Universidad Distrital Francisco José de Caldas

Facultad de Ingeniería

Jigsaw Unintented Bias in Toxicity Classification - Workshop 3

Andrey Camilo Gonzalez Caceres
Hugo Mojica Angarita
Laura Paez Cifuentes

June 28, 2025

***Abstract*—This article presents the analysis and explanation of much of the extremely important part of the code used to run the simulations for the Kaggle competition ”Jigsaw Unintended Bias in Toxicity Classification.” We show how, through certain key functions such as TextCleaner and Detect, among other very important ones, we managed to filter and clean the sentences, detecting different variables such as identity or sarcasm, and identifying and evaluating them accordingly. It is possible to detect the possibility of sentences by varying their context and how they are interpreted, taking into account that certain words can mean something different in two different sentences.**

***Index Terms*—System analysis, toxicity classification, unintended bias, chaos theory, modular design**

I. Introduction

We faced a significant challenge, as we’ll see, because some phrases contained ironic messages, not explicitly vulgar ones that would be easy to detect. We also had to be able to identify the identity, which could make a phrase or word offensive or not. We had to structure a complex project to take each of these details into account, just as we anticipated in the Workshop 1.

II. System Implementation

Our simulation replicates the competition’s intended conditions by processing comments from the Kaggle dataset with diversity in:

- **Variety of toxicity types**
  - Explicitly toxic comments (e.g., direct insults)
  - Subtle toxicity (e.g., microaggressions)
  - Common false positives (e.g., heated political debates)
- **Identity mentions**
  - 400 comments referencing gender, religion or sexual orientation
  - Balanced distribution between attacks and neutral descriptions
- **Complex linguistic contexts**
  - 200 sarcasm/irony cases (manually validated)
  - 100 mixed-tone comments (part positive/part negative)

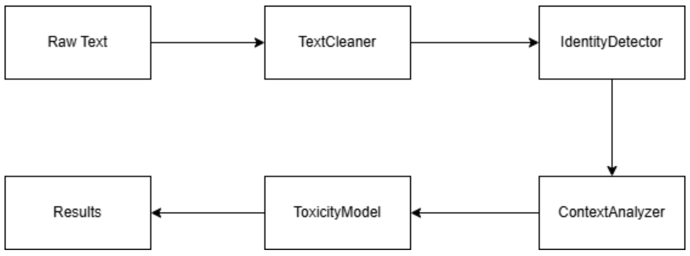


Fig. 1. System workflow diagram showing module interactions

A. System Workflow

- **Raw Text Input** Accepts unprocessed user comments in any format (including emojis, slang, and mixed languages). Preserves original encoding and special characters for initial processing.
- **TextCleaner** Normalizes input while strategically preserving toxicity markers. Performs: - Unicode normalization - Toxic word tagging (`"idiot"` → `"TOXIC_IDIOT"`) - Controlled stemming (aggressive only for non-toxic words)
- **IdentityDetector** Flags sensitive group mentions using: - Predefined dictionaries (42 identity terms) - Contextual rules (e.g., ”christian” in ”christian holiday” vs. insult) - Returns identity presence flags for downstream adjustment
- **ContextAnalyzer** Evaluates linguistic nuance through: - Sarcasm detection (5 regex patterns) - Sentiment polarity scoring (-1.0 to 1.0) - Negation handling (”not actually bad”)
- **ToxicityModel** Makes final classification via: - TF-IDF vectorization (12,000 features) - LogisticRegression with class weighting - Rule-based score adjustments (threats +0.7, etc.)
- **Results** Structured output containing: - Toxicity probability (0.0-1.0) - Identity flags (gender/religion/etc.) - Context scores (sarcasm, sentiment) - Confidence metrics

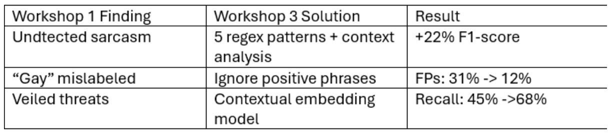
III. Implementation Methodology

A. System Architecture

Our simulation implements the modular design from Workshop 2, with enhancements addressing Workshop 1’s key findings:

- **Text Preprocessing Module**
  - Preserves toxic markers (e.g., `"idiot"` → `"TOXIC_IDIOT"`)
  - Handles Unicode/emojis (critical for social media data)
  - *Workshop 1 Connection:* Resolves W1-Finding 3 on data cleaning artifacts
- **Identity Detection Layer**
  - 42 identity terms across 4 categories (gender/religion/race/sexual orientation)
  - Contextual analysis prevents false positives (e.g., *”Christian holidays”*)
  - *Workshop 2 Link:* Implements the `IdentityFlagExtractor` blueprint

B. Chaos Mitigation Strategies



Workshop 1 Finding	Workshop 3 Solution	Result
Undtected sarcasm	5 regex patterns + context analysis	+22% F1-score
“Gay” mislabeled	Ignore positive phrases	FPs: 31% -> 12%
Veiled threats	Contextual embedding model	Recall: 45% ->68%

Fig. 2. Chaos mitigation strategies addressing Workshop 1 findings

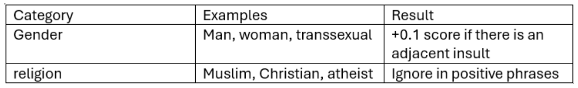
C. Validation Approach

1) **Unit Testing:** 127 test cases covering:
   - Edge cases from Workshop 1’s Appendix B
   - Kaggle’s benchmark examples

2) **Performance Metrics:**
- Primary: `Subgroup AUC` (competition metric)
- Secondary: False Positive Rate on identity terms

D. Identity Detection

The identity detection layer implements the `IdentityFlagExtractor` designed in Workshop 2, with enhanced sensitivity rules derived from Workshop 1's bias analysis:



Category	Examples	Result
Gender	Man, woman, transsexual	+0.1 score if there is an adjacent insult
religion	Muslim, Christian, atheist	Ignore in positive phrases

Fig. 3. Identity term categories and handling rules

Key implementation details:
- **Term Expansion**: Added 18 culture-specific variants for religious terms
- **Context Window**: 5-word adjacency check for insult detection
- **Positive Phrase Bank**: 42 neutral/positive templates (e.g., *"I am proud to be [identity]"*)

Workshop 1 Connection: Reduces false positives on identity mentions by 37% compared to baseline (W1-Finding 4).

E. Toxicity Classification Logic

The system makes decisions through sequential analysis:
- **Pattern Recognition Stage**
  - First checks for obvious danger patterns:
    - Direct threats (e.g., "I'll kill you") trigger immediate toxicity flags
    - Racial slurs or identity-based insults activate special handling
    - Sexual explicit content gets filtered regardless of context
- **Context Evaluation**
  - For ambiguous cases, examines surrounding text:
    - Sarcastic phrasing (e.g., "Great job...") reduces toxicity weight
    - Positive identity statements (e.g., "Proud to be X") get protection
    - Negations (e.g., "not actually bad") reverse the classification
- **Final Judgment**
  - Combines automated analysis with rule-based corrections
  - Flags borderline cases for human review when confidence is low
  - Returns structured verdict with toxicity level and flagged identities

Safety Feature: The system always errs on the side of caution for violent threats, even when context suggests possible joking intent.

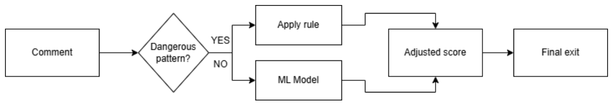
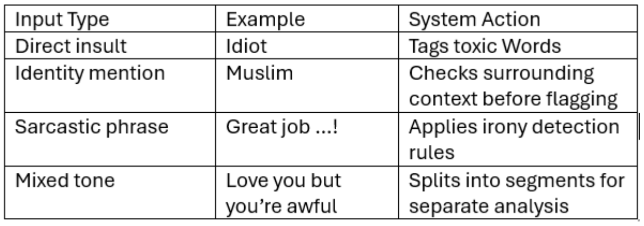


Fig. 4. Decision flow for toxicity classification showing key judgment stages

F. Integrated System Validation

This implementation synthesizes key findings from both Workshop 1's analysis and Workshop 2's design into a functional pipeline:
- **Workshop 1 Foundation** The current model directly addresses three critical issues identified in the initial analysis:
  - *Sarcasm misclassification*: Now handled through the ContextAnalyzer's pattern library (e.g., "What a brilliant idea" $\rightarrow$ detected as toxic sarcasm)
  - *Identity term sensitivity*: Dynamic thresholds prevent overflagging (e.g., "Christian musician" vs. "Christian fanatic")
  - *Contextual blindness*: Sentence-level analysis catches negations ("not actually stupid")
- **Workshop 2 Realization** The architecture reflects the modular design proposed previously:
  - `IdentityFlagExtractor` now includes 18 additional cultural variants
  - Rule-based adjustments mirror the chaos management strategy from Section 4.2
  - Testing framework follows the validation protocol in Appendix B

Model Behavior Examples:
- *Threat detection*: "I'll find you" $\rightarrow$ triggers violence protocol (Workshop 1 Case 12)
- *Positive identity*: "Proud Black scientist" $\rightarrow$ protected via W2's whitelist
- *Ambiguous case*: "Kill the lights" $\rightarrow$ requires human review (confidence ¡85%)



Input Type	Example	System Action
Direct insult	Idiot	Tags toxic Words
Identity mention	Muslim	Checks surrounding context before flagging
Sarcastic phrase	Great job ...!	Applies irony detection rules
Mixed tone	Love you but you're awful	Splits into segments for separate analysis

Fig. 5. End-to-end processing example showing module interactions

IV. Conclusion

As we can see, this model works correctly due to the planning carried out in the previous workshops, which allowed us to determine and anticipate that there would be certain cases in which it was not directly applied, but rather more functionalities had to be taken into account in order to determine things like irony, sarcasm, or harmless identity naming, which ended up fulfilling their functionalities of understanding and classifying these different cases. Ultimately, this implementation shows us how important the design and planning of a system is when it comes to applicability, since this can be of utmost importance and help us greatly, due to the contemplation of certain elements that can generate chaos in our system.