IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 14 February 2023, accepted 2 March 2023, date of publication 16 March 2023, date of current version 21 March 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3253767

RESEARCH ARTICLE

DroMOD: A Drone-Based Multi-Scope Object Detection System

TAKOUA ABDELLATIF[1,2], MOHAMED ALI SEDRINE[3], AND YASSINE GACHA[1,4]

[1]SERCOM Laboratory, University of Carthage, Carthage 1054, Tunisia
[2]ENISo, University of Sousse, Sousse 4002, Tunisia
[3]SACES, Aviation School of Borj El Amri, Borj El Amri 1142, Tunisia
[4]National Center for Mapping and Remote Sensing (CNCT), El Aouina 1080, Tunisia

Corresponding author: Takoua Abdellatif (Takoua.Abdellatif@ept.rnu.tn)

This work was supported in part by the Tunisian Research Project: Big spatio-temporel data for a smart and sustainable city under Reference PR19CNCT02, and in part by the National Center for Mapping and Remote Sensing, CNCT, El Aouina, Tunisia.

ABSTRACT Drone-based Multi-scope Object Detection (DroMOD) system targets an efficient detection of different kinds of objects using drones. DroMOD relies on a cross-platform framework where objects' detection tasks are shared between the drone and the server. The drone stores a set of reference images about the supervised zone. Each time an image is captured with a well-defined spatial frequency, it is compared to its reference image and only 'trigger images' showing a change from the reference image are sent to the server. A Big Data streaming platform is deployed on server-side for scalable and efficient real-time object detection processing based on Deep Learning (DL) models. DroMOD system architecture allows for dynamically upgrading the DL models so that newly considered objects to detect can be added to the drone mission on the fly without modifying the drone embedded software. When compared to existing alternatives, DroMOD presents the best compromise between 1) object detection accuracy, 2) real-time processing, and 3) resource efficiency. Since only lightweight processing is performed on the drone-side, memory and computation are highly optimised on drones. Furthermore, the drone-side image filtering is independent of the objects to detect and the object detection programs are deployed and updated only on the server-side which allows for multi-target detection with minimal engineering efforts and expertise.

INDEX TERMS Big data, deep learning, drone and server-side processing collaboration, image comparison, key frame selection, multi-scope object detection.

I. INTRODUCTION

Multi-scope object detection is about automatically detecting multiple objects from different categories. It is used in several domains, such as video surveillance [1], military zones and public facilities security. For example, in restricted military zones, the surveillance can be about detecting different kinds of intrusions (persons, vehicles, drones, etc.) or different kinds of natural disasters (flood, fire, etc.). Each zone has specific properties to be identified for building accurate object detection surveillance system. Indeed, the nature of the zone (jungle, mountains, city, desert…), acquisition time (night, day) and camera sensor (thermal, RGB, etc.) guide the selection of an object-detection technique. Besides, the system needs permanent updates to adapt to the required changing surveillance tasks and objects to detect.

In this context, drone-based object detection is adopted thanks to embedded drone cameras and resources. Nevertheless, as an embedded system and a flying engine, the drone has limited resources (CPU, memory and energy). This prevents the use of heavy-algorithms such as real time DL algorithms for multi-object detection [2] and complex and convolutional neural network models [3]. In addition, the deployment and the setup of surveillance applications in drones raises the problem of versatility and hardware compatibility. Moreover, it is challenging to update and to extend such systems when adding new targets to detect. This is mainly due to constraints of autonomy, limited resources and to the aeronautical constraints.

To tackle those challenges, server-side processing is adopted. Nevertheless, having the processing exclusively

The associate editor coordinating the review of this manuscript and approving it for publication was Nitin Gupta.

deployed on server-side requires extensive bandwidth usage for the server-drone communication. To countermeasure this issue, hybrid solutions are adopted where image processing is both performed on drone-side and on server-side. Such solutions reach the good compromise between the usage of server resources and the optimisation of the drone-server communication bandwidth [4].

In this work, we build DroMOD, a cross-platform framework that relies on a hybrid solution. Objects' detection tasks are shared between the drone and the server. The drone stores a set of reference images about the supervised zone. Each time an image is captured, it is compared to its reference image and only 'trigger images' showing a change from the reference image are sent to the server. A Big Data streaming platform is deployed on server-side for scalable and efficient real-time object detection processing based on DL models. DroMOD system architecture allows for dynamically upgrading the DL models so that newly considered objects to detect can be added to the drone mission on the fly without modifying the drone embedded software. When compared to existing alternatives, DroMOD presents the best compromise between (i) object detection accuracy, (ii) real-time processing and (iii) resource efficiency. Indeed, Big Data platforms allow for the execution of accurate and resource-consuming DL programs. Since only lightweight processing is performed on the drone-side, memory consumption and computation are reduced on drones while complex model development and deployment are performed on the server-side. Furthermore, the drone-side image filtering is independent of the objects to detect since the object detection programs are deployed and updated only on the server-side. This allows for a simple update of objects to detect without modifying the embedded software.

We evaluate DroMOD by comparing it to both drone-side and server-side solutions as well as to recent hybrid solutions. For performance evaluation, we consider processing latency, object detection's model accuracy and resource consumption. The results show that DroMOD presents the lowest processing latency and an optimal resource consumption without degrading detection accuracy.

Our main contributions are:

- An efficient embedded image filtering method using a lightweight DL model.
- A novel drone-based IoT system that allows for fulfilling three constraints at the same time: (i) real-time object detection, (ii) optimised resource consumption and (iii) high detection accuracy.
- A Big Data system for real-time image processing. We mainly justify the technology choices and propose an integration solution to a scalable Big Data stream technology with an accurate DL processing engine.

The paper is structured into five parts. First, we present the related work on real-time processing in drone-based applications. The second, third and fourth section illustrate respectively DroMOD system presentation, used algorithms and implementation.Then, the evaluation section compares DroMOD with recent state-of-the-art solutions. Finally, we conclude the paper and outline the major works to come.

II. RELATED WORK

Building drone-based real-time applications is a challenging task. Several solutions are proposed to tackle those challenges. Based on data processing location, we can classify those solutions to embedded drone-side processing, server-side processing and hybrid solutions. We end the section with a presentation of image filtering methods for anomaly detection.

A. EMBEDDED DRONE-SIDE PROCESSING

Drone-side embedded processing is efficient for real-time surveillance applications. In recent works, DL programs are intensively used for embedded image processing. Researchers focus mainly at counter-measuring limited resources on drones while increasing their DL models' accuracy. For example, in [5], authors present a drone-embedded DL-based object detection system to detect faults in power line components. To tackle the drone's limited on-board resources, the model inference is done on an embedded GPU-based platform (e.g., Nvidia Jetson AGX Xavier) to accelerate the execution. The experimental results show the effectiveness and efficiency of the method for fully automatic and real-time on-board visual power line inspection. Nevertheless, the use of the GPU platforms induces a high battery consumption so a decrease of drone autonomy. Besides, the deployment the model inference is highly complex.

In [6], authors use the YOLOv3 [7] model trained on multiple eye-sky dataset. To decrease processing latency on embedded hardware, they apply optimisation steps, fusing layers, quantizing calculations to 16-bit floats and 8-bit integers. In addition to the problem of high energy consumption, this solution is not extensible and not really scalable. Furthermore, the object detection models have to be adapted to the specific embedded hardware.

Other solutions consist in designing optimal object detection models for drone on-board implementation. Y. Bazi et al. [8] introduce a novel convolutional support vector machine (CSVM) network for object detection in drone imagery. Compared with others, this model is light and requires a small amount of training data but it is not used of multi-object detection. In [9], authors propose ShuffleDet neural network for real-time vehicle detection to be used on-board by mobile platforms such as UAVs. ShuffleDet network is composed of ShuffleNet [10] and a modified variant of SSD [11] based on channel shuffling and grouped convolution. This model is tested on GPU platform nano Jetson TX2. The results show that this model has a low latency but a low accuracy. In addition, this model is not adapted for multi-target object detection.

B. SERVER-SIDE PROCESSING

All dataflow processing in executed on the server/the cloud. A communication system between drones and servers/clouds are intensively used to transmit data.

In [12], authors propose a cloud implementation of gait human recognition system from drones. The Cloud server uses the input drone continuous streaming video to detect the gait. It uses Single Shot Multibox Detector (SSD) for detecting human and Inception-V3 [13] based transferred learning convolutional neural networks (CNNs) to extract spatial features that are inserted into Long Short-Term Memory (LSTM) [14] deep architecture for action recognition. Dick et al. [15] use an intermediate server at the edge to optimise processing capability and reduce latency for real-time applications. This solution does not solve the problem of limited bandwidth of the server-drone network and the large amount of data over an extended period of time increases system latency.

C. HYBRID SOLUTIONS

Hybrid processing is an interesting solution for drone-based real-time applications. It provides a compromise between resource consumption and processing delay thanks to a two-part processing on the drone-side and on the server-side.

Wang et al. [16] present the EdgeDuet framework that allows for a collaboration between the drone and the edge/cloud to detect objects. Locally, the drone detects large objects using low-resolution images, while on the edge/cloud, the detection is done for small objects via high-resolution uploaded images. EdgeDuet optimises the object detection in tiles with adapted resolution by only keeping the pixel blocks containing small objects in high quality while compressing the rest of the frame to low quality. This avoids delivering the complete frame, resulting in high accuracy and low latency. This framework results in a highly complex embedded development and deployment. In addition, it causes high embedded processing for creating tiles and resolution adaptation, which increases processing latency and resource consumption on the drone-side that is usually resource constrained and has low energy autonomy. Furthermore, in the case of adding another object to detect, EdgeDuet requires an update in the drone embedded code to be able to detect the ''large version'' of this object, which induces a high deployment complexity. DroMOD overcomes these points by a lightweight drone-side algorithm which is independent of the object to detect. In more recent work, Liu et al. [17] present AdaMask, a stream video framework to remote deep neural network (DNN) inference. They propose a frame masking as an effective mechanism to reduce the network consumption of video streams, which keep only regions that potentially contain objects of interest. Additionally, AdaMask controls the resolution, frame rate and quantization rate to reduce the sending data size. This framework is tested for dash cameras, traffic surveillance and drones. The results show an optimisation of network bandwidth which decreases latency and an increase of the accuracy. In the case of drones, AdaMask presents a highly embedded complex algorithm that causes a high resource consumption. In addition, the server-side processing is not treated for multiple object detection and distributed computing. In DroMOD, the embedded code in the drone is the same regardless of the used-for application since it is about detecting a change in terms of an intrusive object (e.g., fire, person, flood, etc.). Alam et al. [18] use a hybrid based solution for real-time abnormal event detection from drones. They propose the use of one stage object detection (tiny YOLO [19]) on the drone-side to detect the presence of a person on the frame. If a person is detected during 3s, the sequence is sent to the server. The detection of abnormal events is done on the server-side. This method is highly sensitive to the accuracy of on-board object detection models which in this case present a low accuracy compared with other models. The DroMOD drone-embedded filtering method has a high accuracy to detect an intrusive object, which avoids the ''False Positives''.

D. IMAGE FILTERING FOR ANOMALY DETECTION

In DroMOD, we propose to perform drone-side image filtering based on a comparison of images taken from the same place at different times. We try to detect intrusions and changes in the same scene, but at different points in the same scene. We propose to use techniques known as visual place recognition. In the literature, we distinguish two main categories of visual place recognition techniques. They differ in the representation of the image. First, appearance-based methods, called handcrafted, have been applied for a long time. Then, more recent works have exploited advances in the field of machine learning and neural networks. The handcrafted techniques use local or global descriptors to model an image from a location. Local descriptors such as SURF, SIFT or FAST are used to build Bags of Visual Words. Each image is modeled by a vector (or histogram) of dimension n, where n is the number of words in the visual vocabulary. Each component of the vector is equal to the number of occurrences of a specific word of the previously constructed vocabulary. Thus, the similarity of two images is given by the scalar product of their respective normalized vectors. Global descriptors like GIST are also used to compact all the information of a scene into a vector. In Murillo and Kosecka et al. [20], the authors applied this descriptor to a set of panoramic set of panoramic images for a large scale localisation. However, recent works, such as [21], show that features generated by CNNs are more robust and discriminative. Thus, the performance of neural network-based visual location recognition methods is superior to that of handcrafted methods.

Currently, Convolutional Neural Networks (CNN) are used as robust feature extractors for location recognition, including for environments. They are able to generate generic and abstract representations of places. They contain more semantic information. In [22], authors introduced an original model. It allows to give a similarity score between two images. It is a Siamese CNN, made of two branches. Each branch is based on YOLOv5 architecture [23]. Authors proposed to keep the YOLOv5 backbone and replace its head by a 3-layers block. Namely, an adaptive average pool layer which down samples

the feature map from [512 × 7 × 7] to [512 × 1 × 1], followed by a flattening layer to get a 1-d feature map which undergoes a linear transformation, and finally a sigmoid activation function layer to bound the output. Each branch output shape is [1000 × 1]. Then, in order to measure the similarity, the contrastive loss [24] between the branches output vectors is computed. The smaller the distance is, the more similar the images are [22] showed that this model is lightweight and fast. Thus, it is suitable to be embedded on drone. Therefore, we decided to use this model in this work.

III. DroMOD SYSTEM DESCRIPTION

Drone Multi-scope Object Detection (DroMOD) architecture is based on three main components in three different environments (Drone, Server, Third-party services) (see figure 1). The drone-recorded data is sent to an on-board Receiver (step 1) which is responsible for raw-data pre-processing. The Receiver sends data to a Controller (step 2) which detects intrusions thanks to image comparison with an on-board-drone reference memory constructed using a Reference Builder module (step 3). In case of intrusion detection, the Controller sends data via a Gateway (step 4) that in turn publishes data to a Publisher/Subscriber broker (step 5), specifically, to a frame processing topic. A Stream processing engine combined with DL models for object detection consumes data from the frame processing topic (step 6). It analyses the frame for fine-grained object detection. In case of an object detection, it publishes a message to the correspondent 'object detection topic' (step 7). The third-party services registered to the 'object detection topic' receive the notification (step 8) and a permanent Storage component stores all the historical data(step 9).

A. DRONE COMPONENTS

The drone is composed of three modules to deal with the drone data stream in order to assure efficient data filtering by sending data when there is a change. The drone components are the following:

1) RECEIVER

The Receiver receives the data flow from on-board-drone captures. It is responsible for raw data pre-processing. It is used in order to ensure the adequate data form for the achievement of next tasks like stream discretization, normalization, verification of empty data, reshap, etc.

2) REFERENCE BUILDER

Reference builder is in charge of creating the reference memory. It identifies and stores a set of key frames representing the no-intrusion case images. This key frames set is used to compare the acquired drone data with the corresponding reference data.

3) CONTROLLER

The drone gets the pre-processed data from the receiver and compares it with the corresponding in-memory reference data. We divide the controller's work into two phases: (i) finding the reference image corresponding to the captured image to analyse and (ii) making a comparison between the two images for intrusion detection using a DL model. The details of the controller algorithm are covered in the next section.

B. SERVER COMPONENTS

The server components are based on Big Data technologies. The technologies are required for the storage and for the processing of the huge volume data of drone images generally sent with high velocity. Furthermore, to fulfil the real-time surveillance applications' requirements, streaming engines allows for the image processing on the fly with short latency. Compared to classic technologies, Big Data solutions are more efficient in optimized distributed computing, high fault tolerance, horizontal scalability and load balancing. Server components are the following:

1) PUBLISHER/SUBSCRIBER BROKER

The Publish/Subscribe broker is a core compound of the Big Data architecture. It is a set of brokers that allows for the transmission of dataflow between different parts of the architecture using connectors. We define a 'frame processing' topic that receives all data sent by the gateway. This data is consumed by the stream processing engine. The broker has other topics related to the events of interest (like fire detection, person detection, etc.). The stream processing engine publishes data on these topics following the detected objects. The subscribers are third-party services. Storage component is a special subscriber for all topics to provide a permanent storage of all data flow and event alerts.

2) STREAM PROCESSING ENGINE WITH DL MODELS

The stream processing engine is the processing server component that enables real-time, low-latency treatment of drone streams. This engine is both a consumer and a producer for the Publish/Subscribe broker. It consumes data from processing topics and uses DL models to analyse and identify objects of interest (fire, person intrusion, etc.). When an event is detected, a message is sent to the corresponding topic in order to notify third-party services.

C. THIRD PARTY SERVICES

As a multi-scope surveillance system, variant services can be concerned about detected events. It is important that each service gets notified about the event occurrence as soon as possible. Thanks to the Publish/Subscribe broker, each service can subscribe to the topic of interest. For example, emergency services can register to 'flood topic' and to 'fire topic'. Necessary actions are then scheduled and executed.

IV. DroMOD ALGORITHMS

In this section, the main algorithms executed at drone-side components are presented, namely the reference memory builder algorithm and the controller's algorithm.

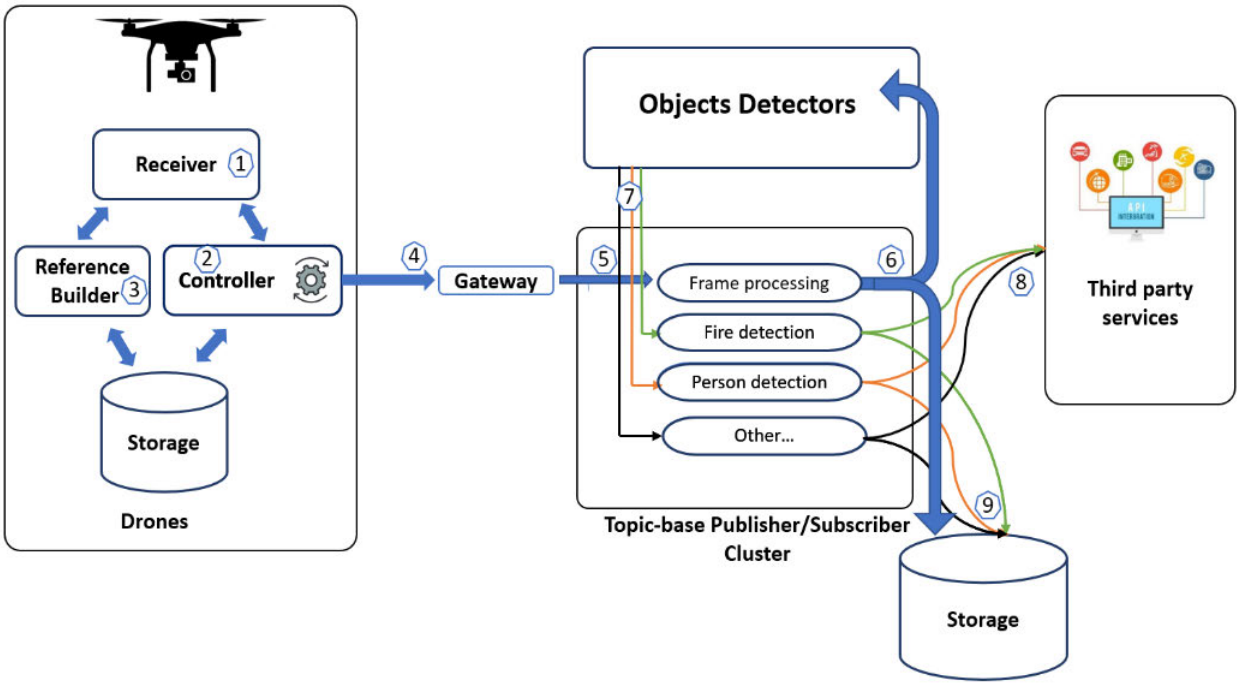

FIGURE 1. DroMOD architecture.

A. REFERENCE MEMORY BUILDER

The generated data by a flying drone in a defined fixed closed trajectory ($\mathcal{T}$) and for a mission range time T can be represented by the set ($\mathbb{D}$) defined in (1).

$$\mathbb{D} = \{data(M_i, t_i) \,/M \in (\mathcal{T}), t_i \in T\} \tag{1}$$

where $data(M_i, t_i)$ is the captured data in the position M_i and at the time t_i. $\mathbb{D}$ is a discrete set since the captured images are measured within an fixed time period which is defined by the camera frame rate (a characteristic of the used Camera).

The reference memory builder allows the detection of 'trigger' image to be sent to the server. A dedicated flight in the area of interest allows to capture reference images and storing them on board. The flight trajectory ($\mathcal{T}$) is split according to a fixed distance step "**a**" (figure 2) that defines ($\mathcal{T}_a$). ($\mathcal{T}_a$) is obtained by a partitioning of ($\mathcal{T}$) with a fixed spatial frequency. It is defined recursively in (2).

$$\begin{cases} M_0 \in (\mathcal{T}_a), \\ M_i \in (\mathcal{T}_a), \quad \text{if } \| \widehat{M_i, M_{i+1}} \| = \mathbf{a} \text{ and } M_0 \notin \widehat{M_i, M_{i+1}}. \end{cases} \tag{2}$$

M_0 the initial point which is the beginning of image capture. $\widehat{M_i, M_{i+1}} \subset (\mathcal{T})$ is a continuous part of the drone trajectory between the point M_i and M_{i+1} and $\| \widehat{M_i, M_{i+1}} \|$ is the arc length of $\widehat{M_i, M_{i+1}}$.

The distance "**a**" is chosen to obtain an optimal reference data representation (figure 3). "**a**" is defined as a function of flight and camera parameters (3).

$$\mathbf{a} = 2(1-r)h \tan\frac{\alpha}{2} = (1-r)T_p h\frac{N}{f} \tag{3}$$

h: Average flight altitude
α: Camera aperture angle
N: Number of image pixels
T_p: Photo-site size of camera
f: Lens focal length of embedded camera
r: Overlap factor between two successive shots (factor to be fixed for our case we propose to be 0.5)

This trajectory partitioning method allows a reduction of acquired data to $\mathbb{D}_a = \{data(M_i, t_i) \in \mathbb{D}/M_i \in (\mathcal{T}_a)\} \subset \mathbb{D}$.

Let T_i be the time range that expresses the time to finish the i^{th} loop by the drone. $T_1, T_2, \ldots T_n$ define a partition of T i.e, $T_i \cap T_j = \varnothing$, $\forall i, j \in [\![1, n]\!] \,/ i \neq j$ *and* $\bigcup_{i=1}^{n} T_i = T$. The reference memory is released in the first drone loop. It is a subset $\mathbb{D}_a^1$ of $\mathbb{D}_a$ defined in eq(4).

$$\mathbb{D}_a^1 = \{data(M_i, t_i) \in \mathbb{D}_a / t_i \in T_1\} \tag{4}$$

As an example of a real drone mission using a camera (with $f = 50\,mm$, $T_p = 8\,\mu m$, and $N = 6000$) at a height of $50\,m$ and a r of 50%. The step is 24 meters long. A higher value of r = 75% results in a step ($12\,m$), which increases the acquisition rate and thus the amount of data sent ($|\mathbb{D}_a^1|$ increase).

Algorithmically, the memory has the structure of a dictionary where the keys are the shot positions and the values are the corresponding images. The visual memory is set up using an algorithm inspired from a recent work of Sedrine et al. [25]

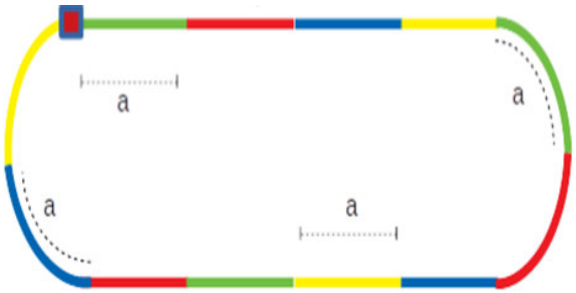

FIGURE 2. UAV trajectory sampling.

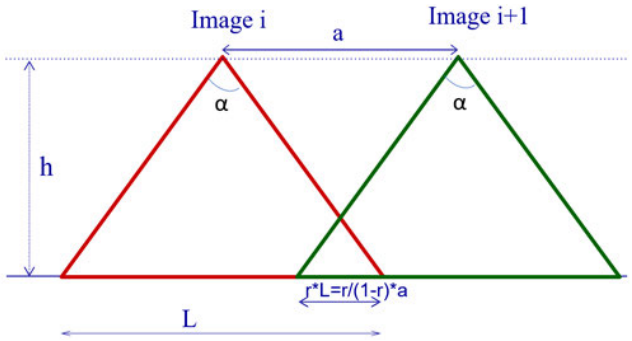

FIGURE 3. Optimal sampling distance.

which builds visual memory using a fixed travel distance for UAV visual odometry application.

B. CONTROLLER ALGORITHM

The controller is about controlling the drone's data flow. It consists of filtering the acquired data and sending it only if it presents a change (that can be caused by an intrusion object). This is done by a comparison between the current captured image (d_c) and its corresponding image. Our approach is to use a DL model **g** that measures the similarity between images. An image is sent to the server side only if the similarity is less than a threshold **s**, which defines the sensibility to detect a change. The filtering of the data is done via a controlling function γ (5).

$$\gamma : \mathbb{D}_a \rightarrow \{0, 1\}$$
$$d_c \mapsto \gamma(d_c) = \begin{cases} 0 & \text{if } \mathbf{g}(d_c, d_0) > \mathbf{s} \\ 1 & \text{Otherwise} \end{cases} \quad (5)$$

where $d_0 \in \mathbb{D}_a^1$ the corespondent reference image of d_c.

The sended data to the server can be mathematically represented as the set $\mathbb{D}_a^\gamma$ (6):

$$\mathbb{D}_a^\gamma = \{data(M_i, t_i) \in \mathbb{D}_a / \gamma(data(M_i, t_i)) = 1\} \quad (6)$$

Our filtering method enables the optimization of the volume of data being transmitted, since:

$$|\mathbb{D}_a^\gamma| \ll |\mathbb{D}_a| \ll |\mathbb{D}| \quad (7)$$

where $|.|$ is the cardinality of a set.

The controller's algorithm has two main steps. First, the system searches the stored reference image representing the same geographic area under supervision. In our case, the use of a dictionary simplifies the task. The position represents the key to find the closest reference image. Once the closer reference image is recovered from drone local disk, the second step starts and consists in a comparison between the reference image and the captured one.

In this work, a deep CNN is used for image silimarity calculation [22]. The controller's algorithm is depicted in Algorithm 1. The function *getDronePosition()* retrieves the UAV position from its navigation system. The function *findReferenceImage( )* finds the reference frame located at the current position. If it exits, then it is compared to the current image. For that, the CNN-based program is called and returns a similarity score. When the latter is under a defined threshold, the processed image is considered interesting to sent to the server for further analysis. Sending only 'trigger images' reduces considerably the bandwidth consumption and the lightweight CNN-based algorithm reduces the drone energy consumption compared to an object detection algorithm. The performance evaluation section confirms these observations.

Algorithm 1 General Controller Algorithm

```
onMission ← True;
while OnMission is True do
    currentPosition ← getDronePosition();
    referenceExist, referenceImage ←
    findReferenceImage(currentPosition);
    if referenceExist is True then
        currentImage ← getDroneImage();
        similarity ← compare(referenceImage,
        currentImage);
        if similarity ≤ thershold then
            sendToServer(currentImage)
        end
        onMission ← getMissionState();
    end
end
```

V. DroMOD IMPLEMENTATION

In this section, we present the implementation details in DroMOD. The Big Data technologies deployed on the server-side are described as well as the parameters of DL models used for object detection and for image comparison. A demo illustrating DroMOD usage is also provided [26].

A. SERVER-SIDE BIG DATA SYSTEM

1) THE PUBLISH/SUBSCRIBE BROKER

Apache Kafka [27] is a defacto standard in IoT systems. It provides a publisher/subscribe broker for collecting data from drones and dispatching them to the system components and to third-party services. The choice of Kafka is based on several comparative studies [28], [29], [30] between different

Publish/Subscribe message systems that show that Kafka provides the highest throughput and lowest latency. In addition, Kafka provides several strength points: scalability, reliability and data injection capabilities.

2) STREAM PROCESSING ENGINE

For real-time data processing, the server-side system is based on Big Data streaming technologies. For that, **Apache Flink** [31] which is a power-full framework for stateful computations over unbounded and bounded data streams, is adopted. Flink is designed to run in all common cluster environments such as Hadoop YARN, Apache Mesos, and Kubernetes or in stand-alone clusters. Flink's batch processing is a special case of streaming processing, which corresponds to a bounded stream of data (stream with a defined start and end). In our implementation, we employ Flink instead of the more well-known, more established and open-source software Apache Spark. Indeed, several works [32], [33], [34] confirm that Flink is more adapted and efficient for stream applications and allows several automated optimisations.

3) STORAGE

For storage, **Apache HBase** database [35], [36] is used in DroMOD. HBase is a distributed NoSQL database built on top of the Hadoop Distributed File System (HDFS) with a column-oriented data structure. The data transfer from Kafka to HBase is done using the Apache HBase Sink Connector. It consumes data from a specified Kafka topic to the corresponding table automatically. This allows permanent storage of data (drone data and detection results) that is mandatory for historical data analysis.

B. IMAGE MODELS LIFE CYCLE

Integration, monitoring and updating tasks are the three main concerns when deploying DL models [37]. In this section, we describe the image models' life cycle: deployment, execution and update.

1) MODELS' DEPLOYMENT

For drone data, object detection is a challenging task because of the mobile nature of drones. The perspective of objects in images is extremely affected by flight parameters (altitude, shot angles, speed…). Several studies attempted to perform object detection to deal with those challenges.

Indeed, for drone-side solution, there are a number of engineering issues that arise during the integration phase to adapt the model to the drone's on-board resources for model deployment. Besides, a DL-model update may require a system scaling capacity, which is not the case for drones (limited energy, aeronautic constraints, etc.) and for non-distributed systems (limited computing resources). For these reasons, DroMOD relies on a server-side Big Data platform for models' detection and update.

2) MODELS' EXECUTION

For the DL-model integration in Flink, the Java DataStream API and the Deep Java Library (DJL) [38] are mutually used. Java DataStream API is a Flink native API that allows the definition of Job GraphFlow. DJL is an open-source, high-level, engine-agnostic, native Java framework for DL. DJL allows the use of DL models in the Java environment. It is designed to use the DL models from different frameworks (TensorFlow, PyTorch, MxNet, etc.) in Java by providing relative DL engines.

Since DroMOD targets multi-scope object detection, several DL models have to be deployed on server-side. For that, TPH-YOLOv5 [39] which is a powerful model that proves good results compared to the baseline model (YOLOv5), is adopted. In the Vis-Drone Challenge (drone object detection benchmark) 2021 [40], TPH-YOLOv5 wins the 5th place and achieves well-matched results with the 1st place model (AP 39.43%). Compared to (YOLOv5), it improves by about 7% in AP. We use a trained PyTorch TPH-YOLOv5 model on the VisDrone dataset [41]. VisDrone is a drone image dataset that provides a large-scale drone-captured images. This dataset is used in four tasks: (i) image object detection; (ii) video object detection; (iii) single object tracking; and (iv) multi-object tracking. Our model is trained on the VisDrone dataset for object detection tasks with a color data augmentation (contrast and brightness). The model detects ten classes of objects: pedestrian, person, car, van, bus, truck, motor, bicycle, awning-tricycle, and tricycle.

The inference of the trained TPH-YOLOv5 models is done using a serialisable and optimised model (TorchScript), which is created using the PyTorch JIT Compiler. Then, the serialisable model is integrated into the Java runtime using the DJL library, which is used with the Java DataStream API for the definition of the Flink job (see figure 4).

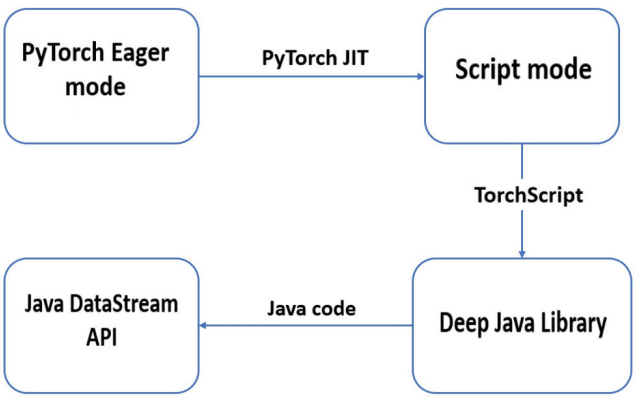

FIGURE 4. PyTorch model deployment in Flink: an integration solution for data stream processing using DL models.

3) MODELS' UPDATE

To understand the need for models' update, let us consider the following use-case where the drone is used to detect disasters. When a fire breaks out, it's critical to find any potentially

hazardous items nearby. DroMOD enables the extension, on the fly, of the drone's mission by adding other objects to detect like humans, animals, automobiles, etc.. This feature is very important for a preliminary assessment of the damage and the evaluation of the emergency actions to take.

For dynamic models' update, Flink hot deployment feature is used in DroMOD. For that, several Flink jobs are created. Each job aims at detecting a specific object category (person, fire, flood, etc.) using a trained DL object detection model. In our implementation, the submission of a job is done using a web server that communicates with the DroMOD server-side system. It uses the SSH protocol to connect to the concerned system server via key-based or user/password authentication. This allows the user to easily run the specific Flink jobs she/he needs even when the drone is flying via a web interface and in a secure manner without requiring technical expertise.

Figure 5 shows the web graphic user interface that is used to update the object detection models. This interface displays the drone's current location above a background map composed of high-resolution satellite images and additional geographic data of the surveillance zone such as roads, buildings, vegetation, etc. A menu shows the existing trained object detection models. It allows the user to activate or deactivate the object to detect and submit the request with a simple click (with the ''submit button''). We provide a demo that demonstrates the DroMOD system in action [26].

C. IMAGE COMPARISON MODEL

As we mentioned above, we use in this work the CNN introduced in [22]. In fact, it is a Siamese CNN, made of two similar branches of modified YOLOv5. Each branch consists of a CSPDarknet, to which we concatenate an adptive average pool layer, a flattening layer and a sigmoid activation function layer, as it is shown in figure 6.

This network was implemented using Pytorch framework. The training was performed on Google Colaboratory platform, using NVidia Tesla K80 GPU.

We used a custom dataset of pairs of images. Half of them are positive, *i.e* representing the same place. In total, we used 7200 pairs for training, and 1800 for validation. The training parameters are displayed in table 1.

TABLE 1. Reference image recognition CNN training parameters.

Parameter	Values
Training epochs	95
Batch size	11
Loss function	Contrastive Loss
Optimizer	Stochastic Gradient Descent
Lerning rate	10^{-1}

Figure 7 presents the confusion matrix of the comparison model. It shows that the model reaches an accuracy of 93%. Its recall is 0.9488, its specificity is around 0.9053, its precision is 0.902 and its F1-score is 0.9246. The figure 8 shows the ROC curve of the used model. Its area under curve is about 0.981, which is relatively high.

VI. DroMOD EVALUATION

In this section, before presenting the performance evaluation, we highlight the key features of DroMOD and contrast them with those of current drone-based systems.

- Multi-scope object detection: DroMOD is based on sharing the processing between the drone and the server. The drone module is in charge of detecting a change in the supervised zone, which is independent of the object's type to be detected. The same embedded algorithm is used for detecting a change caused by fire, flood, person intrusion, etc. Thanks to this feature, DroMOD allows for a multi-scope object detection by using different object detection models that can be dynamically updated on the server-side. For drone-side solutions, it is really hard to provide this feature because of the constrained drone resources and the complexity of embedded software implementation.
- Real-time processing: When compared to other systems, DroMOD has a minimal processing latency. Indeed, DroMOD filters the drone acquired data by a key frame selection algorithm. The execution of heavy-algorithms for object detection is done only on the selected images. However, on exclusive drone-side and server-side solutions, the processing is done on all drone-acquired images. Additionally, the filtering algorithm decreases the network latency, preventing communication channel saturation, which is considered a limit for server-side solutions.
- Stream and batch in one platform: DroMOD processes captured 'trigger images' in real-time and also processes stored data for historical data analysis.
- Optimised use of resources: The execution of object detection models on the drone requires more computing power than image comparison. Usually, a GPU is added to the drone to execute complex DL algorithms which increases drone energy consumption and reduces its autonomy. Consequently, thanks to the drone-side light algorithm, DroMOD uses less energy and increases drone autonomy compared to other systems.
- Low engineering cost: Image processing models' updates take place at the server-side and drone-side embedded programs are rarely modified in DroMOD. Furthermore, models' development and update rely on a high-level API which is very practical and do not require high engineering skills or specific expertise.

The performance evaluation confirms these features thanks to a comparison of DroMOD with three state-of-the art solutions: (i) embedded drone-side processing, (ii) server-side processing and (iii) hybrid solutions.

For each comparison, the following metrics are considered: the processing latency, the consumption of resources and the drone-server communication bandwidth.

For the benchmark, Docker-based deployment [42] is adopted which allows for a practical calculation of resource usage and performance characteristics of each running container. The performance metrics are calculated for different

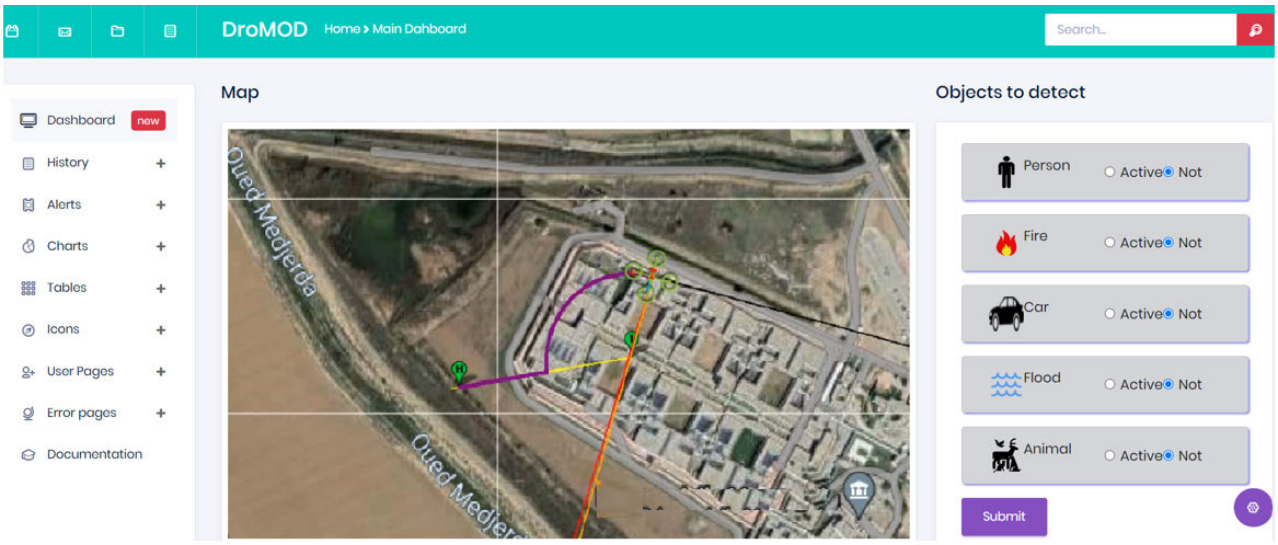

FIGURE 5. Graphic user interface.

architecture components relatively to a PC with the following characteristics:

- CPU: Intel Core i7-1065G7 (1.3 GHz up to 3.9 GHz, 8 Mo Cache, 8 Cores)
- RAM: 8 Go DDR4
- Hard drive: 1 To HDD
- GPU: mx230
- OS: Ubuntu 20.04.1 LTS
- Swap: 16 Go SSD

To ensure reliable and significant tests, a variant range of environments and events are required. For our evaluation, three different environments (city, jungle and mountains) and three kinds of objects to detect (persons, cars and fire) are included in the RGB images' dataset. The environments are represented by 3D models that are generated using a drone stereo photogrammetry process. The images are taken from an altitude of 50m. The 3D models are integrated into a simulation software, allowing for the simulation of real-world environments, events, drones and on-board captures (camera, positioning receiver). Captured images are processed and 'trigger images' are sent to the server with positing information. The following results are calculated as an average of the different obtained test evaluations of scenarios combining the different environments and objects to detect.

A. LATENCY MEASUREMENT

Latency refers to the delay in transmitting (network latency) and processing data (processing latency). In our case, end-to-end latency is calculated from the event launch from the visual field of the drone to a third-party notification.The network latency is not considered because it depends on the network technology and congestion and not on the system's performance. Consequently, the measure of latency is calculated as the sum of the drone-side processing time and the server-side processing time.

Figure 9 shows that one-step drone-side solutions have the highest processing latency with 3.4 s. This can be explained by the DL object detection inference at drone-side. Besides, the server-side solution reaching 2.4 s results of a high fps of camera images compared to the needed processing time. DroMOD presents the lowest latency which demonstrates the efficiency of the two-step approach. Indeed, the image mapping processing is much lighter than the object detection processing. Furthermore, in DroMOD, server-side processing latency is lower than the server-side solution due to the 'trigger image' selection algorithm which reduces the amount of images to process on the server.

B. RESOURCE CONSUMPTION

In this set of tests, we measure the average resources usage (Server CPU RAM, Drone CPU RAM and drone-server network Tx (transmission throughput) for three kind of environments (mountains, city, jungle). In addition, we consider two kinds of intrusion objects (persons and cars). Thanks to the modular deployment of Docker and Docker compose [42] that places each DroMOD component in a separate container, the separate monitoring of each resource is possible and efficient.

Figure 10 shows that DroMOD is a good compromise between drone-side and server-side solutions in resource consumption. DroMOD drone resource consumption is clearly less important than the drone-side solution. Similarly, DroMOD network usage is lower than the server-side solution. These performance evaluation results confirm the relevance of the collaborative processing approach that we adopted in DroMOD.

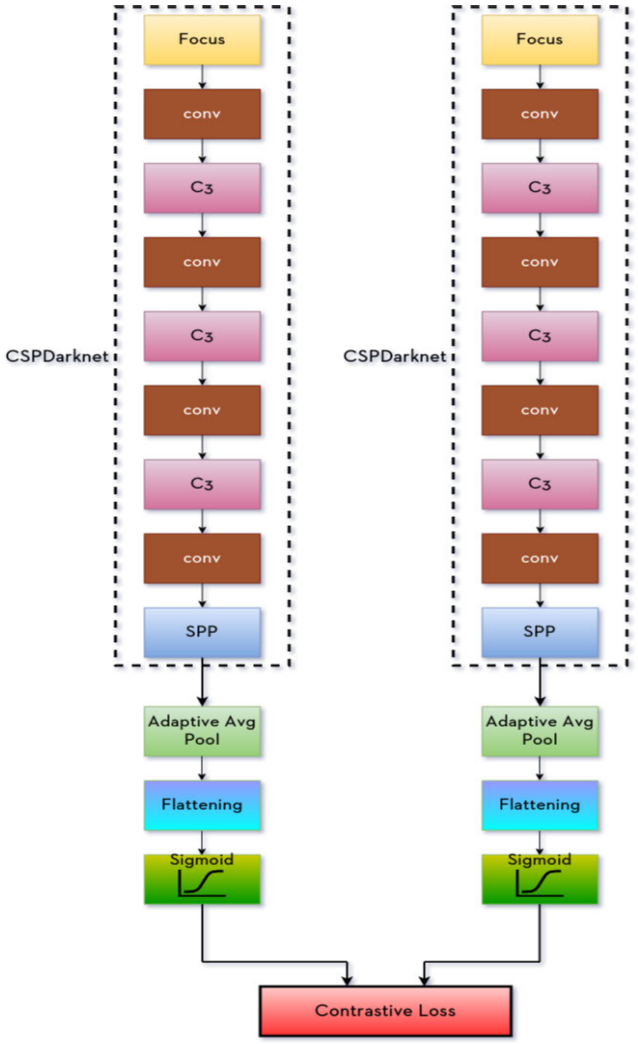

FIGURE 6. Comparison model architecture.

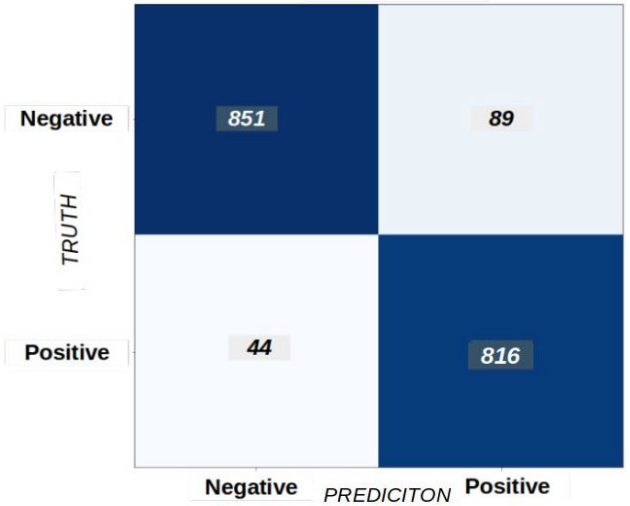

FIGURE 7. Image comparison model confusion matrix.

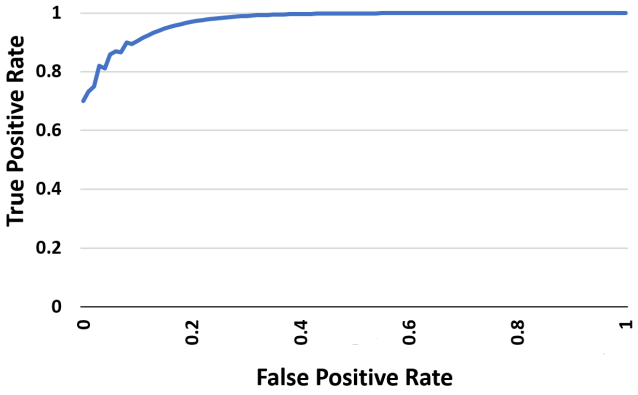

FIGURE 8. Image comparison model ROC curve.

C. IMAGE COMPARISON VS OBJECT DETECTION PROCESSING

The comparison between the two image processing algorithms is based on two evaluations: the inference time and the accuracy.

1) INFERENCE TIME

In this test, we measure the inference time of a classical and recent object detection model (TPH-YOLOv5 [39]) with DroMOD image comparison model. The measure is an average of the inference time of 1000 frames (1000 pairs in case of comparison) with various image sizes (256 × 256, 512 × 512 and 1024 × 1024 pixels). The frames are chosen from different scenes (city, jungle, mountains).

Figure 11 shows that DroMOD comparison model is lighter than the object detection model. Indeed, the inference time for object detection is about 8 times bigger than the comparison model which is inefficient for a real time application. A comparison model is more adapted for a real-time application since it allows a lower processing latency. Indeed, for 512 × 512 images, the maximum fps for one-step drone-side solution the maximum fps is about 0.3 compared with 2.4 for a comparison model. The object detection model inference time varies exponentially with image size, posing several challenges and limits for high-resolution sensors.

2) ACCURACY

In this test, we measure the accuracy of detecting a change (intrusion) in the image using the object detection model and the comparison model. The object detection model detects a change when it detects the intrusive object. The measurement is performed on 1000 frames. The frames are obtained by inserting an intrusion object (a car or a person) in a random position using several scene images (city, jungle, and mountains). The object has a specific area relatively to the total area of the scene image (from 0.01% to 100%). Figure 12 presents the variation of accuracy as a function of object relative area. Equation (8) is used to calculate the accuracy of each model.

$$Accuracy = \frac{N}{T} * 100 \tag{8}$$

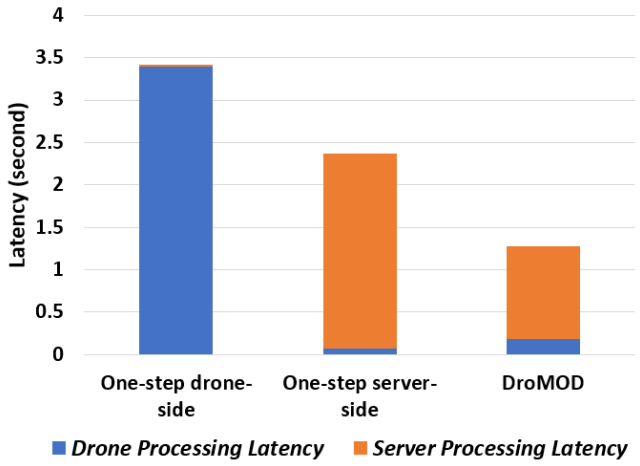

FIGURE 9. Processing latency comparison.

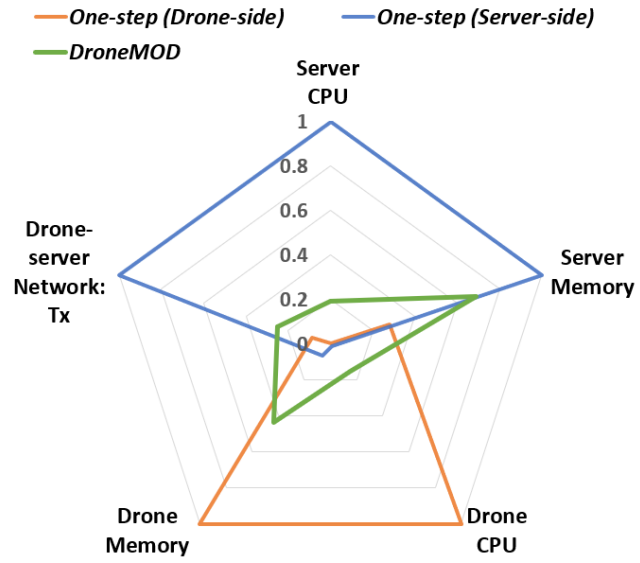

FIGURE 10. Resources consumption comparison.

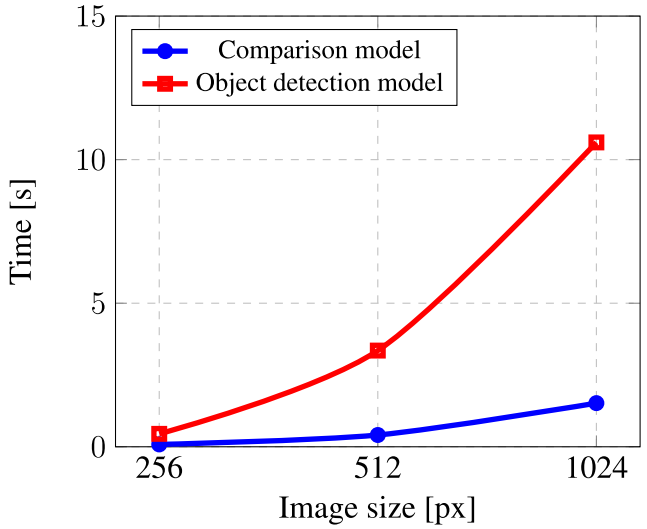

FIGURE 11. Inference time comparison.

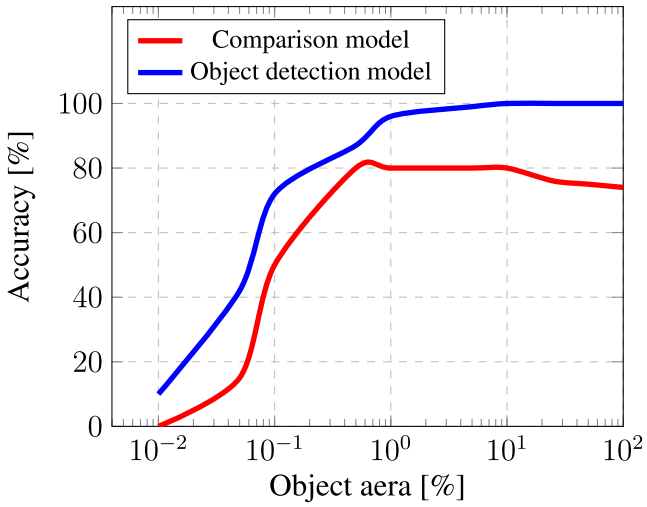

FIGURE 12. Accuracy comparison (x-axis with log10 scale).

where:

- **N**: The total number of true change detection
- **T**: The total number of the test frames

We note that the detection model is efficient for a specific scale (0.5%) with 98% accuracy but for a bigger or smaller scale it has a low accuracy. In contrast, the comparison model has an increased accuracy in function of object area. In general, the accuracy comparison shows that the comparison model is more accurate than the object detection model for different object areas. In the case of drones, the scale of the intrusion object (the changing), relative to the whole image's cover, is variable since it is a flying engine. From this test, we conclude that the on-board comparison model is more effective in identifying an object intrusion in the case of a drone-based surveillance system.

D. DroMOD VS EdgeDuet DRONE-SIDE PROCESSING

In this section, we compare DroMOD to a recent existent work, namely EdgeDuet [16] that adopts, like DroMOD, a hybrid solution. EdgeDuet performs large object detection on board and small objects detection on server-side. Thus, EdgeDuet sends frames continuously from the drone to the server in order to detect small objects while our technique sends data in case of suspicious events only. This leads to a lower bandwidth consumption with DroMOD.

EdgeDuet [16] uses YOLOv3 (640 × 640) [7] as a local detector. In our work, we use the siamese neural network introduced in [22]. The two techniques are different. EdgeDuet is based on object detection whereas DroMOD uses images comparison. A first observation is that our model is significantly smaller than that of EdgeDuet. In fact, *Mseddi et al.* model has a size of **18MB**

and YOLOv3 (640 × 640) used by EdgDuet is around **119MB**. This is a first argument in favor of our method. For processing latency calculation, both CPU and GPU execution environments are adopted. As dataset, VisDrone [41] https://github.com/VisDrone/VisDrone-Dataset that is composed of 548 images captured from a drone, is used. In order to simulate EdgeDuet drone-side, we call YOLOv3 on each image of the dataset, specifying the detection of cars, buses, people, bicycles and motorcycles. Besides, we performed a comparison on each image, using the siamese model. First, we run the comparison on a GPU (Figure 13), then we use the CPU (Figure 14). In both cases, DroMOD is faster than EdgeDuet and the gap is wider on CPU.

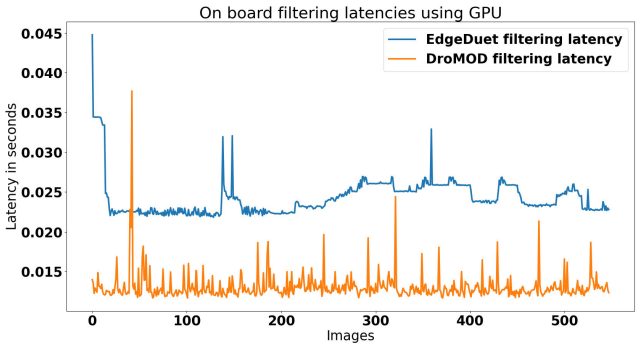

FIGURE 13. EdgeDuet vs DroMOD drone-side processing latency on GPU.

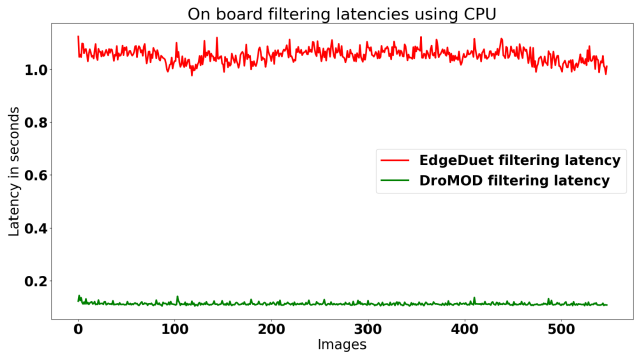

FIGURE 14. EdgeDuet vs DroMOD drone-side processing latency on CPU.

To sum up, our work outperforms EdgeDuet. DroMOD embedded neural network is lighter and faster than the one used by EdgeDuet. The model size is smaller which is practical to embed and its processing time is much faster. On the other hand, EdgeDuet transmits more frames to the server for processing which requires a higher bandwidth.

E. MULTI-SCOPE OBJECT DETECTION

1) PROCESSING LATENCY WITH MULTIPLE MODELS

The multi-scope object detection is assured by the use of several DL models in the server side that are parallel executed in a distributed environment thanks to use of Big Data streaming technologies (Apache Flink and Kafka) in DroMOD.

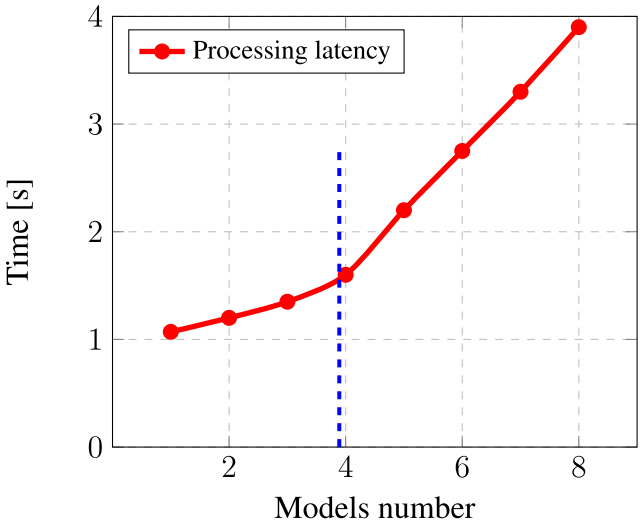

FIGURE 15. DroMOD processing latency as function of models number.

For real-time critical applications, Latency is a key parametric to think about. In this test, we calculate DroMOD's server-side processing latency as a function of the deployment object detection model number.

For the test implementation, YOLOv5s architecture is used to create several models that each detect a single object (e.g. car, person, bus..) which has about 7.22 M trained parameters. The training is done using the MS COCO dataset, which was created by collecting images of typical scenes with common objects in their natural settings. It contains 80 objects. The models are integrated with our DroMOD platform implementation based on the method cited in Subsection VI... We calculate latency by measuring the average amount of time it takes the image to be processed by the deployment models.

The results on figure 15 show that the latency increases with the number of models. This increase isn't linearly proportional to the number of models (which is the case of sequential computing). This is thanks to the scalability and parallel computing of DroMOD, which are results of using streaming big data technologies. From 4 models, the evolution of latency is determined by the saturation of our limited resources of test hardware, which is why our system on the server side balanced the load to assure efficient computing.

2) SINGLE MODEL FOR MULTIPLE OBJECT DETECTION VS. MULTIPLE SINGLE-OBJECT-DETECTION MODELS

Multi-scope object detection can be performed using (meth.1) one model that detects all target objects or (meth.2) different models that each detect a single or a subset of scope objects. DroMOD adopted the second solution. For evaluation, we measure object detection accuracy of each approach, the execution time and the model's computing complexity.

The implementation of this test is done using the YOLOv5 which is a scalable architecture. This model can be scaled in depth and width. The depth of the model is about the

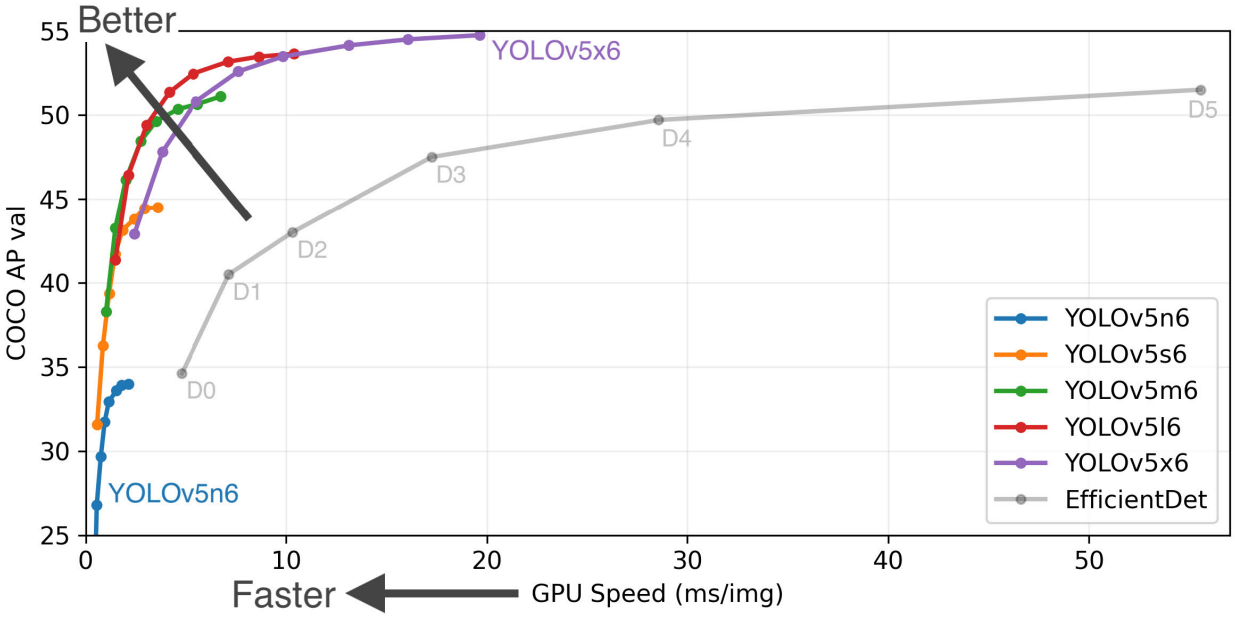

FIGURE 16. COCO AP val of YOLOv5 configurations as function of GPU speed (ms/img) [43].

number of layers and the width is about the number of channels in layers. The complexity of the model is increasing (from YOLOv5n (nano) to YOLOv5X (xlarge)) with the increase of the number of the models' parameters. Note that the higher the complexity is (which improves the objects' detection accuracy), the higher the computing cost and time are. Indeed, the figure 16 shows that the average of mAP on the COCO dataset of different classes (80) increases with the complexity of the model. For the benchmark, we use the large version (YOLOv5l) as a single-model to detect multiple objects (meth.1) and the nano version as single object detection models (meth.2).

The training, validation and test of our models are done with the Microsoft COCO dataset with the native distribution on the Google Colaboratory platform with a Tesla T4 GPU and 16 Go of RAM. We use AP (the average precision) to show the performance (in terms of accuracy of detection) of object detection models since it is the most frequently used. The AP measures the accuracy of detected objects of the detector in terms of classification (attributed a label to an object) and localisation (localisation of the object in the image) of a specific class. The localisation accuracy is measured by the intersect over union metric (IoU) which is the ratio of intersection area of the predicted bounding box (bbox) and the true bbox area by the area of their union. We calculate the AP (mathematically, it is the area under the Precision-Recall curve limited by the abscissa axis relative to a specific class) of car and person with an IoU threshold of 0.5 (a localisation of a detector is considered true if IoU is bigger than 0.5). The deployment of models is done on our DroMOD implementation to measure the execution time.

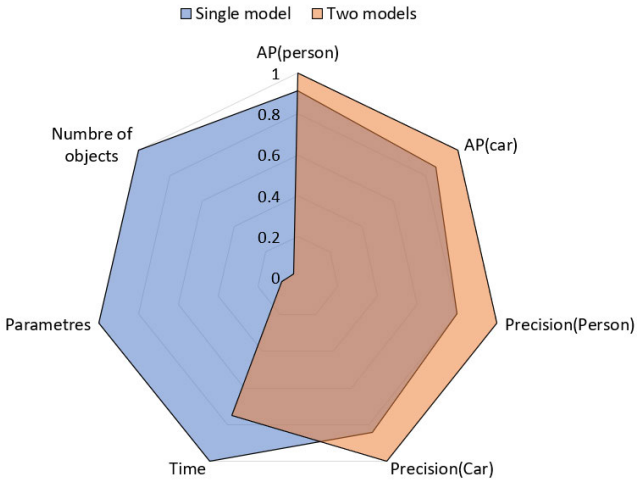

FIGURE 17. Comparison of YOLOv5l (80 objects) and two YOLOv5n (to detect Car and Person) trained using COCO dataset.

The results are presented in the figure 17. By dividing the various comparison variables (x_i) by their maximum values ($max(x_i)$), the variables are normalized since our goal is to compare the two approaches. Based on figure 17, for a single-object detection model (for cars and person), the AP and the accuracy of detecting those objects are better compared with the single object detection. This is a very important point, especially in critical applications such as surveillance. Compared to the execution time on our Big Data platform, those models present a lower execution time compared to the single model, which is important for real-time applications.

We attribute the short execution time to the parallel processing and the use of streaming Big Data technologies.

In addition, the second method ensures that the objects to detect can be updated in a flexible manner. In fact, we can address a specific object (or its subset objects) without affecting the entire set of objects in order to improve, add, or remove a scope object. Multiple models are also well adapted for distributed computations. Furthermore, for a unique model for multi-scope object detection, the tuning of hyper-parameters and regularisation of models to increase the accuracy of a particular object isn't an obvious task.

As a conclusion, we confirm that DroMOD's parallel execution of models allows not only faster execution but also more accurate detection and more flexibility for object detection update.

VII. CONCLUSION

DroMOD is based on a collaboration between the drone and the server processing. On the drone-side, a lightweight 'image trigger' selection algorithm relies on a reference frame comparison. It allows the detection of a change in the observed zone compared to a reference image. On the server-side, the identification and analysis of the change is performed using DL models in order to detect precisely an event or an object of interest. Compared with existent object detection solutions either embedded drone-based, server-based or hybrid solutions, DroMOD shows the lowest processing latency and optimal resource usage while ensuring a high detection accuracy. Furthermore, since the object detection is performed on the server-side, the object detection models can be regularly updated allowing for dynamic multi-scope detection. The performance evaluation of DroMOD considering different surveillance systems contexts and drone parameters confirm the relevance of our approach.

As a future work, we target to test DroMOD on other real use cases where the scope update is required and dynamic. We are also interested in securing the different communication channels and measuring the security overhead on DroMOD performance.

REFERENCES

[1] R. T. Collins, A. J. Lipton, and T. Kanade, "Introduction to the special section on video surveillance," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 8, pp. 745–746, Aug. 2000.

[2] Z. Zou, K. Chen, Z. Shi, Y. Guo, and J. Ye, "Object detection in 20 years: A survey," 2019, *arXiv:1905.05055*.

[3] B. Taylor, V. S. Marco, W. Wolff, Y. Elkhatib, and Z. Wang, "Adaptive deep learning model selection on embedded systems," *ACM SIGPLAN Notices*, vol. 53, no. 6, pp. 31–43, Dec. 2018.

[4] K. Kim and C. S. Hong, "Optimal task-UAV-edge matching for computation offloading in UAV assisted mobile edge computing," in *Proc. 20th Asia–Pacific Netw. Oper. Manage. Symp. (APNOMS)*, Sep. 2019, pp. 1–4.

[5] J. Liu, C. Hu, J. Zhou, and W. Ding, "Object detection algorithm based on lightweight YOLOv4 for UAV," in *Proc. 7th Int. Conf. Intell. Comput. Signal Process. (ICSP)*, Apr. 2022, pp. 425–429.

[6] M. Vandersteegen, K. Van Beeck, and T. Goedeme, "Super accurate low latency object detection on a surveillance UAV," in *Proc. 16th Int. Conf. Mach. Vis. Appl. (MVA)*, May 2019, pp. 1–6.

[7] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," 2018, *arXiv:1804.02767*.

[8] Y. Bazi and F. Melgani, "Convolutional SVM networks for object detection in UAV imagery," *IEEE Trans. Geosci. Remote Sens.*, vol. 56, no. 6, pp. 3107–3118, Jun. 2018.

[9] S. M. Azimi, "ShuffleDet: Real-time vehicle detection network in on-board embedded UAV imagery," in *Proc. Eur. Conf. Comput. Vis. (ECCV) Workshops*, 2018, pp. 1–11.

[10] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 6848–6856.

[11] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot MultiBox detector," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2016, pp. 21–37.

[12] J. P. T. Sien, K. H. Lim, and P.-I. Au, "Deep learning in gait recognition for drone surveillance system," *IOP Conf. Ser., Mater. Sci. Eng.*, vol. 495, no. 1, 2019, Art. no. 012031.

[13] X. Xia, C. Xu, and B. Nan, "Inception-v3 for flower classification," in *Proc. 2nd Int. Conf. Image, Vis. Comput. (ICIVC)*, Jun. 2017, pp. 783–787.

[14] A. Sherstinsky, "Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network," *Phys. D, Nonlinear Phenomena*, vol. 404, Mar. 2020, Art. no. 132306.

[15] J. Dick, C. Phillips, S. H. Mortazavi, and E. de Lara, "High speed object tracking using edge computing," in *Proc. 2nd ACM/IEEE Symp. Edge Comput.*, Oct. 2017, pp. 1–2.

[16] X. Wang, Z. Yang, J. Wu, Y. Zhao, and Z. Zhou, "EdgeDuet: Tiling small object detection for edge assisted autonomous mobile vision," in *Proc. IEEE Conf. Comput. Commun.*, May 2021, pp. 1–10.

[17] S. Liu, T. Wang, J. Li, D. Sun, M. Srivastava, and T. Abdelzaher, "AdaMask: Enabling machine-centric video streaming with adaptive frame masking for DNN inference offloading," in *Proc. 30th ACM Int. Conf. Multimedia*, Oct. 2022, pp. 3035–3044.

[18] M. S. Alam, B. V. Natesha, T. S. Ashwin, and R. M. R. Guddeti, "UAV based cost-effective real-time abnormal event detection using edge computing," *Multimedia Tools Appl.*, vol. 78, no. 24, pp. 35119–35134, Dec. 2019.

[19] C. Djeraba, A. Lablack, and Y. Benabbas, *Multi-Modal User Interactions in Controlled Environments*, vol. 34. Cham, Switzerland: Springer, 2010.

[20] A. C. Murillo and J. Kosecka, "Experiments in place recognition using gist panoramas," in *Proc. IEEE 12th Int. Conf. Comput. Vis. Workshops*, Sep. 2009, pp. 2196–2203.

[21] X. Zhang, L. Wang, and Y. Su, "Visual place recognition: A survey from deep learning perspective," *Pattern Recognit.*, vol. 113, Jan. 2021, Art. no. 107760. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S003132032030563X

[22] W. S. Mseddi, M. A. Sedrine, and R. Attia, "YOLOv5 based visual localization for autonomous vehicles," in *Proc. 29th Eur. Signal Process. Conf. (EUSIPCO)*, Aug. 2021, pp. 746–750.

[23] G. Jocher, A. Stoken, J. Borovec, S. Christopher, and L. C. Laughing, "ultralytics/yolov5: v4. 0—nn. SiLU() activations, Weights & Biases logging, PyTorch Hub integration," Zenodo, 2021.

[24] R. Hadsell, S. Chopra, and Y. LeCun, "Dimensionality reduction by learning an invariant mapping," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, vol. 2, Jun. 2006, pp. 1735–1742.

[25] M. A. Sedrine, W. S. Mseddi, and R. Attia, "Neural network visual odometry based framework for UAV localization in GPS denied environment," *COMPUSOFT, Int. J. Adv. Comput. Technol.*, vol. 9, no. 8, pp. 3798–3809, 2020.

[26] Y. G. T. Abdellatif and M. A. Sedrine. (2022). *Dromod Demo*. [Online]. Available: https://drive.google.com/file/d/1t31QRtoz6aVNXerPGQin4eatO5f5wz4u/view?usp=sharing

[27] K. M. M. Thein, "Apache Kafka: Next generation distributed messaging system," *Int. J. Sci. Eng. Technol. Res.*, vol. 3, no. 47, pp. 9478–9483, 2014.

[28] Confluent. (2022). *Kafka vs. Pulsar vs. Rabbitmq: Performance, Architecture, and Features Compared*. [Online]. Available: https://www.confluent.io/kafka-vs-pulsar/

[29] M. Kaczor and P. Powroźnik, "Comparative analysis of message brokers," *J. Comput. Sci. Inst.*, vol. 23, pp. 89–96, Jun. 2022.

[30] S. Vyas, R. K. Tyagi, C. Jain, and S. Sahu, "Literature review: A comparative study of real time streaming technologies and Apache Kafka," in *Proc. 4th Int. Conf. Comput. Intell. Commun. Technol. (CCICT)*, Jul. 2021, pp. 146–153.

[31] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, "Apache flink: Stream and batch processing in a single engine," *Bull. IEEE Comput. Soc. Tech. Committee Data Eng.*, vol. 36, no. 4, pp. 1–12, Jan. 2015.
[32] A. Sharma, D. Puri, M. Kumar, and G. Soni, "Implementation and comparison of big data analysis on large dataset using SPARK and FLINK," in *Proc. ICCCE*. Cham, Switzerland: Springer, 2022, pp. 385–394.
[33] E. Nazari, M. H. Shahriari, and H. Tabesh, "BigData analysis in healthcare: Apache Hadoop, Apache Spark and Apache Flink," *Frontiers Health Informat.*, vol. 8, no. 1, p. 14, Jul. 2019.
[34] O.-C. Marcu, A. Costan, G. Antoniu, and M. S. Perez-Hernandez, "Spark versus Flink: Understanding performance in big data analytics frameworks," in *Proc. IEEE Int. Conf. Cluster Comput. (CLUSTER)*, Sep. 2016, pp. 433–442.
[35] M. N. Vora, "Hadoop-HBase for large-scale data," in *Proc. ICCSNT*, vol. 1, 2011, pp. 601–605.
[36] (2022). *Apache Hbase*. [Online]. Available: https://hbase.apache.org/
[37] A. Paleyes, R.-G. Urma, and N. D. Lawrence, "Challenges in deploying machine learning: A survey of case studies," *ACM Comput. Surv.*, vol. 55, no. 6, pp. 1–29, Jul. 2023.
[38] (2022). *Deep Java Library*. [Online]. Available: https://github.com/deepjavalibrary/djl/
[39] X. Zhu, S. Lyu, X. Wang, and Q. Zhao, "TPH-YOLOv5: Improved YOLOv5 based on transformer prediction head for object detection on drone-captured scenarios," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. Workshops (ICCVW)*, Oct. 2021, pp. 2778–2788.
[40] Y. Cao, Z. He, L. Wang, W. Wang, Y. Yuan, D. Zhang, J. Zhang, P. Zhu, L. Van Gool, J. Han, S. Hoi, Q. Hu, and M. Liu, "VisDrone-DET2021: The vision meets drone object detection challenge results," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. Workshops (ICCVW)*, Oct. 2021, pp. 2847–2854.
[41] P. Zhu, L. Wen, D. Du, X. Bian, H. Fan, Q. Hu, and H. Ling, "Detection and tracking meet drones challenge," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 11, pp. 7380–7399, Nov. 2022.
[42] (Aug. 2, 2022). *Docker Makes Development Efficient and Predictable*. [Online]. Available: https://www.docker.com/
[43] Ultralytics. (2023). *YOLOv5*. [Online]. Available: https://pytorch.org/hub/ultralyticsyolov5/

TAKOUA ABDELLATIF received the engineering degree in IT from the Engineering School, ENSIMAG, Grenoble, France, in 1998, and the Ph.D. degree in autonomic management in JavaEE clusters from INRIA, in 2006. She worked as a Research and Development Engineer on HP telecom platform with Hewlett Packard, Grenoble, for five years. Then, she was a Research Engineer with Bull/Grenoble on systems' scalability and fault-tolerance. She is currently an Associate Professor in IT with the University of Sousse and the École Polytechnique de Tunisie (EPT), Carthage University. She is also a Senior Researcher with the EPT/SERCOM Research Laboratory, coordinating the laboratory research activities around scalable and secure distributed systems.

MOHAMED ALI SEDRINE received the engineering degree in embedded systems from ISIMA, the master's degree in robotics from Blaise Pascal University (Clermont-Ferrand), France, in 2014, and the Ph.D. degree in vision-based localization techniques for UAVs from the École Polytechnique de Tunisie (EPT), in 2022. He works as a Research and Development Engineer on UAV systems and their applications with the SACES Research Unit and the SERCOM Laboratory. He is also an Assistant Professor in computer science with the Aviation School of Borj El Amri.

YASSINE GACHA received the engineering degree in geomatic from the Aviation School of Borj El Amri, Tunisia, in 2022. He is currently working on surveillance systems using drones. His research interests include big data, deep learning, and quantum computing.