НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
июль–август 2026 Том 26 № 4 http://ntv.ifmo.ru/
SCIENTIFIC AND TECHNICAL JOURNAL OF INFORMATION TECHNOLOGIES, MECHANICS AND OPTICS
July–August 2026 Vol. 26 No 4 http://ntv.ifmo.ru/en/
ISSN 2226-1494 (print) ISSN 2500-0373 (online)

НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

doi: 10.17586/2226-1494-2026-26-4-851-859

Real-time NLP moderation on XMPP with GPU-accelerated inference and adaptive batching

Alaa Aldarf[1]✉, Alaa Shaker[2], Igor A. Bessmertny[3]

[1,3] ITMO University, Saint Petersburg, 197101, Russian Federation
[2] Saint Petersburg, Russian Federation

[1] aaldarf@itmo.ru✉, https://orcid.org/0000-0001-7249-5071
[2] alaashaker11071991@gmail.com, https://orcid.org/0000-0003-2709-0766
[3] bessmertny@itmo.ru, https://orcid.org/0000-0001-6711-6399

Abstract

Real-time communication platforms generate continuous streams of short, latency-sensitive messages, creating a demanding environment for automated toxicity detection. Transformer-based language models offer strong contextual accuracy, but their inference cost makes them difficult to deploy efficiently in decentralized messaging systems such as the Extensible Messaging and Presence Protocol, where requests arrive asynchronously from multiple servers. Traditional batching strategies struggle in this setting because traffic patterns fluctuate, message lengths vary, and strict latency budgets prevent the accumulation of large batches. This work introduces a full-stack moderation system that combines an OpenFire server plugin, a Graphics Processing Unit (GPU) accelerated microservice for toxicity classification, and an adaptive batch processing method suitable for use in multi-server systems. The plugin intercepts messages at the packet-processing layer and forwards them to a lightweight external inference service that can run on either a Central Processing Unit or a GPU. We introduce Adaptive Cross-Domain Batching (ACDB) as a method to dynamically adjust batch sizes based on the state of the queues, characteristics of the messages being processed, and real-time feedback from GPU usage. Experiments demonstrate that GPU inference provides significant improvements in both throughput and latency, with compute accelerations of up to 28× and total end-to-end accelerations of up to 9×. Comparative evaluation against static batching baselines across multiple traffic scenarios shows that ACDB dynamically adapts batch sizes from 3 to 25 depending on load conditions, reducing latency by 52–59 % compared to large-batch static configurations while maintaining 85–94 % of their throughput. Overall, this system enables scalable, real-time toxic content moderation capabilities for federated chat systems using adaptive batching for Transformer-based inference on dynamic workloads, while preserving strict message delivery constraints under realistic operating conditions.

Keywords

toxicity detection, XMPP, OpenFire, GPU acceleration, adaptive batching, NLP moderation, distributed systems, transformer inference

For citation: Aldarf A., Shaker A., Bessmertny I.A. Real-time NLP moderation on XMPP with GPU-accelerated inference and adaptive batching. *Scientific and Technical Journal of Information Technologies, Mechanics and Optics*, 2026, vol. 26, no. 4, pp. 851–859. doi: 10.17586/2226-1494-2026-26-4-851-859

УДК 004.738.5

Модерация NLP в реальном времени в XMPP с GPU-ускоренным инференсом и адаптивным батчингом

Алаа Альдарф[1]✉, Алаа Шакер[2], Игорь Александрович Бессмертный[3]

[1,3] Университет ИТМО, Санкт-Петербург, 197101, Российская Федерация
[2] Санкт-Петербург, Российская Федерация

[1] aaldarf@itmo.ru✉, https://orcid.org/0000-0001-7249-5071
[2] alaashaker11071991@gmail.com, https://orcid.org/0000-0003-2709-0766
[3] bessmertny@itmo.ru, https://orcid.org/0000-0001-6711-6399

Аннотация

Введение. Современные системы мгновенного обмена сообщениями обрабатывают большие объемы коротких текстов, для которых критично время доставки. В таких условиях становится актуальной автоматическая модерация токсичного контента. Примитивные методы, основанные на ключевых словах и ручной проверке, плохо справляются со значительной нагрузкой, многоязычными сообщениями и сложностью контекста. Модели обработки естественного языка, основанные на трансформерах, позволяют более точно определять токсичность, однако их использование в децентрализованных системах обмена сообщениями, таких как протокол расширяемого обмена сообщениями и присутствия (Extensible Messaging and Presence Protocol, XMPP), затруднено высокой вычислительной стоимостью и асинхронным характером запросов. **Метод.** Предложена система модерации сообщений в XMPP-сетях, состоящая из плагина сервера OpenFire, внешнего сервиса инференса, поддерживающего выполнение на центральном процессоре (Central Processing Unit, CPU) и графическом процессоре (Graphics Processing Unit, GPU), и метода адаптивного батчинга (Adaptive Cross-Domain Batching, ACDB). Плагин осуществляет перехват сообщений на уровне обработки пакетов и передает их во внешний сервис инференса для определения токсичности. Предложенный метод адаптивного батчинга динамически подстраивает размер батча в зависимости от текущей нагрузки, состояния очереди и фактического времени выполнения инференса на GPU. **Основные результаты.** Эксперименты проводились на CPU и GPU конфигурациях с текстами сообщений различной длины, соответствующими типичным чат-сообщениям. Полученные результаты показывают, что использование GPU позволяет значительно ускорить инференс — до 28 раз (по сравнению с CPU) и снизить общую задержку до 9 раз. Сравнительная оценка по отношению к базовым решениям со статическим батчингом в различных сценариях трафика показывает, что ACDB динамически адаптирует размеры батчей в диапазоне от 3 до 25 в зависимости от условий нагрузки, снижая задержку на 52–59 % по сравнению со статическими конфигурациями с крупными батчами при сохранении 85–94 % их пропускной способности. **Обсуждение.** Представленный подход ориентирован на реальные условия работы федеративных XMPP-сетей, в которых сообщения поступают асинхронно и неравномерно. В отличие от решений, рассчитанных на централизованные системы с предсказуемым трафиком, данная архитектура обеспечивает устойчивую и масштабируемую модерацию сообщений в реальном времени без увеличения задержек доставки.

Ключевые слова

обнаружение токсичности, XMPP протокол, OpenFire, GPU ускорение, адаптивный батчинг, модерация NLP, распределенные системы, трансформерный инференс

Ссылка для цитирования: Альдарф А., Шакер А., Бессмертный И.А. Модерация NLP в реальном времени в XMPP с GPU-ускоренным инференсом и адаптивным батчингом // Научно-технический вестник информационных технологий, механики и оптики. 2026. Т. 26, № 4. С. 851–859 (на англ. яз.). doi: 10.17586/2226-1494-2026-26-4-851-859

Introduction

The large volumes of real-time, user-generated text from communication applications create a high demand for automated toxicity detection systems with very low latency. Manual review or keyword-based filtering are two traditional approaches to moderating content; however, both methods are ineffective in environments characterized by rapid message flow, multilingual usage, and nuanced forms of hate speech. Because of their strong contextual understanding, Transformer-based language models have been widely adopted for text moderation and provide significant advantages over prior methods for text classification. However, deploying these models at scale is resource-intensive, particularly when there are many small latency-critical requests from users.

Recent research on inference optimization for foundation models emphasizes that the performance of Transformer inference depends not only on model architecture but equally on hardware accelerators, memory bandwidth, and serving-system design [1]. Techniques for improving performance by using multi-Graphics Processing Unit (GPU) inference, managing memory across heterogeneous architectures, and optimizing the execution of kernels have been shown to greatly improve both the latency and throughput associated with inference [2]. However, these strategies are not directly applicable to decentralized real-time messaging systems, where requests originate from multiple servers, exhibit highly variable sequence lengths, and must conform to tight service-level constraints.

The challenge is further compounded when content moderation must integrate with distributed protocols such as Extensible Messaging and Presence Protocol (XMPP). Modern moderation frameworks typically target centralized social-media infrastructures, focusing on large-batch offline inference or content pipelines. Only a few solutions address moderation directly inside federated, multi-server messaging systems. At the same time, Natural Language Processing (NLP)-based moderation is increasingly essential due to the rapid growth of toxic behavior online and the limitations of manual moderation [3–7].

This work introduces a complete system that fills this gap by integrating: a real-time content-moderation plugin for the XMPP OpenFire server, a GPU-accelerated microservice performing toxicity inference using Transformer models, and a new Adaptive Cross-Domain Batching (ACDB) algorithm that aggregates traffic from multiple chat servers to maximize GPU utilization while respecting tight latency bounds. ACDB builds on concepts from dynamic batching in GPU inference [8] and multi-task scheduling for edge-Artificial Intelligence (AI) systems [9], extending these ideas into a distributed, multi-tenant messaging environment. Our system demonstrates that dynamic batching when informed by text-length statistics, server-side arrival patterns, and GPU forward-pass behavior significantly improves inference throughput compared to static batching or naive queue policies. This study focuses

on inference performance and batching behavior rather than model accuracy, using standard pre-trained toxicity classifiers as a baseline.

Related Work and Background

The optimization of Transformer inference has been widely studied due to the computational demands of modern foundation models. [1] discusses system-level strategies such as memory-efficient attention, model compression, and fast decoding techniques suitable for AI accelerators. [2] further introduces multi-GPU parallelism, heterogeneous memory utilization, optimized kernels for small-batch inference, and aggregate-bandwidth strategies enabling trillion-parameter-scale inference. Other works propose specialized runtime systems such as [10], which address variable-length input workloads through fused GPU kernels, efficient memory reuse, and novel sequence-length-aware batching algorithms. Additional research explores quantization, pruning, and software-hardware co-design principles for achieving low-latency inference on resource-constrained systems [11]. These studies collectively demonstrate that GPU throughput is highly sensitive to batch composition, kernel launch overhead, and input-sequence variability — properties that directly motivate the development of our ACDB batching strategy.

Dynamic batching has been recognized as a key strategy for improving inference efficiency. [8] proposes latency-aware dynamic batch sizing on GPUs. [12] adapts batch composition during autoregressive decoding for large language models, enabling insertion/removal of queries with minimal idle computation. Other studies explore batching in edge-AI systems with asynchronous task arrivals. [9] formulates a mixed-integer optimization problem balancing deadlines, upload delays, and GPU compute, highlighting the tradeoff between latency and batch size. [13] introduces bit-serial decomposition techniques that restructure neural-network workloads to achieve higher compute efficiency at low precision, indirectly motivating batch-aware compute restructuring. Finally, research on batching for geometric learning [14] compares static and dynamic batching schemes for Graph Neural Networks, reporting up to 2.7× training-time speedups. Although these works analyze batching in GPU-accelerated Deep Learning workloads, none examine cross-server batching where inference requests originate from multiple independent communication servers. ACDB extends the principles of dynamic batching into this domain, integrating both queue-aware and length-aware heuristics tailored for real-time messaging.

Content moderation has evolved from rule-based filtering and shallow machine-learning models toward Transformer-based architectures due to their superior contextual understanding. Recent surveys [3] emphasize the importance of contextual accuracy, fairness, and interpretability in automated toxic-content detection systems. Studies comparing classical Machine Learning, Convolutional Neural Networks, Long Short-Term Memories, and Transformer-based models consistently show that Transformer architectures outperform earlier approaches in detecting hate speech, spam, and offensive content [4–6]. Applications such as Anonify [6] demonstrate real-time moderation pipelines using modern NLP techniques within anonymous messaging environments. Similarly, GDPR-compliant moderation frameworks [5] explore personalized moderation, counter-speech generation, and regulatory alignment. Group-chat research also shows that multi-user conversational agents require specialized moderation strategies that differ from single-user chatbot interaction patterns [7].

Prior work has explored layered moderation pipelines that combine lightweight rule-based filtering with Transformer-based classifiers to reduce inference cost while maintaining accuracy. In particular, [15] introduces a three-layer GPU-accelerated architecture incorporating string matching, linguistic matching, and a Transformer classifier, demonstrating substantial inference speedups by offloading simpler cases to faster preprocessing stages. While effective at reducing computational overhead, such approaches do not address deployment within federated, multi-server messaging infrastructures, nor do they consider adaptive batching strategies that exploit cross-server traffic aggregation. Our work extends this line of research by integrating GPU-accelerated moderation directly into XMPP-based systems and introducing an adaptive batching mechanism designed for asynchronous, multi-server real-time workloads.

GPU-CPU Inference Microservice

To support real-time toxicity moderation within OpenFire-based XMPP deployments, we implement a standalone inference microservice capable of executing on either Central Processing Unit (CPU) or GPU hardware. The service exposes a lightweight Representational State Transfer interface and performs multilingual toxicity classification using a Transformer-based model. By decoupling inference from the messaging server, the system enables independent scaling, hardware flexibility, and controlled performance evaluation across heterogeneous environments.

The microservice operates as an intermediary between the OpenFire message pipeline and the neural classifier. Incoming messages are grouped into batches, processed for language identification, and evaluated for toxicity. For each message, the service returns a toxicity score in the range 0–1 along with a detected language tag. These outputs are consumed by the OpenFire policy engine to determine whether messages are delivered, annotated, or blocked according to configured moderation rules.

Architecture

The microservice is implemented using FastAPI, providing a low-latency Hypertext Transfer Protocol (HTTP) interface suitable for high-throughput moderation workloads. Toxicity classification is performed using the *unitary/toxic-bert* Transformer model [16], while lightweight statistical language detection is applied prior to inference. The model is loaded once at startup and remains resident in memory for the lifetime of the service.

Requests are submitted as batches of text strings via a single inference endpoint, and responses preserve input order to ensure deterministic mapping between messages

and moderation decisions. In addition to classification outputs, the service records batch-level timing information used for profiling and performance evaluation.

The service is hardware agnostic: when deployed in a GPU-enabled environment, inference is automatically executed on Compute Unified Device Architecture; otherwise, execution proceeds on the CPU without code changes. This design allows direct comparison of CPU and GPU inference behavior under identical software conditions.

Integration with OpenFire

Within OpenFire, the microservice serves as an external moderation engine accessed by a custom server plugin. Message bodies are forwarded to the inference service during packet interception, and moderation decisions are applied synchronously within the routing path once scores are returned. Harmful messages may be blocked or replaced with a standardized moderation notice, while non-toxic messages are forwarded normally.

By externalizing inference, the OpenFire server remains lightweight and independent of model-specific computation. Multiple OpenFire instances can simultaneously submit requests to a shared inference backend, making the design suitable for federated or multi-server deployments.

Design Rationale

This microservice-based design provides three key advantages. First, it enables efficient GPU acceleration without introducing model dependencies into the messaging server. Second, batch-level inference improves throughput while allowing latency-sensitive operation. Third, centralized inference across multiple servers enables shared GPU utilization and supports adaptive batching strategies explored in later sections.

Overall, the GPU-CPU inference microservice forms a modular foundation for real-time Transformer-based moderation in distributed communication systems, enabling systematic evaluation of hardware acceleration and batching behavior under realistic messaging workloads.

OpenFire Moderation Plugin

To enable real-time toxicity moderation within XMPP-based group conversations, we developed a custom OpenFire plugin, referred to as GPU-Filter (GPUF) that integrates directly into the server packet-processing pipeline. The plugin operates at the packet-interception layer using OpenFire PacketInterceptor Application Programming Interface, allowing messages to be analyzed before routing decisions are finalized.

The plugin is implemented as a modular extension that registers with OpenFire interceptor framework at initialization. This design avoids modifications to the server core and allows the plugin to be enabled, disabled, or reconfigured at runtime. As a result, moderation logic can be introduced or updated without disrupting the underlying messaging infrastructure.

Message Interception and Moderation Flow

When a message stanza is intercepted, its textual content is extracted and forwarded to the external inference microservice for toxicity evaluation. Messages are submitted asynchronously and may be grouped into batches to improve inference efficiency.

The microservice returns toxicity scores in the same order as the submitted messages, enabling deterministic association between classification results and message stanzas. Based on configurable policy thresholds, the plugin applies one of three actions: allowing the message to pass unchanged, annotating it with a moderation warning, or blocking it by replacing the message body with a standardized notice.

Backend Communication and Fault Handling

Communication between GPUF and the inference backend is performed over HTTP using JavaScript Object Notation (JSON) serialization. To ensure robustness under network failures or backend unavailability, the plugin supports configurable fault-handling behavior. In fail-open mode, messages are delivered normally if inference cannot be performed; in fail-closed mode, messages are blocked to prioritize safety over availability.

The plugin does not store the message content or user-identifying information. All text is processed transiently in memory and discarded immediately after classification, ensuring privacy preservation and compliance with common data-handling requirements.

Design Considerations

By externalizing inference and restricting the plugin to message interception and policy enforcement, GPUF keeps the OpenFire server lightweight and stateless with respect to model execution. Multiple OpenFire instances can concurrently communicate with a shared inference backend, enabling centralized GPU utilization across federated servers.

This design cleanly separates messaging concerns from compute-intensive moderation logic and provides the necessary integration point for adaptive batching strategies evaluated in later sections.

Experimental Evaluation of CPU and GPU Performance

Experimental Setup

The performance of the proposed toxicity-classification microservice was evaluated on two hardware configurations hosted within identical Hugging Face Space environments:

— CPU Instance — 16 GB Random Access Memory (RAM), no hardware acceleration;
— GPU — NVIDIA Tesla T4 GPU (15 GB Video Random Access Memory (VRAM)) with identical RAM and software configuration.

Both configurations used the same Transformer-based toxicity classifier and runtime code to isolate the impact of hardware acceleration.

Synthetic messages of 10, 50, and 100 tokens were processed using batch sizes ranging from 1 to 128. Token lengths were chosen to reflect common ranges observed in real-time chat messages. For each configuration, end-to-end latency, model forward-pass time, throughput, and resource utilization were recorded. This setup reflects the operational characteristics of real-time moderation systems serving asynchronous requests from multiple XMPP servers.

CPU vs. GPU Latency and Throughput

Fig. 1 presents the latency and throughput trends for both CPU and GPU inference across batch sizes and message lengths. The GPU demonstrates a substantial advantage across all scenarios. Although the raw Transformer forward pass achieves up to 28× acceleration over the CPU, the overall system-level speedup is lower — typically 5–9× — due to the influence of non-compute overheads such as HTTP transport, JSON serialization, and host-side batching.

Table 1 summarizes end-to-end results for 100-token messages. At batch size 1, GPU inference achieves approximately 2× lower latency than CPU inference. As batch size increases, GPU advantages become more pronounced: at batch 16, latency improves by 5.7× and throughput increases from 2.33 to 13.29 text/s. At batch 128, GPU throughput reaches 19.34 text/s compared to 2.39 text/s on CPU, corresponding to a 9× latency reduction and an 8.1× throughput gain.

These results confirm the substantial system-level benefits of GPU deployment for toxicity classification workloads, especially under high-batch or high-traffic conditions.

Forward-Pass Performance

Table 2 reports model-only forward-pass times for 100-token messages. Across all batch sizes, the GPU achieves 15–28× acceleration over the CPU. This confirms that the Transformer inference workload is well suited to GPU execution and that most remaining latency arises from non-compute overheads rather than model execution itself.

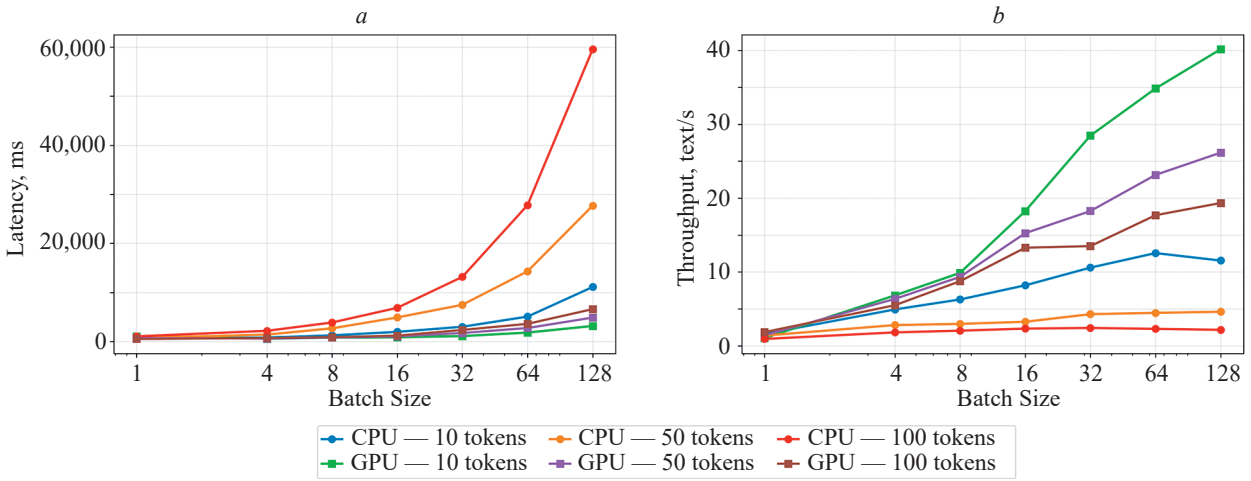

Fig. 1. End-to-end latency (*a*) and throughput (*b*) vs. batch size across CPU and GPU configurations

Table 1. End-to-end latency and throughput for 100-token messages

Batch Size	CPU Latency, ms	GPU Latency, ms	CPU Throughput, text/s	GPU Throughput, text/s	Latency Speedup	Throughput Speedup
1	1,071	541	0.94	1.86	2.0×	2.0×
4	2,196	726	1.82	5.51	3.0×	3.0×
8	3,893	914	2.06	8.75	4.3×	4.2×
16	6,883	1,205	2.33	13.29	5.7×	5.7×
32	13,189	2,377	2.43	13.50	5.5×	5.6×
64	27,707	3,618	2.30	17.71	7.7×	7.7×
128	59,575	6,618	2.39	19.34	9.0×	8.1×

Table 2. Forward-pass computation time for 100-token messages

Batch Size	CPU Forward, ms	GPU Forward, ms	Speedup
1	560.60	19.99	28.0×
4	1,640.31	94.06	17.4×
8	3,091.90	192.52	16.1×
16	6,053.97	393.15	15.4×
32	11,895.76	797.37	14.9×
64	25,625.20	1,616.53	15.9×
128	56,731.82	3,230.99	17.6×

Scaling Behavior and Resource Utilization

CPU latency increases nearly linearly with batch size and message length, reflecting limited parallelism. In contrast, GPU latency scales sub-linearly for batch sizes between 8 and 64, indicating effective kernel-level parallelism. GPU throughput increases until saturation, demonstrating efficient utilization under realistic workloads.

GPU inference also reduces host-side CPU utilization to approximately 15–25 %, compared with roughly 60 % under CPU-only execution. Memory usage follows a similar pattern: GPU deployments consume 420–500 MB of VRAM with minimal CPU RAM usage, while CPU-only runs consume approximately 0.9–1.0 GB of system memory.

By offloading compute-intensive inference to the GPU, CPU resources remain available for batching, HTTP handling, and other server operations. Combined with asynchronous communication between OpenFire and the microservice, this enables pipelined batch processing without blocking message flow, supporting scalable, low-latency moderation under high traffic conditions.

Dynamic Batching Policy for Multi-Server GPU Inference

Real-time moderation workloads differ from conventional deep-learning inference pipelines in three key ways: requests originate from multiple independent XMPP servers, arrive asynchronously, and exhibit high variability in message length. This creates a fundamental tradeoff for GPU-based inference. Large batches improve throughput but increase queueing delay, while small batches minimize latency but underutilize GPU parallelism. Static batching policies cannot resolve this tension when traffic intensity fluctuates across servers.

To address this, we introduce ACDB, a dynamic batching mechanism designed for federated messaging environments. ACDB continuously adjusts batch size using real-time workload characteristics and observed GPU execution behavior, aiming to minimize effective latency while maintaining high utilization. The term *cross-domain* denotes batching across independent XMPP servers that do not coordinate traffic generation or batching decisions.

ACDB Design

In the proposed architecture, multiple XMPP servers forward messages to a shared moderation backend. Incoming texts are placed into a global queue from which ACDB constructs batches for GPU inference (Fig. 2). ACDB maintains two rolling data structures: an arrival time buffer recording timestamps of incoming requests, and a GPU forward time buffer storing observed execution times. Both are implemented as fixed-size circular queues (default capacity: 2,000 entries).

Latency Estimation Model

The core of ACDB is a latency estimation model. The arrival rate λ is estimated from the mean inter-arrival time. The estimated latency for batch size B is:

$$Test = \frac{B}{\lambda} \times 1{,}000 + Tgpu,$$

where the first term represents queuing delay (time to accumulate B messages at rate λ) and $Tgpu$ is the mean GPU forward time from the performance buffer.

Adaptation Rules

ACDB applies threshold-based adaptation: if $Test < 0.7 \times Ttarget$, batch size ($Bnew$) increases by 1.5×; if $Test > Ttarget$, batch size ($Bnew$) decreases by 0.7×. To prevent oscillation, exponential smoothing is applied:

$$Bfinal = \alpha \times Bnew + (1 - \alpha) \times Bcurrent,$$

where $Bnew$ is the candidate batch size produced by the threshold rules; $Bcurrent$ is the batch size used at the previous adaptation step; $Bfinal$ is the resulting smoothed batch size applied to the next batch; α is the smoothing coefficient (α = 0.2 by default). The final batch size is clamped to [Bmin, Bmax] bounds, where Bmin, and Bmax denote the minimum and maximum allowed batch sizes.

Performance Buffer Sizing

Table 3 summarizes buffer sizing trade-offs. The default capacity of 2,000 entries provides 2–3 minutes of history at typical traffic rates, balancing estimation accuracy with adaptation responsiveness.

Experimental Evaluation

To validate ACDB adaptive behavior across different traffic intensities, we conducted benchmarks under three scenarios: moderate traffic (10 servers, normal arrival rates), high traffic (20 servers, fast arrivals), and burst

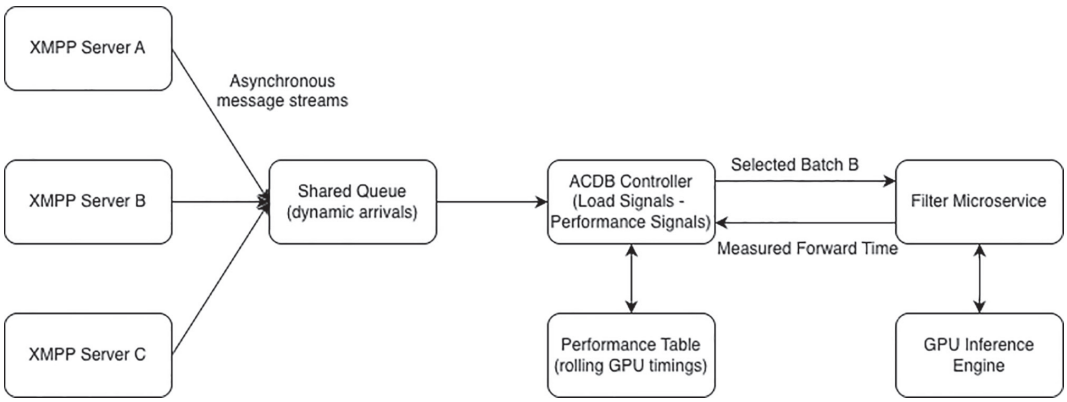

Fig. 2. Architecture of the ACDB controller

Table 3. Performance buffer sizing trade-offs

Buffer Size	Memory, kB	History Window, s	Characteristics
100	about 0.8	5–10	Highly responsive; noisy
2,000 (default)	about 16	100–200	Balanced
5,000	about 40	250–500	Very stable; slow adaptation

Table 4. ACDB performance across traffic scenarios

Scenario	Mode	Throughput, text/s	Mean Latency, ms	P95 Latency, ms	Batch Mean	Batch Max
Moderate	ACDB	21.0	1,538	1,499	3.4	9
Moderate	Static-8	30.1	2,575	2,952	8.0	8
Moderate	Static-32	29.0	10,852	12,542	32.0	32
High	ACDB	72.5	3,176	3,937	12.3	20
High	Static-8	79.7	2,098	2,254	8.0	8
High	Static-32	85.5	6,576	8,641	26.7	32
Burst	ACDB	71.3	2,858	3,633	14.8	25
Burst	Static-8	76.3	1,521	1,696	7.5	8
Burst	Static-32	127.7	6,953	14,776	30.0	32

Table 5. ACDB batch size distribution under burst traffic

Batch Range	Count	Percentage, %	Traffic Condition
1–9	8	13	Low arrival rate periods
10–15	23	38	Moderate arrival rate
16–20	18	29	High arrival rate
21–25	12	20	Burst peaks

traffic (15 servers, very fast arrivals simulating traffic spikes). Each scenario compared ACDB against static batching with fixed batch sizes of 8 and 32. Table 4 summarizes the results across all configurations.

The results demonstrate ACDB key capability: dynamic batch size adaptation based on traffic conditions. Under moderate traffic, ACDB selected small batches (mean 3.4, max 9) to minimize latency. Under high and burst traffic, ACDB scaled up to larger batches (mean 12–15, max 20–25) to improve throughput while maintaining acceptable latency.

Compared to Static-32, ACDB achieved 52 % lower latency under high traffic (3,176 vs. 6,576 ms) and 59 % lower latency under burst traffic (2,858 vs. 6,953 ms). Compared to Static-8, ACDB achieved 40 % lower latency under moderate traffic (1,538 vs. 2,575 ms) while matching 91–94 % of its throughput under high-traffic conditions.

Traffic Variation Response

Table 5 shows ACDB batch size distribution under burst traffic, demonstrating the full dynamic range from 1 to 25.

This distribution confirms that ACDB successfully adapts to varying traffic intensities without manual configuration, automatically selecting batch sizes appropriate to current load conditions.

ACDB addresses the batching challenge in multi-server GPU inference through feedback-driven adaptation.

Key findings:

— ACDB dynamically scales batch sizes from 3 to 25 based on traffic intensity;
— Under high traffic, ACDB reduces latency by 52 % compared to Static-32 while achieving 85 % of its throughput;
— Under moderate traffic, ACDB reduces latency by 40 % compared to Static-8;
— GPU efficiency remains consistent across all configurations, confirming latency improvements arise from reduced queuing.

Conclusion

This work demonstrates an effective approach to integrating automated toxicity moderation directly into federated chat systems. The proposed architecture combines a lightweight OpenFire plugin with a dedicated Graphics Processing Unit (GPU) — Central Processing Unit (CPU) microservice, enabling high-throughput, low-latency Transformer inference without burdening the messaging server. Benchmarks confirm that GPU acceleration provides large performance gains compared to CPU execution, and these gains translate into practical throughput improvements when processing real chat traffic.

A key contribution of this study is the Adaptive Cross-Domain Batching (ACDB) mechanism. Experimental evaluation across three traffic scenarios demonstrates that ACDB dynamically adapts batch sizes from 3 to 25 depending on load conditions. Under high traffic, ACDB achieves 52–59 % lower latency than static large-batch configurations while maintaining 85–94 % of their

throughput. Under moderate traffic, ACDB achieves 40 % lower latency than static small-batch configurations. This makes it particularly well suited for messaging systems in which multiple independent servers generate asynchronous workloads.

Future developments could explore predictive or learning-based approaches that anticipate traffic bursts and proactively adjust batch size. Scaling the system to larger or more complex Transformer architectures may require distributed inference strategies such as tensor parallelism or pipeline parallelism. Additional extensions could incorporate multimodal moderation, more detailed user-context awareness, or enhanced explainability to support regulatory and ethical requirements. Applying ACDB to other distributed inference domains, such as edge Artificial Intelligence, streaming analytics, or federated microservices, also offers promising research opportunities.

Overall, this work establishes a practical foundation for real-time Natural Language Processing moderation in federated communication environments and highlights the importance of dynamic batching for efficient GPU-based inference.

References

1. Park Y., Budhathoki K., Chen L., Kübler J.M., Huang J., Kleindessner M., et al. Inference optimization of foundation models on AI accelerators. *Proc. of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '24)*, 2024, pp. 6605–6615. doi: 10.1145/3637528.3671465
2. Aminabadi R.Y., Rajbhandari S., Awan A.A., Li C., Li D., Zheng E., et al. DeepSpeed-inference: Enabling efficient inference of transformer models at unprecedented scale. *Proc. of the SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2022, pp. 1–15. doi: 10.1109/SC41404.2022.00051
3. Khan Z. Natural language processing techniques for automated content moderation. *International Journal of Web of Multidisciplinary Studies*, 2025, vol. 2, no. 2, pp. 21–27.
4. Liaskovska S., Bacarra R., Martyn Y., Baidych V., Alsayaydeh J. Artificial intelligence for automatic moderation of textual content in online chats and social networks. *International Journal of Electrical and Computer Engineering (IJECE)*, 2025, vol. 15, no. 3, pp. 3396–3409. doi: 10.11591/ijece.v15i3.pp3396-3409
5. Fillies J., Mitsikas T., Schäfermeier R., Paschke A. A hate speech moderated chat application: Use case for GDPR and DSA compliance. *arXiv*, 2024. arXiv:2410.07713. doi: 10.48550/arXiv.2410.07713
6. Mengade S., Chopade P., Tate P., Patil S. Facilitating anonymous communication on social networks via AI-driven content moderation. *Journal of the ACS Advances in Computer Science*, 2025, vol. 16, no. 1, pp. 19–32. doi: 10.21608/asc.2025.351475.1033
7. Wagner N., Kraus M., Tonn T., Minker W. Comparing moderation strategies in group chats with multi-user chatbots. *Proc. of the 4th Conference on Conversational User Interfaces*, 2022, pp. 1–4. doi: 10.1145/3543829.3544527
8. Zhang D., Luo Y., Wang Y., Kui X., Ren J. BatOpt: Optimizing GPU-based deep learning inference using dynamic batch processing. *IEEE Transactions on Cloud Computing*, 2024, vol. 12, no. 1, pp. 174–185. doi: 10.1109/TCC.2024.3350561
9. Cang Y., Chen M., Huang K. Joint batching and scheduling for high-throughput multiuser edge AI with asynchronous task arrivals. *IEEE Transactions on Wireless Communications*, 2024, vol. 23, no. 10, pp. 13782–13795. doi: 10.1109/TWC.2024.3404811
10. Fang J., Yu Y., Zhao C., Zhou J. TurboTransformers: An efficient GPU serving system for transformer models. *Proc. of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '21)*, 2021, pp. 389–402. doi: 10.1145/3437801.3441578
11. Kasoju A., Vishwakarma T.V.C. Optimizing transformer models for low-latency inference: Techniques, architectures, and code implementations. *International Journal of Science and Research (IJSR)*, 2025, vol. 14, no. 4, pp. 857–866. doi: 10.21275/sr25409073105
12. Cong P., Chen Q., Zhao H., Yang T. Baton: Enhancing batch-wise inference efficiency for large language models via dynamic re-batching. *Proc. of the ACM on Web Conference*, 2025, pp. 2309–2318. doi: 10.1145/3696410.3714950
13. Lam M., Yedidia Z., Banbury C.R., Reddi V.J. Precision batching: Bitserial decomposition for efficient neural network inference on GPUs. *Proc. of the 30th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2021, pp. 129–141. doi: 10.1109/PACT52795.2021.00017

Литература

1. Park Y., Budhathoki K., Chen L., Kübler J.M., Huang J., Kleindessner M., et al. Inference optimization of foundation models on AI accelerators // Proc. of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '24). 2024. P. 6605–6615. doi: 10.1145/3637528.3671465
2. Aminabadi R.Y., Rajbhandari S., Awan A.A., Li C., Li D., Zheng E., et al. DeepSpeed-inference: Enabling efficient inference of transformer models at unprecedented scale // Proc. of the SC22: International Conference for High Performance Computing, Networking, Storage and Analysis. 2022. P. 1–15. doi: 10.1109/SC41404.2022.00051
3. Khan Z. Natural language processing techniques for automated content moderation // International Journal of Web of Multidisciplinary Studies. 2025. V. 2. N 2. P. 21–27.
4. Liaskovska S., Bacarra R., Martyn Y., Baidych V., Alsayaydeh J. Artificial intelligence for automatic moderation of textual content in online chats and social networks // International Journal of Electrical and Computer Engineering (IJECE). 2025. V. 15. N 3. P. 3396–3409. doi: 10.11591/ijece.v15i3.pp3396-3409
5. Fillies J., Mitsikas T., Schäfermeier R., Paschke A. A hate speech moderated chat application: Use case for GDPR and DSA compliance // arXiv. 2024. arXiv:2410.07713. doi: 10.48550/arXiv.2410.07713
6. Mengade S., Chopade P., Tate P., Patil S. Facilitating anonymous communication on social networks via AI-driven content moderation // Journal of the ACS Advances in Computer Science. 2025. V. 16. N 1. P. 19–32. doi: 10.21608/asc.2025.351475.1033
7. Wagner N., Kraus M., Tonn T., Minker W. Comparing moderation strategies in group chats with multi-user chatbots // Proc. of the 4th Conference on Conversational User Interfaces. 2022. P. 1–4. doi: 10.1145/3543829.3544527
8. Zhang D., Luo Y., Wang Y., Kui X., Ren J. BatOpt: Optimizing GPU-based deep learning inference using dynamic batch processing // IEEE Transactions on Cloud Computing. 2024. V. 12. N 1. P. 174–185. doi: 10.1109/TCC.2024.3350561
9. Cang Y., Chen M., Huang K. Joint batching and scheduling for high-throughput multiuser edge AI with asynchronous task arrivals // IEEE Transactions on Wireless Communications. 2024. V. 23. N 10. P. 13782–13795. doi: 10.1109/TWC.2024.3404811
10. Fang J., Yu Y., Zhao C., Zhou J. TurboTransformers: An efficient GPU serving system for transformer models // Proc. of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '21). 2021. P. 389–402. doi: 10.1145/3437801.3441578
11. Kasoju A., Vishwakarma T.V.C. Optimizing transformer models for low-latency inference: Techniques, architectures, and code implementations // International Journal of Science and Research (IJSR). 2025. V. 14. N 4. P. 857–866. doi: 10.21275/sr25409073105
12. Cong P., Chen Q., Zhao H., Yang T. Baton: Enhancing batch-wise inference efficiency for large language models via dynamic re-batching // Proc. of the ACM on Web Conference. 2025. P. 2309–2318. doi: 10.1145/3696410.3714950
13. Lam M., Yedidia Z., Banbury C.R., Reddi V.J. Precision batching: Bitserial decomposition for efficient neural network inference on GPUs // Proc. of the 30th International Conference on Parallel Architectures and Compilation Techniques (PACT). 2021. P. 129–141. doi: 10.1109/PACT52795.2021.00017

14. Speckhard D.T., Bechtel T., Kehl S., Godwin J., Draxl C. Training speedups via batching for geometric learning: An analysis of static and dynamic algorithms. *arXiv*, 2025. arXiv:2502.00944. doi: 10.48550/arXiv.2502.00944
15. Aldarf A., Shaker A., Bessmertny I., Sobeih M.Y. Accelerating hate speech classification with GPUs: A multi-layer approach with string matching and transformers. *Lecture Notes in Networks and Systems*, 2025, vol. 5588, pp. 535–548. doi: 10.1007/978-981-96-1918-4_38
16. Yang Z., Grenon-Godbout N., Rabbany R. Towards detecting contextual real-time toxicity for in-game chat. *Findings of the Association for Computational Linguistics: EMNLP*, 2023, pp. 9894–9906. doi: 10.18653/v1/2023.findings-emnlp.663

14. Speckhard D.T., Bechtel T., Kehl S., Godwin J., Draxl C. Training speedups via batching for geometric learning: An analysis of static and dynamic algorithms // arXiv. 2025. arXiv:2502.00944. doi: 10.48550/arXiv.2502.00944
15. Aldarf A., Shaker A., Bessmertny I., Sobeih M.Y. Accelerating hate speech classification with GPUs: A multi-layer approach with string matching and transformers // Lecture Notes in Networks and Systems. 2025. V. 5588. P. 535–548. doi: 10.1007/978-981-96-1918-4_38
16. Yang Z., Grenon-Godbout N., Rabbany R. Towards detecting contextual real-time toxicity for in-game chat // Findings of the Association for Computational Linguistics: EMNLP. 2023. P. 9894–9906. doi: 10.18653/v1/2023.findings-emnlp.663

Authors

Alaa Aldarf — PhD Student, ITMO University, Saint Petersburg, 197101, Russian Federation, https://orcid.org/0000-0001-7249-5071, aaldarf@itmo.ru
Alaa Shaker — PhD, Independent Researcher, Saint Petersburg, Russian Federation, https://orcid.org/0000-0003-2709-0766, alaashaker11071991@gmail.com
Igor A. Bessmertny — D.Sc., Full Professor, ITMO University, Saint Petersburg, 197101, Russian Federation, SC 36661767800, https://orcid.org/0000-0001-6711-6399, bessmertny@itmo.ru

Авторы

Альдарф Алаа — аспирант, Университет ИТМО, Санкт-Петербург, 197101, Российская Федерация, https://orcid.org/0000-0001-7249-5071, aaldarf@itmo.ru
Шакер Алаа — кандидат технических наук, независимый исследователь, Санкт-Петербург, Российская Федерация, https://orcid.org/0000-0003-2709-0766, alaashaker11071991@gmail.com
Бессмертный Игорь Александрович — доктор технических наук, профессор, профессор, Университет ИТМО, Санкт-Петербург, 197101, Российская Федерация, SC 36661767800, https://orcid.org/0000-0001-6711-6399, bessmertny@itmo.ru

Received 15.01.2026
Approved after reviewing 26.02.2026
Accepted 23.07.2026

Статья поступила в редакцию 15.01.2026
Одобрена после рецензирования 26.02.2026
Принята к печати 23.07.2026