

Article

Designing a Scalable YOLO-Based Decision Support Framework for Mitochondrial Analysis in EM Imaging

Gozde Yolcu Oztel ^{1,2,*} , Ismail Oztel ^{2,3} and Celal Ceken ^{3,4} ¹ Department of Software Engineering, Sakarya University, 54050 Serdivan, Türkiye² Intelligent Software Systems Research Lab, Sakarya University, 54050 Serdivan, Türkiye; ioztel@sakarya.edu.tr³ Department of Computer Engineering, Sakarya University, 54050 Serdivan, Türkiye⁴ International Campus, Manash Kozybayev North Kazakhstan University, 150000 Petropavlovsk, Kazakhstan

* Correspondence: gylcu@sakarya.edu.tr

Abstract

This study presents a scalable decision support system (DSS) framework designed to meet the growing demands of instant data-driven decision-making environments. The architecture integrates key technologies, including Apache Kafka for parallel data streaming, a Python-based data analytics module for distributed processing, JWT-based secure user authentication, and WebSocket communication for instantaneous prediction delivery. The system performs mitochondrial localization in electron microscopy (EM) images using multiple versions of the YOLO (You Only Look Once) object detection model. The publicly available CA1 Hippocampus dataset was used for detection evaluation. Among the evaluated models, YOLOv10x achieved the highest detection performance, yielding a mean average precision (mAP) score of 95.2%. Experimental evaluations of the DSS were conducted under simulated load conditions using the Artillery tool to assess the system's scalability and responsiveness. Empirical results indicate consistent low-latency performance across varying consumer group sizes, confirming the architecture's ability to scale the analytics module horizontally without compromising responsiveness. These findings validate the system's suitability for just-in-time decision support applications. In particular, the system may support clinicians in the task of mitochondrial analysis, where structural abnormalities can be indicative of pathological conditions, including cancer. By enabling early detection of such abnormalities, the proposed framework has the potential to contribute to the timely diagnosis of diseases such as cancer. The proposed study differs from existing studies by combining deep learning with real-time scalable data processing technologies, such as Kafka and WebSocket, in a web-based DSS application for mitochondria detection.

Keywords: scalable data streaming; YOLO model; parallel processing; mitochondria detectionAcademic Editors: Thomas Lindner
and Pedro Couto

Received: 8 March 2026

Revised: 26 March 2026

Accepted: 27 March 2026

Published: 2 April 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

1. Introduction

Decision support systems have increasingly evolved into intelligent, real-time platforms capable of assisting decision-making processes across a range of domains [1–5]. As datasets become larger and more dynamic, DSS frameworks must incorporate real-time data processing, concurrent computation, and robust delivery mechanisms. This study presents a scalable decision support system developed as a web-based application. It integrates modern components to support the mitochondria localization task.

Mitochondria produce the energy required for cellular metabolic activities. Abnormal numbers or shapes of mitochondria are associated with cancer and other metabolic

disorders [6–10]. Detecting such differences early may help enable early cancer diagnosis. Although mitochondria images can be obtained using techniques such as electron microscopy, manually counting them is difficult. Because each cell contains many mitochondria and the human body contains many cells, developing fast and automated tools to assist experts in detecting and counting mitochondria can be highly beneficial.

This study presents a comprehensive DSS framework designed to automate mitochondria detection and counting from electron microscopy images using YOLOv10 and YOLO26 variants. The system is low-cost, and it produces objective and countable results. Thus, it can be beneficial to assist experts in detecting and counting mitochondria. This study makes the following contribution to the literature:

(I) **Assists early detection of some disease:** Counts and shapes of mitochondria can be critical biomarkers for early detection of cancer and metabolic disorders. The system reduces the dependency on manual labeling and accelerates diagnosis by using YOLOv10 and YOLO26.

(II) **Integrates SOTA technologies for mitochondria detection:** Integration of Apache Kafka, WebSocket, YOLOv10, and YOLO26 for mitochondria detection contributes to both medical and computer science literature.

The remainder of the paper is organized as follows. Section 2 reviews relevant literature on mitochondrial image analysis and scalable architecture-based studies. Section 3 details the methodology, including system design, YOLOv10 integration, and Kafka-based streaming mechanisms. Section 4 presents the experimental setup and performance evaluation, with a focus on detection accuracy and latency under varying computational loads. It also discusses the implications of the results and potential deployment considerations. The paper is concluded with Section 5 providing final remarks.

2. Related Works

2.1. Scalable Architectures for Instant Data Processing

Recently, technological advances in cameras, sensors, microscopes, and other medical equipment have enabled data collection. Processing of large amounts of data is also possible using powerful hardware. To handle this data efficiently and support real-time applications for multiple users, scalable systems are being developed in many fields. Although the literature is extensive, this section highlights a few representative studies with different architectural approaches.

In [11], the authors proposed a scalable architecture that enables spoken dialogue systems to run on the web. Scalable architectures are also widely used in healthcare. For example, ref. [12] presented an architecture that allows multiple clinical decision support systems (CDSSs) to operate on a common telemonitoring platform without interfering with each other. Similarly, ref. [13] proposed a scalable system combining Twitter, Kafka, Spark, and Cassandra to predict diseases from medical data streams. In that study, the best accuracy was achieved with the Random Forest algorithm after feature selection using the Relief algorithm. Additionally, ref. [14] introduced a scalable e-health architecture that enables clinical devices to communicate automatically via the internet, improving diagnostic speed and accuracy. Ref. [15] presented Personal Health Dashboard (PHD) software that provides scalability for analyzing large biomedical data. This software can store and analyze complex data such as wearable devices, health records, and genetic data. In [16], the authors developed a scalable and intuitive deep learning toolkit called R2D2. R2D2 is focused on semantic segmentation tasks for medical imaging.

2.2. Automated Mitochondria Analysis

Owing to technological development, high-performance medical image analysis studies have been conducted in recent years [17–22]. In classical machine learning, the feature extraction step needs expert knowledge. Recently, owing to deep learning approaches, the feature extraction step is automatically applied. Thus, deep learning makes it possible to carry out more effective image analysis.

The literature includes some mitochondrial analysis studies, including those focusing on mitochondria segmentation [23–25]. In [26], the authors applied supervoxel-based segmentation. This method combines shape features that can describe the 3D shape of the mitochondria. In [27], the authors proposed a method that uses shape information and regional statistics to segment mitochondria in EM images. The algebraic curves and regional information were used to segment the mitochondria at the predicted locations. In [28], a pixel-based approach was developed to analyze mitochondrial movements to examine abnormalities of mitochondria under conditions of cell stress, swelling, etc. In [29], the authors focused on the segmentation of mitochondria using a CNN. Ref. [30] shows high-performance results using pixel-wise segmentation; it operates within a semantic segmentation framework. Also, in [31], a novel recurrent neural network (RNN) was proposed for the mitochondria segmentation task. Moreover, [32] presented MitoSegNet, a software tool that combines segmentation and morphological analysis functions and can run on Windows and Linux systems. Unlike these studies, the system proposed in this paper formulates mitochondria detection as an object detection problem. This brings several practical advantages. This approach enables faster inference and direct quantification of mitochondria as discrete objects. This is particularly beneficial in studies where mitochondrial count, morphology, and spatial distribution are crucial biomarkers.

Some of the previous studies aimed to segment mitochondrial compartments. In [33], a CNN-based system was developed to discriminate four different mitochondrial compartments (matrix, outer, inner, and intermembrane regions). In [34], a system that localizes mitochondrial subdivisions was developed. A bidirectional LSTM with a self-attention mechanism was used in the study.

Also, various studies have focused on mitochondria detection [35]. For example, ref. [36] utilized classical image processing methods such as ellipse detection for mitochondria detection. However, this approach has drawbacks, such as susceptibility to noise and the need for manual intervention. Ref. [37] used the Faster R-CNN algorithm for mitochondria detection in ATUM-SEM images. In [38], using cryo-electron tomography (cryo-ET)-based images, Faster-RCNN was used to detect mitochondria. Faster R-CNN is a relatively heavier detection model and limits real-time usability, especially when processing large-scale EM datasets. In contrast, the proposed YOLO-based system is optimized for real-time, low-latency inference and supports large-scale image processing through Kafka-based data pipelines. Moreover, unlike previous mitochondria analysis studies, by employing Apache Kafka for parallel message processing and WebSocket for asynchronous communication, the proposed architecture enables efficient, scalable, and just-in-time image analysis.

2.3. YOLO and Transformer-Based Approaches for Medical Imaging

YOLO and Transformer-based approaches are widely used for medical imaging tasks in the literature. In [39], a modified YOLOv8 model was used to accurately detect tumors within MRI images. The model performed better than the original YOLOv8 model and also performed better than other object detectors (Faster R-CNN, Mask R-CNN, YOLO, YOLOv3, YOLOv4, YOLOv5, SSD, RetinaNet, EfficientDet, and DETR). Ref. [40] proposed a two-stage deep learning model integrating U-Net, YOLOv8s, and the Swin transformer

to detect lung cancer nodules in computer tomography images. The model demonstrates high accuracy and a reduced false positive rate. Ref. [41] proposed TransUNet, which merges Transformers and U-Net, as a strong alternative for medical image segmentation. TransUNet shows high performance on different medical applications, such as multi-organ segmentation and cardiac segmentation. Ref. [42] used a YOLOv7-based model for kidney detection in medical images. The results show that the model achieves high accuracy, sensitivity, and mean average precision (mAP) values in kidney and tumor detection.

3. Materials and Methods

3.1. Decision Support System Overview

At the core of the system lies a Kafka-based distributed event-streaming backbone, enabling high-throughput and low-latency communication across components. User inputs, submitted via the web interface, are processed in parallel by the Python-based analytics module, ensuring scalability and resilience under varying loads. Figure 1 illustrates the system architecture, including producers, parallel consumers, communication servers, and the persistent storage layer. Predictions are instantly returned to the client interface via WebSocket for an interactive user experience. Access control is enforced through JWT-based authentication and role-based mechanisms, with users, roles, and operational logs persistently stored in a PostgreSQL database.

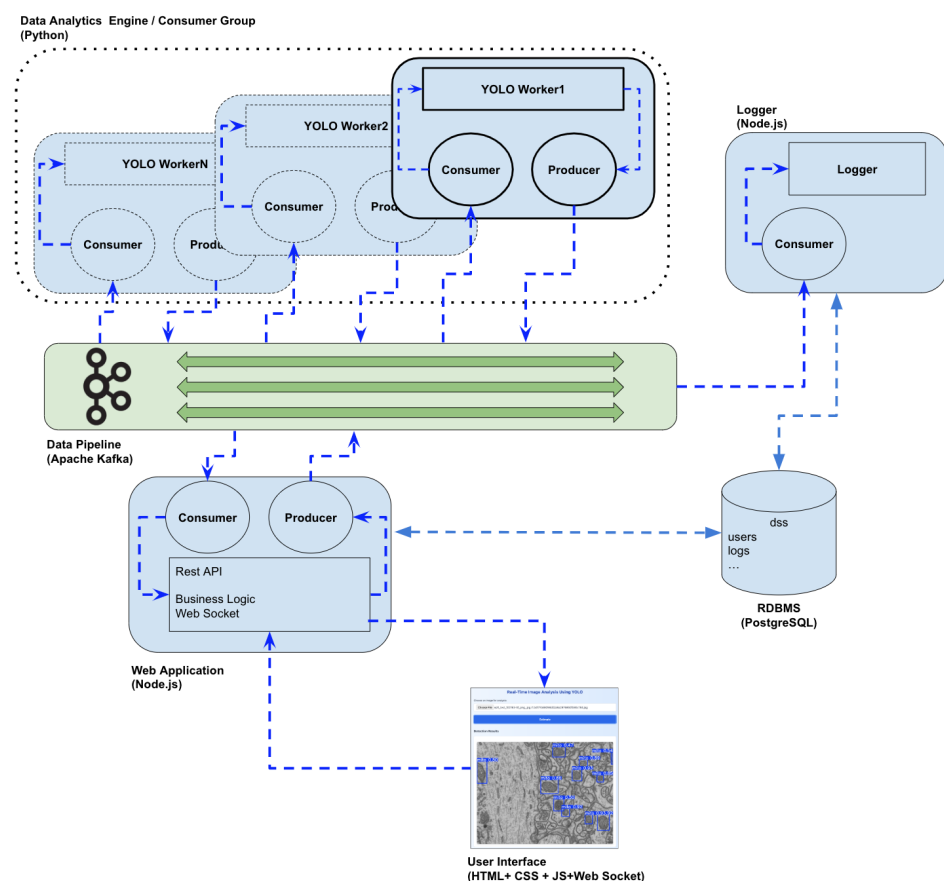


Figure 1. Overall architecture of the proposed real-time DSS, showing the client interface, web service layer, KRaft-based Apache Kafka infrastructure with partitioned topics and consumer groups, YOLO analytics modules, and WebSocket-based result delivery.

3.2. Web Application and User Interfaces

The web application serves as the primary user interaction point within the proposed DSS framework. Designed with an emphasis on usability and responsiveness, the interface

facilitates seamless data entry, model interaction, and real-time visualization of predictions. Upon successful authentication, users assigned with administrator privileges are automatically directed to the dashboard page, which consolidates system functionalities into an accessible and intuitive environment.

The dashboard and model interaction interface are illustrated in Figure 2. This environment enables authorized users to upload mitochondria microscopy images, which are subsequently analyzed using a deployed YOLO-based object detection model. Upon submission, the image data is serialized and transmitted through the Apache Kafka messaging infrastructure, facilitating efficient distribution to parallel data analytics consumers for just-in-time processing.

Once the object detection algorithm processes the image and successfully localizes the mitochondria, the resulting annotations are published to a designated Kafka topic configured for output communication. This asynchronous yet efficient communication mechanism enables downstream components to consume the detection results independently. To support seamless and low-latency feedback to the user interface, the system employs a WebSocket server that listens to this output stream. As a result, the detected locations are immediately sent to the web-based dashboard. This allows users to visualize the predicted regions on the original image almost in real time without manually refreshing the page.



Figure 2. Screenshot of the admin dashboard incorporating the model interaction interface for real-time input and prediction feedback.

3.3. Data Streaming and Analytics

Efficient and reliable data streaming is a critical requirement for modern DSS designed for instant analytical response. In the proposed architecture, Apache Kafka acts as the backbone for data transfer between system components. It enables high-throughput and low-latency communication across distributed modules. Apache Kafka is a distributed event-streaming platform specifically designed to handle large volumes of data with minimal delay. Its scalable architecture allows data to be streamed continuously from producers to consumers. It ensures that the system remains responsive even under increasing workloads.

In the proposed DSS framework, microscopy images uploaded via the web interface are published to designated Kafka topics. The data analytics module, designed around Kafka's consumer group architecture, subscribes to designated topics and retrieves image data for parallel processing. This configuration enables efficient workload distribution and supports instant detection and quantification of mitochondria using a YOLO-based model. As shown in Figure A1, the analytics pipeline is implemented using a Python-based Kafka consumer. It listens for base64-encoded image data, decodes and processes the images with a preloaded YOLO model, and sends the annotated results to an output Kafka topic. The figure illustrates key components of this loop. It includes consumer initialization, instant image parsing, YOLO inference execution, and the serialization of detection results for downstream use. The system runs multiple instances of the consumer using the same group.id parameter (line 31). Through Kafka's partition-aware consumer group mechanism, incoming messages are distributed across independent processing units, enabling parallel processing. This implementation demonstrates the system's practical realization of streaming analytics within a scalable, event-driven architecture.

To achieve effective parallelism and load balancing, Kafka topics are divided into multiple partitions. Each partition acts as a sequential, ordered log that can be consumed independently. Kafka organizes data into multiple partitions. This allows multiple consumers to read from different partitions simultaneously, improving the system's throughput and responsiveness. As described in [43], partitioning plays a key role in scaling data pipelines for high throughput and fault tolerance. Consumer groups are a fundamental concept in Kafka's design for distributed processing. A consumer group is a collection of consumers that work together to consume data from a topic in a coordinated manner. It ensures that each partition is read by only one consumer within the group. This mechanism allows the system to scale horizontally: as the number of consumer instances increases, more data can be processed in parallel, improving overall performance. Figure 1 illustrates how consumer groups are utilized in the data analytics pipeline to manage the parallel processing of incoming data streams. A critical design consideration for maximizing parallelism is ensuring that the number of partitions is greater than or equal to the number of consumers in a consumer group. When properly configured, this arrangement allows each consumer to be assigned a dedicated partition. It prevents bottlenecks and enables true parallel data processing. Conversely, if there are more consumers than partitions, some consumers will remain idle, limiting the system's scalability. The proposed DSS architecture leverages Kafka's topic partitioning and consumer group features. In addition, it integrates a data analytics module for predictive analysis. This design allows the system to efficiently handle real-time data streams and supports scalable, distributed decision-making and near real-time analytics. To validate the real-world applicability of the proposed DSS architecture, the application was deployed onto a dedicated server environment. This deployment allowed end-to-end testing of all components, including the web application, Apache Kafka services, the data analytics module, and the PostgreSQL database.

3.4. YOLOv10 Model for Mitochondria Detection

YOLO is a very popular object detection algorithm owing to its performance [44–53]. Over time, different YOLO versions have been created by researchers. The YOLOv10 version [54] was developed in May 2024. Based on [54], YOLOv10 improves both efficiency and accuracy in object detection compared to previous YOLO versions. In previous versions, dependency on NMS and architectural inadequacies prevented optimal performance. In the YOLOv10 version, training is provided without NMS. Also, the dual label assignments feature is added to the architecture. They incorporated another one-to-one head for YOLO. During training, these heads are optimized with the model. During inference, they discard

the one-to-many head and make predictions using the one-to-one head. It is designed with various model scales to meet different application needs. YOLOv10n is recommended for very limited resources. YOLOv10s is a smaller version that optimizes speed and accuracy. For general-purpose uses, the mid-sized YOLOv10m model is recommended. The YOLOv10b version has an increased width for higher accuracy. The major version, YOLOv10l, is designed for higher accuracy at the cost of increased computational resources. The extra-large version, YOLOv10x, is developed for maximum accuracy. In [54], YOLOv10 was tested on well-known datasets such as COCO and showed superior performance and efficiency. It showed significant improvements in latency and accuracy compared to its previous versions. With this motivation, in this study, YOLOv10 models at different scales were trained and tested for the task of detecting mitochondria. Models of different scales were compared. Thus, the regions containing mitochondria were included in the bounding box, and their locations were determined.

Unlike natural datasets, electron microscope images contain low-contrast, grayscale structures and small mitochondria, making them challenging to detect with YOLOv10. Selecting the most appropriate parameters is crucial to overcoming this challenge. Therefore, the YOLO model that produced the best results was trained and tested with various parameters. These results are also presented in the Experimental Results Section.

3.5. YOLO26 Model for Mitochondria Detection

YOLO26 [55], the latest version released on 14 January 2026 by Ultralytics, is designed from the ground up for edge devices and low-power devices. It eliminates unnecessary complexity and offers a simplified design that is faster, lighter, and more accessible.

NMS, pioneered in YOLOv10, was further developed in YOLO26. Thus, this post-processing step was eliminated, and a faster and lighter model was produced. It uses the MuSGD optimizer, a hybrid of SGD and Muon. This optimizer brings enhanced stability and faster convergence.

The Distribution Focal Loss (DFL) module, included in previous versions, had complicated export and limited hardware compatibility. This module was removed in YOLO26, thus simplifying inference and increasing support for edge and low-power devices. Owing to its improved loss functions, it increases detection accuracy in the field of small object recognition.

Similar to YOLOv10, YOLO26 has five subversions for detection. All of these subversions were trained for mitochondria detection, and their results were compared.

4. Experimental Results

4.1. Dataset and Data Preparation

In this study, the CA1 Hippocampus dataset [56,57] was used. This public dataset includes two volumes of EM images of mouse brains. Each volume consists of 165 slices, each with a resolution of 1024×768 pixels and a voxel size of $5 \times 5 \times 5 \mu$. Each image has corresponding two-class (background and mitochondria) ground truth volumes, as shown in Figure 3. In this study, the goal is the detection and localization of mitochondria. Therefore, the ground truths prepared for segmentation could not be used directly. Instead, the location of each mitochondrion in the training and test sets was labeled with a bounding box using Roboflow [58]. For each image, an original image was submitted to the annotation tool while the corresponding ground truth segmentation mask was viewed alongside. Using the mask as a reference, to locate each mitochondria, bounding boxes were manually drawn on the original image in the annotation tool.

In order to increase the amount of data, some data augmentation approaches were applied. These are given in Table 1. Data augmentation was applied by randomly select-

ing images from the training set and performing random transformations such as flips, rotations, and shear operations, as seen in the table. The augmented dataset included the original images as well, resulting in a threefold increase in size. The training set initially consisted of 165 images and was expanded to 495 images through data augmentation. After the annotation and data augmentation process, the annotated mitochondria number was 7269 in the training set and 2558 in the test set.

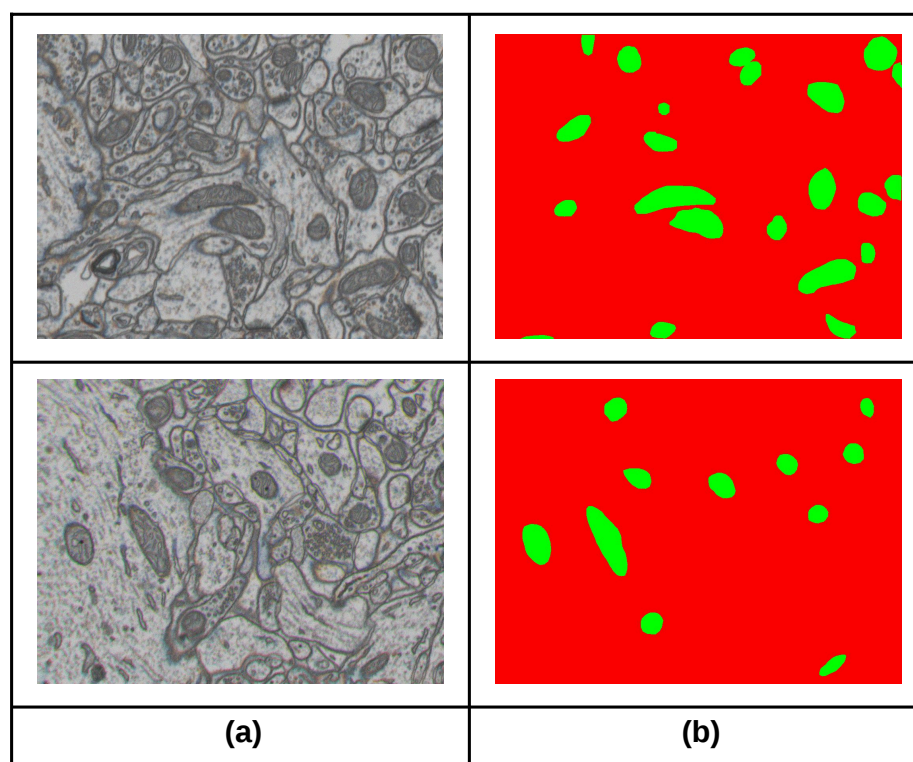


Figure 3. Sample images and their annotations in the proposed dataset. (a) Original images obtained from electron microscopy. (b) Ground-truth labels illustrating the target mitochondria region (green) and background information (red). Annotations were used as a reference during training and evaluation.

Table 1. Data augmentation processes.

Technique	Description
Flipping	Horizontal and Vertical
Rotation	Between -15 and $+15$ degrees
Shearing	± 10 degrees (Vertical and Horizontal)

4.2. Comparative Results of Detection

In this study, experiments were performed on different YOLOv10 and YOLO26 versions, and their results were compared. The used parameters on the versions are reported in Table 2. The training parameters listed in this table were selected based on the official YOLOv10 and YOLO26 implementations and established practices in object detection. The Adam optimizer was chosen due to its effectiveness in stabilizing training for modern YOLO architectures [59]. A learning rate of 0.002, momentum of 0.9, and weight decay of 0.0005 are commonly adopted in recent YOLO variants and have demonstrated robust convergence behavior [54]. An input image size of 640×640 was used, which is a standard resolution in YOLO-based models to balance accuracy and computational efficiency [60]. The same training parameters were consistently applied across all YOLOv10

and YOLO26 variants (from n to x) to ensure a fair comparison. This allows us to isolate the impact of model architecture and scale on performance without confounding effects from varying hyperparameters.

Table 2. Training parameters.

Parameter	Value
Optimizer	AdamW
Learning Rate	0.002
Momentum	0.9
Decay	0.0005
Image Size	640 × 640

In the literature, object detection systems are usually evaluated using the mean average precision (mAP) metric. Thus, in this study, the evaluation was applied using the mean average precision metric (Equation (4)). Also, the inference times of the models were compared.

$$AP = \sum \frac{\text{precision}}{\text{recall}} \quad (1)$$

$$\text{Precision} = \frac{TP}{TP + FP} \quad (2)$$

$$\text{Recall} = \frac{TP}{TP + FN} \quad (3)$$

where *TP* is true positives, *FP* is false positives, *FN* is false negatives, and *AP* is average precision. The mean average precision (*mAP*) is provided in Equation (4).

$$mAP = \frac{1}{N} \sum_{i=1}^N AP_i \quad (4)$$

where *N* is the class number. For evaluation, the PyTorch package was used. The codebase was built with Ultralytics [54], and the comparative results of the YOLOv10 and YOLO26 versions are given in Table 3. All the training operations were started for 200 epochs. It was set to “stop training if no recovery for 10 epochs”. Based on this, the networks were finalized in different epoch numbers. These are reported in Table 3. As can be seen, the best mAP was obtained using YOLOv10x. This is an expected result, considering that YOLOv10x was developed to achieve a maximum success rate. On the other hand, in terms of time, the best performance was obtained with YOLOv26n. This is also an expected result, considering that nano versions were developed to achieve maximum speed. As seen, YOLOv10x and YOLO26x took longer inference times compared to other models. Experiments showed that YOLO26’s various versions consistently showed high and stable performance rates (mAP scores), and training was completed in a shorter time. Although YOLOv10x achieved the highest result in this study, this finding highlights YOLO26 as a stable and robust model.

After identifying YOLOv10x as the best-performing model in terms of mAP, additional re-training experiments were conducted by varying the optimizer, learning rate, and cosine scheduling parameters. As shown in the Table 4, the highest mAP values were achieved with AdamW (*lr* = 0.002, *cos-lr* = false) and SGD (*lr* = 0.01, *cos-lr* = true) configurations. Nevertheless, other parameter settings also produced comparable results, indicating that the model demonstrates stable performance across different hyperparameter choices.

Figure 4 visualizes some sample test results of the YOLOv10x model. Figure 4a shows actual mitochondria labels in a bounding box and Figure 4b. shows predictions of the corresponding sample. In the figure, both the ground-truth labels and the model predictions

are clearly distinguishable. Furthermore, careful examination of the predictions reveals that a structure located in the lower-left region of the image is classified as mitochondria despite the absence of a corresponding ground-truth label. This instance represents a false positive prediction and illustrates one type of error made by the system.

Table 3. System performance comparison.

Model	Epoch Number	Training Time (hours)	mAP	Inference Time (ms)
YOLOv10m	146	1.386	0.923	19.5
YOLOv10b	146	1.431	0.930	24.7
YOLOv10n	198	0.875	0.934	3.8
YOLOv10s	200	1.342	0.944	8.6
YOLOv10l	200	2.810	0.945	32.7
YOLOv10x	143	2.354	0.952	48.5
YOLO26n	52	0.209	0.925	2.8
YOLO26s	60	0.253	0.945	8.2
YOLO26l	68	0.561	0.948	24.6
YOLO26x	85	1.535	0.949	48.5
YOLO26m	58	0.390	0.951	21.3

Table 4. Re-training experiments with the best-performing model (YOLOv10x) under different hyperparameter settings.

Experiment	Optimizer	Learning Rate	cos_lr	mAP
1	AdamW	0.002	False	0.952
2	AdamW	0.002	True	0.939
3	AdamW	0.0015	False	0.942
4	SGD	0.01	True	0.952
5	SGD	0.002	False	0.948
6	SGD	0.001	False	0.950

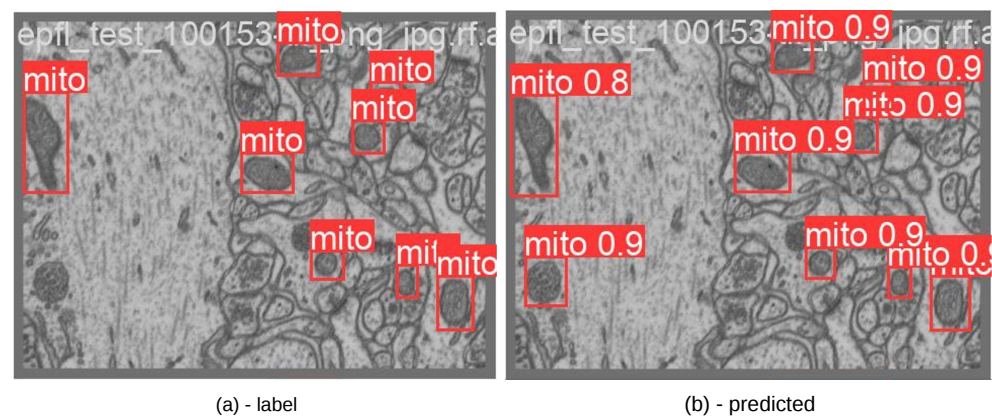


Figure 4. Some samples from the dataset.

To assess the computational complexity of the YOLOv10 and YOLO26 variants, we analyzed two key metrics: the number of model parameters and the floating-point operations per second (FLOPs). The number of parameters reflects the model's capacity and memory footprint, directly influencing training time and storage requirements. Higher parameter counts typically lead to larger models that require more computational resources but may achieve better accuracy. FLOPs, on the other hand, measure the computational cost during inference, indicating how efficiently a model can process input data in real-time. As shown in Table 5, the model size increases significantly from YOLOv10n (2.65M parameters) to YOLOv10x (31.58M parameters), with corresponding FLOPs rising from 8.2 GFLOPs to

169.8 GFLOPs. This trend demonstrates a clear trade-off between model performance and computational efficiency: while larger models like YOLOv10x offer higher accuracy, they demand substantially more hardware resources, making them less suitable for edge devices. In contrast, smaller models such as YOLOv10n are highly efficient and ideal for real-time applications with limited computational power. A similar trend can be observed among the YOLO26 variants.

Table 5. Comparison of the number of parameters and computational load (FLOPs) of different models.

Model	#Parameters	FLOPs
YOLOv10n	2,694,806	8.2 GFLOPs
YOLOv10s	8,035,734	24.4 GFLOPs
YOLOv10m	16,451,542	63.4 GFLOPs
YOLOv10b	20,412,694	97.9 GFLOPs
YOLOv10l	25,717,910	126.3 GFLOPs
YOLOv10x	31,586,006	169.8 GFLOPs
YOLO26n	2,375,031	5.2 GFLOPs
YOLO26s	9,465,567	20.5 GFLOPs
YOLO26m	20,350,223	67.8 GFLOPs
YOLO26l	24,746,511	86.1 GFLOPs
YOLO26x	55,634,703	193.4 GFLOPs

4.3. Evaluation of DSS Architecture Performance

To evaluate the performance of the proposed DSS architecture, a series of load and stress testing experiments was conducted using the Artillery testing tool [61]. The main objective was to assess the system's responsiveness and scalability under simulated real-world conditions, with particular focus on its ability to handle increasing data loads efficiently. The experimental environment was set up on a virtual private server (VPS) machine with an AMD EPYC 7543P processor, 4 cores, and 16 GB of memory, running on Ubuntu 22.04. The Apache Kafka cluster for data streaming was configured to use the KRaft (Kafka Raft) mode in Apache Kafka version 4.0.0. In this setup, the Kafka broker and controller operated within the same KRaft-based process and were executed locally on the same machine as the web application, data analytics modules, and supporting services. This configuration ensures that the observed performance reflects the capabilities of the proposed architecture, without network-induced variability.

In the testing setup, simulated image traffic was directed to the route responsible for sending images to the Apache Kafka cluster. Each virtual user submitted a sample image to the backend for just-in-time prediction by the data analytics module. The load profile was configured to simulate a constant arrival rate of 1 request per second over 60 s. Three distinct scenarios were evaluated:

- Scenario 1: A single consumer instance running within the data analytics module, responsible for processing all incoming prediction requests.
- Scenario 2: Two consumer instances operating within the same consumer group, enabling parallel processing of incoming data streams. To fully utilize all consumers, the Kafka topic was configured with three partitions to ensure each consumer could be assigned a separate partition and work independently for maximum throughput.
- Scenario 3: Three consumer instances operating within the same consumer group, enabling parallel processing of incoming data streams. Similar to Scenario 2, the Kafka topic was configured with five partitions so that each consumer could be assigned a separate partition and work independently for maximum throughput.

Following the experiments, key latency metrics were collected, including the mean, median, P95, and P99 response times. Mean latency represents the average time taken to

process a request across the entire test duration. Median latency is the middle value of all recorded latencies, providing a measure less sensitive to outliers than the mean. P95 latency (95th percentile) represents the maximum latency experienced by 95% of requests, providing insights into typical worst-case performance under load. P99 latency (99th percentile) reflects the latency experienced by 99% of the requests, highlighting extreme outliers and rare delay events. These latency metrics are particularly important in instant decision support environments, where a consistent, low-latency response is critical. While mean and median values provide a general understanding of system behavior, P95 and P99 values reveal how the system performs under stress. It helps to uncover potential bottlenecks that might affect user experience during peak loads.

The detailed experimental outcomes, including graphical representations of the mean, median, P95, and P99 latencies for all scenarios, are presented in Table 6 and Figure 5, respectively.

The performance metrics gathered during the load testing phase reveal a stable system under increasing analytical load. Specifically, latency metrics—mean, median, P95, and P99—remain relatively consistent across tests involving 1, 2, and 3 Kafka consumers. This consistency is expected because all services were deployed on the same server. In addition, the workload of one request per second is low enough for a single consumer to handle without causing queuing or resource contention.

To further validate the scalability characteristics of the proposed architecture, a stress-testing phase was conducted under an enhanced computational configuration. In contrast to the initial single-node CPU-based deployment, two additional GPU-enabled cloud instances were provisioned using Google Colab. Each consumer instance was assigned to a dedicated GPU-backed execution environment. This modification isolates computational scaling effects by alleviating the CPU bottleneck identified in the earlier experiments.

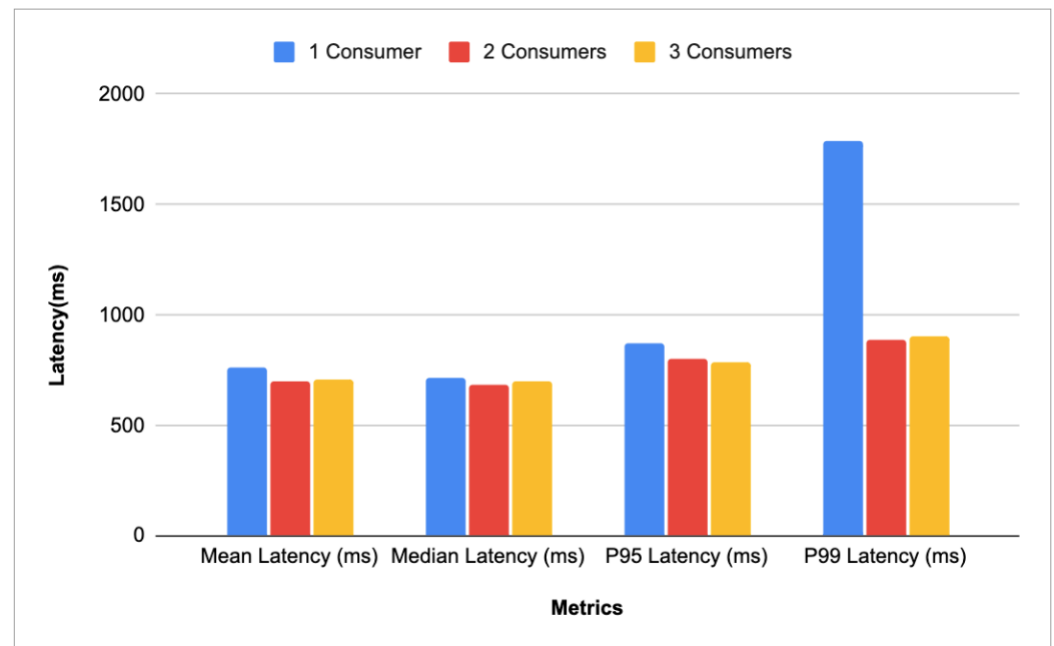
The workload was prepared to ensure a controlled backlog-driven evaluation. Prior to each experiment, relevant Kafka topics were cleared, and all consumers were deactivated to prevent premature message processing. Subsequently, a fixed workload of 3000 requests was generated using Artillery, allowing the consumer group lag to reach its maximum level. Once the backlog was established, consumers were activated, and throughput was measured during the catch-up phase using the Kafka performance tool (`kafka-consumer-perf-test.sh`). The peak throughput was determined from the maximum observed message processing rate (messages/s). This procedure was repeated for configurations with 1, 2, and 3 consumers. Due to the computational characteristics of the YOLO-based inference, where each consumer internally spawns multiple worker processes and heavily utilizes CPU/GPU resources, each consumer instance was deployed on a separate server to avoid resource contention and ensure fair scalability assessment.

For performance testing, the consumer polling interval was reduced to 0.5 s (`consumer.poll(0.5)`) to enable faster message retrieval and minimize idle waiting time. To ensure that the benchmark strictly measures the computational latency of the YOLO-based inference engine and the horizontal scalability of the Kafka consumer group, the output payload was deliberately minimized. Instead of transmitting the annotated output images with detected abnormal mitochondria, the system returns only a lightweight response (e.g., `requestId`). This eliminates external factors such as network serialization overhead and I/O saturation that could otherwise distort the results. Such a design is consistent with real-world practices, where processed images are stored in cloud-based object storage systems, and only a reference URI is returned to the client.

Under this configuration, peak throughput was measured during backlog-drain conditions, thereby capturing the maximum sustainable processing rate of the consumer group. The observed results are depicted in Figure 6.

Table 6. Comparative metrics obtained for all of the scenarios for 60 s.

Metric	1 Consumer	3 Consumers	5 Consumers
Mean Latency (ms)	763.3	703.9	708.4
Median Latency (ms)	713.5	685.5	699.4
P95 Latency (ms)	871.5	804.5	788.5
P99 Latency (ms)	1790.4	889.1	907
Request Rate (req/s)	1/s	1/s	1/s
Total Requests	60	60	60
HTTP 200 Codes	60	60	60
Downloaded Bytes	4020	4020	4020

**Figure 5.** Graphical comparison of mean, median, P95, and P99 latencies under single and parallel consumer configurations.

The results demonstrate a clear superlinear improvement from one to two consumers, followed by continued near-linear scaling from two to three consumers. The increase from 1.53 to 6.09 requests per second indicates that the single-consumer setup was previously limited by computational constraints. These limitations were mainly due to GPU inference throughput and resource contention. Distributing inference workloads across independent GPU-backed instances significantly increased parallel processing capacity. This reduced the processing latency per request and improved overall throughput.

The progression from two to three consumers (6.09 to 11.58 req/s) further confirms the horizontal scalability of the Kafka-based analytical pipeline. The absence of throughput saturation within this range indicates that neither the messaging layer nor the broker became the dominant bottleneck under the tested workload. Instead, system capacity scaled proportionally with the addition of computational resources.

From a scalability validation perspective, these findings confirm that the architecture exhibits effective horizontal scaling when computational resources are provisioned appropriately. To further examine whether horizontal scaling also translates into latency improvements, an additional experiment was conducted under a higher and sustained workload. In this phase, all analytical consumers were deployed exclusively on GPU-enabled Colab servers to eliminate CPU-related bottlenecks and to ensure that inference execution constituted the dominant processing component. This configuration enables

a fair comparison by isolating the impact of consumer parallelism without interference from heterogeneous hardware constraints.

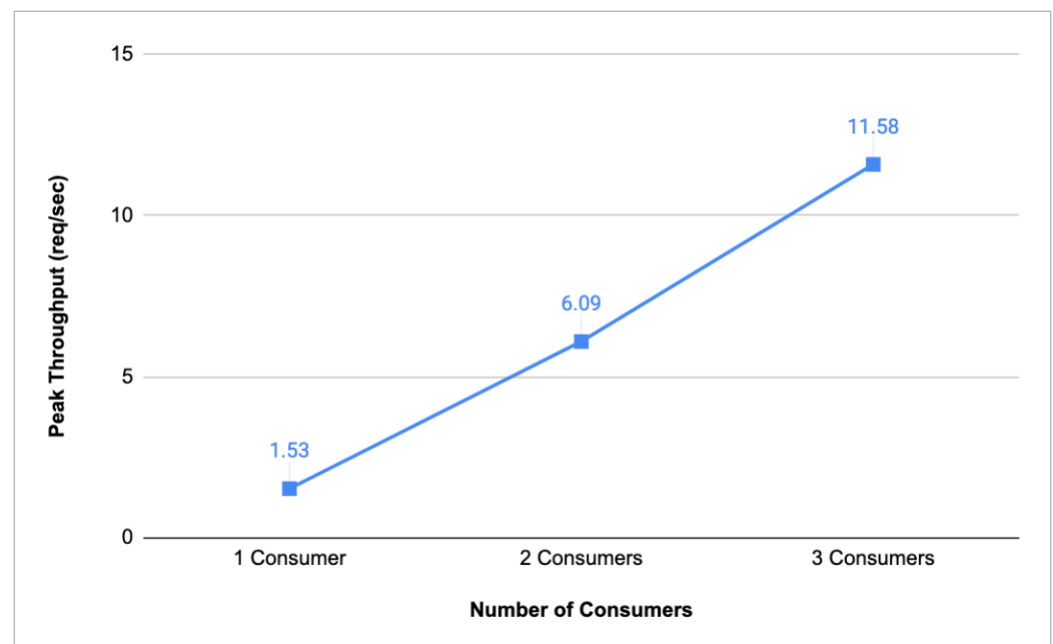


Figure 6. Peak throughput (requests per second) as a function of the number of Kafka consumers, demonstrating the horizontal scalability of the distributed inference architecture under stress-testing conditions.

The workload was increased to a constant rate of three requests per second. This placed the system under significantly higher utilization. Under this elevated load, measurable reductions in latency were observed when the number of consumers was increased from one to two, as illustrated in Figure 7.

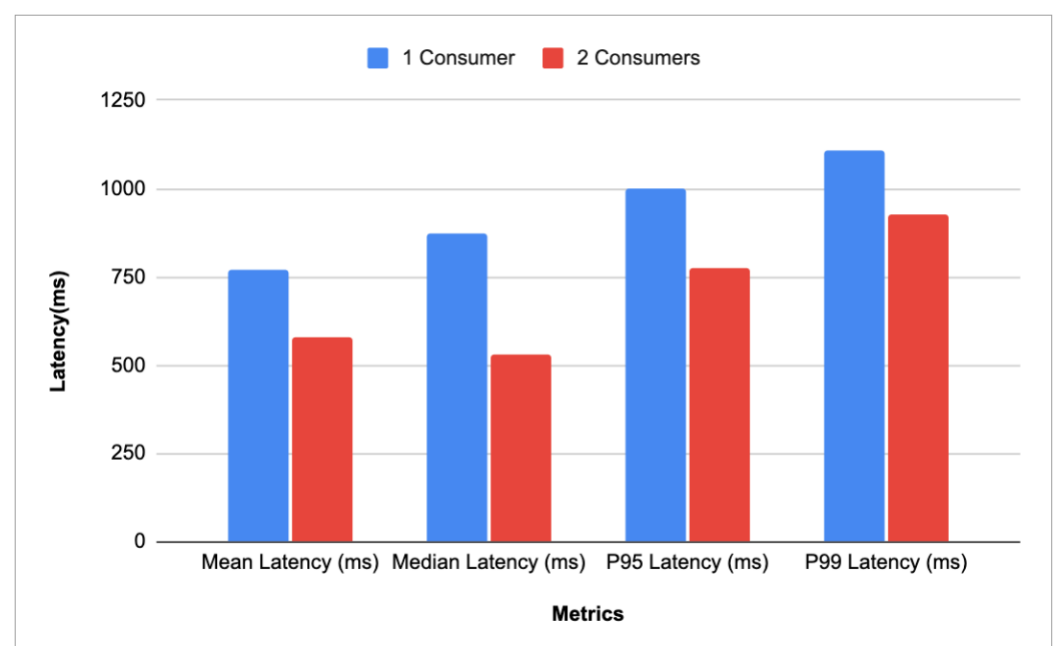


Figure 7. GPU-based latency metrics (mean, median, P95, and P99) as a function of the number of Kafka consumers under constant load, demonstrating reduced response time through horizontal scaling of the distributed inference architecture.

The reduction in mean latency from 770.4 ms to 580.6 ms, along with consistent improvements in median and tail latency metrics, confirms that distributing inference workloads across multiple GPU-backed consumers reduces processing contention and queuing delay. Since the request rate was held constant across both configurations, the observed latency improvements can be directly attributed to the increased parallel processing capacity rather than differences in workload intensity.

Equally important, the total number of requests completed (180) and the number of successful HTTP 200 responses (180) remained identical across both configurations. This result confirms that the system maintained complete processing reliability under increased load, with no failed or dropped requests. The absence of HTTP errors or timeouts demonstrates that the Kafka-based messaging layer, the distributed consumer group, and the web service integration operated in a stable and fault-free manner during the experiment.

These findings provide strong empirical evidence that horizontal scaling of GPU-backed consumers improves latency while preserving full processing correctness and system stability. The simultaneous reduction in latency and preservation of a 100% success rate confirms that the proposed architecture achieves both performance scalability and operational robustness under increased analytical load.

4.4. Discussion

The results demonstrated that the system can perform accurate mitochondria detection with low latency. The findings make it suitable for real-time applications in both research and clinical environments.

From an overall system perspective, these model-level performance characteristics are complemented by the scalability and reliability of the distributed inference infrastructure. The latency reduction observed when increasing the number of GPU-backed consumers was achieved without compromising system reliability or correctness. Under a constant request rate of 3 req/s, both configurations successfully processed all 180 requests, with 100% HTTP 200 response codes and no observed failures or timeouts. This confirms that the latency improvement results directly from enhanced parallel processing capacity and reduced queuing delay, rather than variations in workload or selective request handling. Overall, these results show that the proposed DSS achieves high detection accuracy and fast inference at the model level. It also maintains low-latency, reliable, and scalable performance at the system level. This makes it suitable for real-time deployment in practical biomedical analysis environments.

Despite the demonstrated scalability and low-latency performance, the current study also has several infrastructure-related limitations. First, the experiments were conducted using a single-node Kafka deployment, which may not fully reflect performance behavior in multi-node or geographically distributed clusters. Extending the system to such environments could introduce network latency, partition reassignment overhead, and additional fault-tolerance considerations. Second, while GPU-backed consumers improved throughput and latency, real clinical deployment may present challenges such as heterogeneous hardware availability, integration with hospital information systems, and handling high volumes of diverse EM datasets.

Addressing these limitations will help validate the proposed DSS under realistic operational conditions and guide optimizations for production deployment.

Ethical implications of AI-driven clinical support are also important in this type of study. Using clinical data in AI systems can pose risks to patient privacy and data security. The dataset used in this study contains no personal data. Imbalances in the training data can cause the model to produce erroneous or biased results for specific patient groups. To prevent this, the dataset was used for both training and testing, as originally provided.

Ethical and transparent use is crucial for doctors to trust the system and adopt it in clinical practice. Collaborative work in this area is our future goal.

In summary, the proposed system addresses key gaps in the literature by proposing a high-performance, real-time tool for mitochondria detection. It also contributes a scalable system architecture that can be generalized to broader applications in medical sciences.

In this paper, experimental studies were performed on YOLO-based models. However, the system is suitable for extension to different object detection architectures. In particular, it is possible to integrate models such as Faster R-CNN [62], which uses two-stage detectors, and RetinaNet [63], which adopts a dense prediction approach. However, certain adaptations may be necessary considering the different computational costs of these models. For example, region-recommendation-based models like Faster R-CNN may generate higher inference times and affect real-time performance. In future studies, a comparative evaluation of the proposed method with different detectors will more clearly reveal the generalizability of the approach under different model architectures.

5. Conclusions

This study presents a DSS using mitochondria detection. For the system, YOLOv10 and YOLO26 models were used for detection. Different versions of the YOLOv10 and YOLO26 models were trained and tested for mitochondria detection, and their results were compared. The YOLOv10x model, which was developed to obtain a higher success rate, showed the best performance with a 0.952 mAP score. The smallest inference time was obtained at 2.8 ms with the YOLO26n version. This is a small version developed for faster analysis.

By structuring Kafka topics with multiple partitions and organizing analytics services into consumer groups, the system demonstrates the capacity for horizontal scaling, ensuring stable performance under increasing analytical demand. This scalability was confirmed through throughput and latency measurements in GPU-backed deployments. Peak throughput increased from 1.53 requests per second with a single consumer to 11.58 requests per second with three consumers. Meanwhile, mean latency decreased from 770.4 ms to 580.6 ms when scaling from one to two consumers under a constant load. These improvements were achieved while maintaining a 100% request success rate. This confirms that the performance gains stem from effective workload distribution rather than reduced reliability. The integration of JWT-based authentication and WebSocket communication further complements the architecture by providing secure and low-latency interactions between clients and services. These design choices address common limitations of traditional decision support systems. They result in a flexible, extensible, and production-ready framework. The system delivers accurate, low-latency, and horizontally scalable performance for modern data-intensive applications.

This study also contributes to the literature by comparing different YOLOv10 and YOLO26 versions for mitochondria detection. It can assist experts in detecting and counting mitochondria. Thus, it can be helpful for the early diagnosis of some diseases, such as cancer. Beyond its application-specific contributions, this study also underscores the operational viability of AI-powered scalable image analysis pipelines for biomedical use cases. By leveraging a decoupled Kafka-backed architecture and modern YOLO variants, the system enables just-in-time detection and feedback without sacrificing performance under load.

To implement the proposal in real scenarios, one of the most critical requirements would be access to a sufficiently large and diverse dataset. This includes expert-labeled bounding boxes for mitochondria, ideally verified by domain professionals (e.g., pathologists or cell biologists). Collaboration with medical institutions will be important to collect such datasets under proper ethical and data privacy regulations. It is also important to get feedback about the system from healthcare professionals and to revise the system accordingly.

Author Contributions: G.Y.O., I.O. and C.C. equally contributed to the paper. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The authors declare that all data supporting the findings of this study are available within the article.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A

```

22 # -----
23 # Load The Best Model
24 # -----
25 best_model = YOLO(best_model_path)
26 # -----
27 # Kafka Configuration
28 # -----
29 conf_consumer = {
30     'bootstrap.servers': 'X.101.2.X:9092',
31     'group.id': 'yolo-group',
32     'auto.offset.reset': 'latest'
33 }
34 conf_producer = {
35     'bootstrap.servers': 'X.101.2.X:9092'
36 }
37 # -----
38 # Initialize Kafka Consumer and Producer
39 # -----
40 consumer = Consumer(conf_consumer)
41 producer = Producer(conf_producer)
42 # Subscribe to the input topic
43 consumer.subscribe([INPUT_TOPIC])
44 # -----
45 # Main Kafka Polling + Yolo Prediction Loop
46 # -----
47 try:
48     while True:
49         msg = consumer.poll(1.0)
50         if msg is None:
51             continue
52         if msg.error():...
53         try:
54             # Parse JSON and decode base64 image
55             data = json.loads(msg.value().decode('utf-8'))
56             image_b64 = data['image']
57             image_bytes = base64.b64decode(image_b64)
58             image = Image.open(BytesIO(image_bytes)).convert('RGB')
59             # Run YOLO inference
60             results = best_model(image)
61             # Return resulting image back
62             for r in results:
63                 print(f"Detected {len(r.bboxes)} objects in {data['filename']}")
64                 # Save the annotated result to a BytesIO buffer
65                 annotated_image = r.plot() # This returns a NumPy image (BGR format)
66                 annotated_pil = Image.fromarray(annotated_image[...,:-1]) # Convert BGR to RGB
67                 buffer = BytesIO()
68                 annotated_pil.save(buffer, format="JPEG")
69                 result_image_b64 = base64.b64encode(buffer.getvalue()).decode('utf-8')
70                 # Construct message
71                 result_message = {
72                     'filename': data.get('filename', 'unknown.jpg'),
73                     'num_objects': len(r.bboxes),
74                     'image': result_image_b64,
75                 }
76                 # Send prediction result back to Kafka
77                 producer.produce(OUTPUT_TOPIC, value=json.dumps(result_message).encode('utf-8'))
78                 producer.flush()

```

Figure A1. Partial code snippet of the Kafka consumer used in the analytics module. It demonstrates model loading, message decoding, YOLO-based inference, and result publishing. Parallel processing is achieved by running multiple instances with the same group ID.

References

- Berkani, L. Decision support based on optimized data mining techniques: Application to mobile telecommunication companies. *Concurr. Comput. Pract. Exp.* **2020**, *33*, e5833. [\[CrossRef\]](#)
- Kumari, N.; Acharjya, D.P. A decision support system for diagnosis of hepatitis disease using an integrated rough set and fish swarm algorithm. *Concurr. Comput. Pract. Exp.* **2022**, *34*, e7107. [\[CrossRef\]](#)
- Hafidi, M.; Abdelwahed, E.H.; Qassimi, S. Graph-based tag recommendations using clusters of patients in clinical decision support system. *Concurr. Comput. Pract. Exp.* **2020**, *33*, e5624. [\[CrossRef\]](#)
- Kannan, S. An automated clinical decision support system for predicting cardiovascular disease using ensemble learning approach. *Concurr. Comput. Pract. Exp.* **2022**, *34*, e7007. [\[CrossRef\]](#)
- Singh, A.; Kaur, A.; Dhillon, A.; Ahuja, S.; Vohra, H. Software system to predict the infection in COVID-19 patients using deep learning and web of things. *Softw. Pract. Exp.* **2021**, *52*, 868–886. [\[CrossRef\]](#)
- Campello, S.; Scorrano, L. Mitochondrial shape changes: Orchestrating cell pathophysiology. *EMBO Rep.* **2010**, *11*, 678–684. [\[CrossRef\]](#)
- Cho, D.H.; Nakamura, T.; Lipton, S.A. Mitochondrial dynamics in cell death and neurodegeneration. *Cell. Mol. Life Sci.* **2010**, *67*, 3435–3447. [\[CrossRef\]](#) [\[PubMed\]](#)
- Lesnefsky, E.J.; Moghaddas, S.; Tandler, B.; Kerner, J.; Hoppel, C.L. Mitochondrial Dysfunction in Cardiac Disease: Ischemia-Reperfusion, Aging, and Heart Failure. *J. Mol. Cell. Cardiol.* **2001**, *33*, 1065–1089. [\[CrossRef\]](#) [\[PubMed\]](#)
- Weinberg, R.A. How Cancer Arises. *Sci. Am.* **1996**, *275*, 62–70. [\[CrossRef\]](#)
- Modica-Napolitano, J.; Kulawiec, M.; Singh, K. Mitochondria and Human Cancer. *Curr. Mol. Med.* **2007**, *7*, 121–131. [\[CrossRef\]](#)
- Fuchs, M.; Tsourakis, N.; Rayner, M. A Scalable Architecture For Web Deployment of Spoken Dialogue Systems. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*; Calzolari, N., Choukri, K., Declerck, T., Doğan, M.U., Maegaard, B., Mariani, J., Moreno, A., Odijk, J., Piperidis, S., Eds.; European Language Resources Association (ELRA): Istanbul, Türkiye, 2012; pp. 1309–1314.
- Yazdani, A.; Safdari, R.; Ghazisaeedi, M.; Beigy, H.; Sharifian, R. Scalable Architecture for Telemonitoring Chronic Diseases in Order to Support the CDSSs in a Common Platform. *Acta Inform. Med.* **2018**, *26*, 195. [\[CrossRef\]](#)
- Ed-daoudy, A.; Maalmi, K.; El Ouazizi, A. A scalable and real-time system for disease prediction using big data processing. *Multimed. Tools Appl.* **2023**, *82*, 30405–30434. [\[CrossRef\]](#)
- Omar Said, A.T. SEAIoT: Scalable E-Health Architecture based on Internet of Things. *Int. J. Comput. Appl.* **2012**, *59*, 44–48. [\[CrossRef\]](#)
- Bahmani, A.; Alavi, A.; Buerge, T.; Upadhyayula, S.; Wang, Q.; Ananthakrishnan, S.K.; Alavi, A.; Celis, D.; Gillespie, D.; Young, G.; et al. A scalable, secure, and interoperable platform for deep data-driven health management. *Nat. Commun.* **2021**, *12*, 5757. [\[CrossRef\]](#)
- Guedria, S.; De Palma, N.; Renard, F.; Vuillerme, N. R2D2: A scalable deep learning toolkit for medical imaging segmentation. *Softw. Pract. Exp.* **2020**, *50*, 1966–1985. [\[CrossRef\]](#)
- Toprak, A. Determination of Colorectal Cancer and Lung Cancer Related lncRNAs based on Deep Autoencoder and Deep Neural Network. *Int. J. Comput. Exp. Sci. Eng.* **2024**, *10*, 1893–1900. [\[CrossRef\]](#)
- Rao, B.D.; Madhavi, K. BCDNet: A Deep Learning Model with Improved Convolutional Neural Network for Efficient Detection of Bone Cancer Using Histology Images. *Int. J. Comput. Exp. Sci. Eng.* **2024**, *10*, 988–998. [\[CrossRef\]](#)
- Huang, W.; Cai, X.; Yan, Y.; Kang, Y. MA-DenseUNet: A Skin Lesion Segmentation Method Based on Multi-Scale Attention and Bidirectional LSTM. *Appl. Sci.* **2025**, *15*, 6538. [\[CrossRef\]](#)
- undefinedelebi, S.B.; Emiroğlu, B.G. A Novel Deep Dense Block-Based Model for Detecting Alzheimer's Disease. *Appl. Sci.* **2023**, *13*, 8686. [\[CrossRef\]](#)
- Soylu, E. A Deep Transfer Learning-Based Comparative Study for Detection of Malaria Disease. *Sak. Univ. J. Comput. Inf. Sci.* **2022**, *5*, 427–447. [\[CrossRef\]](#)
- Guler, R.; Karapınar Senturk, Z.; Gamsızkan, M.; Özcan, Y. Diagnosis of Lichen Sclerosus, Morphea, and Vasculitis Using Deep Learning Techniques on Histopathological Skin Images. *Sak. Univ. J. Comput. Inf. Sci.* **2025**, *8*, 312–321. [\[CrossRef\]](#)
- Xiao, C.; Chen, X.; Li, W.; Li, L.; Wang, L.; Xie, Q.; Han, H. Automatic Mitochondria Segmentation for EM Data Using a 3D Supervised Convolutional Network. *Front. Neuroanat.* **2018**, *12*, 92. [\[CrossRef\]](#) [\[PubMed\]](#)
- Yuan, Z.; Ma, X.; Yi, J.; Luo, Z.; Peng, J. HIVE-Net: Centerline-Aware Hierarchical View-Ensemble Convolutional Network for Mitochondria Segmentation in EM Images. *arXiv* **2021**, arXiv:2101.02877. [\[CrossRef\]](#)
- Oztel, I.; Yolcu, G.; Ersoy, I.; White, T.A.; Bunyak, F. Deep learning approaches in electron microscopy imaging for mitochondria segmentation. *Int. J. Data Min. Bioinform.* **2018**, *21*, 91. [\[CrossRef\]](#)
- Lucchi, A.; Smith, K.; Achanta, R.; Knott, G.; Fua, P. Supervoxel-Based Segmentation of Mitochondria in EM Image Stacks with Learned Shape Features. *IEEE Trans. Med. Imaging* **2012**, *31*, 474–486. [\[CrossRef\]](#)

27. Seyedhosseini, M.; Ellisman, M.H.; Tasdizen, T. Segmentation of mitochondria in electron microscopy images using algebraic curves. In *Proceedings of the 2013 IEEE 10th International Symposium on Biomedical Imaging, San Francisco, CA, USA, 7–11 April 2013*; IEEE: Piscataway, NJ, USA, 2013; pp. 860–863. [\[CrossRef\]](#)
28. Zahedi, A.; On, V.; Phandthong, R.; Chaili, A.; Remark, G.; Bhanu, B.; Talbot, P. Deep Analysis of Mitochondria and Cell Health Using Machine Learning. *Sci. Rep.* **2018**, *8*, 16354. [\[CrossRef\]](#)
29. Oztel, I.; Yolcu, G.; Ersoy, I.; White, T.; Bunyak, F. Mitochondria segmentation in electron microscopy volumes using deep convolutional neural network. In *Proceedings of the 2017 IEEE International Conference on Bioinformatics and Biomedicine (BIBM), Kansas City, MO, USA, 13–16 November 2017*; IEEE: Piscataway, NJ, USA, 2017; pp. 1195–1200. [\[CrossRef\]](#)
30. Casser, V.; Kang, K.; Pfister, H.; Haehn, D. Fast Mitochondria Detection for Connectomics. In *Proceedings of the Third Conference on Medical Imaging with Deep Learning, Montreal, QC, Canada, 6–8 July 2020*; Arbel, T., Ben Ayed, I., de Bruijne, M., Descoteaux, M., Lombaert, H., Pal, C., Eds.; PMLR, Proceedings of Machine Learning Research: Cambridge, MA, USA, 2020; Volume 121, pp. 111–120.
31. Liu, J.; Li, L.; Yang, Y.; Hong, B.; Chen, X.; Xie, Q.; Han, H. Automatic Reconstruction of Mitochondria and Endoplasmic Reticulum in Electron Microscopy Volumes by Deep Learning. *Front. Neurosci.* **2020**, *14*, 599. [\[CrossRef\]](#)
32. Fischer, C.A.; Besora-Casals, L.; Rolland, S.G.; Haeussler, S.; Singh, K.; Duchen, M.; Conradt, B.; Marr, C. MitoSegNet: Easy-to-Use Deep Learning Segmentation for Analyzing Mitochondrial Morphology. *iScience* **2020**, *23*, 101601. [\[CrossRef\]](#)
33. Savojardo, C.; Bruciaferri, N.; Tartari, G.; Martelli, P.L.; Casadio, R. DeepMito: Accurate prediction of protein sub-mitochondrial localization using convolutional neural networks. *Bioinformatics* **2019**, *36*, 56–64. [\[CrossRef\]](#)
34. Hou, Z.; Yang, Y.; Li, H.; Wong, K.c.; Li, X. iDeepSubMito: Identification of protein submitochondrial localization with deep learning. *Briefings Bioinform.* **2021**, *22*, bbab288. [\[CrossRef\]](#) [\[PubMed\]](#)
35. Yolcu Oztel, G. Automated detection of mitochondria in EM images with YOLOv10 for robust mitochondria segmentation. In *Proceedings of the 3rd International Paris Applied Science Congress Proceedings Book, Paris, France, 14–16 December 2024*; p. 118.
36. Mumcuoglu, E.; Hassanpour, R.; Tasel, S.; Perkins, G.; Martone, M.; Gurcan, M. Computerized detection and segmentation of mitochondria on electron microscope images. *J. Microsc.* **2012**, *246*, 248–265. [\[CrossRef\]](#)
37. Hu, J.; Xiao, C.; Shen, L.; Xie, Q.; Chen, X.; Han, H. Automatic detecting and connecting the mitochondria from the serial EM images. In *Proceedings of the 2017 IEEE International Conference on Mechatronics and Automation (ICMA), Takamatsu, Japan, 6–9 August 2017*; IEEE: Piscataway, NJ, USA, 2017; pp. 1632–1637. [\[CrossRef\]](#)
38. Li, R.; Zeng, X.; Sigmund, S.E.; Lin, R.; Zhou, B.; Liu, C.; Wang, K.; Jiang, R.; Freyberg, Z.; Lv, H.; et al. Automatic localization and identification of mitochondria in cellular electron cryo-tomography using faster-RCNN. *BMC Bioinform.* **2019**, *20*, 132. [\[CrossRef\]](#)
39. Dulal, R.; Dulal, R. Brain Tumor Identification using Improved YOLOv8. *arXiv* **2025**, arXiv:2502.03746. [\[CrossRef\]](#)
40. Wang, X.; Wu, H.; Wang, L.; Chen, J.; Li, Y.; He, X.; Chen, T.; Wang, M.; Guo, L. Enhanced pulmonary nodule detection with U-Net, YOLOv8, and swin transformer. *BMC Med. Imaging* **2025**, *25*, 247. [\[CrossRef\]](#)
41. Chen, J.; Lu, Y.; Yu, Q.; Luo, X.; Adeli, E.; Wang, Y.; Lu, L.; Yuille, A.L.; Zhou, Y. TransUNet: Transformers Make Strong Encoders for Medical Image Segmentation. *arXiv* **2021**, arXiv:2102.04306. [\[CrossRef\]](#)
42. Anari, P.Y.; Obiezu, F.; Lay, N.; Dehghani Firouzabadi, F.; Chaurasia, A.; Golagha, M.; Singh, S.; Homayounieh, F.; Zahergivar, A.; Harmon, S.; et al. Using YOLO v7 to Detect Kidney in Magnetic Resonance Imaging. *arXiv* **2024**, arXiv:2402.05817. [\[CrossRef\]](#)
43. Apache Software Foundation. A Distributed Streaming Platform. 2025. Available online: <http://kafka.apache.org/documentation/> (accessed on 26 March 2026).
44. Amin, R.A.; Hasan, M.; Wiese, V.; Obermaisser, R. FPGA-Based Real-Time Object Detection and Classification System Using YOLO for Edge Computing. *IEEE Access* **2024**, *12*, 73268–73278. [\[CrossRef\]](#)
45. Vijayakumar, A.; Vairavasundaram, S. YOLO-based Object Detection Models: A Review and its Applications. *Multimed. Tools Appl.* **2024**, *83*, 83535–83574. [\[CrossRef\]](#)
46. Atıcı, H.; Kocer, H.E.; Sivrikaya, A.; Dagli, M. Analysis of Urine Sediment Images for Detection and Classification of Cells. *Sak. Univ. J. Comput. Inf. Sci.* **2023**, *6*, 37–47. [\[CrossRef\]](#)
47. Ragab, M.G.; Abdulkadir, S.J.; Muneer, A.; Alqushaibi, A.; Sumiea, E.H.; Qureshi, R.; Al-Selwi, S.M.; Alhussian, H. A Comprehensive Systematic Review of YOLO for Medical Object Detection (2018 to 2023). *IEEE Access* **2024**, *12*, 57815–57836. [\[CrossRef\]](#)
48. Ahmadyar, Y.; Kamali-Asl, A.; Arabi, H.; Samimi, R.; Zaidi, H. Hierarchical approach for pulmonary-nodule identification from CT images using YOLO model and a 3D neural network classifier. *Radiol. Phys. Technol.* **2023**, *17*, 124–134. [\[CrossRef\]](#)
49. Tarimo, S.A.; Jang, M.A.; Ngasa, E.E.; Shin, H.B.; Shin, H.; Woo, J. WBC YOLO-ViT: 2 Way—2 stage white blood cell detection and classification with a combination of YOLOv5 and vision transformer. *Comput. Biol. Med.* **2024**, *169*, 107875. [\[CrossRef\]](#)
50. Sriram, N.; Jayalakshmi, V.; Preethi, P.; Shoba, B.; Shenbagavalli, K. Navigating the Future with YOLOv9 for Advanced Traffic Sign Recognition in Autonomous Vehicles. *Int. J. Comput. Exp. Sci. Eng.* **2024**, *10*, 1424–1436. [\[CrossRef\]](#)

51. Liang, S.; Xu, H.; Liu, J.; Li, J.; Pan, H. YOLOv8n-GSS-Based Surface Defect Detection Method of Bearing Ring. *Sensors* **2025**, *25*, 6504. [[CrossRef](#)]
52. Kim, H.; Kim, T.K. Design and Implementation of a YOLOv2 Accelerator on a Zynq-7000 FPGA. *Sensors* **2025**, *25*, 6359. [[CrossRef](#)]
53. Yang, C.; Shen, Y.; Wang, L. EMFE-YOLO: A Lightweight Small Object Detection Model for UAVs. *Sensors* **2025**, *25*, 5200. [[CrossRef](#)]
54. Wang, A.; Chen, H.; Liu, L.; Chen, K.; Lin, Z.; Han, J.; Ding, G. YOLOv10: Real-Time End-to-End Object Detection. *arXiv* **2024**, arXiv:2405.14458. [[CrossRef](#)]
55. Jocher, G.; Qiu, J. *Ultralytics YOLO26*; Ultralytics Inc.: Frederick, MD, USA, 2026.
56. Lucchi, A.; Li, Y.; Fua, P. Learning for Structured Prediction Using Approximate Subgradient Descent with Working Sets. In *Proceedings of the 2013 IEEE Conference on Computer Vision and Pattern Recognition, Portland, OR, USA, 23–28 June 2013*; IEEE: Piscataway, NJ, USA, 2013; pp. 1987–1994. [[CrossRef](#)]
57. Lucchi, A.; Marquez-Neila, P.; Becker, C.; Li, Y.; Smith, K.; Knott, G.; Fua, P. Learning Structured Models for Segmentation of 2-D and 3-D Imagery. *IEEE Trans. Med. Imaging* **2015**, *34*, 1096–1110. [[CrossRef](#)]
58. Roboflow. Roboflow: Computer Vision Tools for Developers and Enterprises. Available online: <https://roboflow.com/> (accessed on 15 September 2025).
59. Ultralytics. Adam Optimizer: Deep Learning. Available online: <https://www.ultralytics.com/glossary/adam-optimizer> (accessed on 15 September 2025).
60. Adam Optimizer. Available online: <https://github.com/ultralytics/yolov5> (accessed on 15 September 2025).
61. Artillery Software. Artillery Docs. 2025. Available online: <https://www.artillery.io/docs/> (accessed on 15 September 2025).
62. Ren, S.; He, K.; Girshick, R.; Sun, J. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *arXiv* **2016**, arXiv:1506.01497. [[CrossRef](#)] [[PubMed](#)]
63. Lin, T.Y.; Goyal, P.; Girshick, R.; He, K.; Dollár, P. Focal Loss for Dense Object Detection. *arXiv* **2018**, arXiv:1708.02002. [[CrossRef](#)]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.