelForSequenceClassification.from_pretrained("NathanDB/toxic-bert-dsti")

Hosting the model on the Hugging Face Hub offers several methodological advantages:

- **Versioning and traceability:** each upload is automatically versioned, ensuring reproducibility across experiments and deployments.
- **Bundled dependencies:** model weights, configuration files, and tokenizer artifacts are stored together, preventing inconsistencies across setups.
- **Universal accessibility:** the model can be retrieved from any environment using a single command, without requiring manual downloads.
- **Repository cleanliness:** large binary artifacts are decoupled from the main project, keeping the codebase lightweight and maintainable.

b) Inference Script

This script is the inference module we use to run the fine-tuned toxicity detection model on new pieces of text. It first loads a BERT-based classifier (*NathanDB/toxic-bert-dsti*) and its tokenizer from the Hugging Face Hub, automatically choosing GPU if available or falling back to CPU. The code then tries to load a set of per-label decision thresholds (for *toxic, severe_toxic, obscene, threat, insult, and identity_hate*) from a local JSON file; if that file is not present, it fetches the same thresholds from the model repository on Hugging Face, and if that also fails it defaults to a threshold of 0.5 for all labels. When calling *predict_toxicity*, the function tokenizes one or several input comments, runs them through the model, and applies a sigmoid to obtain a probability between 0 and 1 for each toxicity category. These probabilities are then compared to the pre-computed thresholds to produce binary predictions (0 or 1) for each label. Finally, the function returns three elements: the raw probability matrix, the binary prediction matrix, and a list of dictionaries that map each label to its predicted value for each input text, which makes the output easy to log, interpret, or plug into a user interface.

c) Gradio App

This file is the Gradio web interface we built on top of the toxicity classifier so that people can test the model directly in a browser at this adress. When the app starts, it loads the fine-tuned Hugging Face model *NathanDB/toxic-bert-dsti* and its tokenizer on CPU or GPU, and it also loads a vector of decision thresholds for each label, either from a local JSON file or, if that is missing, from the model repository on Hugging Face (otherwise it falls back to 0.5). The main callback *analyze* takes the user's text from the textbox, runs it through *predict_toxicity* to get a probability for each class, applies the thresholds to obtain binary decisions, and then passes everything to a helper that builds a detailed HTML "report" for the interface. This report shows a global risk badge (low/medium/high), a short verbal summary, a bar chart per label with the threshold marked, and a block of the original text where "risky" words are highlighted using a small hand-crafted dictionary of toxic terms; each label is also given a short explanation so the scores are easier to interpret. For transparency and analysis, every request is logged server-side in a CSV file that contains a UTC timestamp, the raw text, the list of predicted labels, and the per-label probabilities encoded as JSON. On Hugging Face Spaces this log is written to /data, which is persistent but not part of the public repository, so normal users cannot browse it. Finally, the app exposes two download features: any user can download a CSV file for the current prediction only, and there is a separate "Admin tools" accordion where an administrator can enter a secret key (provided via an environment variable) to download the full history file; if the key is missing or incorrect, the download is simply refused.

d) Docker-Based Deployment

To ensure that the toxicity detection system can run reliably in any environment, we packaged the inference pipeline into a Docker container. The goal was to obtain a setup that behaves identically on a laptop, on a server, or on a cloud-hosted instance such as Hugging Face Spaces. The container wraps the entire execution environment of the model, Python version, system libraries, Python dependencies, and the Gradio application, inside a single reproducible artifact.

The container is built from the official *python:3.12* base image and installs only the system tools required for lightweight deep learning inference (*build essential, git, curl*). All Python dependencies are installed directly from a *requirements.txt* file, which guarantees that the versions used during development are exactly the ones running in production. The application code is copied inside */app*, which serves as the working directory for the container.

At runtime, the model is automatically downloaded from the Hugging Face Hub during the first execution of *app.py*. This avoids embedding large model weights inside the container and ensures that the exact version stored in the Hub is always used. The Gradio interface exposes its API on port *7860*, which is declared in the image through the *EXPOSE* instruction.

Because everything, from the Python interpreter to the Hugging Face libraries, is fixed inside the image, the container behaves the same regardless of where it is deployed. This eliminates "works on my machine" issues and makes the application portable across teams, machines, and platforms. In practice, the Docker setup provides a clean and dependable way to run the toxicity classifier locally, in a CI pipeline, or in any cloud environment that supports containers.

Results

This section reports the empirical performance of the three main modeling stages:

1. TF-IDF + logistic regression (baseline),
2. DistilBERT,
3. BERT-base (final model with class weighting and oversampling).

We focus on **multi-label metrics**, especially macro F1, per-label scores, and comparisons between models.

1. Baseline: TF-IDF + Logistic Regression

Table 8 : Baseline model scores on the validation set

F1 micro	F1 macro	Hamming loss	Subset accuracy	Macro average precision	Macro ROC-AUC
0.7108	0.5393	0.0177	0.9210	0.6197	0.9756

The high subset accuracy and ROC-AUC mostly reflect the dominance of the non-toxic class. A finer view is obtained by examining per-label F1 scores:

Table 9: F1 scores

Label	F1 (baseline)
toxic	0.770
severe_toxic	0.396
obscene	0.766
threat	0.357
insult	0.656
identity_hate	0.291

The baseline already captures **general toxicity and obscenity** reasonably well, but performance for **rare and subtle categories** such as *threat* and *identity_hate* remains low (F1 ≈ 0.29–0.36). This confirms that more expressive models are necessary.

2. DistilBERT

The DistilBERT model, fine-tuned for 10 epochs with a maximum length of 256 tokens and a batch size of 32, significantly improves upon the baseline:

- **F1 micro**: 0.7662
- **F1 macro**: 0.6355

The gain in macro F1 is about +0.10 absolute compared to logistic regression (0.535 → 0.636), which shows that pre-trained contextual embeddings help capture nuances beyond what bag-of-words features can represent, especially for minority labels.

However, DistilBERT is still a distilled model with fewer parameters. We therefore explored whether full BERT-base could provide further improvements at a reasonable computational cost.

3. BERT-base: Learning Dynamics and Global Metrics

The final BERT-base model is trained for **4 epochs** with class weighting and oversampling. Validation macro F1 after each epoch is:

Table 10 : F1 macro validation after each epoch

Epoch 1	Epoch 2	Epoch 3	Epoch 4
0.5940	0.6556	0.6750	0.6727

Performance improves quickly during the first two epochs, then stabilizes around 0.67. There is no clear sign of overfitting within 4 epochs, but gains beyond epoch 3 are marginal, justifying the chosen number of epochs.

The best validation performance obtained is:

- **Final validation F1 macro: 0.6727**

This is about **+0.037** absolute above DistilBERT and **+0.133** above the TF-IDF baseline, confirming the benefit of the larger model combined with tailored imbalance handling.

Although micro F1 is not explicitly logged for BERT in the notebook, its per-label precision and recall (see below) indicate a strong overall trade-off, with especially clear gains on the rare classes.

4. Per-Label Performance of BERT-base

The classification report for the final BERT-base model on the validation set yields the following per-label metrics:

Table 11 : BERT-based classification report (validation)

Label	Precision	Recall	F1	Support
toxic	0.809	0.855	0.831	1480
severe_toxic	0.463	0.466	0.465	148
obscene	0.838	0.836	0.837	836
threat	0.500	0.622	0.554	37
insult	0.773	0.781	0.777	791
identity_hate	0.612	0.537	0.572	147

Compared to the baseline logistic regression, the F1 gains are substantial across all labels:

Table 12 : F1 improvement over logistic regression

Labels	*toxic*	*severe_toxic*	*obscene*	*threat*	*insult*	*identity_hate*
Baseline logistic regression	0.77	0.40	0.77	0.36	0.66	0.29
BERT-base	0.83 (+0.06)	0.47 (+0.07)	0.84 (+0.07)	0.55 (+0.20)	0.78 (+0.12)	0.57 (+0.28)

The largest improvements are observed precisely on the most challenging and rare labels (*threat* and *identity_hate*). This indicates that the combination of contextual embeddings from BERT, positive class weighting in the loss, and sample-level oversampling successfully addresses part of the imbalance problem.

5. Model Comparison Summary

Putting all models side by side:

Table 13 : Overall comparison of models

Model	F1 micro	F1 macro
TF-IDF + Logistic Regression	0.7108	0.5393
DistilBERT	0.7662	0.6355
BERT-base (final)	N/A	0.6727

Even without the exact micro F1 for BERT, the macro F1 trajectory clearly shows:

- A strong baseline provided by TF-IDF + logistic regression,
- A first major leap with DistilBERT (+0.10 macro F1),
- A second, smaller but meaningful gain with BERT-base (+0.037 macro F1 above DistilBERT).

From a cost-benefit perspective:

- DistilBERT is lighter and faster, already giving good results.
- BERT-base is heavier but offers noticeably better performance, especially on minority labels, which are often the most important in moderation scenarios (threats, identity-based hate).

Given the project's objectives and available resources, we chose BERT-base as the final model, implemented threshold tuning and rebalancing on top of it, and deployed this version on Hugging Face and in the Gradio app.

6. Practical Behavior and Observations

Beyond numerical metrics, several practical observations emerge:

- The model is confident and accurate on clearly abusive content, often assigning high probabilities (>0.9) to *toxic*, *obscene* or *insult* when appropriate.
- For more borderline or sarcastic comments, probabilities are more moderate and spread across labels, which justifies the use of label-specific thresholds instead of a blunt 0.5 cut-off.
- The tuned thresholds make the model stricter on generic toxicity (*toxic* and *obscene*, threshold ≈ 0.9) while remaining sensitive to rare but critical labels like *threat* and *identity_hate* (thresholds 0.10 and 0.15).

Discussion

1. Interpretation of Results

The experimental results reveal several important insights regarding the modelling choices and the intrinsic difficulty of multi-label toxicity detection.

a) Impact of Model Architecture

The progression from TF-IDF + Logistic Regression to DistilBERT and finally BERT-base highlights a consistent and significant improvement in model performance. This confirms several findings reported in the scientific literature:

- **Contextual embeddings are essential**: Transformers capture deeper semantic and syntactic patterns that remain inaccessible to traditional statistical models or RNNs (Devlin et al., 2018). As a result, rare or implicit forms of toxicity—such as indirect threats or identity-based hate expressed through coded language—are detected more reliably.
- **Model capacity matters, but in diminishing returns**: the performance gain obtained when moving from DistilBERT (66M parameters) to BERT-base (110M) confirms that increased model capacity improves expressiveness and generalization. However, the marginal improvement begins to taper off. In particular, Sanh et al. (2019) show that a distilled model such as DistilBERT retains approximately 97% of BERT-base's performance despite being 40% smaller, illustrating the phenomenon of diminishing returns.In practice, this means that extremely large architectures (e.g., RoBERTa-large or DeBERTa-xlarge) may not yield proportionally higher performance for this dataset, especially when considering deployment constraints, inference speed, and resource limitations.

b) Effects of Class Imbalance Techniques

The combination of class weighting and oversampling proved decisive. Without these strategies, the rarest labels (especially *threat*, ~0.3%) had near-zero recall. Weighted loss alone was insufficient, as rare examples simply did not appear often enough during training. Conversely, oversampling alone amplified noise and led to unstable gradients.

The hybrid approach ensured that:

- rare classes became "visible" during training,
- the model learned meaningful patterns specific to minority labels,
- performance improved without causing catastrophic overfitting.

c) Importance of Per-Label Threshold Tuning

One of the most striking findings is the impact of per-label threshold optimization. The naive global threshold of 0.5 severely undervalued minority classes, leading to poor recall. After tuning:

- **threat** recall increased significantly with a threshold as low as 0.10
- **obscene** required a high threshold (0.90) to avoid over-prediction due to lexical biases
- **identity_hate**, a nuanced category, benefitted from a moderate threshold (0.15)

This underscores that toxicity categories have heterogeneous statistical behaviors. Treating them identically is suboptimal both mathematically and operationally. In the context of moderation systems—where false negatives on threats or hate speech are critical—threshold calibration becomes not just a technical step, but a safety requirement.

d) Error Patterns and Moderation Challenges

Several recurrent error patterns were observed:

- Sarcastic toxicity remains difficult even for BERT-like models.
- Borderline cases ("heated but not abusive discussions") lead to confusion between insult and general toxic.
- Identity terms sometimes trigger false positives.

These findings resonate with the limits of supervised learning when annotations reflect social or annotator biases. They also highlight the importance of continuous monitoring and post-deployment evaluation for any moderation system.

e) Generalization and Real-World Robustness

The model performs well on the validation set, but real moderation environments exhibit: rapidly evolving slang, multimodal or contextual toxicity… Transformers capture many nuances, but remain vulnerable to adversarial spelling and domain shifts. Threshold tuning and oversampling help mitigate this but cannot fully eliminate these robustness limits.

2. Limitation

Despite the strong performance of the final model, several limitations must be acknowledged, both in terms of data quality and modeling constraints. These limitations influence the model's robustness, fairness, and generalization capabilities, and they highlight the inherent challenges of automatic toxicity detection.

a) Dataset Biases

A central limitation stems from the Jigsaw dataset itself. As widely documented in prior research, the dataset exhibits demographic and identity-term biases. As a result, the model tends to overpredict identity_hate for comments referencing minority groups, thereby raising concerns about fairness across demographic subpopulations. Furthermore, the dataset offers limited linguistic variety for rare labels, especially threat and identity_hate. Even with oversampling and class weighting, the scarcity of clean, diverse positive instances restricts the model's ability to generalize to novel threats, paraphrased insults, or emerging coded language. This limitation is structural and cannot be fully resolved without expanding or rebalancing the training corpus.

b) Modeling and Thresholding Constraints

Several constraints arise from the modeling choices. First, although Transformers provide excellent contextual representations, their computational cost imposed practical restrictions, such as limiting the maximum sequence length to 128–256 tokens. This led to the truncation of longer comments. Additionally, while the combination of oversampling and loss weighting significantly improved recall on minority labels, the model remains highly sensitive to thresholding decisions. Although threshold tuning improves the model's predictions, it also creates certain challenges: The thresholds may not work everywhere: if the model is used on another platform (Twitter, Reddit, YouTube…), the distribution of toxic comments may be different, and the thresholds might no longer be optimal. The thresholds require maintenance: because language evolves and new forms of toxicity appear, the thresholds may need to be updated regularly. In summary, the predictions are not absolute, they are probabilities and turning them into binary labels always depends on the chosen threshold.

c) Limitations of Interpretability and Robustness

Even though BERT-based models outperform classical approaches, they remain difficult to interpret. Their internal decision process is not directly observable, and without additional tools such as LIME, SHAP, or attention visualization, it is hard to explain why a prediction was made. This is a common limitation of deep learning models and was beyond the scope of the present project, but it is an important aspect for reliable moderation systems.

In addition, the model remains sensitive to adversarial or creative forms of toxicity, such as altered spellings ("1d!ot"), emerging slang, or changes in online discourse. Transformers help capture many linguistic variations, but they cannot fully anticipate new toxic expressions or domain shifts. While our preprocessing steps and threshold tuning reduce some of these vulnerabilities, they cannot eliminate them entirely. Continuous monitoring and periodic retraining would be required in real-world settings.

d) Future work

Future work could explore interpretability approaches such as attention-weight visualization or gradient-based explanation techniques, which would help better understand the model's decisions without requiring additional training. Larger or more balanced datasets, or architectures designed for long sequences, could also improve performance on rare or contextual toxicity. Robustness could be strengthened through adversarial data augmentation and dynamic threshold calibration. Finally, regular monitoring and retraining would be essential in a real moderation pipeline, as online language evolves rapidly.

Conclusion

The project explored how different NLP approaches behave when confronted with the complexity of toxic online conversations. Starting with a simple TF-IDF + logistic regression baseline helped clarify the strengths and limitations of traditional models: although they handled the most common forms of toxicity reasonably well, they quickly reached their limits on subtle categories such as threats or identity-based hate.

The transition to Transformer-based models marked a clear step forward. DistilBERT already brought a substantial gain by leveraging contextual word representations, and BERT-base pushed performance further, especially on the rarest labels. These improvements did not come “for free”: careful handling of class imbalance, thoughtful preprocessing, and per-label threshold tuning were all essential to obtain a model that behaves reliably across different forms of toxicity.

Beyond the numerical results, the project highlighted several broader lessons. First, automatic moderation is not simply a technical problem; it reflects the ambiguity and diversity of how people express themselves online. Second, even strong models require regular monitoring, as language evolves quickly, and new patterns of abuse appear over time. Finally, deploying the system through a clean inference script and a Gradio interface showed that machine-learning models can be made accessible to non-technical users when the right tools and design choices are in place.

Overall, the final model delivers robust and interpretable predictions, while remaining light enough to be deployed in practical environments. Although challenges remain, biases in the dataset, sensitivity to domain shifts, and limited interpretability, the work carried out here provides a solid foundation for a real-world toxicity detection pipeline and opens the door to further improvements such as larger datasets, better explainability tools, and periodic retraining. In that sense, the project successfully balances methodological rigor with operational pragmatism and demonstrates how modern NLP techniques can meaningfully support safer and more constructive online interactions.

Appendix

Table of figures