SoftwareX 31 (2025) 102325

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

AffectStream: Kafka-based real-time affect monitoring system using wearable sensors

Jeonghyun Kim, Duri Lee, Uichin Lee *

KAIST, 291 Daehak-ro, Yuseong-gu, Daejeon 34141, Republic of Korea

ARTICLE INFO

Keywords:
Real-time affect monitoring
Kafka-based systems
Wearable sensors
Personalized affect classification

ABSTRACT

Real-time affect monitoring is essential for personalized and adaptive applications in fields like education, healthcare, and customer service. However, existing systems often struggle with scalability and low-latency requirements for processing high-frequency sensor data. To address these challenges, we propose *AffectStream*, a Kafka-based real-time affect monitoring system that processes wearable sensor data through a cloud-based pub/sub architecture to the applications. *AffectStream* ensures scalability, fault tolerance, and personalized emotional state analysis. Its robust performance is demonstrated through trace-based evaluations using three public datasets (i.e., WESAD, AMIGOS, and GalaxyPPG). This open-source framework advances real-time emotion recognition, paving the way for large-scale affective computing applications.

Code metadata

Current code version	1.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-25-00173
Permanent link to Reproducible Capsule	–
Legal Code License	MIT License
Code versioning system used	Git
Software code languages, tools, and services used	Python, Java, Amazon Web Services, Terraform, Locust, Docker, SQL.
Compilation requirements, operating environments & dependencies	Debian (Slim), Python 3.9.16, OpenJDK 21 (Early Access, Build 11), other requirements provided in requirements.txt
If available Link to developer documentation/manual	–
Support email for questions	jeonghyun.kim@kaist.ac.kr

1. Motivation and significance

Real-time affect monitoring refers to the process of tracking and analyzing a user's affect state (e.g., feelings, moods, and stress levels) via real-time sensor data analysis. Affect monitoring involves collecting various behavioral (e.g., facial expression and voice [1]), physiological (e.g., heart rate, skin conductance [2]), and psychological sensor data (e.g., self-reported stress and mood [3]) to monitor the current affect state using wearable sensors (e.g., Samsung Watch and Google Fitbit).

Understanding and responding to users' affect states is important, as it allows for more personalized and adaptive interactions across various applications. Personalized and adaptive services using real-time affect monitoring can be applied in multiple fields, such as education [4], military training [5], healthcare [6], and empowerment at workplaces [7]. For example, educational software can monitor students' cognitive and emotional state, and dynamically adjust the difficulty of the content or offer assistance that is appropriate to their emotions and concentration levels to maximize learning effectiveness [4,8]. In military and medical training, it could provide a targeted intervention by automatically monitoring trainees' affect condition in highly stressful environments, helping the trainees manage their emotions [5,9].

In general, a real-time affect monitoring system consists of four continuous and iterative stages: (1) data acquisition, (2) data streaming, (3) data processing, and (4) data storage (see Fig. 1) [10,11]. In the data acquisition stage, user data is collected in real time from one or multiple wearable devices (e.g., smartwatch and chest band). During the data streaming stage, large volumes of sensor data are transmitted

* Corresponding author.
E-mail addresses: jeonghyun.kim@kaist.ac.kr (Jeonghyun Kim), duri.lee@kaist.ac.kr (Duri Lee), uclee@kaist.ac.kr (Uichin Lee).

https://doi.org/10.1016/j.softx.2025.102325
Received 17 March 2025; Received in revised form 11 August 2025; Accepted 18 August 2025
Available online 28 August 2025
2352-7110/© 2025 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

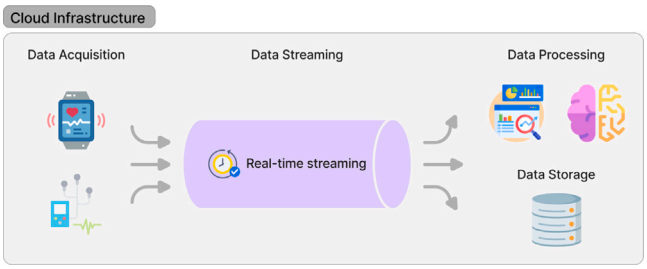


Fig. 1. Four stages of real-time affect monitoring system.

and processed in real time, requiring mechanisms to ensure data ordering and integrity. In the data processing stage, features are extracted and machine learning (ML) models are applied to estimate users' affect states from the collected data in real time. In the data storage stage, both the raw sensor data and the inferred affect states are stored in a large-scale storage system (e.g., cloud service) for future use.

To perform such complex data processing for a real-time affect monitoring system, a scalable open-source platform for real-time sensor data processing is necessary. Among well-known architectures, in this work, we use the publisher/subscriber (pub/sub) architecture. The pub/sub architecture utilizes one of the messaging patterns that allows for loose coupling between publishers and subscribers. Therefore, it enhances the efficiency of data transmission and processing, providing scalability and flexibility for the system. This architecture has been widely used in prior studies. Lohitha et al. [12] employed a cloud-based IoT platform utilizing a pub/sub architecture for real-time sensor data analysis, and Haque et al. [13] proposed a distributed pub/sub architecture for real-time remote patient monitoring using Movesense [14] sensor, which is a wearable sensor for measuring electrocardiogram (ECG), heart rate and movement. Various messaging systems, such as RabbitMQ, ActiveMQ, and Kafka, implement the pub/sub architecture.

- **RabbitMQ**: A message broker based on the Advanced Message Queuing Protocol (AMQP), known for its reliable message queuing and complex routing capabilities. Although RabbitMQ supports pub/sub messaging, its queuing system architecture is less suited for handling high-frequency, large-scale data streams [15].
- **ActiveMQ**: A Java Message Service (JMS)-supported message broker that offers both queue and pub/sub models, making it suitable for transactional messaging and enterprise applications [16].
- **Kafka**: A *distributed streaming* platform optimized for real-time data streaming and high throughput. Its architecture excels in log processing and scalable data pipelines [17]. Notably, Kafka supports excellent real-time throughput and provides superior performance in environments that require large-scale real-time data processing and streaming [15].

Compared to RabbitMQ's queuing mechanism and ActiveMQ's transactional messaging, Kafka's high throughput and scalability make it the optimal choice for real-time affect monitoring.

While prior benchmarking studies [15] have shown Kafka's superior throughput and scalability compared to RabbitMQ and ActiveMQ, our system's architecture was primarily driven by a strict functional requirement; i.e., per-user data ordering must be preserved across the entire real-time data processing pipeline. RabbitMQ and ActiveMQ can guarantee message order only within a single queue and under sequential consumption [18,19]. However, ordering may be violated under parallel consumption or load balancing across multiple queues. Unlike these systems, Kafka can preserve the exact order of each user's data by grouping messages by user ID. This ensures strict per-user ordering even under high-frequency, large-scale streaming workloads. This requirement constrained the choice of alternative messaging systems for *AffectStream*, with Kafka emerging as the optimal architecture to satisfy both the performance and ordering guarantees necessary for real-time affect monitoring.

Table 1
Kafka terminology.

Term	Description
Broker	A Kafka server storing data and serving producers/consumers.
Topic	A named feed where records are published.
Partition	A division of a topic's log for parallel processing.
Producer	A client publishing records to Kafka topics.
Consumer	A Client reading records from Kafka topics.
Consumer Group	A group of consumers that work together to consume data from a set of topics.
Cluster	A group of brokers working together.
Replication	The process of duplicating data across brokers for fault tolerance.
Throughput	The amount of data processed in a given period.
Latency	The time for a record to travel from producer to consumer.
Stream Processing	A real-time processing of continuous data from a topic.

Therefore, this paper proposes *AffectStream*, a real-time affect monitoring system built on a Kafka-based architecture. The system enables real-time affect tracking and is designed to be highly adaptable across various applications. It leverages a real-time distributed system to handle sensor data collection and storage in an end-to-end manner within a cloud environment. *AffectStream* can reliably process sensor data without performance degradation, even when multiple users access it simultaneously. Unlike conventional systems that classify emotions solely based on collected data, *AffectStream* integrates a real-time distributed framework to seamlessly manage processes from data collection to emotional classification within a cloud-based environment. By applying *AffectStream* in various fields, including education, healthcare, customer service, and psychological therapy, we can understand users' emotional states in real time and provide customized responses accordingly. The following sections will provide a detailed introduction to the architecture and applications of *AffectStream*.

2. Software description

AffectStream supports an end-to-end pipeline for real-time affect analysis, from sensor data collection to modeling and management. It operates on a Kafka-based cloud service and is designed to enable real-time classification of personalized models as well as real-time affect recognition in the workflow of Fig. 2. In this section, we provide an overview of *AffectStream*. For detailed explanations, please refer to our GitHub repository.

2.1. Software architecture

AffectStream is built on Kafka, designed to support parallel data processing. The entire architecture operates in a cloud environment for increased flexibility in case of more sensors and users.

2.1.1. Basic architecture illustration

Kafka is based on a pub/sub architecture, where the producer acts as the publisher and the consumer acts as the subscriber. Fig. 3 illustrates the basic structure and Table 1 summarizes the terminologies of Kafka. The *producer* is the client that sends data received from users (i.e., wearable or IoT sensor devices) to the broker in message units. Each data record contains a key, value, timestamp, and optional metadata. After hash processing, the data is serialized before being sent as messages to the broker. The *broker* stores the data published by the producer and provides the requested data to the consumer, thus managing the storage of records. Producers publish data to topics, which brokers

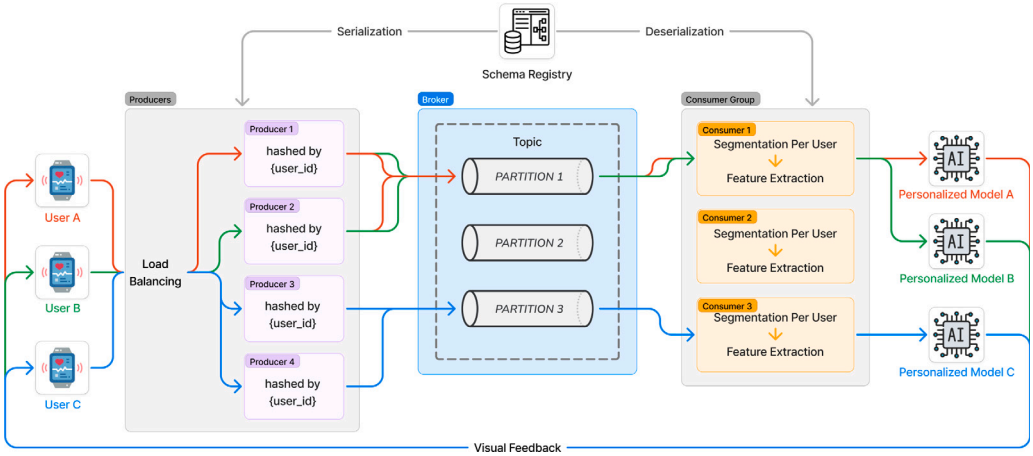


Fig. 2. The architecture of *AffectStream*.

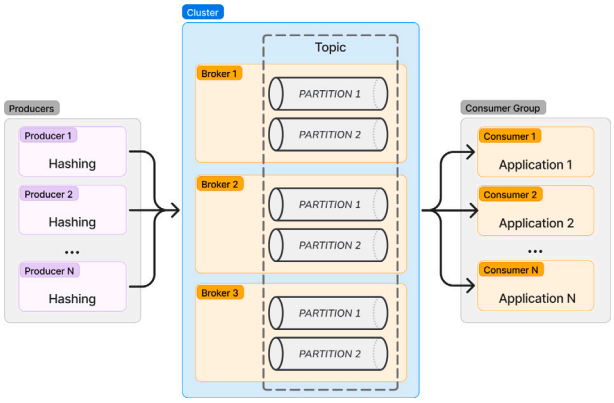


Fig. 3. The architecture of Kafka.

organize into partitions. These partitions are distributed across multiple brokers, enabling scalable and fault-tolerant storage. Scalability is achieved by distributing partitions across brokers, allowing the system to handle large data volumes efficiently. Fault tolerance is ensured through replication: partitions are replicated across multiple brokers in a cluster, so even if one broker fails, data remains accessible. This architecture allows Kafka to maintain high availability and reliability in data-intensive environments. The *consumer* is the client that consumes the records generated by the producer, processing the records received.

The operation of Kafka-based *AffectStream* is as follows (see Fig. 3). The producer (client side) creates records and uses a hash function to determine the partition where the record will be stored. Records with the same key are processed sequentially, and after hashing, they are serialized and sent to the broker. The broker uses the key from the producer to partition the records and then store them in the appropriate location. When a consumer sends a request, the broker retrieves the relevant records and sends them to the consumer. Within the same partition, the order of the data is guaranteed, allowing for data streaming. Records arriving at the consumer undergo deserialization before being utilized in the application. Following Kafka's basic operation principles, consumers within a single consumer group cannot read the same partition of the topic, which allows the system to be designed so that specific data can be read appropriately by designated applications.

2.1.2. Rationales for core elements of affectstream

Kafka-based pub/sub architecture. Kafka is selected because it can handle large volumes of sensor data by processing it in parallel. This is achieved by partitioning the data, which distributes the load across multiple brokers to prevent overload and reduce latency, making real-time services possible. Kafka also excels at managing time-series data, where the order of messages is important. By partitioning messages and preserving their order, Kafka facilitates smooth data streaming for machine learning models. Additionally, Kafka provides reliable data delivery through data replication, ensuring continuous service even if a subset of brokers fails.

Cloud-based scalable computing. *AffectStream* works in a cloud environment. Cloud services are easy to set up and highly scalable, enabling rapid adjustments to the required resources. In addition, the cloud environment facilitates the management and monitoring of remote device life cycles. Furthermore, data stored and processed in the cloud can be accessed from anywhere, improving data mobility. High-speed networks and data transfer technologies enable real-time data transmission, supporting the effective processing of sensing data and enabling quick decision-making.

Sensor data schema registry. For efficient processing of sensed data, *AffectStream* is designed for each component of the system to check the structure and format of the sensor data through a predefined schema in the schema registry. This allows for parallel construction of various data fields, making it easy to modify if the types of data change. Such flexibility facilitates the scalability and maintenance of the system.

2.2. Software functionalities

This section outlines key components and their roles in the workflow.

Producer. The producer in *AffectStream* is an API server that transmits user sensor data from various sensors to the broker. Multiple producer pods use a load balancer to distribute user data and manage traffic evenly. Data is collected from sensors and sent in predefined segments, with the producer using a user ID as the key for the hash function to assign partitions. Records with the same key are stored sequentially in the same partition, ensuring that data from the same user is stored sequentially in the same partition. The data is then serialized based on the schema defined in the Schema Registry before transmission to the broker. Note that the Schema Registry manages data schema, allowing producers and consumers to share a common format for serialization and deserialization.

Broker. The broker in *AffectStream* stores sensor data in partitions based on the key and transmits it to consumers. Partitioning is performed to ensure that the data from all users is evenly distributed across the partitions. Kafka maintains order within partitions, ensuring sequential storage of a user's data. Additionally, data replication across

multiple brokers enhances system reliability despite failures. By using the user ID as the key in the hash function, the producer ensures that data from the same user is stored in the same partition of the broker. In *AffectStream*, the number of consumer pods on Kubernetes matches the number of broker partitions, allowing Kafka to *guarantee the order of data for each person* within the same partition. However, uneven user activity can cause partition skew, leading to overloaded partitions and increased latency. To address this, *AffectStream* can employ partition scaling and user-level partition redistribution, enabling balanced load distribution while preserving the one-to-one mapping between partitions and consumers. This maintains per-user data ordering and ensures scalable, low-latency real-time affect monitoring.

Consumer. A consumer group in *AffectStream* has multiple consumers reading data from the same topic. Each consumer subscribes to specific partitions, ensuring exclusive processing per partition. Matching the number of consumers to partitions in the broker assigns each user's data consistently to one consumer, ensuring sequential processing and the order of each user's data.

Each consumer pod independently processes and analyzes data in real time through three steps. (1) Deserialization parses data using the schema defined in the Schema Registry for structured formatting. (2) Feature extraction applies a sliding window, dividing the incoming data stream into fixed-size segments (windows) and overlapping intervals that shift forward by a set step size. Through this segmentation, user-specific features can be extracted. (3) Affect classification uses extracted features and a pre-trained model for continuous, real-time affect state detection.

2.3. Implementation

The producer, consumer, and simulator were deployed on Kubernetes using Amazon Web Services (AWS) [20]. Kubernetes is an open-source container orchestration platform that automates the deployment, scaling, and management of containerized applications. It also supports various cloud resources and infrastructures for the Kafka system [21–23]. The producer was implemented based on a Spring-based API server that handles data transmission to the broker.

AffectStream secures sensitive physiological data using SCRAM-SHA-512 authentication within the Simple Authentication and Security Layer (SASL) framework, supported by AWS Secrets Manager. It runs in multiple Virtual Private Cloud (VPC) private networks with Amazon Elastic Compute Cloud (Amazon EC2) security groups for access control. Data in transit is encrypted via Transport Layer Security (TLS) between brokers and clients, with configurable encryption options. These measures ensure strong data protection suitable for privacy-sensitive applications like healthcare.

3. Illustrative examples

As an example use case, we selected a scenario where *AffectStream* is implemented to detect stress [24–26] of call agents at a typical call center with around 100 employees. We set up the evaluation environment using widely used datasets for stress detection, and the system was tested for real-time stress detection in this scenario. Note that our evaluation focuses on validating the performance and scalability of *AffectStream* itself, rather than directly measuring stress reduction or collecting user feedback in the call center setting. Our evaluation scenario is based on the prior research that has demonstrated that just-in-time (JIT) personalized interventions and data-driven feedback can positively impact stress management and user experience [7,27–30]. These works support the potential of real-time, personalized affective feedback, which *AffectStream* is designed to enable by providing a scalable and low-latency data streaming platform.

Table 2
System configuration for trace-based evaluation. vCPU (virtual CPU) represents a virtualized processing unit assigned to instances, where 1 vCPU = 1000 m in Kubernetes.

Components	Configurations
Producer	10 * {1 vCPU, 4 GiB memory}
Broker	3 * {2 vCPUs, 2 GiB memory}, 12 partitions
Consumer	12 * {1 vCPU, 1.5 GiB memory}
Load test simulator	6 * {1 vCPU, 2 GiB memory}

Table 3
Three trace datasets. SR indicates the sampling rate of each data.

Dataset	Wearable device	Data	SR (Hz)
WESAD	RespiBAN	ACC	700
		TEMP	
		EMG	
		EDA	
		ECG	
		RESP	
AMIGOS	Emotiv EPOC Neuro headset	EEG	128
	the Shimmer 2R4 platform	ECG	256
	the Shimmer 2R platform	GSR	128
GalaxyPPG	Empatica E4	ACC	32
		BVP	64
		HR	1
		TEMP	4
	Galaxy Watch 5	ACC	25
		HR	1
		PPG	25
		SkinTemp	1/60
	Polar H10	ACC	200
		ECG	130
		HR	1

3.1. Trace-based evaluation setup

System configuration. To evaluate the system's performance under real-world conditions, we deployed the producer, broker, consumer, and load test simulator on a distributed infrastructure. Table 2 summarizes the system configuration used for trace-based evaluation. Similar to a prior work [31], the system consisted of 10 producer instances, each with 1 vCPU (virtual CPU) and 4 GiB memory, responsible for handling high-frequency sensor data ingestion. The broker was deployed with 3 instances, each having 2 vCPUs and 2 GiB memory, and configured with 12 partitions. The consumer consisted of 12 instances, each with 1 vCPU and 1.5 GiB memory, processing incoming messages in parallel. Additionally, 6 load test simulator worker instances, each with 1 vCPU and 2 GiB memory, were used to generate traffic, simulating real-world workloads.

Trace datasets. We used three multimodal physiological sensor datasets to evaluate the generalizability of our approach: WESAD [32], AMIGOS [33], and GalaxyPPG [34]. Details of the sensing modalities, sampling rates, and experimental protocols are summarized in Table 3.

Since *AffectStream* aims to detect stress for individual users, we added a unique user identifier (UUID) to each data record via Python's `uuid.uuid4()` method. Using random numbers, it produces 128 bit identifiers with unique possibilities 2^{122}, effectively eliminating the risk of collision. This allows the system to analyze stress patterns on a per-user basis and supports real-time stress monitoring for multiple users, as required in scenarios such as call centers. Nonetheless, uneven user activity may cause partition skew and overload some consumers. Mitigation strategies such as dynamic partition scaling and user reassignment help balance load while maintaining data consistency and low latency.

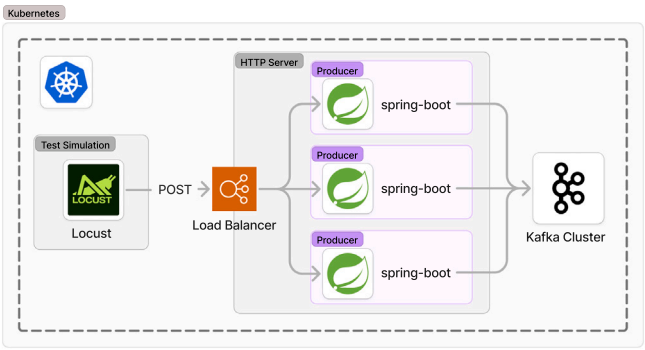


Fig. 4. Trace-based evaluation setup for load testing.

Sensor data generator. Fig. 4 illustrates a simulation implemented using Locust [35], an open-source tool for load testing of the system using HTTP and other protocols, to generate traffic for producers that closely resembles real-world scenarios. In this simulation, a total of 100 users were simulated by configuring Locust to spawn virtual users at a rate of 20 users per second. Sensor data was transmitted at the sampling rates specified for each sensor in the corresponding datasets (Table 3), with each segment spanning 1 s to the producer via POST.

Consumer of AffectStream. A sliding window [36] was applied to perform feature extraction in the consumers of *AffectStream*, with a window size of 2s and an overlap of 1s.

3.2. Functionality evaluation of AffectStream

This section evaluates three key functionalities of *AffectStream*: (1) validating schema and maintaining data consistency, (2) enabling real-time personalized affect classification, and (3) ensuring data order during processing. These were evaluated through latency, throughput, and metadata analysis.

Schema validation and data consistency. The schema defines data structure and types, ensuring that all messages adhere to a predefined format. AWS Glue Schema Registry is used to enforce schema validation during serialization and deserialization between producers and consumers. The validation process includes format verification, field presence checks, and data type validation to prevent schema violations. Producers serialize data according to the registered schema before sending it to the broker, and consumers deserialize the data while verifying its integrity against the schema. Inconsistent or incompatible data is rejected during this process, ensuring data consistency across the system.

We also measured the serialization overhead introduced by integrating AWS Glue Schema Registry with Kafka and found it to be negligible – less than 1.43% of total end-to-end latency of *AffectStream* – and thus it did not significantly affect the overall performance trends of the system.

Personal affect classification in real time. *AffectStream*'s capability to support real-time personalized affect classification was evaluated using end-to-end pipeline (E2E) latency and throughput. E2E latency, from data generation in the simulator to classification completion in the consumer, includes transmission through the Producer–Broker–Consumer pipeline. The throughput is the number of records processed per second. Timestamps recorded at data generation completion and classification completion provided the basis for calculating latency and throughput. Fig. 5 shows the latency results for each dataset, verifying low-latency operation. In the WESAD dataset, the mean latency was 353.14ms and the 99th percentile latency was 821.02ms. The AMIGOS dataset shows a mean latency of 98.14ms and a 99th percentile latency of 256.0ms, similar to the GalaxyPPG dataset, which has a mean latency of 55.89ms and a 99th percentile latency of 141.0ms. Fig. 6 confirms no data loss, as it matches Locust's 20 users/s spawn rate for all datasets, demonstrating robustness and scalability.

Table 4
Sustainable workload performance for three datasets under varying numbers of concurrent users. # Users indicates the number of concurrent users.

# Users	Latency		Throughput
	Mean (ms)	99% percentile (ms)	Mean (records/s)
100	353.14	821.02	20.02
500	1,279.92	4,002.0	19.97
1,000	38,262.82	200,664.16	16.77

(a) WESAD dataset.

# Users	Latency		Throughput
	Mean (ms)	99% percentile (ms)	Mean (records/s)
100	98.14	256.0	20.0
500	332.12	820.0	20.01
1,000	439.84	1,343.0	20.0

(b) AMIGOS dataset.

# Users	Latency		Throughput
	Mean (ms)	99% percentile (ms)	Mean (records/s)
100	55.89	141.0	20.0
500	296.39	750.0	20.0
1,000	317.59	989.0	20.0

(c) GalaxyPPG dataset.

Data order guarantee. To maintain data order, each message includes a user ID, timestamp, and offset value. The offset is a unique identifier assigned to each message within a partition, incrementing sequentially as new messages arrive. The consumer processes messages in order by tracking offsets, ensuring that records are consumed in the same sequence as they were produced. The experimental results were validated by checking that all messages belonging to the same user were in the correct order within a partition.

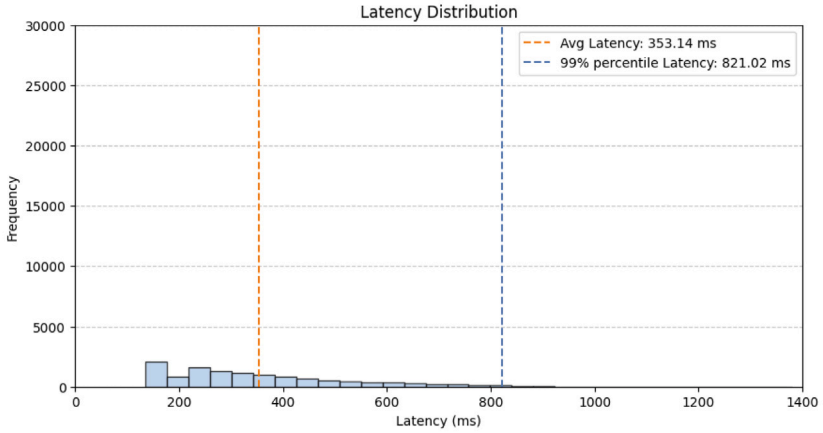
3.3. Trade-offs of kafka

This section evaluates the maximum workload that *AffectStream* can sustain while maintaining acceptable latency and throughput. We focus on identifying the load threshold beyond which performance degradation becomes significant, as well as analyzing the operational implications of approaching this limit. The evaluation combines queuing theory analysis with empirical measurements across multiple datasets.

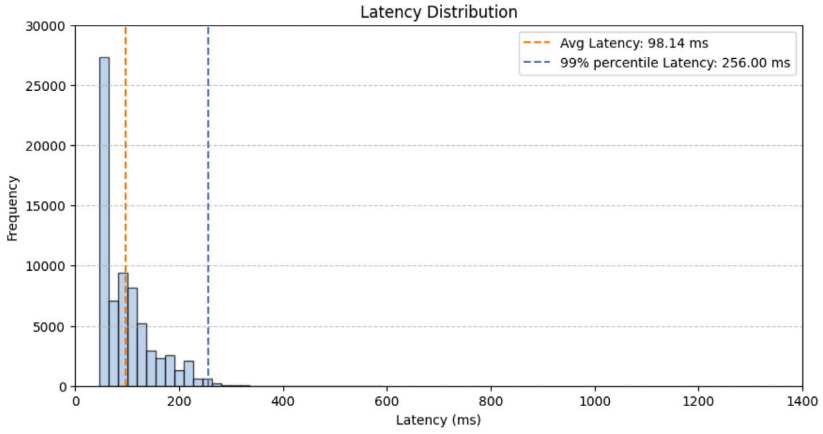
Sustainable workloads. In streaming data pipelines, all queuing systems eventually face performance bottlenecks as the input load increases beyond the processing capacity of the deployed infrastructure. This behavior aligns with queuing theory: in an M/M/1 model [37], as server utilization ρ approaches 1, the expected delay grows asymptotically to infinity. In practice, once broker I/O capacity or consumer fetch rates are saturated, queue backlogs grow without bound, and if sustained, can lead to disk exhaustion and forced broker shutdowns.

To analyze *AffectStream*'s sustainable workload, we incrementally increased the number of concurrent users while keeping the per-user data generation rate constant at each dataset's native sampling rates (Table 3). For each configuration, we measured the end-to-end mean latency, the 99th percentile latency, and the average throughput to identify the load threshold at which performance degradation becomes significant.

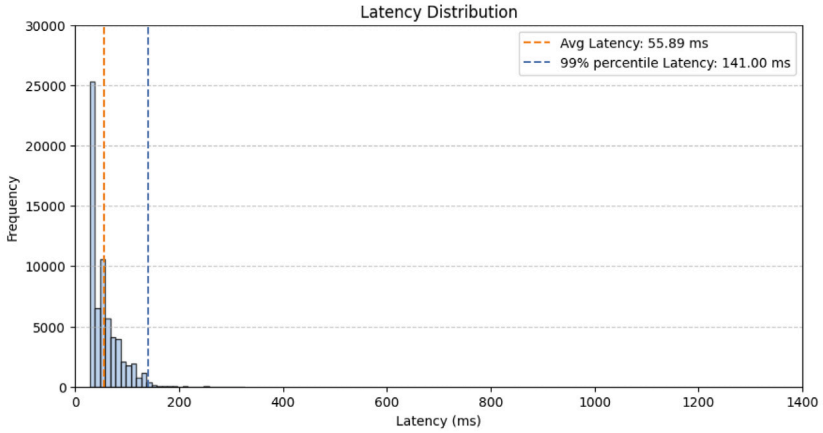
Table 4 shows the performance of *AffectStream* with 100, 500, and 1,000 concurrent users for each dataset. When using the WESAD dataset, which contains sensor data with the highest sampling rate, *AffectStream* exhibited latency exceeding 1,000ms with more than 500 concurrent users, and throughput dropped below 20.0 records/s. For



(a) WESAD dataset.



(b) AMIGOS dataset.

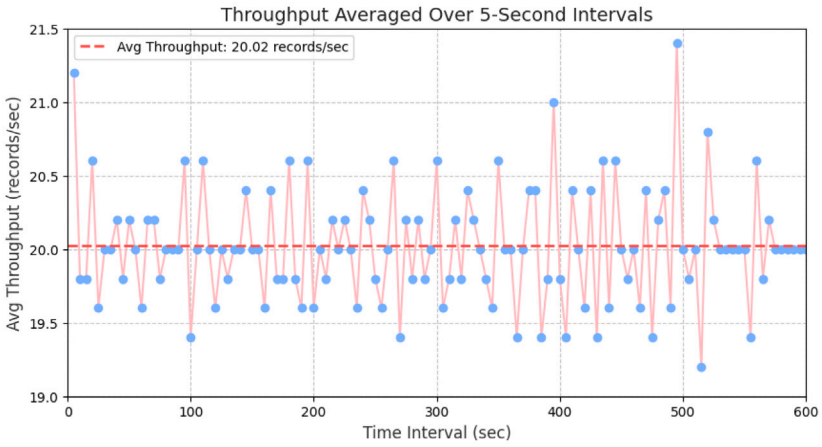


(c) GalaxyPPG dataset.

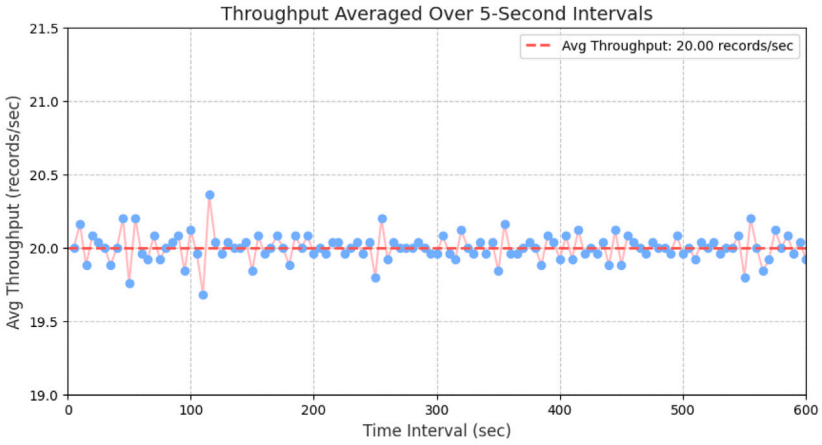
Fig. 5. End-to-end latency distribution for each dataset.

the AMIGOS and GalaxyPPG datasets, no significant increase in latency or decrease in throughput was observed due to the relatively low sampling rates of their sensor data. These results indicate that *AffectStream* can operate stably when collecting sensor data similar to the WESAD dataset with fewer than 500 concurrent users.

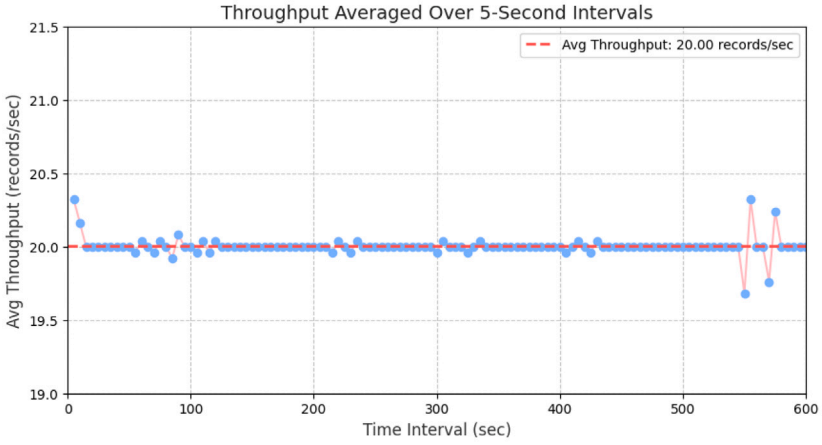
Operational trade-offs. While *AffectStream* achieves low-latency and stable throughput under normal workloads, operating a Kafka-based streaming pipeline in production entails inherent trade-offs in management complexity, cost, and performance variability under different loads. Kafka cluster management requires continuous monitoring and tuning of broker replication, partition assignments, and leader elections to ensure fault tolerance and balanced throughput. Furthermore, under extreme workloads, traffic surges can saturate broker I/O capacity and consumer fetch rates, leading to queue backlogs that grow without bound. If queue depths exceed available disk capacity, brokers are forced to shut down, interrupting processing entirely. Thus, to avoid reaching the workload limit, it is required to scale broker instances or increase partition counts, but this comes at the cost of higher cloud infrastructure expenses and increased operational overhead; for example, Amazon AWS charges per broker instance, data storage, and transfer amount. These trade-offs highlight that achieving high



(a) WESAD dataset.



(b) AMIGOS dataset.



(c) GalaxyPPG dataset.

Fig. 6. Throughput averaged over 5-s intervals for each dataset.

performance in a cloud-deployed streaming system requires not only optimizing throughput and latency, but also balancing cost efficiency.

4. Impact

AffectStream, a system that monitors users' affect states in real time using wearable sensor data, can be used to realize affective computing applications across various domains such as empowerment at workplaces, education, military training, and healthcare. For example, integrating *AffectStream* with voice assistants or chatbots in a call center setting can enable real-time detection of customer dissatisfaction or confusion, improving interactions by responding appropriately [38,39]. This would enhance the user experience and contribute to improved service quality. Notably, there is significant potential for integrating *AffectStream* into systems designed for education. Previous studies have shown that emotion recognition software in education can maximize learning outcomes by adjusting the difficulty of the material or providing encouragement based on the student's emotional state [4,8,40].

Table 5
Integration plan for various domains.

Domain	Methods	Key challenges
Education	API connection to LMS, WebSocket for real-time dashboard updates	Classroom dynamics tolerance, supporting variable network conditions in remote learning
Healthcare	FHIR integration with EMRs, secure VPN channel	HIPAA/GDPR compliance, maintaining data integrity during asynchronous updates
Customer Service	Kafka-to-CRM pipeline, integration with chatbot frameworks	Handling peak-hour traffic spikes, maintaining low response time during high concurrency
Psychological Therapy	Secure connection for remote sessions, emotion-aware feedback API	Ensuring stable multimodal sensing, privacy preservation in sensitive contexts

By combining such software with *AffectStream*, which monitors real-time changes in a student's emotional state using multimodal data, it is possible to meet individual emotional needs and create a more effective learning environment. This approach is beneficial in military and medical training, where trainees often practice in simulated high-stress conditions [5,9]. With *AffectStream*, trainers can adjust the training process in response to trainees' emotional states, providing just-in-time interventions to help them manage stress more effectively. Given the recent trend of actively introducing virtual and augmented reality into simulation training in these fields [41–43], systems for real-time affect recognition such as *AffectStream* are essential. Further, when combined with AI-driven human–robot interaction (HRI), it can offer more natural interactions [44,45].

To concretize the integration potential across domains and address domain-specific challenges, Table 5 outlines practical integration methods and considerations. For example, in education, *AffectStream* can link to learning management systems (LMS) via standardized APIs and stream real-time dashboards to adapt content to students' emotional states. In healthcare, it can connect to electronic medical records (EMRs) via FHIR [46] APIs and transmit biosignals securely to ensure regulatory compliance. In customer service, a Kafka-based pipeline can interface with customer relationship management (CRM) or chatbot systems for rapid detection and response to customer dissatisfaction. In psychological therapy, secure connections can support remote multimodal sensing and deliver emotion-aware feedback while preserving privacy.

5. Conclusions

We proposed *AffectStream*, a real-time affect monitoring system that uses a Kafka-based cloud infrastructure for large-scale affect analysis in real time. Notably, Kafka-based pub/sub architecture in cloud environments supports the real-time processing of large-scale user data through a high-performance distributed system, effectively detecting emotional states such as stress. We demonstrated the applicability of *AffectStream* for real-time affect recognition by conducting a trace-based evaluation with data from wearable sensors. *AffectStream* has the potential to deliver a more personalized user experience across fields such as education, healthcare, customer service, and military training, thereby enhancing learning efficiency, user satisfaction, and emotional responsiveness. To this end, future work should explore extending *AffectStream* to include additional modalities that are active in emotion detection research, such as audio and visual data. Systems should be developed to support these modalities (e.g., edge computing based on device processing to enhance scalability with a growing number of users and modalities). Furthermore, such a system is recommended to be validated in real-world environments.

CRediT authorship contribution statement

Jeonghyun Kim: Writing – original draft, Writing – review & editing, Conceptualization, Software. **Duri Lee:** Writing – review & editing, Supervision. **Uichin Lee:** Writing – review & editing, Supervision, Conceptualization, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korean government (MSIT) (No. 2022-0-00064, Development of Human Digital Twin Technologies for Prediction and Management of Emotion Workers' Mental Health Risks).

References

[1] Park CY, Cha N, Kang S, Kim A, Khandoker AH, Hadjileontiadis L, et al. K-emocon, a multimodal sensor dataset for continuous emotion recognition in naturalistic conversations. Sci Data 2020;7(1):293.
[2] Hovsepian K, Al'Absi M, Ertin E, Kamarck T, Nakajima M, Kumar S. Cstress: towards a gold standard for continuous stress assessment in the mobile environment. In: Proceedings of the 2015 ACM international joint conference on pervasive and ubiquitous computing. 2015, p. 493–504.
[3] Zhang P, Jung G, Alikhanov J, Ahmed U, Lee U. A reproducible stress prediction pipeline with mobile sensor data. Proc the ACM Interact Mob Wearable Ubiquitous Technol 2024;8(3):1–35.
[4] Sarmiento-Calisaya E, Ccori PC, Parari AC. An emotion-aware persuasive architecture to support challenging classroom situations. In: 2022 IEEE international conference on consumer electronics. IEEE; 2022, p. 1–2.
[5] Linssen L, Landman A, van Baardewijk JU, Bottenheft C, Binsch O. Using accelerometry and heart rate data for real-time monitoring of soldiers' stress in a dynamic military virtual reality scenario. Multimedia Tools Appl 2022;81(17):24739–56.
[6] Elvitigala DS, Scholl PM, Suriyaarachchi H, Dissanayake V, Nanayakkara S. Stressshoe: a diy toolkit for just-in-time personalised stress interventions for office workers performing sedentary tasks. In: Proceedings of the 23rd international conference on mobile human-computer interaction. 2021, p. 1–14.
[7] Rivera-Pelayo V, Fessl A, Müller L, Pammer V. Introducing mood self-tracking at work: Empirical insights from call centers. ACM Trans Computer-Human Interact (TOCHI) 2017;24(1):1–28.
[8] Ez-Zaouia M, Tabard A, Lavoué E. Emodash: A dashboard supporting retrospective awareness of emotions in online learning. Int J Hum-Comput Stud 2020;139:102411.
[9] Lai K, Yanushkevich SN, Shmerko VP. Intelligent stress monitoring assistant for first responders. IEEE Access 2021;9:25314–29.
[10] McDuff D, Rowan K, Choudhury P, Wolk J, Pham T, Czerwinski M. A multimodal emotion sensing platform for building emotion-aware applications. 2019, arXiv preprint arXiv:1903.12133.
[11] Choksi K, Chen H, Joshi K, Jade S, Nirjon S, Lin S. Sensemo: Enabling affective learning through real-time emotion recognition with smartwatches. 2024, arXiv preprint arXiv:2407.09911.
[12] Lohitha NS, Pounambal M. Integrated publish/subscribe and push-pull method for cloud based iot framework for real time data processing. Meas Sensors 2023;27:100699.
[13] Haque KN, Islam J, Ahmad I, Harjula E. Decentralized pub/sub architecture for real-time remote patient monitoring: A feasibility study. In: Nordic conference on digital health and wireless solutions. Springer; 2024, p. 48–65.
[14] Movesense. Wearable sensor — movesense. 2023, https://www.movesense.com/. (Accessed 11 August 2025).
[15] Maharjan R, Chy MSH, Arju MA, Cerny T. Benchmarking message queues. Telecom 2023;4:298–312.
[16] Chy MSH, Arju MAR, Tella SM, Cerny T. Comparative evaluation of java virtual machine-based message queue services: A study on kafka, artemis, pulsar, and rocketmq. Electronics 2023;12(23):4792.
[17] Foundation AS. Apache kafka. 2012, https://kafka.apache.org/, accessed .(Accessed 11 August 2025).
[18] RabbitM Q. Queues. 2025, https://www.rabbitmq.com/docs/queues. (Accessed 11 August 2025).

[19] Apache. Activemq. 2025, https://activemq.apache.org/components/classic/documentation/how-do-i-preserve-order-of-messages .(Accessed 11 August 2025).
[20] Amazon. Amazon web service (aws). 2024, https://aws.amazon.com. (Accessed 11 August 2025).
[21] Wu H, Shang Z, Wolter K. Performance prediction for the apache kafka messaging system. In: 2019 IEEE 21st international conference on high performance computing and communications; IEEE 17th international conference on smart city; IEEE 5th international conference on data science and systems (HPCC/smartCity/DSS). IEEE; 2019, p. 154–61.
[22] George J. Build a realtime data pipeline: Scalable application data analytics on amazon web services (aws). JETIR 2024.
[23] Boscain S. Aws cloud: infrastructure, devops techniques, state of art (Ph.D. thesis), 2023, Politecnico di Torino.
[24] Yurtay Y, Demirci H, Tiryaki H, Altun T. Emotion recognition on call center voice data. Appl Sci 2024;14(20):9458.
[25] Hernandez J, Morris RR, Picard RW. Call center stress recognition with person-specific models. In: Affective computing and intelligent interaction: 4th international conference, ACII 2011, memphis, TN, USA, October (2011) 9–12, proceedings, part i 4. Springer; 2011, p. 125–34.
[26] Bromuri S, Henkel AP, Iren D, Urovi V. Using ai to predict service agent stress from emotion patterns in service interactions. J Serv Manag 2021;32(4):581–611.
[27] Howe E, Suh J, Bin Morshed M, McDuff D, Rowan K, Hernandez J, et al. Design of digital workplace stress-reduction intervention systems: Effects of intervention type and timing. In: Proceedings of the 2022 CHI conference on human factors in computing systems, association for computing machinery, new york, NY, USA. 2022, p. 1–16.
[28] Nahum-Shani I, Smith SN, Spring BJ, Collins LM, Witkiewitz K, Tewari A, et al. Just-in-time adaptive interventions (jitais) in mobile health: key components and design principles for ongoing health behavior support. Ann Behav Med 2018;52(6):446–62.
[29] Sano A, Johns P, Czerwinski M. Designing opportune stress intervention delivery timing using multi-modal data. In: 2017 seventh international conference on affective computing and intelligent interaction. IEEE; 2017, p. 346–53.
[30] Neupane S, Saha M, Ali N, Hnat T, Samiei SA, Nandugudi A, et al. Momentary stressor logging and reflective visualizations: Implications for stress management with wearables. In: Proceedings of the CHI conference on human factors in computing systems, association for computing machinery, new york, NY, USA. 2024, p. 1–19.
[31] Raptis TP, Passarella A. On efficiently partitioning a topic in apache kafka. In: 2022 international conference on computer, information and telecommunication systems. IEEE; 2022, p. 1–8.
[32] Schmidt P, Reiss A, Duerichen R, Marberger C, Van Laerhoven K, wesad Introducing. A multimodal dataset for wearable stress and affect detection. In: Proceedings of the 20th ACM international conference on multimodal interaction. 2018, p. 400–8.
[33] Miranda-Correa JA, Abadi MK, Sebe N, Patras I. Amigos: A dataset for affect, personality and mood research on individuals and groups. IEEE Trans Affect Comput 2018;12(2):479–93.
[34] Park S, Zheng D, Lee U. A ppg signal dataset collected in semi-naturalistic settings using galaxy watch. Sci Data 2025;12(1):892.
[35] Locust. Locust: An open source load testing tool. 2020, https://locust.io. (Accessed 11 August 2025).
[36] Datar M, Gionis A, Indyk P, Motwani R. Maintaining stream statistics over sliding windows. SIAM J Comput 2002;31(6):1794–813.
[37] Thomas MU, systems Queueing. Volume 1: Theory (leonard kleinrock). SIAM Rev 1976;18(3):512–4.
[38] Liu C, Agrawal P, Sarkar N, Chen S. Dynamic difficulty adjustment in computer games through real-time anxiety-based affective feedback. Int J Hum-Comput Interact 2009;25(6):506–29.
[39] Henkel AP, Bromuri S, Iren D, Urovi V, human Half. Half machine–augmenting service employees with ai for interpersonal emotion regulation. J Serv Manag 2020;31(2):247–65.
[40] Wu C-H, Huang Y-M, Hwang J-P. Review of affective computing in education/learning: Trends and challenges. Br J Educ Technol 2016;47(6):1304–23.
[41] Onu P, Pradhan A, Mbohwa C. Potential to use metaverse for future teaching and learning. Educ Inf Technol 2024;29(7):8893–924.
[42] Kuleto V, Ilić MP, Ranković M, Radaković M, Simović A. Augmented and virtual reality in the metaverse context: The impact on the future of work, education, and social interaction. In: Augmented and virtual reality in the metaverse. Springer; 2024, p. 3–24.
[43] Yu D. Designing effective learning environments in the educational metaverse: The role of augmented and virtual reality. In: Augmented and virtual reality in the metaverse. Springer; 2024, p. 81–100.
[44] Ottoni LTC, d. J. F. Cerqueira J. A systematic review of human–robot interaction: The use of emotions and the evaluation of their performance. Int J Soc Robot 2024;1–20.
[45] Obaigbena A, Lottu OA, Ugwuanyi ED, Jacks BS, Sodiya EO, Daraojimba OD. Ai and human–robot interaction: A review of recent advances and challenges. GSC Adv Res Rev 2024;18(2):321–30.
[46] HL7. Hl7 fhir v5.0.0. 2023, https://hl7.org/fhir/. (Accessed 11 August 2025).