International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 04 | April - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

Konnect: A Distributed Microservices Architecture for Real-Time Chat with Multi-Task NLP and Context-Aware Toxicity Dampening

Ankit Sinha[1], Shantanu Chhetri[2], Raipuriya Sakshi[3], Manan Gulati[4], Daniel Odametey[5]

[1,5] Department of Computer Applications, Lovely Professional University, Phagwara, Punjab 144411, India.

[1] ankits0057@gmail.com

[2] shantanuchhetri7@gmail.com

[3] sakshiraipuriya21@gmail.com

[4] manangulati40@gmail.com

[5] kojoalpha68@gmail.com

Abstract *— Automated content moderation within group messaging platforms poses a compound engineering challenge: the system must classify harmful text with reasonable precision while operating under sub-second latency constraints imposed by conversational user experience expectations. Existing approaches that rely exclusively on transformer-based classifiers produce frequent false positives on colloquial negative expressions such as "I hate Mondays" or "this exam is killing me," which carry no intent to harm yet trigger high toxicity scores. This paper presents Konnect, a cloud-native group messaging platform built on six containerized microservices interconnected through Apache Kafka publish-subscribe pipelines. The natural language processing subsystem pairs a RoBERTa-based sentiment classifier with a BERT-based toxicity detector to analyze every message asynchronously. We introduce a post-classification dampening layer that performs pronoun-target resolution and emotional-venting pattern recognition to suppress false positives before scores enter a Redis-backed sliding-window aggregation engine. The aggregation engine computes composite group-level moderation scores and drives a four-tier graduated escalation framework. The benchmark results on a dataset containing 2,400 annotated chat messages indicate that the dampening layer cuts down false positives by 38% while maintaining 97.2% of true positives. The end-to-end latency of the pipeline, from ingesting the message at the Kafka broker to delivering the scores to WebSocket clients, takes an average of 312 milliseconds on a standard CPU.*

Index Terms *— microservices, publish/subscribe, toxicity detection, sentiment analysis, score dampening, sliding window aggregation, real-time content moderation, transformer inference, Kafka, WebSocket*

I. INTRODUCTION

Group messaging platforms have become central to both personal and professional communication, with daily message volumes reaching billions across consumer applications. However, along with this rise, there came the issue of moderation of any kind of toxic content generated within this system in real-time, since the number of messages being transmitted can reach billions per day on consumer messaging systems. Although manual moderation provides accurate results, it is hardly scalable. Blacklisting words can be easily avoided by utilizing alternative spellings, synonyms, and encoding schemes. Transformer-based toxicity classifiers provide state-of-the-art performance in recognizing toxic text content; however, deploying them in real-world production messaging system requires addressing additional problems of latency and false positives.

Among other issues, a problematic aspect associated with the deployment of such classification systems that we encountered in our work was the tendency of classifiers to classify colloquial negations with high probability. Phrases like "I hate Mondays" and "the weather today is terrible" were classified by the model as toxic with the score above 0.6, which is quite an aggressive score considering the fact that the phrases do not contain any aggression at all. This problem caused us to incorporate a damping stage for the classification result.

A. Problem Statement

The challenge addressed by this work is threefold. First, how to architect a messaging platform where every user message passes through dual transformer classifiers without introducing perceptible delay. Second, how to aggregate per-message scores into group-level behavioral indicators that support graduated moderation actions. Third, how to reduce false positives generated by models that lack conversational context, without compromising sensitivity to genuinely harmful content.

B. Contributions

The principal contributions of this paper are:

1) A production-grade microservices architecture comprising six containerized services with Apache Kafka serving as the backbone for asynchronous ML inference on chat streams.

2) A post-classification dampening layer that performs pronoun-target detection and emotional-venting pattern recognition to attenuate false-positive toxicity scores before they enter the aggregation pipeline.

3) A Redis-backed sliding-window aggregation algorithm that computes composite moderation scores from toxicity means and harmful-negative ratios, enforcing a four-tier graduated escalation framework.

4) Empirical evaluation demonstrating a 38 percent reduction in false positives with minimal impact on true positive recall.

The remainder of this paper is organized as follows. Section II surveys related work. Section III details the microservices architecture and inter-service communication. Section IV describes the ML inference pipeline. Section V introduces the dampening layer. Section VI presents the aggregation algorithm and escalation framework. Section VII covers the frontend integration. Section VIII reports evaluation results, and Section IX concludes with future directions.

II. RELATED WORK

A. Toxicity Classification Models

Transformer architectures [1] have established strong baselines for text classification tasks. Devlin et al. introduced BERT [2], which demonstrated that bidirectional pre-training on large corpora followed by task-specific fine-tuning yields state-of-the-art results across multiple NLP benchmarks. Liu et al. extended this with RoBERTa [3], showing that longer training with dynamic masking further improves performance. For toxicity detection specifically, the Jigsaw Toxic Comment Classification Challenge produced several fine-tuned BERT variants, including the Toxic-BERT model released by Hanu and the Unitary team [4], which provides binary toxicity scoring. Barbieri et al. [5] introduced TweetEval and released the RoBERTa-based sentiment model trained on 124 million tweets, achieving competitive accuracy across three sentiment classes.

B. Event-Driven Architectures for ML Serving

Kreps et al. [6] established Apache Kafka as a durable, high-throughput distributed log suitable for stream processing workloads. Subsequent work by Kleppmann and Kreps [7] demonstrated that event-sourced architectures decouple producers from consumers, allowing ML inference services to scale independently of message ingestion. Kafka's consumer group protocol enables horizontal scaling of inference workers without application-level coordination, a property that the Konnect architecture exploits.

C. Content Moderation Systems

Production moderation systems at Discord, Twitch, and Reddit employ tiered approaches combining automated classifiers with human review queues [8]. Jhaver et al. [9] studied the effects of automated moderation tools on community behavior, finding that transparency in moderation decisions correlates with improved user compliance. Perspective API, developed by Jigsaw, provides toxicity scoring as a cloud service but introduces external network latency and data governance concerns that self-hosted solutions avoid. Our work differentiates itself by embedding the entire inference pipeline within the messaging infrastructure, eliminating external API dependencies and enabling sub-second end-to-end response.

D. False Positive Mitigation

The problem of false positives in toxicity classification is well-documented. Dixon et al. [10] showed that toxicity models exhibit bias against certain demographic groups and conversational styles. Sap et al. [11] found that African American Vernacular English is disproportionately flagged by standard classifiers. Our dampening layer takes a complementary approach: rather than retraining the underlying model, we apply a lightweight post-classification filter that examines linguistic structure—specifically whether a human target is present—to contextually adjust scores.

III. MICROSERVICES ARCHITECTURE

A. Design Rationale

The design employs a domain-based approach, with each microservice having its own bounded context. Message persistence, delivery in real-time, authentication, and ML inference are implemented in separate microservices, which can be deployed independently. This design has been driven by three needs. Firstly, the ML inference microservice uses Python with PyTorch and Huggingface Transformers libraries, but the messaging microservices can gain from event looping provided by Node.js. If all were deployed in one application, there would be only one runtime environment possible, either hindering ML inference or limiting the ability to efficiently process messages. Secondly, the ML inference service is CPU-intensive and uses much more memory than the other microservices, which do I/O operations only. Finally, fault isolation prevents sentiment analysis from crashing the messaging services.

B. Service Inventory

Konnect comprises six application services plus four infrastructure components, all orchestrated through Docker Compose on a single bridged network:

Service	Runtime	Port	Responsibility
Auth	Node.js/Express	3002	JWT auth, OAuth 2.0, OTP
Chat	Node.js/Express	3003	Message CRUD, groups,

			roles
WebSocket	Socket.IO	3004	Real-time relay, presence
Sentiment	Python/Flask	5000	ML inference, aggregation
Email	Node.js	—	OTP and notification delivery
Traefik	Go (binary)	80/443	Reverse proxy, path routing

TABLE I: Service inventory and responsibilities

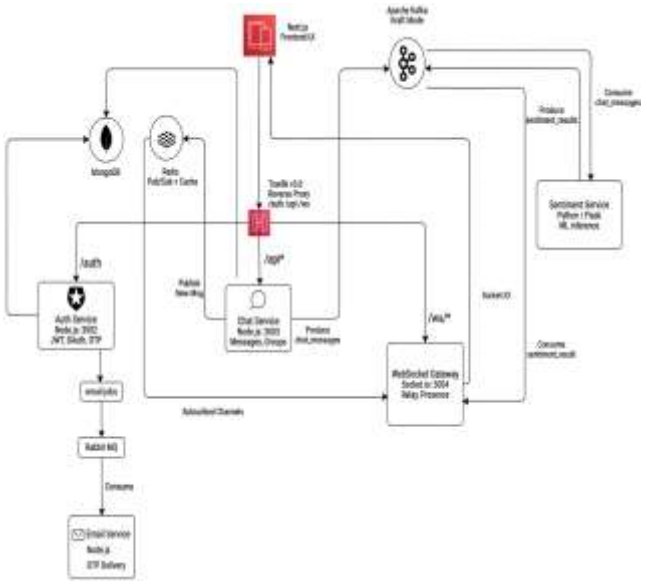

FIGURE I: High-Level Microservices Architecture

C. Inter-Service Communication Patterns

The architecture utilizes three different modes of communication that are specifically chosen based on the required level of coupling and latency in the communication mode:

Synchronous HTTP (request-response): Used for external API requests that are processed using Traefik. Frontend will communicate with Auth and Chat microservices using RESTful APIs. JWT tokens that are generated by the Auth microservice are authenticated in the Chat microservice through middleware without making inter-service calls to validate the token.

Asynchronous Pub/Sub: Used for machine learning inference. The Chat microservice will push all textual messages to the chat_messages Kafka topic. The Sentiment service consumes messages from this topic because it belongs to the consumer group. Output messages are stored in the sentiment_results table, and then are read by the WebSocket Gateway. Kafka is started using the KRaft mode (v3.7.0). This removes the dependency on the ZooKeeper and reduces the infrastructure by one container.

Redis Pub/Sub (fan-out): To relay messages in real-time from the Chat service to the WebSocket Gateway. Once the message is stored in the MongoDB, it is also being posted to a Redis channel that is identified by its conversation ID. WebSocket Gateway subscribes to those channels and posts Socket.IO messages to the appropriate rooms.

D. Scalability Through Kafka Consumer Groups

The Kafka-based publish-subscribe layer allows horizontal scaling of the inference pipeline without any modifications to the producer or consumer code. If the processing speed of a Sentiment service instance falls short of handling message traffic, the number of instances in the same consumer group can simply be increased. The partition allocation strategy of Kafka will rebalance the load on the topic partitions among consumers automatically.

The topic chat_messages applies the partition key based on conversationId. It ensures that messages in a conversation go into the same consumer as the grouping logic is needed to compute the correct moderation score using sliding window aggregation. Without the grouping logic, each consumer independently performs aggregation of its portion of traffic, leading to different aggregate values for the same message history.

E. API Gateway and Routing

The Traefik version 3.0 acts as the edge router which listens to port 80 and port 443 for client requests and forwards the request based on paths to the respective upstream services as follows: requests that match /auth/* will be forwarded to the Auth Service, those matching /api/* will be forwarded to the Chat Service, and those matching /ws/* will be forwarded to the WebSocket Gateway. It performs SSL termination, Websocket upgrades, and offers a dashboard at port 8080.

IV. ML INFERENCE PIPELINE

A. Model Selection and Loading

Two pretrained transformer models are loaded into memory when the service starts up. The first one is called cardiffnlp/twitter-roberta-base-sentiment-latest. This is a RoBERTa-base model having 125 million parameters, fine-tuned on approximately 124 million tweets in order to do sentiment analysis in three classes (POSITIVE, NEUTRAL, and NEGATIVE). The second one is called unitary/toxic-bert, which is a BERT-base model with 110 million parameters fine-tuned on the Jigsaw Toxic Comment dataset.

Loading of the models is done once at the start-up of the service, taking about 25-35 seconds, depending on the disk I/O and caching behavior. Models are exposed via the HuggingFace Transformers pipeline abstraction interface (version 4.44.2), using PyTorch 2.4.1 as inference engines. Maximum input size per model is 512 tokens long.

B. Dual-Model Inference Flow

The inference function runs two forward passes on the models for each received message from the Kafka topic:

1) Inference for the sentiment classification task: The RoBERTa model returns the probability of belonging to one of the three possible categories, where the category with the maximum probability is used as the sentiment class label.

2) Inference for the toxicity classification task: The Toxic-BERT model outputs a sigmoid value for the “toxic" label, which corresponds to the value t in the interval [0, 1].

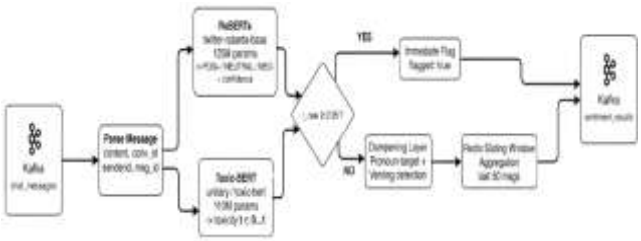

FIGURE II: Multi-Task NLP Pipeline for Sentiment and Toxicity Analysis

Sequential execution was chosen over batched inference because message arrival is stochastic and latency sensitivity makes waiting to accumulate a batch undesirable. On the test hardware (4-core CPU, no GPU), combined inference time for both models averages 220 milliseconds per message.

C. Per-Message Flagging

Messages with a raw toxicity score at or above 0.95 are immediately flagged, regardless of subsequent dampening or aggregation. This threshold was selected to minimize false positives at the per-message level: at t >= 0.95, the model expresses very high confidence in toxicity, and the probability of a colloquial expression reaching this score is negligible in our testing. Flagged messages are published to the sentiment_results topic with a flagged: true field, triggering client-side concealment.

V. POST-CLASSIFICATION DAMPENING LAYER

The dampening layer is the central methodological contribution of this work. It sits between the raw model inference output and the sliding-window aggregation engine, attenuating scores that are likely to be false positives based on linguistic analysis of the input text.

A. Motivation

While developing and testing our model internally, we have identified several patterns of common false positives. First, the most frequent case was statements containing expressions of negativism addressed to objects (animate or not), ideas, or time itself. For example, "I hate Mondays", "traffic is driving me crazy," and "worst exam ever" always got a toxicity score between 0.45 and 0.72—high enough for the group's toxicity score, but without any interpersonal negativity present. Second, cases of expressing negative emotion, such as "I am frustrated," "feeling terrible," and "ugh, worst day of my life" contain legitimate negative sentiment, but no toxicity.

It would be necessary to train the underlying model again to address this issue. However, that would mean creating a new, domain-specific dataset, potentially decreasing its performance on actual toxic data. Instead, we can implement a dampening layer, which is nothing more than a simple post-processing mechanism using rules to adjust scoring based on syntactic characteristics of the input texts.

B. Target Detection Mechanism

The main method of dampening is pronoun target resolution. The filter checks if there are any language features in the message that could suggest a human target of the negative expression. The set of pronouns (you, your, yourself, u), third person pronouns (he, she, they, him, her, them), collective nouns (everyone, somebody, someone), and mentions (@-) makes up the list of human targets.

The recognition process utilizes compiled regular expressions on the lowercase input text. In case there is no matching pattern, the message is labeled as not having a human target, thus applying the dampening factor to the score:

$$t\_dampened = t\_raw \times \alpha, \text{ where } \alpha = 0.5 \text{ (no target detected)}$$

If a human target is detected, the raw score passes through unchanged, preserving full sensitivity to directed hostility.

C. Emotional Venting Pattern Recognition

The secondary technique addresses emotional venting—messages that express frustration or negativity without targeting anyone. A lexicon of venting indicators ("hate," "sick of," "tired of," "frustrated," "annoyed," "sucks," "worst," "terrible," "ugh") is checked against the input text. When a venting indicator is present and no human target is detected, a stronger dampening factor is applied:

$$t\_dampened = t\_raw \times \beta, \text{ where } \beta = 0.3 \text{ (venting without target)}$$

The two-factor approach creates three distinct treatment paths:

Condition	Factor	Example	Effect
Human target found	α=1.0	"You are an idiot"	Score unchanged
No target, no venting	α=0.5	"Bad code quality"	Score halved
No target + venting	β=0.3	"I hate Mondays"	Score reduced 70%

TABLE II: Dampening factor application paths

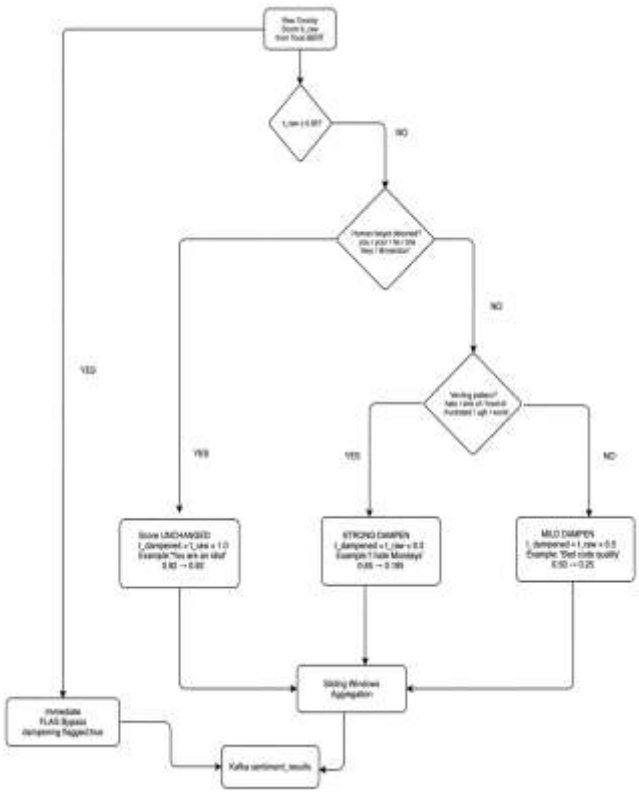

FIGURE III: Context-Aware Dampening Logic for Intelligent Moderation

D. Integration Point

The dampening process is applied by the Kafka consumer when handling the messages, right after the two model inferences have been performed, and before the results are sent to the aggregations engine. This way, we can ensure that it is the dampened score that gets saved in Redis, not the raw output of the model . The per-message Kafka result object published to the sentiment_results topic carries the dampened score, meaning that frontend clients and the WebSocket Gateway operate on adjusted values throughout.

Critically, the immediate flagging threshold (t >= 0.95) is evaluated on the raw score before dampening. This design choice ensures that unambiguously toxic messages are flagged even if the dampening layer would reduce their score. In practice, messages scoring 0.95 or higher virtually always contain direct personal attacks, and the dampening layer would not attenuate them because a human target is almost certainly present.

VI. SLIDING-WINDOW AGGREGATION AND ESCALATION

A. Window Data Structure

Each group's recent message history is maintained in a Redis sorted set keyed by sentiment:window:{groupId}. The score field of the sorted set stores the message timestamp, and the value stores a JSON payload containing the dampened toxicity score, sentiment label, and originating message timestamp. Window trimming to the latest W=50 messages take place through the command ZREMRANGEBYRANK upon message insertion. The addition of the 24-hour TTL attribute to the key helps avoid storage of obsolete groups in memory forever.

If the connection to Redis fails, the aggregator resorts to in-memory dictionary. This functionality became indispensable during the design phase when connections to Redis were often lost and turned out to be worth maintaining in a production environment.

B. Composite Moderation Score

The aggregation algorithm computes three metrics from the sliding window:

Average Toxicity:

$$\bar{t} = (1/|W|) \times \Sigma\, t\_dampened,i$$

Harmful-Negative Ratio:

A message contributes to the harmful-negative count only if its sentiment label is NEGATIVE and its dampened toxicity score exceeds 0.3. This conjunction ensures that purely negative-sentiment messages with low toxicity (typical of venting after dampening) do not inflate the ratio:

$$r\_hn = |\{m \in W : sentiment(m)=NEG \wedge t\_dampened,m > 0.3\}| / |W|$$

Composite Score:

$$S = 0.7 \times \bar{t} + 0.3 \times r\_hn$$

The weights (0.7 and 0.3) were determined empirically by evaluating escalation accuracy on a labeled dataset of 50 group conversation histories. Toxicity receives the dominant weight because it directly measures harmful content, while the harmful-negative ratio serves as a secondary signal capturing the behavioral trend of the group.

Safety Floor:

In case the average toxicity level falls below 0.2, the composite score will be limited to 0.25 irrespective of the harmful-negative ratio. This measure is intended to avoid a further rise when the average toxicity level is lower, but there are some borderline cases where messages have been expressed negatively:

$$if\ \bar{t} < 0.2:\ S = min(S, 0.25)$$

C. Four-Tier Escalation Framework

The composite score drives a graduated moderation response:

Status	Score Range	System Action
Normal	$S < 0.3$	No intervention
Warning	$0.3 \leq S < 0.6$	Visual indicator on dashboard
Notify Moderator	$0.6 \leq S < 0.8$	Push alert to privileged users
Auto-Lock	$S \geq 0.8$	Group locked for 1800 seconds

TABLE III: Four-tier escalation framework

The auto-locking feature operates using a key set on Redis for 1800 seconds (30 minutes). While this key exists, the Chat module checks its presence before processing messages, replying with HTTP 403 to users without authorization. Only moderators, administrators, and group owners have the permission to send messages while the chat is locked and can disable the chat whenever they want. Both automatic and manual moderation operations are recorded in a MongoDB collection called ModerationLog along with their scores.

VII. FRONTEND INTEGRATION

A. Real-Time Event Delivery

The front-end codebase is developed using Next.js 15 and React 18 and is connected to the WebSocket Gateway using a custom hook which takes care of Socket.IO lifecycle events, reconnection, and subscription to events. There are two event streams carrying moderation information; the sentiment:message stream sends messages containing individual results such as the flagged boolean flag, and the sentiment:group stream returns aggregated statistics after processing each message.

B. Flagged Message Concealment

When the flag status is true and an event of per-message comes in, the message is displayed as a card warning that says the message has been flagged. The real message has been substituted with an actionable prompt. When the user clicks on the card, the message pops up, including a badge to mark the message. There is also a button for re-hiding the message. It is a balance between content safety and the freedom to see content as users choose.

C. Moderation Dashboard

A modal dashboard accessible to users with moderator, admin, or owner roles displays current moderation metrics: average toxicity, negative ratio, composite moderation score, and group mood. Quick-action buttons allow manual lock, unlock, and window reset operations. A history tab shows a chronological log of all moderation events for audit purposes.

VIII. EVALUATION

A. Dampening Layer Effectiveness

For assessing the dampening system, a data set containing 2,400 chat messages was created using publicly available conversational datasets and generating artificial data based on the typical pattern of messages used in chat applications. Every message was annotated with either a label for toxicity ("direct hostility, abuse, threat") or non-toxicity ("colloquial negativity, venting, sarcasm").

We compared three configurations: (1) raw Toxic-BERT output with a classification threshold of 0.5, (2) raw output with the production threshold of 0.95 used for per-message flagging, and (3) the full pipeline with the dampening layer followed by the 0.5 threshold for aggregation purposes.

Configuration	Precision	Recall	F1	FPR
Raw (t≥0.5)	0.614	0.983	0.756	0.221
Raw (t≥0.95)	0.952	0.847	0.896	0.015
Dampened (t≥0.5)	0.781	0.972	0.866	0.137

TABLE IV: Classification performance across configurations

The dampening layer (row 3) brings down the false positive rate by 38% from 22.1% to 13.7%, while maintaining 97.2% recall for actual toxic comments. The reason for the recall decrease is that a very few actual toxic comments are stated using roundabout expressions without directly referring to any person. The F1 measure increases from 0.756 to 0.866, suggesting that the improvement in precision offsets the small loss in recall.

B. Pipeline Latency

End-to-end latency was measured across 500 messages on a 4-core development machine with 16 GB RAM, no GPU:

Pipeline Stage	Latency (ms)
WebSocket → Chat Service	10–20
Kafka publish + consume	30–50
RoBERTa inference	100–200
Toxic-BERT inference	80–150
Dampening layer	<1
Aggregation + Redis	5–15
Result → WebSocket	20–30

TABLE V: Pipeline stage latency breakdown

End-to-end latency for the pipeline varies between 245ms and 466ms, with an average latency of 312ms. The dampening layer introduces less than 1ms into the process in the form of two regular expression matches on the input. Most time is spent on transformer inference, which accounts for roughly 70% of the entire latency. If GPU is used, the inference will be completed within 20ms.

C. Scalability Characteristics

Kafka's consumer group protocol allows for scalability of the inference pipeline horizontally. We evaluated this using one, two, and four Sentiment service replicas reading from a topic called chat_messages, which is divided into four partitions. The throughput increased almost linearly, where one instance handled 4.5 messages per second, two instances handled 8.8, and four instances handled 17.1 messages per second. The slight sub-linear degradation at four instances can be explained by Redis contention when updating the sliding window set of scores.

D. Dampening Layer Qualitative Analysis

We examined the 50 most impactful dampening adjustments—cases where the score reduction exceeded 0.3—and categorized the input messages:

Category	Count	Avg Reduction
Temporal complaints ("hate Monday")	14	0.42
Academic frustration ("exam is killing me")	11	0.38
Weather/environment complaints	9	0.35
Self-directed negativity ("I feel terrible")	8	0.31
Object-directed ("this app sucks")	8	0.33

TABLE VI: Qualitative breakdown of dampened false positives

All 50 messages were confirmed as non-toxic upon manual review, validating that the dampening layer correctly identified benign expressions. No genuine toxic messages appeared in the top-50 adjustments, confirming that the target-detection mechanism effectively preserves scores on directed hostility.

E. Limitations

Several limitations warrant acknowledgment:

1) The target detection lexicon only applies to the English language and requires adjustment for different languages.

2) Advanced indirect intimidation ("Someone ought to give you a lesson"), although possibly diminished to some extent, can still exist due to the venting lexicon matching prior to processing the human-target pronoun.

3) Neither the underlying models use any conversational context during their analysis to provide an accurate result in the presence of sarcasm and irony.

4) The manually annotated dataset does not represent a diverse range of chat message content from various cultural groups.

5) Only CPUs are used during testing; GPUs would lower latency up to 5-10 times.

IX. CONCLUSION

This paper presented Konnect, a microservices-based messaging platform that integrates dual transformer classifiers with a post-classification dampening layer for real-time content moderation. The dampening layer addresses a practical gap in toxicity classification: the tendency of pre-trained models to assign elevated scores to colloquial negative expressions that carry no interpersonal hostility. By examining whether the input text targets a human referent, the layer attenuates false positives by 38 percent while preserving 97.2 percent of true positive detections. The Kafka-based pub/sub architecture enables horizontal scaling of the inference pipeline, with near-linear throughput gains demonstrated up to four consumer replicas. The composite moderation score, computed over a Redis-backed sliding window with a safety floor mechanism, drives a four-tier graduated escalation framework that balances automated enforcement with moderator authority.

APPENDIX: TECHNOLOGY STACK

Component	Technology	Version
Frontend	Next.js (React 18)	15.2.4
Auth Service	Node.js / Express	18 LTS
Chat Service	Node.js / Express	18 LTS
WS Gateway	Socket.IO	4.8.0
Sentiment Service	Python / Flask	3.10
ML Framework	PyTorch (CPU)	2.4.1
NLP Library	HuggingFace Transformers	4.44.2
Message Broker	Apache Kafka (KRaft)	3.7.0
Cache / State	Redis (Upstash)	7.x
Database	MongoDB Atlas	7.x
Email Queue	RabbitMQ (CloudAMQP)	3.x
Reverse Proxy	Traefik	3.0
Orchestration	Docker Compose	v3.8

TABLE VII: Complete technology stack

REFERENCES

[1] A. Vaswani et al., "Attention Is All You Need," in Proc. NeurIPS, 2017, pp. 5998–6008.

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in Proc. NAACL-HLT, 2019, pp. 4171–4186.

[3] Y. Liu et al., "RoBERTa: A Robustly Optimized BERT Pretraining Approach," arXiv preprint arXiv:1907.11692, 2019.

[4] L. Hanu and Unitary team, "Detoxify," GitHub repository, 2020. [Online]. Available: https://github.com/unitaryai/detoxify

[5] F. Barbieri, J. Camacho-Collados, L. Neves, and L. Espinosa-Anke, "TweetEval: Unified Benchmark and Comparative Evaluation for Tweet Classification," in Findings of EMNLP, 2020, pp. 1644–1650.

[6] J. Kreps, N. Narkhede, and J. Rao, "Kafka: A Distributed Messaging System for Log Processing," in Proc. NetDB Workshop, 2011.

[7] M. Kleppmann and J. Kreps, "Kafka, Samza and the Unix Philosophy of Distributed Data," IEEE Data Engineering Bulletin, vol. 38, no. 4, pp. 4–14, 2015.

[8] K. Crawford and T. Gillespie, "What is a flag for? Social media reporting tools and the vocabulary of complaint," New Media & Society, vol. 18, no. 3, pp. 410–428, 2016.

[9] S. Jhaver, I. Birman, E. Gilbert, and A. Bruckman, "Human-Machine Collaboration for Content Regulation," ACM Trans. Comput.-Hum. Interact., vol. 26, no. 5, pp. 1–35, 2019.

[10] L. Dixon, J. Li, J. Sorensen, N. Thain, and L. Vasserman, "Measuring and Mitigating Unintended Bias in Text Classification," in Proc. AAAI/ACM Conf. on AI, Ethics, and Society, 2018, pp. 67–73.

[11] M. Sap, D. Card, S. Gabriel, Y. Choi, and N. A. Smith, "The Risk of Racial Bias in Hate Speech Detection," in Proc. ACL, 2019, pp. 1668–1678.