DOI 10.54596/2958-0048-2025-2-194-206
UDK 621.867
IRSTI 45.53.00

SCALABLE PIPELINES FOR INSTANT OBJECT DETECTION: DEPLOYING YOLO MODELS WITH APACHE KAFKA FOR HIGH-THROUGHPUT INFERENCE

Ismail Oztel[1,2], Celal Ceken[1,3*]
[1]Sakarya University, Department of Computer Engineering, Faculty of Computer and Information Sciences, Sakarya, Türkiye*
[2] Sakarya University, Intelligent Software Systems Research Lab, Sakarya, Türkiye
[3]Manash Kozybayev North Kazakhstan University NPLC, International Campus Petropavlovsk, Kazakhstan*
[*]Corresponding author: celalceken@sakarya.edu.tr

Abstract

Instant object detection is a critical capability in modern applications where timely decision-making is essential, such as in emergency medicine, autonomous systems, and intelligent surveillance. Efficiently handling high-throughput image streams with minimal delay presents significant challenges, particularly under demanding conditions. This study presents a scalable object detection pipeline that integrates YOLO models with Apache Kafka, a distributed streaming platform, to support just-in-time inference. The proposed architecture leverages Kafka's partitioning and consumer group mechanisms to enable parallel processing, ensuring high throughput without requiring complex load-balancing logic. The system is deployed on a Virtual Private Server to demonstrate practical implementation. Two configurations are presented to illustrate Kafka's native scalability: one with a single partition and a single consumer, and another with five partitions and five consumers. These setups visually demonstrate how Kafka efficiently distributes workloads across multiple consumers. Although specific latency or throughput metrics are not reported, the architecture effectively showcases how Kafka's design enables prompt responses to high-volume input. This pipeline is well-suited for time-sensitive object detection tasks and can be extended to a wide range of instant analytics applications where rapid feedback is critical.

Keywords: Instant object detection, High-Throughput Inference, Apache Kafka, YOLO model, Scalable stream processing.

ЛЕЗДІК ОБЪЕКТІНІ АНЫҚТАУҒА АРНАЛҒАН МАСШТАБТАЛАТЫН КОНВЕЙЕРЛЕР: YOLO МОДЕЛЬДЕРІН APACHE KAFKA АРҚЫЛЫ ЖОҒАРЫ ӨТКІЗУ ҚАБІЛЕТТІ ИНФЕРЕНС ҮШІН ЕНДІРУ

Ісмаил Өзтел[1, 2], Желял Чекен[1, 3*]
[1]Сакария университеті, Компьютерлік инженерия кафедрасы, Компьютер және ақпараттық ғылымдар факультеті, Сакария, Түркия*
[2] Сакария университеті, Зияткерлік бағдарламалық жүйелер зертханасы Сакария, Түркия
[3]«Манаш Қозыбаев атындағы Солтүстік Қазақстан университеті» КеАҚ Халықаралық кампус, Петропавл, Қазақстан*
[*]Хат-хабар үшін автор: celalceken@sakarya.edu.tr

Аңдатпа

Лездік объектіні анықтау – шұғыл шешім қабылдау маңызды болатын қазіргі заманғы қосымшаларда, мысалы, жедел медициналық көмек, автономды жүйелер және интеллектуалды бақылау салаларында аса маңызды мүмкіндік. Жоғары өткізу қабілеті бар кескін ағындарын кідіріссіз тиімді өңдеу – әсіресе күрделі жағдайларда – үлкен қиындықтар туғызады. Бұл зерттеуде YOLO модельдерін Apache Kafka ағындық платформасымен біріктіретін, масштабталатын объектіні анықтау конвейері ұсынылады. Ұсынылған архитектура Kafka-ның бөлшектеу (partitioning) және тұтынушылар топтары (consumer groups)

механизмдерін пайдалана отырып, параллель өңдеуді қамтамасыз етеді және күрделі жүктеме теңгеру логикасын қажет етпейді. Жүйе нақты іске асыруды көрсету үшін Виртуалды Жеке Серверде (VPS) орналастырылған. Kafka-ның табиғи масштабталуын көрсету үшін екі конфигурация ұсынылады: біреуі – бір бөліктен және бір тұтынушыдан тұрады, ал екіншісі – бес бөліктен және бес тұтынушыдан тұрады. Бұл конфигурациялар Kafka-ның жұмыс жүктемесін бірнеше тұтынушыға қалай тиімді бөлетінін көрнекі түрде көрсетеді. Нақты кідіріс немесе өткізу қабілеті метрикалары көрсетілмегенімен, бұл архитектура Kafka-ның құрылымы жоғары көлемді деректерге жедел жауап қайтаруға қалай мүмкіндік беретінін тиімді түрде көрсетеді. Бұл конвейер уақытқа сезімтал объектіні анықтау тапсырмаларына өте қолайлы және жедел кері байланысты қажет ететін лездік аналитикалық қолданбаларға кеңейтілуі мүмкін.

Ключевые слова: Лездік объектіні анықтау, Жоғары өткізу қабілетті инференс, Apache Kafka, YOLO моделі, Масштабталатын ағынды өңдеу.

МАСШТАБИРУЕМЫЕ КОНВЕЙЕРЫ ДЛЯ МГНОВЕННОГО ОБНАРУЖЕНИЯ ОБЪЕКТОВ: РАЗВЕРТЫВАНИЕ МОДЕЛЕЙ YOLO С APACHE KAFKA ДЛЯ ВЫСОКОПРОИЗВОДИТЕЛЬНОГО ИНФЕРЕНСА

Исмаил Озтел[1, 2], Джелял Чекен[1, 3*]

[1]Университет Сакарьи, Кафедра компьютерной инженерии,*
Факультет компьютерных и информационных наук, Сакарья, Турция
[2]Университет Сакарьи, Лаборатория исследований интеллектуальных программных систем, Сакарья, Турция
[3]НАО «Северо-Казахстанский университет имени Манаша Козыбаева»*
Международный кампус, Петропавловск, Казахстан
[*]Автор для корреспонденции: celalceken@sakarya.edu.tr

Аннотация

Мгновенное обнаружение объектов является критически важной возможностью в современных приложениях, где необходимо оперативное принятие решений, например, в неотложной медицине, автономных системах и интеллектуальном видеонаблюдении. Эффективная обработка потоков изображений с высокой пропускной способностью и минимальной задержкой представляет собой значительную задачу, особенно в условиях высокой нагрузки. В данном исследовании представлена масштабируемая конвейерная архитектура для обнаружения объектов, интегрирующая модели YOLO с Apache Kafka – распределённой потоковой платформой, поддерживающей инференс в режиме just-in-time. Предлагаемая архитектура использует механизмы партиционирования и групп потребителей Kafka для обеспечения параллельной обработки, что позволяет достичь высокой производительности без необходимости в сложной логике балансировки нагрузки. Система развернута на виртуальном частном сервере (VPS) для демонстрации практической реализации. Представлены две конфигурации для иллюстрации масштабируемости Kafka: одна с одним разделом и одним потребителем, и другая с пятью разделами и пятью потребителями. Эти настройки наглядно демонстрируют, как Kafka эффективно распределяет рабочую нагрузку между несколькими потребителями. Хотя конкретные показатели задержки или пропускной способности не приведены, архитектура эффективно демонстрирует, как дизайн Kafka обеспечивает оперативную реакцию на вход с высоким объемом. Этот конвейер хорошо подходит для задач обнаружения объектов, чувствительных ко времени, и может быть расширен для широкого круга приложений моментальной аналитики, где критически важна быстрая обратная связь.

Ключевые слова: Мгновенное обнаружение объектов, Высокопроизводительный инференс, Apache Kafka, Модель YOLO, Масштабируемая потоковая обработка.

Introduction

Instant object detection has become a pivotal capability in numerous real-world systems where time-sensitive decisions must be made based on high-velocity image or video streams. Such systems are increasingly employed in emergency medicine (e.g., identifying abnormalities in radiological scans in emergency rooms), autonomous driving, smart

manufacturing, intelligent surveillance, unmanned aerial vehicles, and robotic vision applications, etc. In all these domains, detection pipelines must not only deliver high accuracy but also respond to a continuous and often unpredictable flow of visual input with minimal delay.

While deep learning models such as the YOLO (You Only Look Once) family have achieved significant success in object detection tasks due to their balance of speed and accuracy [1], integrating such models into scalable, responsive pipelines remains challenging. This is especially true when the objective is to maintain responsiveness under high-throughput conditions without relying on resource-intensive orchestration or complex load-balancing mechanisms.

Prior work has explored different strategies for deploying deep learning inference at scale. For example, Redmon et al. [1] introduced the original YOLO architecture with real-time detection capabilities, which has since evolved through multiple versions emphasizing speed and deployment efficiency. Han et al. [2] proposed a cloud-based object detection service using Apache Storm for real-time stream processing, though the system exhibited scalability limitations under increased load. Similarly, Shi et al. [3] presented a real-time pedestrian detection framework integrated with Apache Flink, focusing on processing performance but without emphasizing system deployment simplicity or adaptability. Although these systems successfully address aspects of responsiveness or scalability, they often require extensive configuration or fail to exploit native streaming platform features that facilitate parallelism.

In contrast, this study presents an instant object detection pipeline that combines the efficiency of YOLO-based models with the distributed streaming capabilities of Apache Kafka. Kafka offers a topic-partitioning and consumer group mechanism that inherently supports parallelism and load balancing without introducing additional middleware complexity. This makes it an ideal choice for building scalable and responsive inference pipelines.

The main contributions of this study are as follows:

- A scalable pipeline architecture that integrates YOLO inference with Apache Kafka, supporting high-throughput image stream processing with minimal latency.
- A practical implementation and deployment scenario using a Virtual Private Server (VPS), demonstrating real-world applicability and ease of setup.
- A demonstration of system behavior under two Kafka configurations—single partition with a single consumer, and five partitions with five consumers—highlighting the platform's native parallel processing capabilities.
- A modular design that can be extended to support various instant decision-making applications, particularly those requiring just-in-time analytics without complex orchestration layers.

Through this architecture, the study aims to offer a lightweight and adaptable solution for time-critical detection tasks, emphasizing just-in-time responsiveness over raw computational performance metrics.

The remainder of this paper is organized as follows: Section 2 describes the overall architecture of the proposed instant object detection pipeline, detailing its key components including the Kafka cluster, YOLO inference engine, and data streaming mechanism. Section 3 outlines the development and deployment process, emphasizing practical implementation on a VPS. Section 4 presents and discusses the results, with a particular focus on illustrating how Kafka's native parallelism contributes to scalable inference. Finally, Section 5 concludes the study and suggests possible directions for future work.

System Architecture

The proposed system architecture is designed to support instant object detection over image streams, with scalability and deployment simplicity as primary design goals. It is composed of modular components that collectively enable distributed data ingestion, parallel model inference, and instant result dissemination. The architecture and interaction between the core components are illustrated in Figure 1, including the data analytics engine, Apache Kafka cluster, data producers, parallel consumers, and the web server.

At the center of the architecture is an Apache Kafka cluster, which serves as the backbone for data streaming and coordination. Kafka topics are configured with a user-defined number of partitions, enabling message-level parallelism and horizontal scalability. Each partition can be consumed independently by separate inference workers, allowing the system to process multiple image streams concurrently.

Data producers are responsible for acquiring and publishing image frames to Kafka topics. These producers may include video capture nodes, camera gateways, or external systems that encode and forward visual data for detection. The decoupling of data producers from consumers ensures that frame generation and processing rates do not have to be synchronized, which improves system robustness under burst traffic conditions.

Parallel Kafka consumers, which host the YOLO inference engine, subscribe to the input topic and perform object detection on incoming frames. Each consumer instance is assigned one or more partitions depending on the deployment configuration. The inference engine is stateless and designed to be lightweight, facilitating easy replication across multiple nodes or containers.

The web application facilitates the ingestion of image data into the data analytics engine via Kafka and also acts as a consumer to receive inference outputs from the backend. As illustrated in Figure 1, this inference data is consumed by the web application and can then be transmitted to the user interface using real-time communication technologies such as WebSockets or Server-Sent Events. This enables immediate visualization and supports timely decision-making processes. By centralizing communication through the web cluster, the system ensures seamless integration between Kafka streaming components and downstream analytics or visualization services, maintaining efficient data flow and real-time accessibility of detection results.

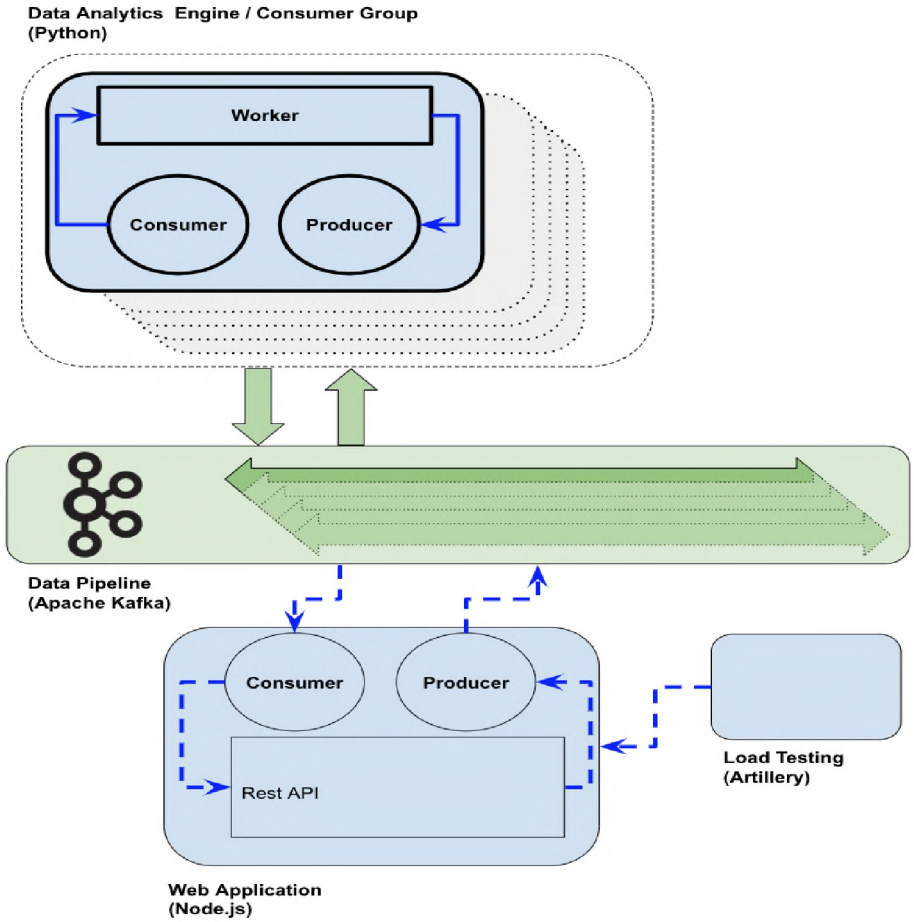

Figure 1. Main components of the proposed scalable object detection pipeline

This modular architecture offers several benefits: it simplifies scalability through Kafka's native consumer group mechanism; isolates each stage of the data pipeline for independent development and deployment; and supports extensibility by allowing additional consumers or analytics modules to be integrated without disrupting existing operations.

Kafka Cluster

The Kafka cluster, illustrated in Figure 2, constitutes the core streaming infrastructure of the proposed object detection pipeline. Kafka is a distributed messaging platform composed of four principal components [4]:

- Producer: An entity that generates and publishes messages to Kafka topics. In this pipeline, producers are responsible for sending image frames to designated topics.
- Consumer: A client that subscribes to Kafka topics and processes the received messages. Consumers in this system execute the YOLO inference engine on the image data.
- Broker: The Kafka server responsible for receiving messages from producers, persisting them to disk, and serving them to consumers on request. A Kafka cluster comprises multiple brokers to distribute workload and enhance reliability.
- Zookeeper: A coordination service that manages the Kafka cluster metadata, including broker membership, topic configuration, and partition leader election. Zookeeper ensures the cluster's consistency and fault tolerance by monitoring broker health and orchestrating failover procedures.

Kafka organizes messages into topics, each of which is subdivided into partitions to facilitate parallelism. Producers append messages to specific partitions, while consumers independently retrieve and process messages from these partitions. This design supports concurrent message consumption and aligns with scalable streaming paradigms.

The Kafka cluster architecture is designed to provide scalability, fault tolerance, and high availability. Data streams are distributed across multiple brokers, allowing the system to handle large volumes of incoming data and providing redundancy against node failures. This horizontal scalability enables the addition of brokers to meet increasing workload demands without service disruption.

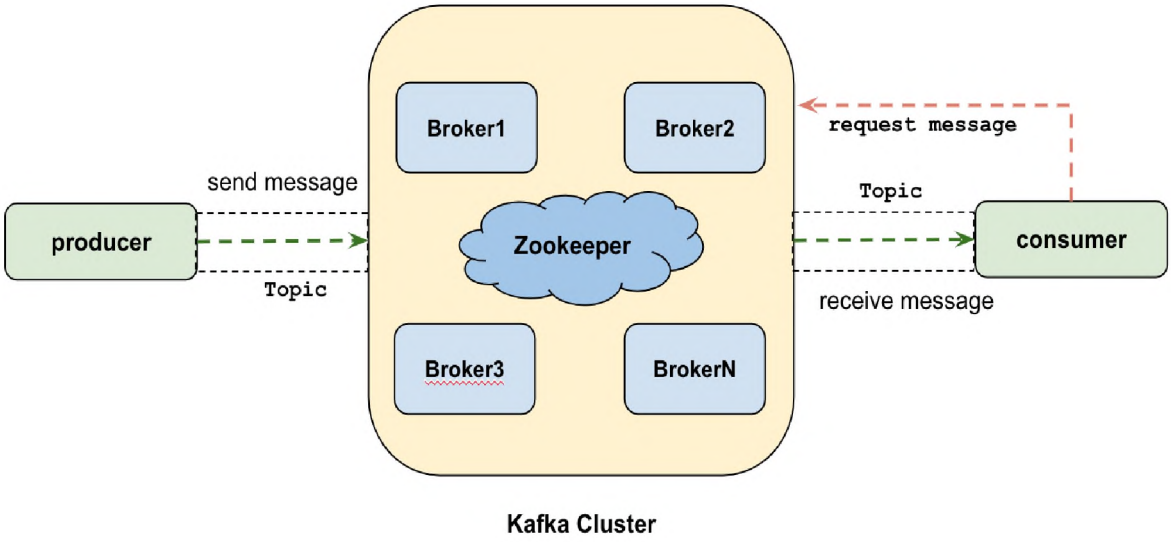

Figure 2. Schematic representation of the Kafka cluster architecture

Fault tolerance is primarily ensured through partition replication as depicted in Figure 3. Each partition is replicated across several brokers, forming multiple copies called replicas. One replica is designated as the leader, responsible for handling all read and write operations for that partition. In the event of a broker failure, Kafka automatically promotes one of the in-sync follower replicas (ISR) to leader status, maintaining continuous data availability and consistency. This replication mechanism underpins Kafka's reliability and ensures durable message delivery in distributed deployments. The replication factor must be less than or equal to the number of brokers in the Kafka cluster, since each replica must reside on a different broker to provide true fault tolerance.

In Figure 3, the replication factor is set to 3, meaning each partition has three copies distributed across the cluster to ensure fault tolerance and high availability. As shown in the figure, in the event of a failure in Kafka Broker 1, the replica of partition 0 on Kafka Broker 3 is promoted to Leader, allowing the system to continue operating without interruption.

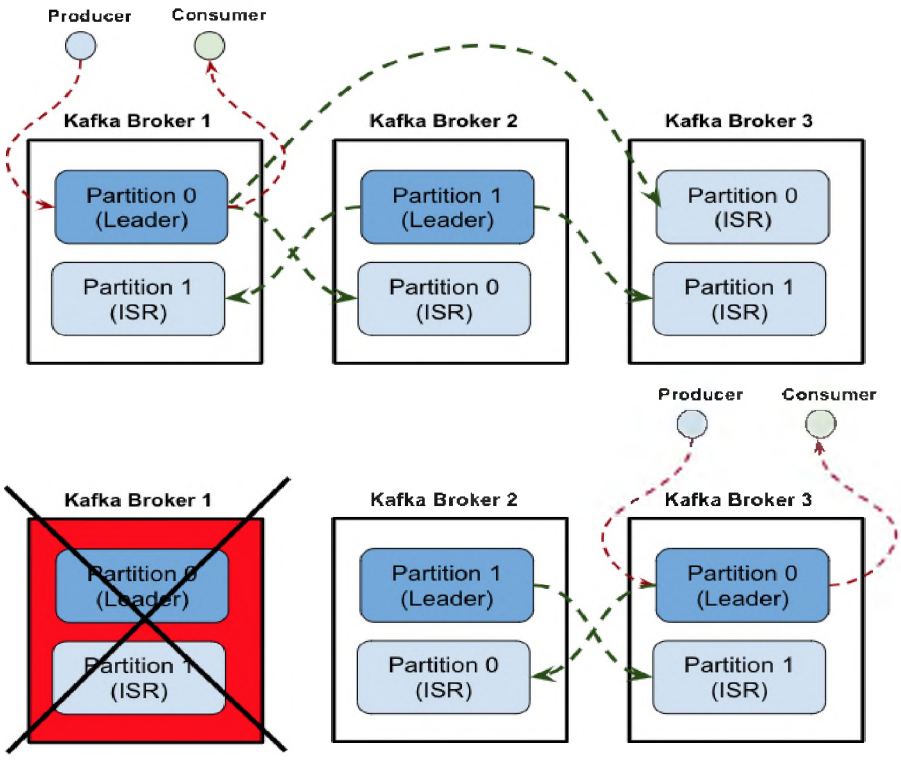

Figure 3. Partition replication in Kafka with a replication factor of 3. In the event of a failure in Kafka Broker 1, the in-sync replica of partition 0 on Kafka Broker 3 is promoted to Leader, ensuring uninterrupted service

Kafka's design principles and core components are described in detail in the official documentation, which serves as a comprehensive resource on the platform's scalability and fault-tolerance mechanisms [4].

Through these mechanisms, the Kafka cluster forms a resilient, scalable backbone for streaming large-scale image data and detection results in real time, making it well-suited for the high-throughput requirements of instant object detection.

YOLO Inference Engine

The YOLO model [1, 5] is a popular algorithm used in object detection in computer vision problems such as medical object detection [6], agriculture field [7], education [8], etc. Unlike previously introduced models such as Fast RCNN [9], it analyzes the image once. It basically divides the image into grids and determines whether the desired object is present in each grid. If there is an desired object, it detects its class and marks its location with a bounding box.

Recently, the YOLO model is improved by developers at short intervals and new versions are released. The latest version is the YOLOv12 model [10]. These versions are presented in different sub-versions. For example, YOLOv12n was developed to work on devices such as mobile applications, it can produce faster results but its accuracy is lower than larger models. On the other hand, YOLOv12l works with higher accuracy but works slower.

Data Streaming and Analytics

Within Kafka, topics are logically divided into multiple partitions, a fundamental feature that enables efficient data distribution across brokers and facilitates parallel processing. Each partition acts as an ordered, immutable sequence of messages that can be independently written and read by producers and consumers, respectively. This partitioning mechanism allows multiple consumers to concurrently process data by assigning different partitions to separate consumer instances, thereby significantly improving the system's scalability and throughput [4].

An increase in the number of partitions directly corresponds to greater parallelism, as more partitions enable a higher degree of concurrent data handling across the Kafka cluster. This capability is essential for high-throughput applications such as instant object detection pipelines, where rapid processing of large data volumes is required.

Figure 4 illustrates a Kafka topic configured with five partitions, demonstrating how messages are distributed and processed in parallel. Kafka topics can be defined and managed using the Kafka command-line tool. For instance, a Kafka topic configured with five partitions can be defined using the following command-line instruction:

bin/kafka-topics.sh --bootstrap-server localhost:9092 --create --topic dss-image-stream --partitions 5

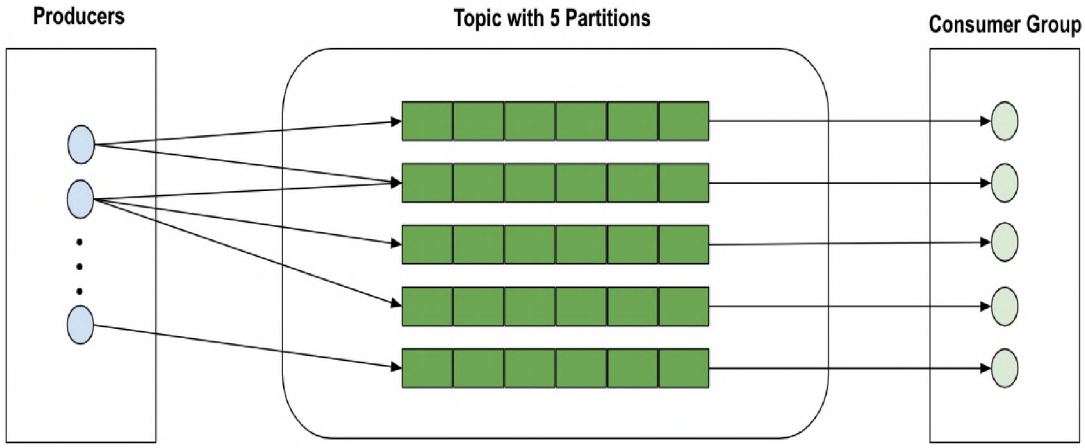

Figure 4. Illustration of a Kafka topic with five partitions enabling parallel message distribution and processing

Furthermore, Kafka allows the number of partitions for existing topics to be increased to accommodate growing workloads, though the number of partitions cannot be decreased once set. The partition count can be altered via the following command-line instruction:

bin/kafka-topics.sh --bootstrap-server localhost:9092 --alter --topic dss-image-stream --partitions 7

These administrative capabilities offer flexibility in scaling Kafka-based streaming systems dynamically, ensuring sustained performance as demand fluctuates.

Development and Deployment Details

Kafka Cluster Setup:

To support scalable, real-time data ingestion and distribution in the proposed object detection pipeline, a Kafka cluster was deployed on a Virtual Private Server (VPS) running Ubuntu 22.04. Kafka serves as the backbone for streaming image data and detection results between distributed components. This section outlines the installation and initialization of Kafka version 3.9.0.

Kafka is a distributed event-streaming platform that requires a coordination service — Apache ZooKeeper – for managing broker metadata and maintaining cluster consistency. Although Kafka is evolving toward a ZooKeeper-free architecture, version 3.9.0 retains ZooKeeper as a mandatory dependency for traditional deployments.

The Kafka binary package compiled for Scala 2.13 was downloaded from the official Apache Kafka repository:

```
wget https://downloads.apache.org/kafka/3.9.0/kafka_2.13-3.9.0.tgz
tar -xzf kafka_2.13-3.9.0.tgz
cd kafka_2.13-3.9.0
```

This unpacks the Kafka distribution into a working directory containing executables, configuration files, and documentation.

Kafka employs ZooKeeper to coordinate cluster state and manage broker metadata. Both services must be launched in independent terminal sessions.

```
# Start ZooKeeper (Terminal 1)
bin/zookeeper-server-start.sh config/zookeeper.properties

# Start Kafka broker (Terminal 2)
bin/kafka-server-start.sh config/server.properties
```

These commands start a single-node Kafka cluster with default configuration parameters. For multi-node clusters, configurations such as broker ID, log directories, and advertised listeners can be adjusted in config/server.properties.

Kafka topics are the fundamental unit for organizing and streaming data. Each topic can be divided into multiple partitions, allowing data to be distributed across brokers and processed concurrently by multiple consumers. For the definition of a Kafka topic configured with five partitions, refer to Section 2.3

YOLO Model Integration:

To implement instant object detection within the streaming pipeline, the YOLO model was integrated into the consumer-side application logic. The integration was performed within a Python environment on the same VPS used for Kafka deployment.

A Python virtual environment was employed to ensure isolation and reproducibility of the software environment. The following commands were executed on the remote VPS:

```
python3 -m venv venv
source venv/bin/activate
```

The first command defines a virtual environment named venv, encapsulating all Python dependencies locally within the project directory. The second command activates the environment, thereby ensuring that all subsequent package installations are confined to this context.

The following Python packages were installed using pip, Python's package manager, to satisfy the runtime dependencies required for model inference and data streaming integration:

pip install numpy==1.26.4 ultralytics opencv-python confluent-kafka

- numpy==1.26.4: Provides numerical computing capabilities required for image array manipulation and inference output processing.
- ultralytics: Contains the official implementation of YOLOv10 and associated utility functions for loading models, processing inputs, and retrieving detections.
- opencv-python: Enables image decoding, frame processing, and visualization functions. It is essential for handling raw image data within the pipeline.
- confluent-kafka: A high-performance Kafka client for Python, used to subscribe to Kafka topics and consume image streams for inference.

Figure 5 presents the code implementation of the data analytics consumer and worker, a core component of the object detection pipeline. This module is responsible for ingesting image data from the Kafka cluster, executing object detection tasks using a previously trained YOLOv10-based model, and streaming the resulting outputs to a designated Kafka topic.

The consumer is configured to subscribe to a specified input topic, from which it receives serialized image frames. Upon message arrival, the worker reconstructs the image from the byte stream and processes it using the previously trained model. The inference output is then encoded and published to an output topic. This enables downstream systems or monitoring agents to consume and utilize the detection results instantly.

Importantly, the worker does not maintain internal state across messages and can be deployed across multiple consumer instances under a common consumer group. Kafka's partitioning mechanism ensures that incoming data is distributed across available workers, enabling parallelism and horizontal scalability without centralized coordination. If multiple consumer instances are assigned the same group ID, they are treated as members of the same consumer group. As a result, launching multiple instances of this program with the same group ID forms a set of logically coordinated consumers, each acting as an independent worker operating on a disjoint subset of partitions.

```python
while True:
    msg = consumer.poll(1.0)
    if msg is None:
        continue
    if msg.error():
        print(f" Consumer error: {msg.error()}")
        continue
    try:
        # Parse JSON and decode base64 image
        data = json.loads(msg.value().decode('utf-8'))
        image_b64 = data['image']
        image_bytes = base64.b64decode(image_b64)
        image = Image.open(BytesIO(image_bytes)).convert('RGB')
        # Run YOLO inference
        results = best_model(image)
        # Return resulting image back
        for r in results:
            print(f"Detected {len(r.boxes)} objects in {data['filename']}")
            # Save the annotated result to a BytesIO buffer
            annotated_image = r.plot()  # This returns a NumPy image (BGR format)
            annotated_pil = Image.fromarray(annotated_image[..., ::-1])  # Convert BGR to RGB
            buffer = BytesIO()
            annotated_pil.save(buffer, format="JPEG")
            result_image_b64 = base64.b64encode(buffer.getvalue()).decode('utf-8')
            # Construct message
            result_message = {
                'filename': data.get('filename', 'unknown.jpg'),
                'num_objects': len(r.boxes),
                'image': result_image_b64
            }
            # Send prediction result back to Kafka
            producer.produce(OUTPUT_TOPIC, value=json.dumps(result_message).encode('utf-8'))
            producer.flush()
```

Figure 5. Kafka consumer and worker implementation for instant object detection

The following command is used to instantiate five parallel workers executing the same YOLO inference program:

pm2 start load-yolo-model.py --name yolo-python-model --interpreter ../venv/bin/python -i 5

In this command:

- **load-yolo-model.py** is the consumer-worker program given in Figure 5.
- **-i 5** specifies that five instances of the script should be launched concurrently.

Results and Discussions

To evaluate the scalability and parallel processing capability of the proposed architecture, two configurations were deployed using the Kafka client tool. Both configurations utilized the same topic name but differed in the number of partitions: one with a single partition and the other with five partitions. The resulting partition structures and message distributions are shown in Figures 6 and 7, respectively. To evaluate the performance of the proposed system, a series

of load testing experiments were conducted using the Artillery testing tool [11]. During these experiments, images were sent periodically to the system via Artillery to simulate real-time data input for object detection tasks.

In the **single-partition configuration** (Figure 6), all messages are routed to a single partition. Consequently, only one consumer instance within the group is eligible to process incoming data. This enforces a strictly sequential processing regime, limiting the system's ability to handle large volumes of data efficiently. The configuration does not permit concurrent consumer operation on the same topic, thereby constraining horizontal scalability and increasing susceptibility to processing delays under high load.

Figure 6. Kafka topic (yolo-model-input) configured with a single partition. All messages are serialized into one processing stream, restricting throughput and preventing parallelism

In contrast, the five-partition configuration (Figure 7) demonstrates Kafka's capacity for parallel stream processing. By defining five partitions within the topic, Kafka enables up to five consumer instances – each assigned to a separate partition within the same consumer group – to process data concurrently. The figure confirms that messages are distributed across all partitions, supporting balanced workload allocation without requiring manual load distribution logic.

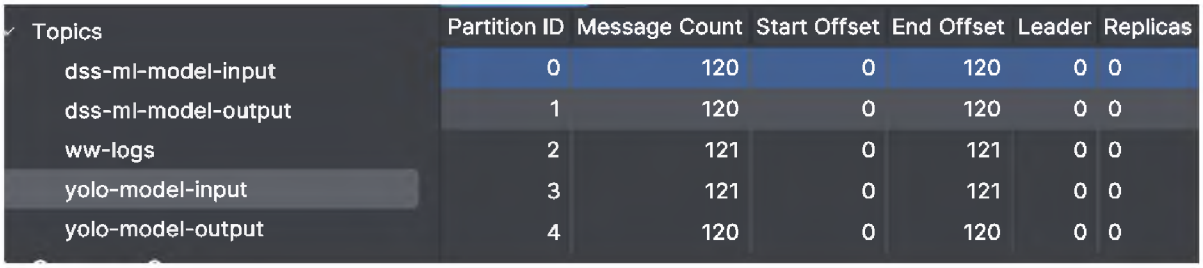

Figure 7. Kafka topic (yolo-model-input) configured with five partitions.
Messages are distributed across partitions, allowing parallel processing by multiple consumers within a group

This partitioned architecture substantially enhances the pipeline's throughput. It supports scalable deployment of multiple inference workers, thereby improving responsiveness and reducing overall system latency. Furthermore, it enables elastic resource utilization, where additional consumer instances can be introduced in response to increased traffic volume, constrained only by the number of partitions defined.

Conclusions

This study has presented a modular and scalable pipeline for instant object detection, integrating YOLO-based models with Apache Kafka as the backbone for streaming and

workload distribution. The system architecture exploits Kafka's partitioning and consumer group coordination mechanisms to enable parallel processing of high-throughput image streams without requiring external load balancers or centralized control.

Through deployment on a VPS, the pipeline has demonstrated operational feasibility and adaptability to practical constraints. Comparative configurations with single and multiple topic partitions empirically illustrate Kafka's role in supporting distributed inference across multiple consumer instances, thereby enhancing system throughput and scalability. While the study does not report detailed latency or throughput metrics, the experimental setups and system behavior validate Kafka's effectiveness in facilitating concurrent processing and maintaining instant responsiveness under load. Future work will focus on extending the architecture with quantitative performance profiling, dynamic scaling strategies, and integration with downstream analytics components. This architecture serves as a robust foundation for time-sensitive object detection tasks and may be extended to broader domains requiring scalable and instant data analytics.

References:

1. Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 779–788.
2. Han, S., Lee, H., & Park, Y. (2018). A Real-Time Object Detection System Based on Cloud and Edge Computing. In International Conference on Cloud Computing and Big Data (CloudCom), pp. 153–160.
3. Shi, Y., Ding, G., & Wu, Q. (2019). Edge Computing: Vision and Challenges. IEEE Internet of Things Journal, 6(3), 4724–4737.
4. Apache Kafka. Apache Kafka Documentation. Available at: https://kafka.apache.org/39/documentation.html (Accessed: May 2025).
5. J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger," 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). IEEE, pp. 6517–6525, Jul. 2017. doi: 10.1109/cvpr.2017.690.
6. A. Soni and A. Rai, "YOLO for Medical Object Detection (2018–2024)," 2024 IEEE 3rd International Conference on Electrical Power and Energy Systems (ICEPES). IEEE, pp. 1–7, Jun. 21, 2024. doi: 10.1109/icepes60647.2024.10653506.
7. C.M. Badgujar, A. Poulose, and H. Gan, "Agricultural object detection with You Only Look Once (YOLO) Algorithm: A bibliometric and systematic literature review," Computers and Electronics in Agriculture, vol. 223, p. 109090, Aug. 2024, doi: 10.1016/j.compag.2024.109090.
8. H. Chen, "YOLO Algorithm in Analysis and Design of Athletes' Actions in College Physical Education," 2024 International Conference on Interactive Intelligent Systems and Techniques (IIST). IEEE, pp. 764–768, Mar. 04, 2024. doi: 10.1109/iist62526.2024.00002.
9. R. Girshick, "Fast R-CNN," 2015 IEEE International Conference on Computer Vision (ICCV). IEEE, Dec. 2015. doi: 10.1109/iccv.2015.169.
10. Ultralytics, "YOLOv12 Models," Ultralytics Documentation. Available at: https://docs.ultralytics.com/tr/models/yolo12/. (Accessed: May 2025).
11. Artillery Docs. Available at: https://www.artillery.io/docs (Accessed: May 2025).

Information about the authors:

Celal Ceken – corresponding author, Professor, Department of Computer Engineering, Sakarya University, Sakarya, Türkiye; e-mail:celalceken@sakarya.edu.tr; Professor, Manash Kozybayev North Kazakhstan University NPLC, International Campus, Petropavlovsk, Kazakhstan; e-mail: cceken@ku.edu.kz;
Ismail Oztel – Assistant Professor, Department of Computer Engineering, Sakarya University, Sakarya, Türkiye; e-mail: ioztel@sakarya.edu.tr.