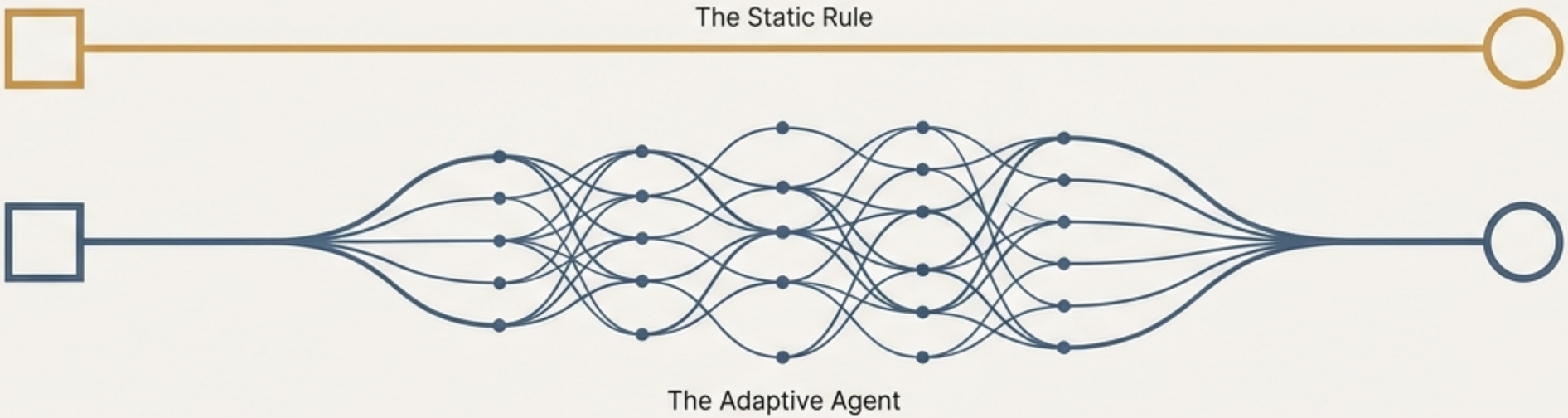
Learning to Alert: Can an AI Agent Outsmart a Simple Rule for Detecting Coordinated Harassment?

An experimental evaluation of a Reinforcement Learning policy for group-level cyberbullying detection.



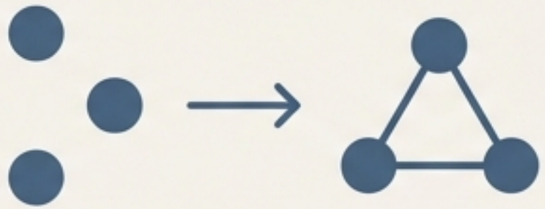
NotebookLM

The Challenge: Detecting Coordinated “Raids” at Scale

Social platforms face the problem of **group-level cyberbullying**: “**pile-ons**,” raids, or coordinated harassment where many similar, harmful messages arrive in a short time.

High Cost & Latency

Per-message inference is expensive at scale. Running powerful LLM moderation inline on high-volume streams is often unfeasible.



Lack of Group Context

Basic tools like Retrieval-Augmented Generation (RAG) are similarity-based but do not produce a *set-level* risk score that captures the critical signals of **density** and **concentration**.

Static Rules are Brittle

Simple, hand-coded rules can be effective but often lack the adaptability to respond to novel attack patterns.

Our focus: To develop a fast, discriminative monitoring loop with an intelligent policy that learns *when* to alert.

NotebookLM

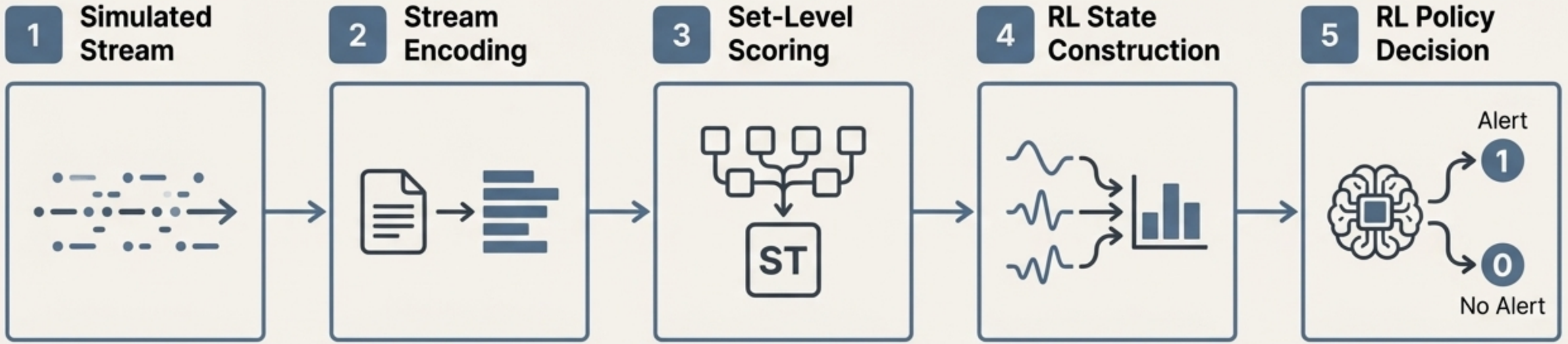
Our Hypothesis: An Adaptive Agent Can Learn a Superior Alerting Policy

> We hypothesised that a Reinforcement Learning (RL) agent, trained on signals from a stream, could learn a more effective and dynamic alerting policy than a static, hand-tuned threshold rule.

Can an agent learn to **optimally trade off the costs** of **false positives** (alerting on safe content) versus **missed toxic events** (failing to alert on harassment), outperforming a fixed baseline?

NotebookLM

The Experimental Pipeline: From Raw Messages to an Alert Decision



Simulated Stream

A live message stream is simulated using `civil_comments` data, with synthetic harassment "raids" injected.

Stream Encoding

Each message is converted into a numerical embedding vector (`all-MiniLM-L6-v2`).

Set-Level Scoring

A pretrained **Set Transformer** analyses a rolling window of messages to produce group-level risk signals.

RL State Construction

These signals are combined into a low-dimensional state vector for the agent.

RL Policy Decision

A PPO agent observes the state and makes a binary decision: **Alert (1)** or **No Alert (0)**.

NotebookLM

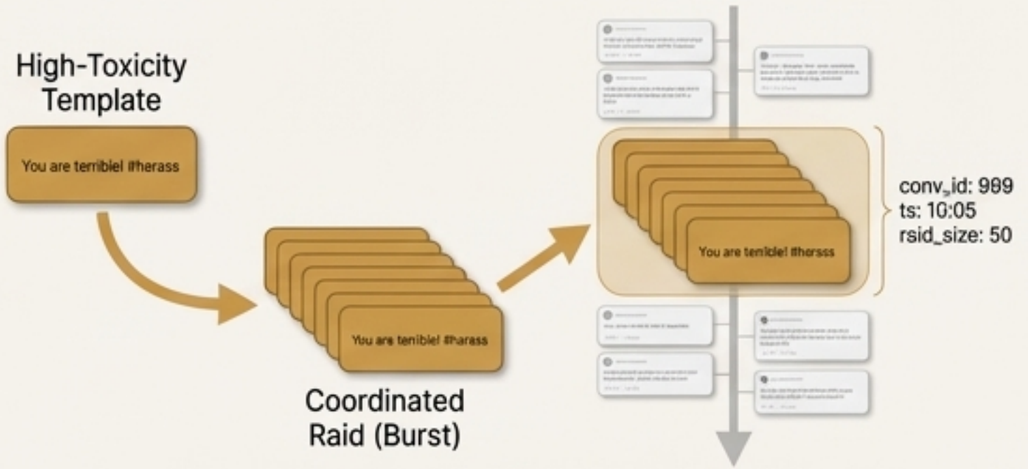
Pipeline Step 1: Simulating a Realistic Harassment Environment

Base Stream

- **Source:** Hugging Face `civil_comments` dataset.
- **Purpose:** Provides a high-volume stream of real-world text. The `toxicity` score is used as a proxy for ground truth during reward calculation.
- Synthetic `conv_id` and `ts` (timestamps) are added to emulate **concurrent** conversation threads and arrival times.

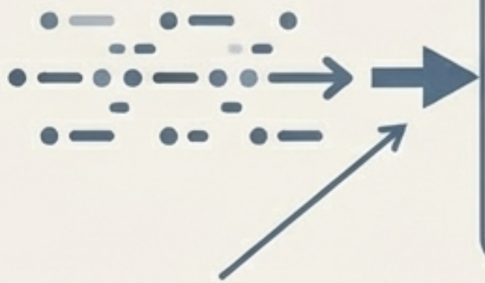
Injecting Coordinated Raids

- The `inject_raid(..)` function simulates “crowd harassment” bursts.
- **Mechanism:** A high-toxicity message is chosen as a template, and a burst of near-duplicate messages (`raid_size`) is injected over a short time window (`dt`) into a dedicated conversation (`conv_id=999`).
- **Goal:** To test if the system can detect **dense and semantically consistent** harassment bursts without relying on expensive per-message analysis.



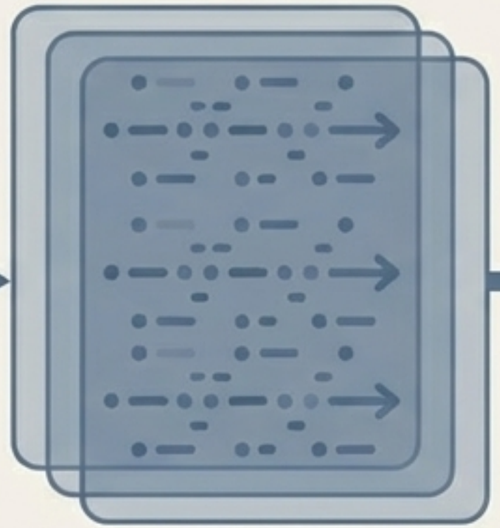
NotebookLM

Pipeline Step 2 & 3: Scoring Message Sets, Not Just Single Messages



1. **Encoding**: Each message is encoded using the `sentence-transformers/all-MiniLM-L6-v2` model.

Rolling Context Window / Set



2. Set-Level Scoring: A pretrained, dual-head Set Transformer operates on a rolling buffer of these message embeddings. This model, a key artifact from a previous project, is designed to analyse an entire set of messages at once.

Set Transformer

`mean_hat`

An estimate of the set's average toxicity or intensity.

`conc_hat`

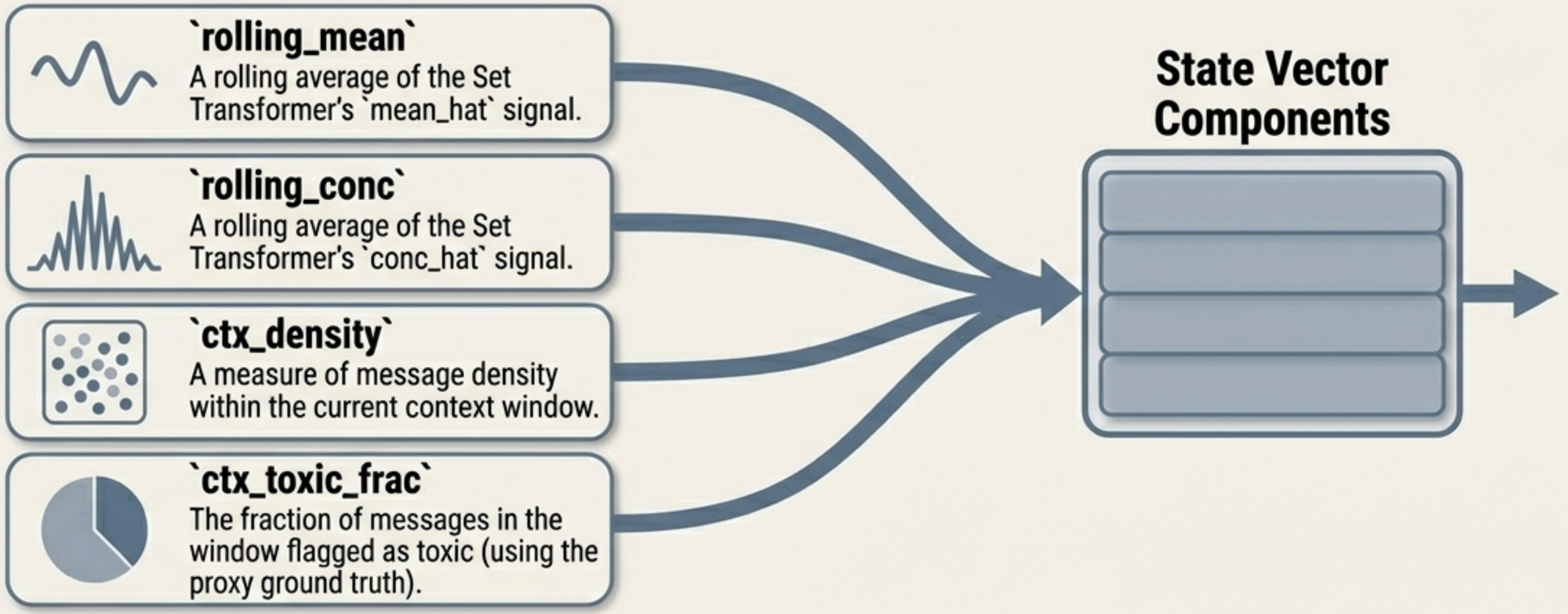
An estimate of the set's concentration or 'burstiness'.

3. **Key Outputs**: The Set Transformer produces two critical, set-level signals.

NotebookLM

Pipeline Step 4: Constructing the Agent's View of the World

Before the agent can act, the signals from the monitoring loop are compiled into a concise, 4-dimensional state vector (the 'observation').



Purpose: This vector provides the agent with a snapshot of the current group-level threat landscape at each time step.

NotebookLM

The Agent's Task: Learning the Optimal Alerting Strategy

The Environment

A custom `Gymnasium` environment (`AlertEnv`) defines the rules of the task.

- Action Space: A binary choice at each step:
  - `0`: Do not alert.
  - `1`: Issue an alert.
- Episode: A single episode consists of a full pass over the entire simulated message stream dataframe (`out_df`), with a mean length of ~5,040 steps.

The Reward Signal

The agent learns based on a simple reward function that uses the message's `toxicity` score as a proxy for ground truth.

Agent Action	Message Condition (Proxy)	Reward	Consequence
Alert (1)	`toxicity > 0.7`	+1.0	Correctly identified a threat.
Alert (1)	`toxicity <= 0.7`	-0.5	False positive.
No Alert (0)	`toxicity > 0.7`	-0.2	Missed a threat.
No Alert (0)	`toxicity <= 0.7`	0.0	Correctly ignored safe content.

NotebookLM

The Training Run: Clear Evidence of Learning

Algorithm

The agent was trained using the **Proximal Policy Optimisation (PPO)** algorithm from Stable-Baselines3.

- **Policy**: `MlpPolicy`
- **Total Timesteps**: 10,000

Interpretation

The agent learned to make better decisions according to the rules it was given.

Learning Progress

The agent's performance, measured by its average reward per episode, steadily improved during training. This demonstrates that the PPO algorithm was successfully optimising its policy to maximise the defined reward signal.

PPO Iteration	Total Timesteps	Mean Episode Length	Mean Episode Reward
20	5,120	~5040	-216
40	10,240	~5040	**-118**

NotebookLM

The **Final Evaluation:** **The Adaptive Agent** vs. The Static Rule

The PPO Policy (The Adaptive Agent)

How it works: Uses the trained PPO model to make a `predict` call on the 4D state vector at each step.

Logic: A complex, non-linear function learned from thousands of simulated interactions.

```
action = model.predict(obs, deterministic=True)
```

The Naive Baseline (The Static Rule)

How it works: A simple, hand-tuned rule based on the core set-level signals.

Logic: `alert IF rolling_mean > 0.6 OR rolling_conc > 0.5`

Represents a standard, straightforward heuristic one might implement without machine learning.

NotebookLM

The Result: A Surprising Tie

Both the trained PPO policy and the naive baseline were evaluated on the same stream of data, using the exact same reward function to calculate a final score.

Policy	Total Reward
Naive Threshold	**-19.8**
PPO (RL) Policy	**-19.8**

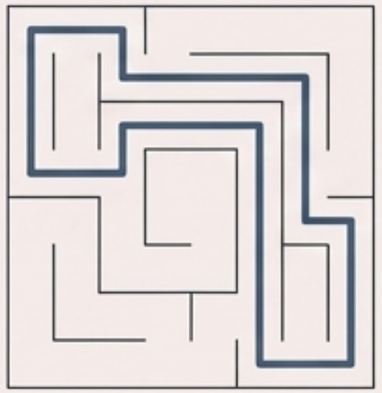
In this experiment, the trained PPO policy **did not outperform** the naive threshold baseline. Both strategies achieved the identical final score.

NotebookLM

Why a Tie is the Most Important Finding

The Key Insight

This result is not a failure of Reinforcement Learning. It's a powerful demonstration of the specific conditions under which a complex learning agent may struggle to outperform a simpler heuristic.



What We Learned

- For a **single, deterministic data stream** where the agent sees the **same sequence** in every episode...
- ...and with a **simple, per-step proxy reward signal**...
- ...the PPO agent can learn to perfectly optimise for that specific sequence and reward, yet the resulting policy may not be more robust or effective than a well-tuned rule on the final aggregate score.

The agent learned to solve the puzzle perfectly, but the puzzle itself was too simple to allow for a more general, superior strategy to emerge.

NotebookLM

Building a Better Experiment: Conditions for Future Success

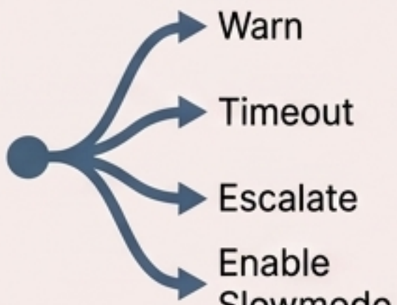
The results point directly to the improvements needed for an RL agent to beat a strong baseline in this domain:

1. Stochastic Environments

Introduce randomness through multiple, randomized episodes. Vary raid locations, sizes, and timings to force the agent to learn a more generalised policy.

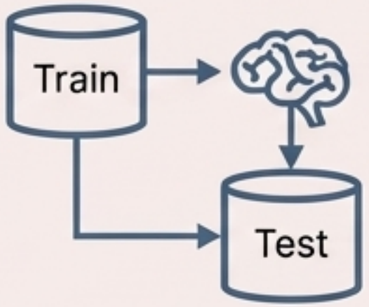
2. Richer Reward Shaping

Move beyond per-step rewards. Introduce sequence-level costs for factors like 'alert fatigue' (penalising too many alerts in a short time) to better reflect real-world operational goals.



3. Richer Action Space

Expand the agent's choices beyond a binary alert. Actions like `warn`, `timeout`, `escalate`, or `enable_slowmode` would allow for more nuanced, long-term conversation management.



4. Explicit Evaluation Splits

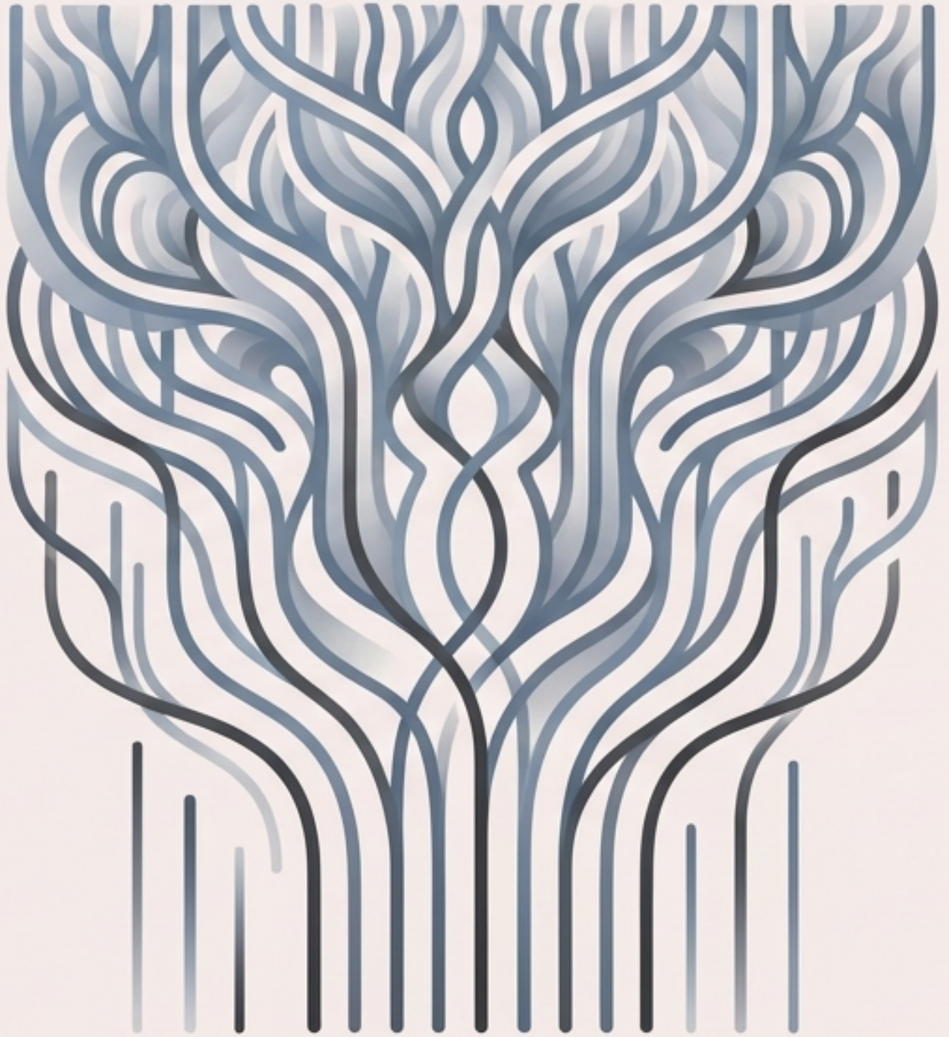
Use separate, held-out datasets for final evaluation to ensure the agent's performance is not just an artefact of overfitting to the training stream.

NotebookLM

The Path Forward: Next-Generation Moderation Systems

This experiment provides a strong foundation for more sophisticated group-level moderation. Future work will focus on:

- **Per-Thread Aggregation**: Building context sets that are aware of individual conversation threads, rather than mixing all messages into a single buffer.
- **Conversation-Level RL**: Optimising for the long-term health of a conversation, not just immediate message-level threats.
- **Time-Window Modelling**: Incorporating explicit temporal features (e.g., message rates over 30s, 5m, 1h) for true density estimation.
- **Target Classification**: Moving beyond toxicity to identify *who* is being targeted and the specific *type* and *severity* of the harm.



NotebookLM