AffectStream: Kafka-based Real-time Affect Monitoring System using Wearable Sensors

Jeonghyun Kim[a], Duri Lee[b], Uichin Lee[c,*]

[a]*KAIST, 291 Daehak-ro, Yuseong-gu, Daejeon 34141, Republic of Korea, jeonghyun.kim@kaist.ac.kr*
[b]*KAIST, 291 Daehak-ro, Yuseong-gu, Daejeon 34141, Republic of Korea, duri.lee@kaist.ac.kr*
[c]*KAIST, 291 Daehak-ro, Yuseong-gu, Daejeon 34141, Republic of Korea, uclee@kaist.ac.kr*

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 the WESAD dataset. This open-source framework advances real-time emotion recognition, paving the way for large-scale affective computing applications.

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

Metadata

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

*Corresponding author

Nr.	Code metadata description	Metadata
C1	Current code version	1.0
C2	Permanent link to code/repository used for this code version	`https://github.com/Kaist-ICLab/AffectStream`
C3	Permanent link to Reproducible Capsule	-
C4	Legal Code License	MIT License
C5	Code versioning system used	None
C6	Software code languages, tools, and services used	Python, Java, Amazon Web Service, Terraform, Locust, Docker, SQL.
C7	Compilation requirements, operating environments & dependencies	Debian (Slim), Python 3.9.16, OpenJDK 21 (Early Access, Build 11), other requirements provided in requirements.txt
C8	If available Link to developer documentation/manual	-
C9	Support email for questions	jeonghyun.kim@kaist.ac.kr

Table 1: Code metadata.

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 [9, 5].

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 Figure 1) [10, 11]. First, in the data acquisition stage, user data is collected in real time from one or multiple wearable

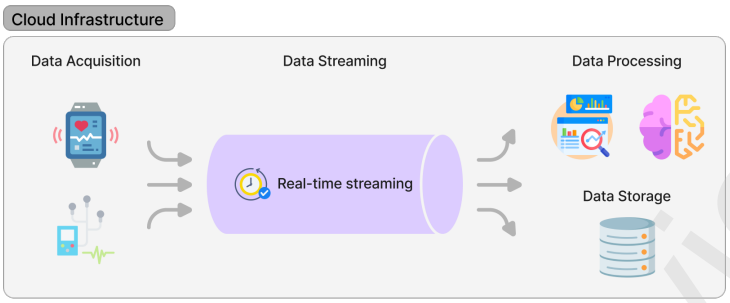

Figure 1: Four stages of real-time affect monitoring system.

devices (e.g., smartwatch and chest band). Second, during the data streaming stage, large volumes of sensor data are transmitted and processed in real time, requiring mechanisms to ensure data ordering and integrity. Third, 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. Lastly, 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 further analysis and 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 is a message broker based on the Advanced Message Queuing Protocol (AMQP) that excels in reliable message queuing and complex routing. Although RabbitMQ supports pub/sub messaging, its queuing system architecture limits high-frequency, high-volume continuous data processing [15]. ActiveMQ is 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. In contrast, Kafka is a *distributed streaming* platform optimized for high throughput and scalability, particularly for real-time data

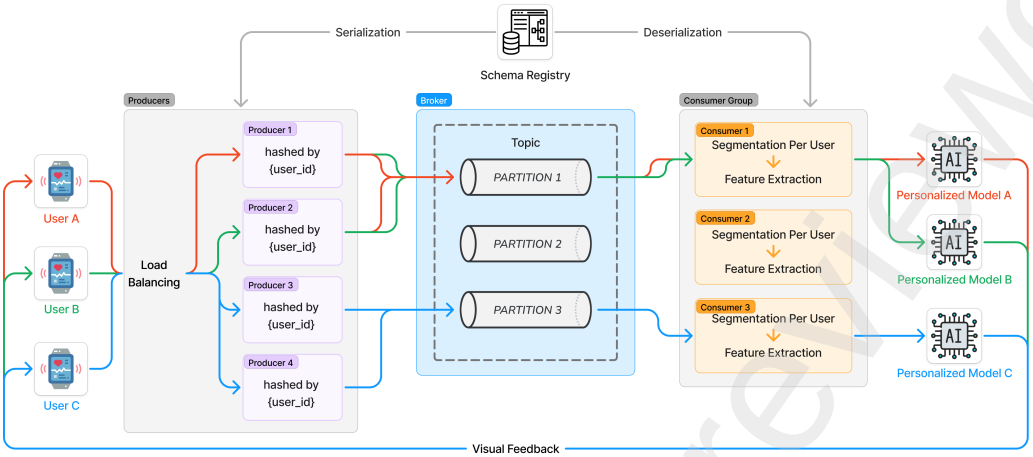

Figure 2: The architecture of *AffectStream*.

streaming and log processing [16]. Notably, Kafka supports excellent real-time throughput and provides superior performance in environments that require large-scale real-time data processing and streaming [15].

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. This system 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 simultaneously engage with the system. 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. Applying *AffectStream* in various fields such as education, healthcare, customer service, and psychological therapy, we can understand the emotional states of users 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 Figure 2. In this section, we provide an overview of *AffectStream* and briefly

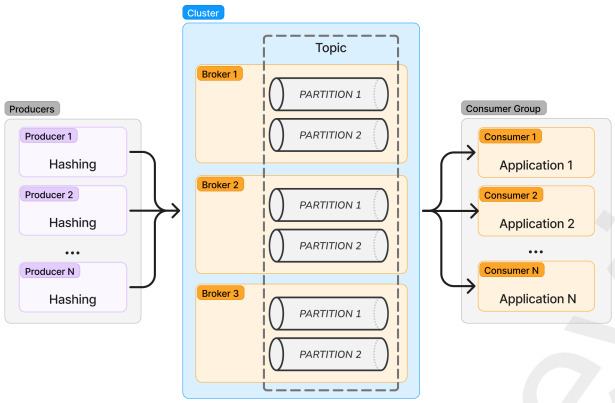

Figure 3: The architecture of Kafka.

describe the functionalities that the system supports. For detailed explanations, please refer to the technical documents in 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 a more flexible system in case of an increase in 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. Figure 3 illustrates the basic structure and Table 2 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 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.

Term	Description
Broker	A Kafka server that stores data and serves producers and consumers.
Topic	A category or feed name to which records are published.
Partition	A division of a topic's log, enabling parallel processing.
Producer	A client that publishes records to one or more Kafka topics.
Consumer	A client that reads records from one or more 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 multiple brokers for fault tolerance.
Throughput	The amount of data processed in a given time period.
Latency	The time it takes for a record to travel from producer to consumer.
Stream Processing	The real-time processing of data continuously from a Kafka topic.

Table 2: Kafka terminology.

The operation of Kafka-based *AffectStream* is as follows (see Figure 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, significantly 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 sensing 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 sensing data fields, making it easy to modify if the types of sensing 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 evenly distribute user data and manage traffic. 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.

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 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) [17]. 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 [18, 19, 20]. The producer was implemented based on a Spring-based API server that handles data transmission to the broker.

3. Illustrative examples

As an example use case, we selected a scenario where *AffectStream* is implemented to detect stress [21, 22, 23] of call agents at a typical call center with around 100 employees. We set up the evaluation environment using a widely used dataset for stress detection, and the system was tested for real-time stress detection in this scenario.

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

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 3 summarizes the system configuration used for trace-based evaluation. Similar to a prior work [24], 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 dataset. We used the WESAD wearable dataset [25], which includes multimodal physiological sensor data such as acceleration, temperature, muscle activity, skin conductance, heart activity, and respiration. These data points were collected at 700 Hz during stress-inducing tasks to classify affective states like neutral, stress, and amusement. Since *AffectStream* is designed to detect stress for individual users, we added a unique user identifier (UUID) to each data record. The original WESAD dataset does not contain any user-specific identifiers, so we generated UUIDs to distinguish data from different users. 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.

Sensor data generator. Figure 4 illustrates a simulation implemented using Locust [26], 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

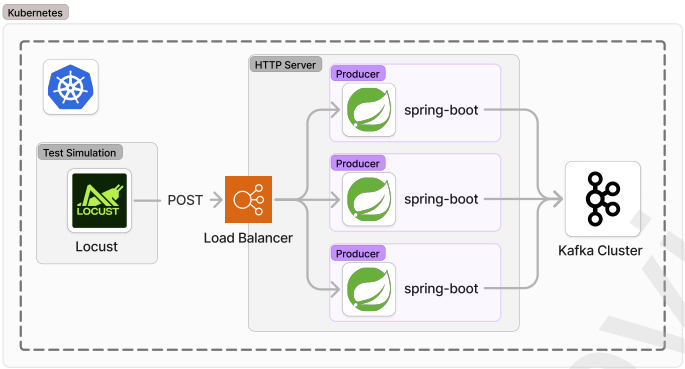

Figure 4: Trace-based evaluation setup for load testing.

per second. Sensor data was transmitted at a rate of 700 Hz with each segment spanning 1 second to the producer via POST.

Consumer of AffectStream. A sliding window [27] was applied to perform feature extraction in the consumers of *AffectStream*, with a window size of 2 seconds and an overlap of 1 second.

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.

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

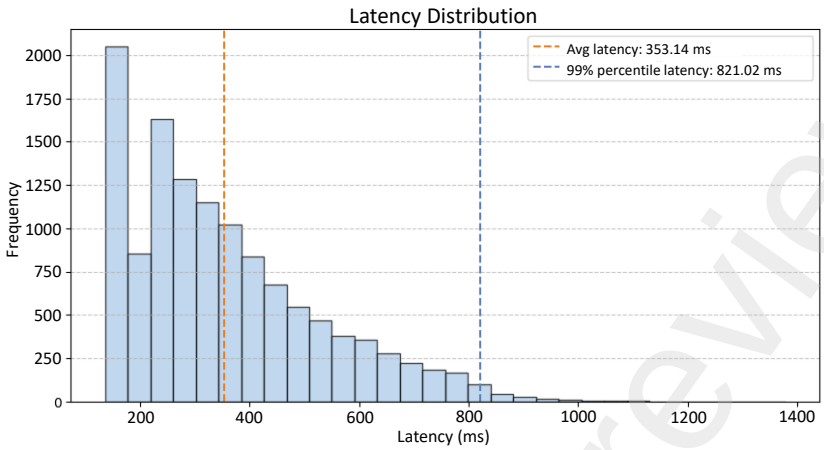

Figure 5: End-to-end latency distribution.

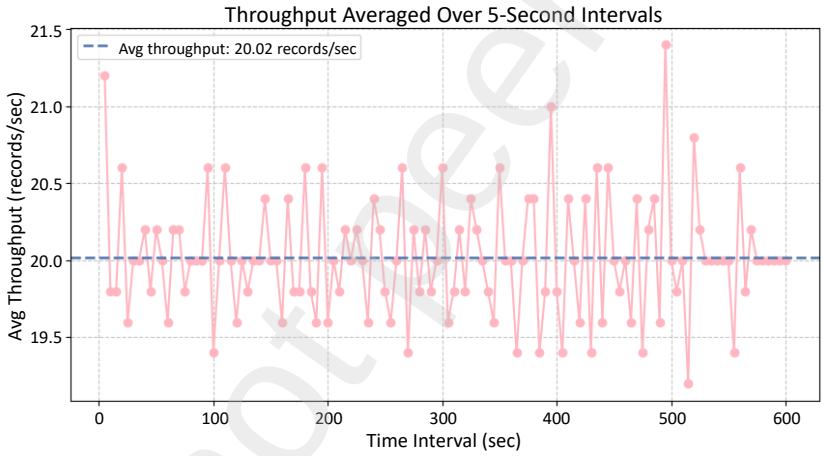

Figure 6: Throughput averaged over 5-second intervals.

generation in the simulator to classification completion in the consumer, includes transmission through the Producer-Broker-Consumer pipeline. 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. Figure 5 shows a mean latency of 55.89 ms and a 99th percentile of 141.00 ms, verifying low-latency operation. Figure 6 confirms no data loss, as it matches Locust's 20 users/s spawn rate, demonstrating robustness and scalability.

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.

4. Impact

AffectStream, a system that monitors users' affect state in real time using wearable sensor data, can be used to realize affective computing applications in various domains such as education, military training, healthcare, and empowerment at workplaces. 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 [28, 29]. 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 and training. 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, 30]. 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 particularly useful in military and medical training, where trainees often practice in simulated high-stress conditions [9, 5]. 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 [31, 32, 33], 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 [34, 35].

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 enables the processing of large-scale user data in real time thanks to a high-performance distributed system architecture, 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.

CRediT authorship contribution statement

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

Acknowledgements

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] C. Y. Park, N. Cha, S. Kang, A. Kim, A. H. Khandoker, L. Hadjileontiadis, A. Oh, Y. Jeong, U. Lee, K-emocon, a multimodal sensor dataset for continuous emotion recognition in naturalistic conversations, Scientific Data 7 (1) (2020) 293.

[2] K. Hovsepian, M. Al'Absi, E. Ertin, T. Kamarck, M. Nakajima, S. Kumar, 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, pp. 493–504.

[3] P. Zhang, G. Jung, J. Alikhanov, U. Ahmed, U. Lee, A reproducible stress prediction pipeline with mobile sensor data, Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies 8 (3) (2024) 1–35.

[4] E. Sarmiento-Calisaya, P. C. Ccori, A. C. Parari, An emotion-aware persuasive architecture to support challenging classroom situations, in: 2022 IEEE International Conference on Consumer Electronics (ICCE), IEEE, 2022, pp. 1–2.

[5] L. Linssen, A. Landman, J. U. van Baardewijk, C. Bottenheft, O. Binsch, Using accelerometry and heart rate data for real-time monitoring of soldiers' stress in a dynamic military virtual reality scenario, Multimedia Tools and Applications 81 (17) (2022) 24739–24756.

[6] D. S. Elvitigala, P. M. Scholl, H. Suriyaarachchi, V. Dissanayake, S. Nanayakkara, 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, pp. 1–14.

[7] V. Rivera-Pelayo, A. Fessl, L. Müller, V. Pammer, Introducing mood self-tracking at work: Empirical insights from call centers, ACM Transactions on Computer-Human Interaction (TOCHI) 24 (1) (2017) 1–28.

[8] M. Ez-Zaouia, A. Tabard, E. Lavoué, Emodash: A dashboard supporting retrospective awareness of emotions in online learning, International Journal of Human-Computer Studies 139 (2020) 102411.

[9] K. Lai, S. N. Yanushkevich, V. P. Shmerko, Intelligent stress monitoring assistant for first responders, IEEE Access 9 (2021) 25314–25329.

[10] D. McDuff, K. Rowan, P. Choudhury, J. Wolk, T. Pham, M. Czerwinski, A multimodal emotion sensing platform for building emotion-aware applications. arxiv 2019, arXiv preprint arXiv:1903.12133.

[11] K. Choksi, H. Chen, K. Joshi, S. Jade, S. Nirjon, S. Lin, Sensemo: Enabling affective learning through real-time emotion recognition with smartwatches, arXiv preprint arXiv:2407.09911 (2024).

[12] N. S. Lohitha, M. Pounambal, Integrated publish/subscribe and push-pull method for cloud based iot framework for real time data processing, Measurement: Sensors 27 (2023) 100699.

[13] K. N. Haque, J. Islam, I. Ahmad, E. Harjula, Decentralized pub/sub architecture for real-time remote patient monitoring: A feasibility study, in: Nordic Conference on Digital Health and Wireless Solutions, Springer, 2024, pp. 48–65.

[14] Movesense, Wearable sensor — movesense, accessed March 17, 2025(2023).
URL https://www.movesense.com/

[15] R. Maharjan, M. S. H. Chy, M. A. Arju, T. Cerny, Benchmarking message queues, in: Telecom, Vol. 4, MDPI, 2023, pp. 298–312.

[16] A. S. Foundation, Apache kafka, accessed March 17, 2025(2012).
URL https://kafka.apache.org/

[17] Amazon, Amazon web service (aws), accessed March 17, 2025(2024).
URL https://aws.amazon.com

[18] H. Wu, Z. Shang, K. Wolter, 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, pp. 154–161.

[19] J. George, Build a realtime data pipeline: Scalable application data analytics on amazon web services (aws) (2024).

[20] S. Boscain, Aws cloud: Infrastructure, devops techniques, state of art., Ph.D. thesis, Politecnico di Torino (2023).

[21] Y. Yurtay, H. Demirci, H. Tiryaki, T. Altun, Emotion recognition on call center voice data, Applied Sciences 14 (20) (2024) 9458.

[22] J. Hernandez, R. R. Morris, R. W. Picard, Call center stress recognition with person-specific models, in: Affective Computing and Intelligent Interaction: 4th International Conference, ACII 2011, Memphis, TN, USA, October 9–12, 2011, Proceedings, Part I 4, Springer, 2011, pp. 125–134.

[23] S. Bromuri, A. P. Henkel, D. Iren, V. Urovi, Using ai to predict service agent stress from emotion patterns in service interactions, Journal of Service Management 32 (4) (2021) 581–611.

[24] T. P. Raptis, A. Passarella, On efficiently partitioning a topic in apache kafka, in: 2022 International Conference on Computer, Information and Telecommunication Systems (CITS), IEEE, 2022, pp. 1–8.

[25] P. Schmidt, A. Reiss, R. Duerichen, C. Marberger, K. Van Laerhoven, Introducing wesad, a multimodal dataset for wearable stress and affect detection, in: Proceedings of the 20th ACM international conference on multimodal interaction, 2018, pp. 400–408.

[26] Locust, Locust: An open source load testing tool, accessed March 17, 2025(2020).
URL `https://locust.io`

[27] M. Datar, A. Gionis, P. Indyk, R. Motwani, Maintaining stream statistics over sliding windows, SIAM journal on computing 31 (6) (2002) 1794–1813.

[28] C. Liu, P. Agrawal, N. Sarkar, S. Chen, Dynamic difficulty adjustment in computer games through real-time anxiety-based affective feedback, International Journal of Human-Computer Interaction 25 (6) (2009) 506–529.

[29] A. P. Henkel, S. Bromuri, D. Iren, V. Urovi, Half human, half machine–augmenting service employees with ai for interpersonal emotion regulation, Journal of Service Management 31 (2) (2020) 247–265.

[30] C.-H. Wu, Y.-M. Huang, J.-P. Hwang, Review of affective computing in education/learning: Trends and challenges, British Journal of Educational Technology 47 (6) (2016) 1304–1323.

[31] P. Onu, A. Pradhan, C. Mbohwa, Potential to use metaverse for future teaching and learning, Education and Information Technologies 29 (7) (2024) 8893–8924.

[32] V. Kuleto, M. P. Ilić, M. Ranković, M. Radaković, A. Simović, 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, pp. 3–24.

[33] D. Yu, 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, pp. 81–100.

[34] L. T. C. Ottoni, J. d. J. F. Cerqueira, A systematic review of human–robot interaction: The use of emotions and the evaluation of their performance, International Journal of Social Robotics (2024) 1–20.

[35] A. Obaigbena, O. A. Lottu, E. D. Ugwuanyi, B. S. Jacks, E. O. Sodiya, O. D. Daraojimba, Ai and human-robot interaction: A review of recent advances and challenges, GSC Advanced Research and Reviews 18 (2) (2024) 321–330.