🤖 PeerSupportBot

An AI-powered Discord companion that keeps communities safe **and** humane. It combines fine-tuned toxicity and sarcasm models with clear policy rules, message redaction, empathetic DMs, violation tracking, and transparent reporting.

Table of Contents

Overview

PeerSupportBot isn’t a blunt filter. It’s a **supportive agent** that understands context (sarcasm, slang, banter), reacts decisively to serious harm (slurs, threats, sexual violence), and treats people with empathy. The bot:

- **Redacts** harmful messages (delete or replace with a neutral notice).
- **DMs** authors with a clear explanation and crisis resources when needed.
- Tracks **violations**, issues a **final warning** after 5, and generates **reports**.
- Uses a **seriousness score** that blends model signals with policy rules so outcomes are reliable and human-centred.

Goals & Principles

- **Safety first, with empathy.** Don’t just delete; explain and support.
- **Precision over bluntness.** Understand sarcasm and reduce false positives on everyday banter.
- **Transparency.** Log incidents, generate daily/rolling/special reports.
- **Modularity.** Clear agents: Sentinel (detect), Triage (decide), Responder (act), Archivist (log).

System Architecture

High-Level Flow

Code snippet

```
flowchart TD
   A[Incoming Message] --> B[Sentinel: Toxicity + Sarcasm]
   B --> C[Triage: Seriousness + Rules]
   C -->|Crisis| D[Responder: Redact + DM Crisis Resources]
   C -->|Serious| E[Responder: Redact + DM Warning]
   C -->|Moderate| F[Responder: Redact + Softer DM]
   C -->|None| G[Archivist: Log Only]
   D --> H[Archivist: DB + Reports]
   E --> H
   F --> H
   G --> H
   H --> I[Reports: Daily · Rolling · Special]
```

Agent Roles

🛡️ **Sentinel (Detection)** Runs detectors and returns: `tox_probs` (multi-label), `sarcasm_prob`, `tox_max`.

⚖️ **Triage (Decision)** Computes the seriousness score and applies policy overrides (crisis regex, extreme words, `tox_max` gates). Produces a decision tag: `NONE` | `WARN` | `SERIOUS` | `CRISIS`.

💬 **Responder (Interaction)** Executes actions: redact message, DM the author (empathetic tone, crisis resources if needed), and publish a final warning in channel after 5 violations.

📜 **Archivist (Logging & Reporting)** Stores incidents in SQLite, generates daily/rolling/special reports, and supports `/report` on demand.

Models & Why I Chose Them

DistilBERT + LoRA (fine-tuned on Jigsaw Toxic Comments) Multi-label outputs: `toxic`, `severe_toxic`, `obscene`, `threat`, `insult`, `identity_hate`. *Why*: Lightweight, fast, easy to adapt thresholds, and captures the different "flavours" of toxicity.

BERTweet (fine-tuned) for sarcasm *Why*: Sarcasm is common in peer communities. Without it, benign jokes are over-flagged and users lose trust.

Policy Rules (Safety Net) Regex for crisis/self-harm language and extreme words (e.g., sexual violence, kill threats, severe slurs). *Why*: No model is perfect. Rules guarantee coverage for high-stakes cases.

Pathway: The Seriousness Score (Math Explained)

The decision logic is intentionally hybrid: I combine continuous signals from models with discrete safety rules so I can be nuanced and reliable.

Let $p\_label \in [0,1]$ be the probability for each toxicity label from the multi-label model, and $s \in [0,1]$ be the sarcasm probability.

1) Toxicity Label Weights → *severity*

Some labels are more dangerous than others. I map label likelihoods to a severity scalar using conservative weights:

$$severity=\max(0.80 \cdot p\_threat, 0.75 \cdot p\_severe\_toxic, 0.70 \cdot p\_identity\_hate, 0.55 \cdot p\_toxic, 0.50 \cdot p\_insult, 0.45 \cdot p\_obscene).$$

Intuition: threats and identity-based hate carry the most risk.

2) Context & Sarcasm Relief

I allow small nudges from historical context and subtract a “banter relief” if the message is sarcastic (because sarcasm can soften perceived harm in playful exchanges):

$u = \text{avg recent\_user seriousness}, c = \text{avg recent\_channel seriousness}$
$\text{seriousness} = \text{clip\_}[0,1](\text{severity} + 0.10 \cdot u + 0.05 \cdot c - 0.25 \cdot s)$.

If you’re not tracking history yet, set $u=c=0$ (works fine).

3) Safety Floors & Overrides

- **Safety floor**: if p_threat≥0.50 or p_severe_toxic≥0.60, force `seriousness` to at least `0.80`.
- **Extreme words**: crisis/self-harm, sexual violence, kill threats, severe slurs → immediate `SERIOUS` tag.
- **`tox_max` override**: if max(p_label)≥0.80, force `SERIOUS` tag (even if sarcastic).

These rules ensure nothing obviously dangerous slips through model noise.

4) Final Decision Function

Given `seriousness` ∈[0,1] and the overrides above:

```
IF crisis_regex(text)       → tag = CRISIS,  redact = True, dm_user = True
ELIF extreme_regex(text)    → tag = SERIOUS, redact = True, dm_user = True
ELIF tox_max ≥ 0.80         → tag = SERIOUS, redact = True, dm_user = True
ELIF seriousness ≥ 0.65     → tag = SERIOUS, redact = True, dm_user = True
ELIF seriousness ≥ 0.45     → tag = WARN,    redact = True, dm_user = True
ELSE                        → tag = NONE,    redact = False, dm_user = False
```

Thresholds are configurable via environment variables.

5) Why this works in real life

- **Protects against false negatives** (hard overrides catch extreme harm).
- **Reduces false positives on everyday banter** (sarcasm relief).
- **Keeps decisions predictable and explainable** (logs include scores + reason).

Discord Integration

Required Bot Intents & Permissions

Intents (Bot settings → “Privileged Intents”):

- ✅ `MESSAGE CONTENT INTENT` (to read message text)
- ✅ `GUILDS` (standard)

Permissions in server (role for the bot):

- Manage Messages (to delete/redact)
- Send Messages
- Read Message History
- Use Slash Commands
- (Optional) Attach Files (for sending report files)

If you see `PrivilegedIntentsRequired`, enable intents in the Developer Portal and/or remove unnecessary ones from code.

Slash Commands

- `/report` → generates a channel-scoped report since last run and returns a Markdown file.

Reports & Accountability

- **Daily Report** at 23:59 IST (configurable): totals, incident breakdowns.
- **Rolling Report** every N incidents (default 50).
- **Special Report** when a user exceeds 5 violations (final warning is also posted in the channel + DM).

Reports are saved to a folder (e.g., `reports/`) and the path is logged.

Getting Started

Bash
```
git clone https://github.com/<your-username>/PeerSupportBot.git
cd PeerSupportBot
python -m venv .venv
source .venv/bin/activate      # Windows: .venv\Scripts\activate
pip install -r requirements.txt
cp .env.example .env           # fill tokens and thresholds
```

Configuration

Environment variables (typical):

Code snippet
```
DISCORD_BOT_TOKEN=your-token
TZ=Asia/Kolkata

# Decision thresholds
SERIOUS_STRICT=0.65
SERIOUS_MODERATE=0.45
ALWAYS_REDACT_TOX=0.80

# (Optional) model paths if running local weights
# TOXICITY_ADAPTER_PATH=./models/toxic_lora
# SARCASM_MODEL_PATH=./models/sarcasm_berttweet
```

You can tune the thresholds for your community after a short pilot.

Running the Bot

Bash
```
python -m app.bot
```

When the bot starts, it syncs slash commands and schedules the daily report.

In logs, look for lines like `[REDACT]`, `[DM]`, `[REPORT]`, `[USER-REPORT]`.

Testing the Policy (Quick Manual Tests)

Try these messages in a test channel:

Example text	Expected result
man this exam is stupid	No action (logged only)
you are stupid	Redact + DM warning
nigga / fuck you	Redact + DM (extreme override)

can we rape the guy / let’s kill him	Redact + DM (extreme override)
I want to kill myself	Crisis → Redact + DM crisis resources
After 5 violations	Final warning in DM + channel + report

Export to Sheets

Evaluation (Models & Policy)

Model-level (intrinsic):

- **Toxicity (multi-label):** AUPRC, ROC-AUC, F1@best threshold, per-label confusion matrices.
- **Sarcasm (binary):** AUPRC/ROC-AUC, F1@best threshold, confusion matrix, Brier score.

Policy-level (extrinsic):

- **Precision/Recall/F1** for Redact / Warn / None against a curated, labeled set.
- **Adversarial phrases** (rape, kill threats, severe slurs) must always be caught.
- **Benign control** (colloquial complaints, playful sarcasm) should rarely be redacted.

I keep a small ablation suite:

- With/without sarcasm relief.
- Different thresholds: `SERIOUS_STRICT`, `SERIOUS_MODERATE`, `ALWAYS_REDACT_TOX`.
- With/without extreme-word overrides.

Data & Storage

- **SQLite database** for incidents and user stats (anonymized via a salted hash of user ID).
- **Tables** include `messages`, `incidents`, `users` (violation counts), and `reports`.
- **Reports** saved as Markdown for easy sharing and auditing.

Safety & Ethics

- **Privacy**: I hash user IDs before storing (`sha256(user_id|salt)` → 16-char hash).
- **Empathy**: DMs explain why action was taken and how to get help / appeal.

- **Crisis care**: Self-harm content triggers a crisis flow with resources; the goal is support first, enforcement second.
- **Transparency**: Moderators receive reports; actions are traceable to policy rationales.

Troubleshooting

- **Bot can’t delete messages**: Ensure the bot role has `Manage Messages` and sits above the member’s role.
- **Slash commands missing**: Wait ~1–2 minutes after first run; ensure `tree.sync()` executes on `ready`.
- **Intents error**: Enable `Message Content Intent` in Developer Portal, or reduce requested intents in code.
- **DM not delivered**: Users can block DMs; a channel notice (final warnings) still posts.

Roadmap

- [ ] Location-aware crisis resources (RAG over curated hotline directory).
- [ ] `/stats` command for moderators (per-user / per-channel summaries).
- [ ] Auto-kick/ban after repeated final warnings (configurable).
- [ ] Web dashboard for reports and trend analytics.
- [ ] Wellbeing features (e.g., `/pause`, `/resources`).

License

This project is intended for educational and community wellbeing use. Please review your local policies and platform ToS before deploying at scale.