POLITECNICO DI TORINO
Repository ISTITUZIONALE

Efficient Tensor Compression and Reconstruction in Split DNNs for Edge-Based Object Detection

Original
Efficient Tensor Compression and Reconstruction in Split DNNs for Edge-Based Object Detection / Yu, Y., Mendula, M., Levorato, M., Papatriantafilou, M., Chiasserini, C.F.. - In: IEEE INTERNET OF THINGS JOURNAL. - ISSN 2327-4662. - 13:8(2026), pp. 16085-16101.

Availability:
This version is available at: 11583/3006868 since: 2026-01-23T08:28:18Z

Publisher:
IEEE

Published
DOI:

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright
IEEE postprint/Author's Accepted Manuscript

©2026 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collecting works, for resale or lists, or reuse of any copyrighted component of this work in other works.

(Article begins on next page)

06 November 2026

Efficient Tensor Compression and Reconstruction in Split DNNs for Edge-Based Object Detection

Yenchia Yu, *Student Member, IEEE*, Matteo Mendula, *Member, IEEE*, Marco Levorato, *Senior Member, IEEE*, Marina Papatriantafilou, Carla Fabiana Chiasserini, *Fellow, IEEE*

***Abstract*—Computer Vision (CV) tasks are among the most pivotal, yet challenging, operations for Uncrewed Aerial Vehicles (UAVs), especially in mission-critical applications. They require processing complex image data through Deep Neural Networks (DNNs), which demand computational resources far beyond UAVs' capacity. To address this limitation, Split DNNs offer a promising solution by partitioning the model into: (i) a lightweight *Head*, deployed on the UAV for rapid, albeit less precise, initial image representations, and (ii) a more complex *Tail*, executed at the network edge for refined, higher-accuracy results. However, this solution necessitates transmitting large tensor data from the UAV to the edge server, leading to significant bandwidth consumption. We tackle this challenge by introducing a goal-oriented framework named Compressed Tensor-based DNN Split (CoTeD). Our framework integrates an application- and system-aware optimization model that orchestrates computing and transmission resources in real time. At the UAV, CoTeD dynamically selects relevant tensor information and optimally trades-off between DNN detection quality and bandwidth consumption, guided by application requirements and system operational conditions. At the edge server, CoTeD reconstructs the tensor, enabling efficient inference by the Tail model. This approach effectively balances bandwidth usage with quality of the CV task output. Experimental results, obtained through our hardware-software testbed and using datasets with different sizes and characteristics, show that CoTeD can reduce data transmission over the radio link by up to 90% without noticeable loss in object detection quality and inference latency by up to 70% compared to local DNN deployment onboard the UAV. Also, CoTeD yields an inference request success rate of at least 90%, with an increase of 20%-80% compared to direct DNN splitting, static JPEG compression, and DNN model quantization.**

***Index Terms*—Edge computing, UAVs, Orchestration, Bandwidth utilization**

I. Introduction

Advancements in Uncrewed Aerial Vehicle (UAV) technology have substantially broadened its adoption, leveraging it for a wide range of applications including disaster rescue, infrastructure inspection, environmental monitoring, and delivery services. In such application scenarios, computer vision (CV) tasks such as object detection are identified as essential, yet computationally demanding and performance-critical, for execution on UAV platforms [1]. On one hand, they provide essential input for streamlined applications, such as object tracking and UAV autopilot tasks. On the other hand, the variability in the performance of CV tasks in terms of accuracy and latency can substantially impact downstream applications and even cause system failures. The task performance strongly depends on the quality of the data samples (e.g., images) provided as input and on the scale of the relevant objects, which can widely vary depending on the captured scene, the weather conditions, and the UAV position, making stable inference quality hard to achieve. Deep Neural Networks (DNNs) have been shown to outperform traditional CV approaches in accuracy and robustness for UAV CV tasks [2]. However, high-quality inference using complex DNNs on resource-constrained UAVs remains a challenging and costly task.

A promising approach towards coping with these issues is to break down the original DNN architecture into segments, allowing subsets of layers to be deployed on different computing nodes, distributing the processing between onboard and off-board parts. This way, the original processing pipeline is preserved by linking these segments through a suitable communication protocol [3], [4]. Specifically in a UAV setting, the DNN can be split into a small, low-complexity *Head* segment executed at the UAV and a larger, more complex, *Tail* segment running at an edge server. This approach enables two processing options: (i) local refinement of the Head output (hereinafter also referred to as *activation tensor* or simply *tensor*) on the UAV using a simplified Tail model [5], or (ii) transmission of the Head output to the edge server for full Tail processing. Together, the two options cover diverse relevant system and application requirements. The former represents a reliable solution in the presence of impaired connectivity while resulting in a limited energy footprint, but it comes with lower accuracy. In contrast, the latter allows for a higher level of inference accuracy, thus making it possible to fulfill more stringent application requirements. However, it implies a larger inference latency and requires not only reliable connectivity between the UAV and the edge server, but also, due to the large size of the activation tensor, the use of a data rate that can easily reach Gbps level. It is worth emphasizing that transmitting raw data collected by the UAV cameras to the edge server is not a desirable option. Such an approach would necessitate duplicating the processing already performed at the UAV, consequently increasing the overall energy consumption. Additionally, it could compromise data privacy. Conversely, due to the non-linear operations of the DNN, transmitting the tensor better preserves privacy, as reverting it to the original input is a challenging task [6].

Y. Yu and C. F. Chiasserini are with Politecnico di Torino, Torino, Italy. M. Mendula is with Centre Tecnològic de Telecomunicacions de Catalunya (CTTC), Barcelona, Spain. M. Levorato is with University of California, Irvine, USA. M. Papatriantafilou and C. F. Chiasserini are with Chalmers University of Technology, Sweden. Y. Yu's work was suported by Leonardo S.p.A. The work was also supported by the SNS JU 6G-INTENSE project (Grant Agreement No. 101139266), the CSI-Future project under the NRRP PRIN 2022 program (D.D.1409 del 14/09/2022 MUR), and the NRRP MOST CNMS (CN00000023) Spoke 6. Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the MUR and European Union.

In light of the above observations, it is essential to carefully consider the following factors regarding DNN splitting: *(i)* the strategy for partitioning the DNN, *(ii)* the radio bandwidth required for transferring the activation tensor from the UAV to the edge server, and *(iii)* the overall inference latency associated with the transmission of the tensor and the distributed processing. Techniques to reduce the radio bandwidth needed for the tensor transfer have been proposed in recent studies. The most well-known approach is to introduce "bottleneck" layers at the end of the DNN Head [7], [8], which can significantly reduce the tensor size. However, this approach cannot fully address some key factors: *(i)* once the "bottleneck" layers are set, the compression ratio on the tensor is fixed and cannot be adapted to the available radio bandwidth, *(ii)* DNN retraining is required for every splitting point, which introduces a high computation cost and limits the flexibility on adjusting the DNN partitioning point in runtime. Other approaches leverage general compression solutions like Joint Photographic Experts Group (JPEG) [9] or quantization [10] to directly compress the Head tensor, which is DNN retraining free. Nonetheless, these approaches suffer from high compression loss and, yet, limited compression ratio, making the split DNN inference quality and latency very hard to be guaranteed.

In this work, we fill the above gaps by addressing the following technical challenges: *(i)* executing DNN inference efficiently by distributing the computational load between the UAV and the edge server; *(ii)* maintaining stable system performance despite limited and time-varying radio bandwidth for tensor transmission; *(iii)* keeping the overall inference latency (including both the data transfer and process latencies) within user requirements, and *(iv)* achieving an optimal trade-off between DNN detection quality and bandwidth consumption.

We achieve this multi-faced goal by accurately modeling the real-world cooperative UAV-edge CV system, and designing a low-complexity, *goal-oriented framework* named COmpressed TEnsor-based DNN split (CoTeD). *First,* CoTeD removes redundant information leveraging the correlation between consecutive video frames, which, as demonstrated later, is retained by consecutive tensors. *Second,* given the time-varying operational context (i.e., radio link quality, available bandwidth, and acquired video frames), CoTeD selects the appropriate compression technique, and configures the associated parameters, in such a way that it minimizes bandwidth usage while ensuring that the target values of the application-level key performance indicators are met. *The key idea we propose is a modeling that enables the transformation of the complex (non-linear) optimization problem into a simpler, tractable one, that admits efficient solutions, balancing the aforementioned multi-faced trade-offs. Third,* by exploiting a minimal amount of information sent by the transmitter on the performed processing, CoTeD analyzes the original tensor at the edge server and provides it as input to the DNN Tail, allowing for a highly accurate inference output.

Our contribution can be summarized as follows:

- We design the CoTeD framework for efficient split computing as well as optimization-driven tensor compression and transmission from the UAV to the edge server. Remarkably, CoTeD can *be easily adapted to different pre-trained DNNs without any DNN model modification*;
- We develop a UAV-to-5G-edge testbed in which an object-detection DNN is split and deployed across the UAV and the edge server to process the video frames captured by the UAV (the code will be made available to accompany the paper publication);
- Through our testbed, we demonstrate that the tensors generated by the DNN Head on the UAV, which corresponds to consecutive video frames, preserve a correlation level comparable to that of the original frames. This property enables effective data reduction through a simple yet dynamic data-deduplication scheme;
- Next, we focus on three relevant compression techniques, namely, JPEG, CANDECOMP/PARAFAC decomposition, and polynomial regression. Using our testbed, we experimentally characterize how detection quality and sensitivity degrade under different pruning factors, compression methods, and compression parameters, which jointly produce a different signal-to-noise ratio (SNR) on the decompressed tensor. We then use this characterization to derive an analytical relationship among these quantities;
- We exploit the aforementioned experimental findings to formulate an optimization model that, accounting for the system and application constraints, dynamically determines the tensor sparsity, compression technique, and corresponding configuration that minimize tensor transmission bandwidth. Furthermore, we propose a novel, yet simple, strategy to efficiently solve the resulting non-linear optimization problem. By delegating this optimization to the edge orchestrator, we enable real-time adjustment of tensor transmission settings, thereby *effectively trading off radio resource consumption with application performance in dynamic scenarios*;
- We evaluate the overall CoTeD framework by means of our testbed using three different video datasets, proving the effectiveness and robustness of our solution. Our experimental results show that CoTeD can reduce up to 90% the data to be transmitted over the radio link without significant impact on the overall object detection quality. Further data compression can be achieved under controlled degradation of the DNN detection quality. Also, CoTeD guarantees an inference request success rate of at least 90%, with an increase of 20%–80% when compared to direct DNN splitting, static JPEG compressions, and DNN model quantization.

In the rest of the paper, Sec. II describes our reference scenario, while Sec. III introduces the CoTeD framework. Sec. IV presents our empirical investigation of the relation between tensor compression loss and object detection performance. We then leverage these results to formulate a mathematically tractable optimization of the system orchestration. Sec. V validates CoTeD and compares its performance against baseline solutions. Finally, we discuss some relevant related work in Sec. VI and draw our conclusions in Sec. VII.

II. Reference Scenario and Developed Testbed

Our reference system scenario, depicted in Fig. 1, includes an edge server, a 5G base station (gNB) and Core Network (CN), and a UAV acting as a 5G User Equipment (UE). The

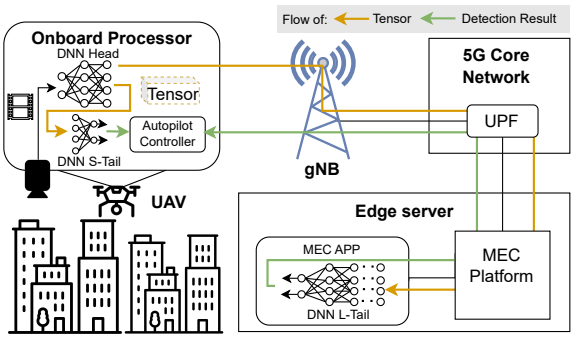

Fig. 1: System scenario and application example: UAV-edge cooperative video object detection for UAV flight control.

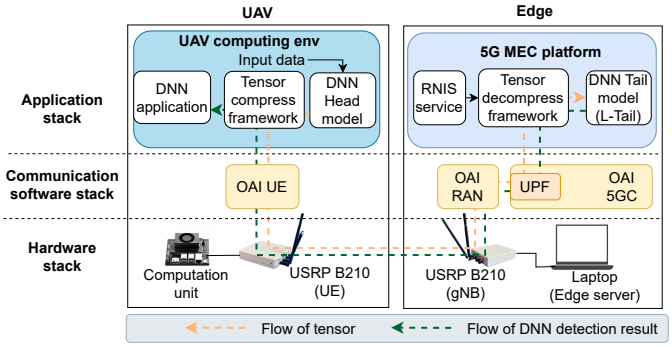

Fig. 2: Schematic representation of our testbed.

multi-access edge computing (MEC) platform uses the 5G-CN's User Plane Function (UPF) as virtualized data plane, thereby ensuring minimal latency and communication distance between the edge application and the 5G UE. For concreteness, we consider an autopilot application navigating the UAV, which is enabled by a DNN for video object detection on the frames captured by the UAV's camera.

Motivated by the resource-constrained nature of onboard processors of UAVs, we distribute execution between the UAV and the edge server, by leveraging CoTeD as a novel DNN split solution. We place the DNN Head and small, lightweight Tail (S-Tail) models at the UAV and the larger, heavy Tail (L-Tail) model at the edge server [11]. The DNN Head model takes the raw data as input, processes it with the first few layers of the DNN, and outputs the activation tensor. The activation tensor can be input to the S-Tail model, which generates detection results swiftly but with low quality. For a better quality result, the activation tensor can be transmitted to the edge server over the 5G network and processed by the L-Tail model on a more powerful computing unit, thus obtaining higher quality detection but with communication need in-between. Notice that the acceptable latency for object detection-based applications (e.g., UAV autopilot or object recognition) is determined by the UAV flying dynamic and/or the frame rate of the video captured by the UAV, hence it is strictly bounded to the sub-second range [12]. Consequently, given the application latency requirements and that the size of the tensor output by the DNN Head is in the order of Kilobytes to Megabytes, the tensor transmission would require substantial bandwidth – i.e., a 5G-connection at least one order of magnitude faster than the one available today (current public 5G networks provide, on average, 100s Mbps in uplink [13]).

To enable the system experimental analysis and performance assessment, we develop the testbed in Fig. 2, which consists of two parts: **UAV** and **edge server**. Each part includes three stacks: the **hardware stack**, the **communication software stack**, and the **application stack**.

Hardware Stack: We utilize an Nvidia Jetson Orin Nano as the UAV's computation unit. This choice is driven by its ability to deliver server-class AI performance under low power consumption, which is critical for extending UAV flight endurance. The edge server is a laptop featuring an Intel I7-7700HQ CPU and Nvidia GTX 1050Ti GPU as the edge server. This setup serves as a representative state-of-the-art [14] mid-tier edge node, providing the necessary multi-core processing power to host the OAI-5GC and MEC platforms simultaneously. Each of them is connected to a Universal Software Radio Peripheral (USRP) device, model B210, to establish a wireless communication channel.

Software Stack: We set up a private 5G connection between the UAV and the edge server leveraging the open-source OpenAirInterface (OAI) project (https://openairinterface.org/). On the UAV side, we configure the USRP as a 5G user using OAI-UE and connect it to our private 5G gNB. On the edge server side, we create our private base station by configuring the USRP as a 5G gNB through the OAI-RAN (Radio Access Network) and run the OAI-5GC (Core Network) on the same server. In addition, to host edge services, we integrate the OAI-MEC platform (https://gitlab.eurecom.fr/oai/orchestration/oai-mec) to the edge server. This architecture enables edge services to directly access the 5G network through the OAI-5GC User Plane Function (UPF), while network-level metrics are exposed to our framework via the OAI-RNIS (Radio Network Information Service).

Experimental Setup: To ensure consistent channel conditions, we first recorded a 5G trace while the UE was in motion within the laboratory. We then emulated the corresponding available bandwidth using the `tc` traffic control tool.

In the application stack, we deploy a DNN Head model and the corresponding DNN application (e.g., UAV autopilot application) on the UAV side and the larger DNN Tail (L-Tail) model on the MEC platform at the edge server. Since the behavior and the performance of the lightweight S-Tail are predictable, the S-Tail has not been implemented to keep the testbed construction leaner. Also, we limit the computational power that can be consumed by the DNN Head model to 10% of the UAV device to mimic the workload that is expected in a multi-task system. To achieve this, we simulate the high GPU load by running a concurrent GPU stress-test during the experiment. We then deploy our tensor compression and decompression framework (see Sec. III) on the UAV and edge server, respectively, aiming at adapting the tensor compression ratio to the real-time quality of the 5G connection. We remark that the above three stacks of our testbed are well decoupled, which provides high flexibility for evaluating our framework across different platforms and connectivity settings. Further details about the testbed can be found in our open-source repository: https://github.com/Rexyyj/CoTeD.

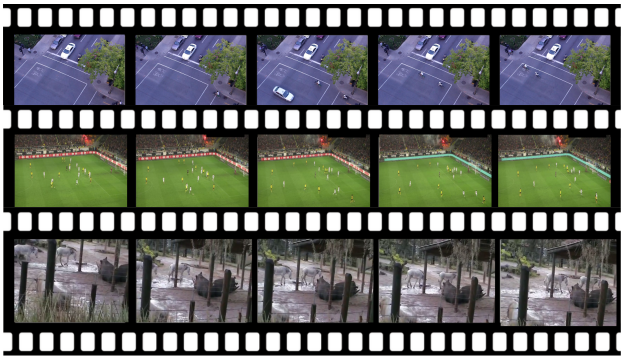

Fig. 3: Datasets samples: St. Marc dataset (top); BEV dataset (middle); ImageNet-VidVRD (bottom).

As a DNN model that is suitable for the application at hand and as part of the testbed instantiation for our analysis and benchmarking, we selected well-established pre-trained real-time video object detection DNNs, namely, YOLOv3 [15] and YOLOv3-tiny [16]. The pre-trained YOLOv3 model is composed of 107 layers, with 62M parameters in `float32` format. Instead, YOLOv3-tiny is designed as a more computationally efficient version of YOLOv3, enabling its deployment in resource-limited scenarios as the mobile network edge with pre-trained model is composed of 24 layers with 8.9M parameters in `float32` format. Table I presents the DNN split point analysis we performed on these models. We found that, for the YOLOv3 model, due to special shortcut branches in the DNN input layers, the only layers with a single activation tensor split point are at the 1st, 3rd, and 10th layer, and the minimum DNN Head output tensor size split point can be found at the 1st and the 10th layers. We thus select the 10th layer as optimal split point for the YOLOv3 mode, with the activation tensor to be transmitted between the Head and L-Tail models in the shape of $[128\times104\times104]$, which results in around 5540 KB. For the YOLOv3-tiny model, all the layers between the 1st and the 8th layer present a single activation tensor to be transmitted between the DNN Head and Tail, after the 8th layer, the DNN starts to branch and requires multiple activation tensors to be transmitted. As the tensor size decreases as the split point is pushed later in the DNN structure, the optimal split point for the YOLOv3-tiny model is thus the 8th layer, with the activation tensor to be transmitted between the Head and L-Tail models in the shape of $[128\times26\times26]$, i.e., around 350 KB.

For the YOLO-v3-tiny model training and testing, we use three data sets with different size and characteristics: (i) the St. Marc dataset (https://www.jpjodoin.com/urbantracker/dataset.html), (ii) the Bird Eye View (BEV) (https://universe.roboflow.com/mohamed-badreldin-cp2tc/bird-eye-view), and (iii) the ImageNet-VidVRD dataset (https://xdshang.github.io/docs/imagenet-vidvrd.html). Sample frames from the three datastes are shown in Fig. 3. In all three datasets, the original frame rate is equal to 30 fps, while the image resolution is, respectively, 1280×720, 640×480, and 1920×1080. The St. Marc dataset includes 1,000 continuous video frames containing three classes, shot by a static camera pointing at a traffic intersection. With reference to our use case this emulates a UAV traffic monitoring scenario. The BEV dataset comprises over 4,000 video frames captured from football matches, serving as a reference for a two-class object detection problem. In this context, we consider the use of UAVs as a complement to, or replacement for, cable robot applications [17]. Finally, the ImageNet-VidVRD dataset includes 200 videos with labels on each frame that may belong to 35 different object classes. As depicted in Fig. 3, this represents a UAV scenario for wildlife monitoring, where human intervention may unintentionally disturb animals, or wildlife habitats may be difficult to access. For each data set, 75% of the video frames are used for model training (including 5% for validation) and 25% for testing. Random shuffling is applied on the training frame set, while the testing set is ordered in time sequence as the original video. Importantly, using the above three datasets allows us to evaluate the robustness of our approach for diverse tasks with different characteristics and levels of complexity.

TABLE I: DNN model split point analysis

YOLOv3		YOLOv3-tiny	
Split layer	Tensor size [kB]	Split layer	Tensor size [kB]
1-st	5540	2-nd	2770
3-rd	11080	4-th	1380
10-th	5540	6-th	690
-	-	8-th	350

III. The CoTeD Framework

This section introduces the functional architecture of our CoTeD framework, depicted in Fig. 4. The notation we use is presented in Table II. As illustrated in Fig. 4, an onboard camera captures a sequence of video frames, denoted by $[\mathbf{F}_1,\ldots,\mathbf{F}_t,\mathbf{F}_{t+1},\ldots]$, where t is the generic frame index. Video frames are given as input to the DNN Head deployed on the UAV. Upon processing $\mathbf{F}_t$, the Head model generates an activation tensor, $\mathbf{V}_t\in\mathbb{R}^{I\times J\times K}$, that is *(i)* input to and processed by the DNN S-Tail model, and *(ii)* transmitted to the edge server and processed by the DNN L-Tail model.

A. Overview of CoTeD

As shown in Fig. 4, CoTeD is composed of two main modules: CoTeD-Mobile and CoTeD-Edge, deployed on the UAV and the edge server, respectively. Each module encompasses a framework manager responsible for configuring the framework settings, facilitating inter-module communication, and interacting with external services. Below, we give an overview of these components and their key functionality, while in the subsequent subsections we detail and motivate the specific algorithmic design choices.

• *Prior to transmission from CoTeD-Mobile to CoTeD-Edge.* $\mathbf{V}_t$ is processed by a sequence of low-complexity operators in CoTeD-Mobile: *(i)* `Difference`, yielding $\mathbf{\Delta}_t$, *(ii)* `Pruning`, generating $\mathbf{P}_t$, and *(iii)* `Compression`, producing $\mathbf{C}_t$ (corresponding, resp., to operators 1 to 3 in Fig. 4). This outcome, $\mathbf{C}_t$, is a highly compressed representation of $\mathbf{V}_t$ that can be efficiently transmitted to CoTeD-Edge.

• *Concurrently with the above operations.* The `Monitor` module (component I in Fig 4) measures: *(i)* tensor compression quality, i.e., compression ratio and reconstructed tensor SNR from $\mathbf{P}_t$, $\widehat{\mathbf{P}}_t$, and $\mathbf{C}_t$, and *(ii)* resource availability and usage, e.g., communication bandwidth. The compressed tensor

TABLE II: Notation of symbols used in the paper

Symbol	Description	Symbol	Description
t	Generic video frame index	d_t^{mAP}	User experienced mAP_F drop
$\mathbf{F}_t$	Video frame sequence	d_t^{sen}	User experienced sen_F drop
$\mathbf{V}_t \in \mathbb{R}^{I\times J\times K}$	Activation tensor generated by $\mathbf{F}_t$	γ_t	Measured SNR between $\mathbf{V}_t$ and $\hat{\mathbf{V}}_t$
$\hat{\mathbf{V}}_t \in \mathbb{R}^{I\times J\times K}$	Reconstructed activation tensor	γ_t^*	Target reconstructed tensor SNR
$\boldsymbol{\Delta}_t \in \mathbb{R}^{I\times J\times K}$	Difference tensor	$\gamma_{\mathbf{a}_t}(p_t, q_{\mathbf{a}_t})$	Estimated SNR between $\mathbf{V}_t$ and $\hat{\mathbf{V}}_t$ for $\mathbf{a}_t$ configuration
$\mathbf{P}_t \in \mathbb{R}^{I\times J\times K}$	Pruned tensor	r_t	Measured compression ratio between $\mathbf{V}_t$ and $\mathbf{C}_t$
$\widehat{\mathbf{P}}_t \in \mathbb{R}^{I\times J\times K}$	Reconstructed pruned tensor	$r_{\mathbf{a}_t}^*$	Target compression ratio for different compression techniques
$\boldsymbol{\mathcal{M}} \in \mathbb{R}^{I\times J\times K}$	Pruning mask	$r_{\mathbf{a}_t}(p_t, q_{\mathbf{a}_t})$	Estimated compression ratio between $\mathbf{V}_t$ and $\mathbf{C}_t$
$\mathbf{C}_t$	Compressed pruned tensor	σ	Size of activation tensor $\mathbf{V}_t$, in bytes
p_t	Pruning threshold, in range [0, 1]	L_t	Available bandwidth for tensor transmission
$\theta_{\mathbf{P}_t}$	Sparsity of the pruned tensor $\mathbf{P}_t$	$l_t(p_t, \mathbf{a}_t, q_{\mathbf{a}_t})$	Used bandwidth for tensor transmission
$\mathbf{a}_t = \{a_t^{\text{jpeg}}, a_t^{\text{decom}}, a_t^{\text{reg}}\}$	Tensor compression technique selector, binary	T_t^{ifr}	DNN inference deadline
$q_{\mathbf{a}_t} = \{q_t^{\text{jpeg}}, q_t^{\text{decom}}, q_t^{\text{reg}}\}$	Quality of compression techniques	$T_{\mathbf{a}_t}^{\text{tr}}$	Compressed tensor transmission time when different compression technique is selected
mAP_F	Frame-level DNN detection mAP	T^{Head}	DNN Head model processing time
sen_F	Frame-level DNN detection sensitivity	$T_{\mathbf{a}_t}^{\text{CoTeD}}$	CoTeD-framework processing time when compression technique $\mathbf{a}_t$ is selected
D_t^{mAP}	User tolerable mAP_F drop	T^{Tail}	DNN Tail model processing time
D^{sen}	User tolerable sen_F drop	$S = (p_s, \mathbf{a}_s, q_{\mathbf{a}_s})$	Sample configs for optimization solver

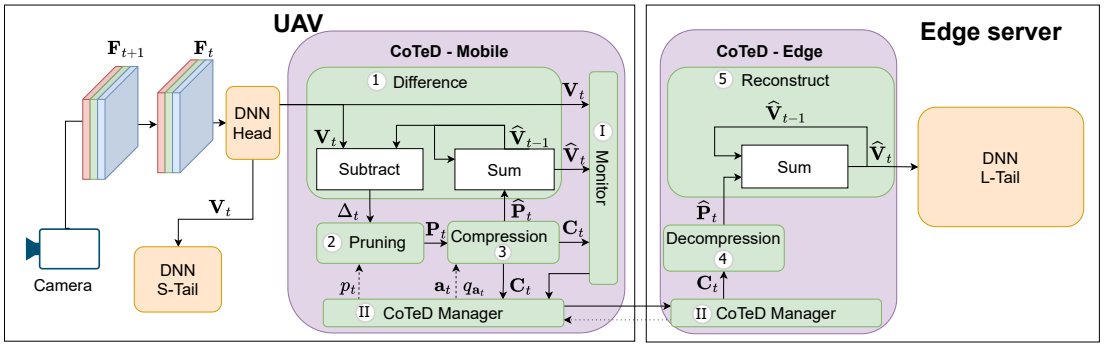

Fig. 4: Scheme of the CoTeD framework: DNN Head and tensor difference, pruning, and compression at the UAV; tensor decompression and reconstruction and DNN L-Tail at the edge server.

$\mathbf{C}_t$ and the measurements in the `Monitor` are next passed to the CoTeD `Manager` (component II in Fig. 4), which is the orchestration module of the framework that manages the communication to CoTeD-Edge and configures the framework setting in real-time to fulfill the application requirements and the dynamic system conditions.

• *Upon reception by CoTeD-Edge.* CoTeD-Edge processes $\mathbf{C}_t$ by the following operators: *(i)* `Decompression`, yielding $\widehat{\mathbf{P}}_t$, and *(ii)* `Reconstruction`, $\hat{\mathbf{V}}_t$, (resp., operators 4 and 5 in Fig. 4). The output of `Reconstruction` is an estimated activation tensor, which is input to the DNN Tail running at the edge server to produce the object detection outcome for frame $\mathbf{F}_t$. The result is relayed back to the UAV via CoTeD, to be used by the application (e.g., UAV's autopilot).

In the following, we detail the framework components by addressing three key questions: *(i)* how to effectively reduce the data volume that the UAV has to transmit to the edge server (Sec. III-B), *(ii)* how to optimize tensor processing at both the UAV and the edge server (Sec. III-C), and *(iii)* how to ensure that the overall latency – including both data transmission and data processing – is within application requirements (Sections III-D and III-E).

B. Tensor difference, pruning, and reconstruction

Motivated by the observation that video frame sequences with high adjacent similarity are commonly compressed by data-deduplication algorithms, a key question focuses on the potential differences between corresponding tensors. To address this question, we performed extensive tests using our testbed and the datasets introduced in Sec. II.

Fig. 5 summarizes the outcome for a representative example: a 10-s test video with 30 fps, i.e., a total of 200 frames. Each video frame contains one target object to be detected.

Specifically, Fig. 5 (left) shows the similarity between consecutive video frames as well as between activation tensors output by the DNN Head ($\mathbf{V}_t$) with different DNN split layers, using the cosine similarity metric, defined as $\frac{\mathbf{V}_t \cdot \mathbf{V}_{t-1}}{\|\mathbf{V}_t\|\|\mathbf{V}_{t-1}\|}$. When consecutive video frames $\mathbf{F}_t$ have high similarity, the corresponding activation tensors ($\mathbf{V}_t$) exhibit high similarity as well. Also, the level of adjacent tensor similarity does not directly relate to the DNN split layer, but strictly follows the similarity trend of the input video frames. These findings

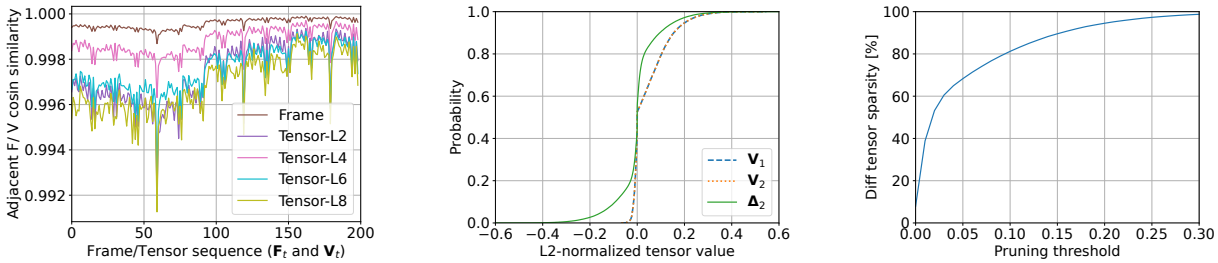

Fig. 5: Representative example of our analysis on the (left) similarity between consecutive input frames and between consecutive YOLOv3-tiny's activation tensors, (middle) CDF of the values in the activation tensor and difference tensor, and (right) impact of the pruning threshold on the difference tensor sparsity.

demonstrate that the activation tensors bear the potential to be compressed with deduplication algorithms when the input video frames have high adjacent similarity.

Next, Fig. 5 (middle) illustrates the cumulative distribution function (CDF) of the L2-normalized values of the activation tensors ($\mathbf{V}_1$, $\mathbf{V}_2$), and of their difference tensor ($\mathbf{\Delta}_2$). Most of the elements of the activation tensors take on values close to zero and have a non-symmetric distribution, which makes their compression and transmission difficult to perform. However, looking at the CDF of the difference tensors, one can see that the values of the $\mathbf{\Delta}_2$ follow a normal distribution with zero mean and very small standard deviation. As $\mathbf{\Delta}_2$ is a difference tensor, it is also fair to assume that its small absolute values will have a negligible impact on the DNN inference and can be removed. It follows that $\mathbf{\Delta}_2$ is a tensor that can be easily compressed, hence transmitted, while retaining the relevant information for an accurate application-level result. Furthermore, as shown in Fig. 5 (right), the sparsity of the difference tensor increases significantly as the pruning threshold increases, thus underlining the effectiveness of this configuration parameter in making the tensor more compressible [18].

Based on the above observations, **it is evident that the `Difference`, `Pruning`, and `Reconstruction` are essential operators as pre-processing steps** at the UAV prior to compression. Hence, we design CoTeD so that, upon receiving the activation tensor ($\mathbf{V}_t$) from the DNN Head, the `Difference` operator in CoTeD-Mobile *(i)* subtracts $\widehat{\mathbf{V}}_{t-1}$ from $\mathbf{V}_t$, generating the difference tensor $\mathbf{\Delta}_t$, and *(ii)* sums the decompressed pruned tensor $\widehat{\mathbf{P}}_t$ with $\widehat{\mathbf{V}}_{t-1}$. CoTeD then stores the summation result, $\widehat{\mathbf{V}}_t$, in the internal memory for the subtract operation in the next time instance. Here, $\widehat{\mathbf{V}}_{t-1}$ denotes the reconstructed pruned activation tensor in the previous time instance (which is initialized as a zero vector).

After `Difference`, the tensor $\mathbf{\Delta}_t$ is fed as input to `Pruning`, generating the pruned tensor $\mathbf{P}_t \in \mathbb{R}^{I \times J \times K}$ by setting to zero the elements of $\mathbf{\Delta}_t$ with absolute normalized value smaller than a threshold p_t. For an efficient algorithmic implementation that can use hardware SIMD (Single Instruction, Multiple Data) parallelism, `Pruning` (i) creates a binary mask $\mathcal{M}$ of values larger than p_t, and (ii) calculates the element-wise product between mask $\mathcal{M}$ and $\mathbf{\Delta}_t$.

In CoTeD-Edge, the `Reconstruction` operator sums the reconstructed pruned tensor $\widehat{\mathbf{P}}_t$ with $\widehat{\mathbf{V}}_{t-1}$ and generates the reconstructed activation tensor $\widehat{\mathbf{V}}_t$. Specifically, $\widehat{\mathbf{V}}_{t-1}$ represents the reconstructed activation tensor in CoTeD-Edge at the previous time instance (initialized to a zero tensor). $\widehat{\mathbf{V}}_t$ is then processed by the DNN L-Tail at the edge server.

Notice that our proposed architecture makes CoTeD able to perform the differential operation on the activation tensors *without the need to cache the original video frame streams*. Also, leveraging $\widehat{\mathbf{V}}_{t-1}$ as reference for the Difference operation reduces the pruning and compression loss effect on the overall detection quality, thus increasing CoTeD's robustness. Indeed, values in the activation tensor $\mathbf{V}_t$ that have a small variation rate can significantly affect the final detection result. Using ($\mathbf{V}_{t-1}$) as a reference would make such values become small in the difference tensor and eventually be pruned by the pruning operator. **Instead, in CoTeD, the small value variations in the activation tensor accumulate over time and, hence, they will eventually be transferred to the DNN L-Tail model.**

C. Tensor compression and decompression

The `Compression` is the last operator in CoTeD-Mobile: it compresses and encodes the high-dimensional tensor $\mathbf{P}_t$ to a data sequence for transmission. To facilitate efficient compression for different application scenarios and edge system states, it is critical to ensure flexibility in adjusting and choosing among different compression options. Below, we outline the possibility for an adjustable process, employing three configurable representative compression algorithms, i.e., JPEG, tensor decomposition, and polynomial regression.

The `Compression` operator selects in real-time the most suitable compression strategy (i.e., one of the possible techniques denoted by $\mathbf{a}_t = \left\{a_t^{\text{jpeg}}, a_t^{\text{decom}}, a_t^{\text{reg}}\right\}$) based on the input quality level $q_{\mathbf{a}_t}$, and then it passes the compressed output to the CoTeD `Manager`. Notice that a_t^{jpeg}, a_t^{decom}, and a_t^{reg} are binary selectors, and only one algorithm can be selected at one time. Also, the quality level $q_{\mathbf{a}_t}$, which ranges in [0, 1], represents the normalized compression settings in the considered quality range for each compression algorithm (namley, JPEG: [50, 100], Decomposition: [1, 5], and Regression: [1,5]). Given a compression technique, the higher the $q_{\mathbf{a}_t}$ value, the higher the compression quality, or, equivalently, the lower the compression ratio, the lower the compression loss. We remark that the resulting compression ratio and quality depend non-linearly on the above configuration parameters. Therefore, we design a dynamic, event-driven, decision-making strategy within the CoTeD `Manager` to manage these parameters, as detailed in Sec. III-E.

The compressed tensor and the metadata (including the selected algorithm, quality level, and implementation settings) are combined in a data structure denoted by $\mathbf{C}_t$. Additionally, the `Compression` operator performs two steps: *(i)* it generates $\widehat{\mathbf{P}}_t$ from $\mathbf{C}_t$, which will be consumed by the Difference operator as described in Sec. III-B, and *(ii)* it transmits $\mathbf{C}_t$ to CoTeD-Edge via CoTeD `Manager`.

On CoTeD-Edge, upon receiving $\mathbf{C}_t$, the `Decompression` operator identifies the compression settings through the metadata in $\mathbf{C}_t$, decompresses the tensor correspondingly, and generates the reconstructed pruned tensor $\widehat{\mathbf{P}}_t$. We remark that the compression process is not always lossless, i.e., $\widehat{\mathbf{P}}_t \neq \mathbf{P}_t$. Below, we outline and motivate the choice of the three representative algorithms we consider, namely, JPEG, Decomposition, and Regression.

JPEG (https://en.wikipedia.org/wiki/JPEG). It is a well-known image compression algorithm that can achieve a high compression ratio with adjustable loss in image quality, denoted as q_t^{JPEG}. We selected it since, as noted above, tensors retain the same characteristics as video frames. We use `simplejpeg` (https://pypi.org/project/simplejpeg/) to implement JPEG compression for tensor $\mathbf{P}_t$, in three steps: *(i)* quantizing the `float` type tensor to `int8` type (due to the `simplejpeg` input requirement), *(ii)* reshaping the quantized tensor to a 3D-3channel tensor (as RGB image) or 2D-1channel tensor (as GRAY scale image), and *(iii)* supplying the reshaped tensor and the JPEG quality setting as input to an off-the-shell JPEG implementation. Thus, the metadata includes the compression selector, the quantization bases and bins, and the reshaped tensor shape. In the `Decompression` operator, we reverse the JPEG compression operation, i.e., from step *(iii)* to *(i)*, and obtain the reconstructed pruned tensor $\widehat{\mathbf{P}}_t$.

Decomposition [19]. As representative of compression techniques specifically designed for tensors, we adopt the CANDECOMP/PARAFAC (CP) decomposition. This is a variant of the tensor rank decomposition, in which, for an arbitrary value R, referred to as decomposition rank, the tensor is expressed as a sum of R rank-1 tensors, which significantly reduces the tensor size. Thus, we use the rank R as the compression quality indicator for the decomposition technique, denoted by q_t^{decom}. For a third-order tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I \times J \times K}$ and a given value of R, the rank-one tensors can be obtained by solving the following problem: $\min_{\boldsymbol{\mathcal{X}}} \left\| \boldsymbol{\mathcal{X}} - \widehat{\boldsymbol{\mathcal{X}}} \right\|_F$, where $\|\boldsymbol{\mathcal{Y}}\|_F = \sqrt{\sum_{i=1}^{I} \sum_{j=1}^{J} \sum_{k=1}^{K} |\mathcal{Y}_{ijk}|^2}$. To implement the decomposition procedure and solve the above optimization, we adopt TensorLy library (https://tensorly.org), which can be accelerated using [20]. To further accelerate the decomposition and reduce the re-composition loss, the `Compression` operator implements the decomposition compression at the tensor slice level, i.e., it splits the pruned tensor $\mathbf{P}_t$ in N slices, with $N{=}I$), and it performs CP decomposition on each slice. Then the `Decompression` operator reconstructs tensor $\widehat{\mathbf{P}}_t$ slice by slice. It follows that the compression ratio for decomposition can be expressed as, $r^{\text{decom}} = \frac{J \cdot K}{(J+K) \cdot R}$. Importantly, if R is equal to the rank of tensor $\boldsymbol{\mathcal{X}}$, then $\widehat{\boldsymbol{\mathcal{X}}} = \boldsymbol{\mathcal{X}}$. Also, as the R decreases, the compression ratio increases while the estimation quality decreases, and vice versa, which provides a flexible way to adjust the compression performance. In the `Compression` operator, we indeed use the decomposition rank R as the compression quality indicator for the decomposition technique, denoted by q_t^{decom}.

Regression. As a representative of data-driven analytics approaches, we adopt polynomial regression to summarize the data in tensor $\mathbf{P}_t$ and use the polynomial degree as the compression quality indicator q_t^{reg}. We implement regression compression for tensor $\mathbf{P}_t$ in four steps. *(i)* Slice $\mathbf{P}_t$ into N slices, with the total number of elements in the slice not larger than 65,536 (to store the value index in `int16` format). *(ii)* Reshape the tensor slices to 1-D vectors and mark the tensor values as dependent variable sets and the corresponding indexes as the independent variable sets. *(iii)* To guarantee the polynomial equation only models the trend of meaningful (i.e., non-zero) values in the tensor, remove the zero value and corresponding position indexes from the dependent variable sets and the independent variable sets. *(iv)* Input the independent variables, the dependent variable, and the regression degree to a regression solver (e.g., Numpy polyfit function https://numpy.org/doc/stable/reference/generated/numpy.polyfit.html). The results of this operation are the polynomial coefficients, denoted by $\mathbf{Q}_t$, and the non-zero value indexes (the positive and negative value indexes are separated), which represent our compressed data. Again, the `Decompression` operator reconstructs tensor $\widehat{\mathbf{P}}_t$ slice by slice. For every slice, it calculates the value of each positive and negative index, and it inserts them into a zero vector of the same size as the tensor slice. Finally, it reshapes the vector to the shape of the slice. The metadata thus includes the compression algorithm selector, the number and shape of each tensor slice, and the vectors' positive and negative value indexes. The compression ratio of the regression compression is expressed as $r^{\text{reg}} = \frac{\text{Size}(\mathbf{P}_t)}{\text{Size}(\mathbf{P}_t) \cdot \theta_{\mathbf{P}_t} / 2 + \text{Size}(\mathbf{Q}_t)} \approx \frac{2}{\theta_{\mathbf{P}_t}}$ where $\theta_{\mathbf{P}_t}$ is the sparsity of the tensor $\mathbf{P}_t$. Since the size of the polynomial coefficients $\mathbf{Q}_t$ is typically much smaller than the tensor size, it can be ignored during the compression ratio estimation. Thus, the polynomial compression ratio r^{reg} is approximated to $\frac{2}{\theta_{\mathbf{P}_t}}$.

D. CoTeD Monitor

On CoTeD-Mobile, the `Monitor` component is responsible for monitoring the tensor compression quality and system resource consumption, providing the necessary information for the `Manager` to make precise and effective real-time framework configurations.

For tensor *compression quality monitoring*, on one hand, the `Monitor` measures the SNR (denoted by γ_t and measured in dB), between the activation tensor $\mathbf{V}_t$ and the reconstructed activation tensor $\widehat{\mathbf{V}}_t$, as

$$\gamma_t = 10 \log_{10} \left(\frac{\mathbb{E}[{\mathbf{V}_t}^2]}{\mathbb{E}[(\widehat{\mathbf{V}}_t - \mathbf{V}_t)^2]} \right) \tag{1}$$

where $\mathbb{E}[\cdot]$ denotes the expectation operator. Our observations indicate that most values in the tensor $\mathbf{V}_t$ are small and have a negligible impact on the performance of the object detection DNN. In contrast, a few significant values primarily determine the DNN's detection quality. Thus, common error metrics, like

the mean square error (MSE), cannot be used to represent the tensor reconstruction quality in CoTeD, since the distortion of tensor significant values will vanish in such computation due to the average operation. Instead, we consider the SNR, which can well capture the significant distortions in the tensor thanks to the square operation in the signal and noise power calculation. The `Monitor` also measures the compression ratio r_t between the activation tensor $\mathbf{V}_t$ and the compressed tensor $\mathbf{C}_t$, which is given by:

$$r_t = \frac{\texttt{size}(\mathbf{V}_t)}{\texttt{size}(\mathbf{C}_t)} \tag{2}$$

where we have underlined the dependency of the compression ratio (through $\mathbf{C}_t$) on the pruning factor, the compression technique, and the compression quality. Additionally, the `Monitor` queries the mobile network (e.g., the MEC platform) for real-time values of the available bandwidth, denoted by L_t. All such measurements are aggregated and transferred to the `Manager` component every time an activation tensor $\mathbf{V}_t$ is processed by CoTeD-Mobile.

E. CoTeD `Manager`

The `Manager` component runs on both CoTeD-Mobile and CoTeD-Edge, and it plays a pivotal role in the orchestration of the overall framework. When the `Manager` is initiated on CoTeD-Edge, it acts as a RESTful server and listens to the requests from the UAV. Notably, upon the arrival of a request, it extracts and forwards the compressed tensor $\mathbf{C}_t$ to the `Decompression` operator, collects the DNN L-Tail detection output, and returns the collected result to the UAV. When instead the `Manager` operates on CoTeD-Mobile, it performs three major tasks: *(i)* it transmits the compressed tensor $\mathbf{C}_t$ to CoTeD Edge using a RESTful request, *(ii)* it collects the compression quality and system measurements from the `Monitor`, and *(iii)* it updates the framework configuration (i.e., the pruning threshold p_t, the compression technique selector $\mathbf{a}_t$, and the compression quality q_{a_t}) according to the real-time network condition and application requirements.

To perform the above third task, we consider three *reconfiguration triggering events* that imply the need for an update of the framework configuration: *(i)* a change in the available radio bandwidth larger than a threshold, e.g., 20% w.r.t. the previous update, *(ii)* a drop in the measured reconstructed tensor SNR γ_t below the target value γ_t^* (detailed in Sec. IV), and *(iii)* a change in the application requirements, e.g., accuracy of object detection. At each framework update, the `Manager` finds the configuration t hat optimally trades off application requirements with radio bandwidth consumption (see Sec. IV).

In general, for CV tasks, the DNN detection performance is measured as detection mean averaged precision ("mAP" for short), and as detection sensitivity ("sen" for short), across the whole test dataset. The Manager should however guarantee frame-level detection performance, i.e.,

$$\text{mAP}_F = \frac{1}{N}\sum_{n=1}^{N}\text{AP}_n, \quad \text{sen}_F = \frac{\text{TP}}{\text{TP}+\text{FN}}\,. \tag{3}$$

In (3), N denotes the target object classes in a video frame, AP_n indicates the averaged precision on the detection of class n, and TP and FN denote, respectively, the number of true-positive and false-negative detections within the video frame. Then we define the experienced detection performance in splt DNN as detection mAP drop, denoted by d_t^{mAP}, and detection sensitivity drop, denoted by d_t^{sen}, with respect to the optimal mAP and sensitivity values that the DNN can achieve (i.e., when the DNN is not split) for the target video frame. We measure the detection mAP and sensitivity for each input frame, and impose that neither the object detection mAP drop nor the sensitivity drop exceed their respective maximum tolerable value. More formally, we must have: $d_t^{\text{mAP}} \leq D_t^{\text{mAP}}$ and $d_t^{\text{sen}} \leq D_t^{\text{sen}}$, $\forall t$.

Furthermore, given the inference deadline T_t^{ifr} required by the application at hand and the adopted compression technique a_t, the `Manager` first calculates the time left for transmitting the compressed tensor $\mathbf{C}_t$, as

$$T_{\mathbf{a}_t}^{\text{tr}} = T_t^{\text{ifr}} - T^{\text{Head}} - T^{\text{Tail}} - T_{\mathbf{a}_t}^{\text{CoTeD}}, \tag{4}$$

where T^{Head}, T^{Tail}, and $T_{\mathbf{a}_t}^{\text{CoTeD}}$ are, respectively, the processing time at the DNN Head and L-Tail and through the whole sequence of CoTeD operators. Notice that these values can be considered as constant and measured in advance. Then, given the framework pruning threshold, compression technique, and the compression setting to be used at t, the corresponding compression ratio can be expressed as $r_{\mathbf{a}_t}(p_t, q_{\mathbf{a}_t})$. The bandwidth required to meet the transmission deadline, $T_{\mathbf{a}_t}^{tr}$, is thus:

$$l_t(p_t, \mathbf{a}_t, q_{\mathbf{a}_t}) = \frac{\sigma}{T_{\mathbf{a}_t}^{\text{tr}} \cdot r_{\mathbf{a}_t}(p_t, q_{\mathbf{a}_t})} \tag{5}$$

where σ is the size of the original tensor.

Finally, the `Manager` selects the configuration to be used so that: *(i)* the bandwidth occupancy is minimized and *(ii)* the available bandwidth limit L_t is not exceeded, and *(iii)* the application-layer constraints are fulfilled. To this end, it applies the decision-making strategy defined and analysed in the next section.

IV. CoTeD Configuration Strategy

As stated above, the `Manager` is in charge of dynamically selecting the optimal framework configuration, i.e., pruning threshold p_t, adopted compression technique $\mathbf{a}_t$, and compression quality q_{a_t}. To this end, it solves the following optimization problem, $\mathbf{O}(t)$, at any time t if any of the three reconfiguration triggering events specified in Sec. III-E occurs:

$$\mathbf{O}(t): \quad \min_{\{p_t, \mathbf{a}_t, q_{\mathbf{a}_t}\}} \; l_t(p_t, \mathbf{a}_t, q_{\mathbf{a}_t}) \tag{6}$$

$$s.t. \quad d_t^{\text{mAP}} \leq D_t^{\text{mAP}} \tag{7}$$

$$d_t^{\text{sen}} \leq D_t^{\text{sen}} \tag{8}$$

$$l_t(p_t, \mathbf{a}_t, q_{\mathbf{a}_t}) \leq L_t \tag{9}$$

$$a_t^{\text{jpeg}} + a_t^{\text{decom}} + a_t^{\text{reg}} = 1 \tag{10}$$

$$p_t \in [0,1) \quad ; \quad q_{\mathbf{a}_t} \in \mathbb{N}^+\,. \tag{11}$$

The objective function $l_t(p_t, \mathbf{a}_t, q_{\mathbf{a}_t})$ is given in (5). Constraints (7)–(9) guarantee that under the selected compression technique, compression quality, and pruning threshold setting, the maximum mAP and sensitivity drop values and the

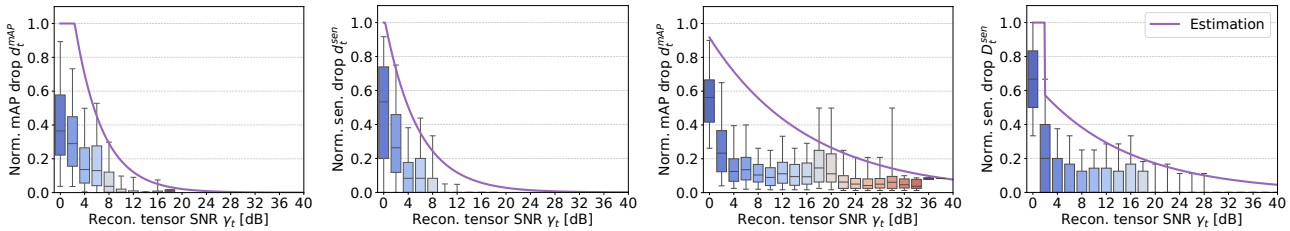

Fig. 6: Normalized DNN object detection mAP/sensitivity drop vs. SNR of the reconstructed tensor $\widehat{\mathbf{P}}_t$ for the BEV (left: mAP drop; middle-left: sensitivity drop) and, St. Marc (middle-right: mAP drop; right: sensitivity drop) datasets.

available bandwidth are not exceeded. Also, constraint (10) imposes that only one compression technique is selected at a time. Once an optimal solution for $\mathbf{O}(t)$ is found, the `Manager` updates the framework configuration accordingly. The video frames are then processed with the same configuration until the next triggering event occurs. When no feasible solution is found for $\mathbf{O}(t)$, the `Manager` keeps the previous configuration and notifies the UAV application about the associated constraint violation. We remark that:

• The objective function in (6) is nonlinear as both the tensor compression ratio $r_{\mathbf{a}_t}(p_t, q_{\mathbf{a}_t})$ and the transmission time $T^{\text{tr}}_{\mathbf{a}_t}$ (determining $l_t(p_t, \mathbf{a}_t, q_{\mathbf{a}_t})$ as per (5)) depend on the compression framework configuration in a non-linear manner;
• No exact analytical relation exists between the mAP (sensitivity) drop d_t^{mAP} (d_t^{sen}) and the framework configuration parameters p_t, $\mathbf{a}_t$, and $q_{\mathbf{a}_t}$.

To overcome the above challenges, we proceed as follows.

First, we decompose problem $\mathbf{O}(t)$ in three sub-problems, denoted by $\widehat{\mathbf{O}}(\mathbf{a}_t)$, one for each compression technique. *Second,* given $\mathbf{a}_t$, $T^{\text{tr}}_{\mathbf{a}_t}$ is a constant. We then leverage (5) to replace the minimization of the bandwidth occupancy with the maximization of the compression ratio $r_t(p_t, \mathbf{a}_t, q_{\mathbf{a}_t})$. *Third,* to make constraints (7)–(9) solvable, we model the relationship between the mAP (sensitivity) drop d_t^{mAP} (d_t^{sen}) with the reconstructed tensor $\widehat{\mathbf{V}}_t$'s SNR, denoted by γ_t, using (12), and estimate γ_t from p_t, $\mathbf{a}_t$, and $q_{\mathbf{a}_t}$ through dynamic linear interpolation. *Finally,* when feasible solutions exist for the sub-problems, the compression technique ($\mathbf{a}_t$) and the corresponding sub-problem solutions (p_t and $q_{\mathbf{a}_t}$) that correspond to the highest compression ration are selected as the solution of $\mathbf{O}(t)$.

Sensitivity and mAP drop modeling. As the intuition suggests and our experimental investigation confirmed, the DNN detection performance drop (for both d_t^{mAP} and d_t^{sen}) decreases with the increase of the reconstructed tensor $\widehat{\mathbf{V}}_t$'s SNR γ_t. This relationship, however, is strongly related to the input frame complexity and affected by the non-linearity of the DNN model, which makes it difficult to derive an analytical formulation thereof. Importantly, through an extensive experimental campaign on our testbed, we found that, for $\gamma_t{>}0$, this relationship can be modeled by a piecewise negative exponential equation, i.e.,

$$D_t^{(y)} = \begin{cases} 1 & \text{if } \gamma_t < \max(b^{(y)}, \frac{\ln(h^{(y)})}{k^{(y)}}) \\ h^{(y)} \cdot e^{-k^{(y)} \cdot \gamma_t} & \text{else} \end{cases} \tag{12}$$

where $D_t^{(y)}$ ((y)=mAP, sens) represents the DNN performance drop, and $b^{(y)}$, $h^{(y)}$, and $k^{(y)}$ ((y)=mAP, sens) are parameters that can be experimentally estimated.

To give an intuition, we illustrate some example measurements, using the BEV, the St. Marc, and the ImageNetVidVRD datasets mentioned in Sec. II. Fig. 6 shows the box plots of DNN performance drop measurements for different reconstructed tensor SNR bins and the corresponding fitting curves. Each bin includes the measurements with the reconstructed tensor SNR within 2 dB windows; the results have been obtained considering 200 video frame samples in each test dataset. The left and middle-left plots correspond to the DNN detection mAP drop and sensitivity drop for BEV dataset, while the middle-right and right plots correspond to the St. Marc dataset. More in detail, boxes show the inter-quartile range (IQR) of the SNR measurements, with the box edges corresponding to the first and third quartiles, respectively. The whiskers extend to the smallest and largest values within the 5th and the 95th percentile, while the measurements beyond the whiskers are considered to be outliers. During the experiment, we performed the split inference on the test video frame sets using all possible combinations of the framework settings. The measurements thus represent the general CoTeD's performance.

Fig. 6 shows that both the normalized DNN detection mAP drop and sensitivity drop exhibit negative correlation with the reconstructed tensor SNR γ_t for all datasets. However, such a relation varies across the datasets. To better describe it, we filter out the outliers in each SNR bin and fit (12) with the top 1% measurements of the mAP drop and sensitivity drop in each reconstructed tensor SNR bin by minimizing the MSE between the estimated and the measured values. We emphasize that, fitting the estimation curve with the top 1% DNN performance drop measurements in each SNR bin helps the model to capture the general upper bound of the DNN performance drop and makes our approximation more robust in dynamic scenarios. The parameters used for approximating the DNN detection performance drop are reported in Table III.

Decomposition into sub-problems. To ensure that the drop of the DNN detection metrics for video frame t does not exceed the tolerable values, we use them in (12), along with the parameters presented in Table III. By doing so, we obtain the corresponding minimum values for the tensor SNR, denoted by γ_t^{mAP} and γ_t^{sen} (resp.). Then the two constraints on the SNR of the reconstructed tensor $\widehat{\mathbf{P}}_t$ can be expressed by a single constraint, as: $\gamma_t^* = \max(\gamma_t^{\text{mAP}}, \gamma_t^{\text{sen}})$. Since the SNR of the reconstructed tensor is directly related to the selected compression technique and configuration, we highlight such dependencies in the tensor SNR notation as $\gamma_{\mathbf{a}_t}(p_t, q_{\mathbf{a}_t})$.

TABLE III: Approximation function parameters for mAP and sensitivity drop (D_t^{mAP} and D_t^{sen})

	BEV	St. Marc	ImageNetVidVRD
$\mathbf{b^{mAP}}$	0	0	6
$\mathbf{h^{mAP}}$	1.674	0.917	2.385
$\mathbf{k^{mAP}}$	0.218	0.062	0.176
$\mathbf{b^{sen}}$	0	2	6
$\mathbf{h^{sen}}$	1.044	0.654	3.451
$\mathbf{k^{sen}}$	0.182	0.067	0.210

Based on the above findings, we decompose the optimization problem $\mathbf{O}(t)$ in three sub-problems, one for each compression technique, and write the sub-problem for $\mathbf{a}_t$ as:

$$\widehat{\mathbf{O}}(\mathbf{a}_t): \quad \max_{\{p_t, \mathbf{a}_t, q_{\mathbf{a}_t}\}} \; r_{\mathbf{a}_t}(p_t, q_{\mathbf{a}_t}) \tag{13}$$

$$s.t. \quad \gamma_{\mathbf{a}_t}(p_t, q_{\mathbf{a}_t}) \geq \gamma_t^* \tag{14}$$

$$r_{\mathbf{a}_t}(p_t, q_{\mathbf{a}_t}) \geq r_{\mathbf{a}_t}^* \tag{15}$$

$$p_t \in [0, 1) \quad ; \quad q_{\mathbf{a}_t} \in \mathbb{N}^+ . \tag{16}$$

In the above formulation, given technique $\mathbf{a}_t$ and exploiting (5), we defined the target minimum compression ratio as $r_{\mathbf{a}_t}^*=\sigma/(L_t \cdot T_{\mathbf{a}_t}^{\text{tr}})$ and rewrote the constraint on the maximum bandwidth occupancy as Constraint (15).

Dynamic interpolation for sub-problem solving. To solve $\widehat{\mathbf{O}}(\mathbf{a}_t)$, we empirically characterize the relation between $\gamma_{\mathbf{a}_t}(p_t, q_{\mathbf{a}_t})$ ($r_{\mathbf{a}_t}(p_t, q_{\mathbf{a}_t})$) and the framework configuration parameters p_t and $q_{\mathbf{a}_t}$. According to our experimental tests, such relations can be conveniently approximated through linear functions. Specifically, we used a dynamic linear interpolation method over a history window size denoted by W. Initially, the `Manager` selects S sample configuration points $(p_s, \mathbf{a}_s, q_s)$ (uniformly distributed on the sample space), processes the first $W \cdot S$ frames (W frames per sample), and collects the reconstructed tensor SNR and compression ratio measurements in two maps, i.e., SNR-map and compression (CMP)-map. In such maps, each sample point corresponds to W history measurements stored in ring buffers. The `Manager` uses the mean value of each sample point's history to perform interpolation and estimate the $\gamma_{\mathbf{a}_t}$. A similar method, but using the minimum value, is followed to estimate $r_{\mathbf{a}_t}$. During runtime, the `Manager` inserts the measured SNR γ_t (and compression ratio r_t) of the reconstructed tensor, along with the corresponding framework configuration, into the ring buffer of the SNR-map (and CMP-map). This ensures that the maps are always up to date.

We tested our estimation method with JPEG compression technique on 200 video frame samples using the St. Marc dataset. In the test, we configured CoTeD so as to process the test video frames with pre-set dynamic framework configurations and obtain continuous varying reconstructed tensor SNR and compression ratio measurements. Then we applied the dynamic interpolation method described above to estimate the reconstructed tensor SNR and tensor compression ratio, using different history window sizes, i.e., $W=\{3, 6, 9\}$.

The right and left plots in Fig. 7 show the estimation error versus the tested video frame sequence for the reconstructed tensor SNR and the tensor compression ratio, respectively. From Fig. 7(left), we observe that the SNR estimation error varies with the video frame sequence and the value of W, with the average estimation error being 7.63%, 8.14%, and 8.49% for $W=3, 6, 9$, respectively. Similarly, Fig. 7(right) underlines that the compression ratio estimation error varies with the video frame sequence, and the average estimation error is 12.69%, 13.17%, 14.11% for $W=3, 6, 9$, respectively. We conclude that, the smaller the history window size, the smaller the estimation error and the shorter the warm-up phase, thus making the reconstructed tensor SNR and tensor compression ratio estimation more responsive to the system dynamics. Also, we remark that we obtained similar performance when we applied our dynamic interpolation method to estimate the reconstructed tensor SNR and tensor compression ratio for the Decomposition and Regression compression techniques.

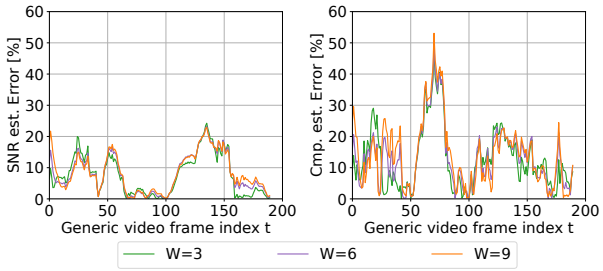

Fig. 7: Reconstructed tensor SNR (left) and tensor compression ratio (right): measurement and estimation.

Final configuration selection. Given the solution to each sub-problem, the solution to the overall optimization problem $\mathbf{O}(t)$ is given by the compression technique $\mathbf{a}_t$ whose optimal configuration (i.e., p_t and $q_{\mathbf{a}_t}$) yields the highest compression ratio. Importantly, each $\widehat{\mathbf{O}}(\mathbf{a}_t)$ is a linear optimization problem with real-valued decision variables (LP), and can thus be efficiently solved using, e.g., the `pymoo` solver [21].

The overall CoTeD configuration strategy is summarized in Fig. 8. As discussed in Sec. III-E, this strategy is triggered by the *reconfiguration triggering events*, which are checked at every time instance before processing the activation tensor $\mathbf{V}_t$ on CoTeD-Mobile. When a feasible solution exists for $\mathbf{O}(t)$, the corresponding framework configuration is applied to process $\mathbf{V}_t$; otherwise, the framework configuration remains unchanged from the previous time instance.

V. Experimental Results

This section presents the experimental performance evaluation of CoTeD that we conducted using our testbed, described in Sec. II. The datasets used in the study, with different size and characteristics, are also described in that section.

We first compare in Sec. V-A the performance achieved under our CoTeD framework and the split DNN paradigm against the case where the DNN is deployed entirely at the UAV. We then demonstrate in Sec. V-B the real-time orchestration performance of the CoTeD's `Manager` under different operational conditions, showing the capability and robustness of our orchestration algorithm to address system-level and application-level requirements. We remind that the constraints

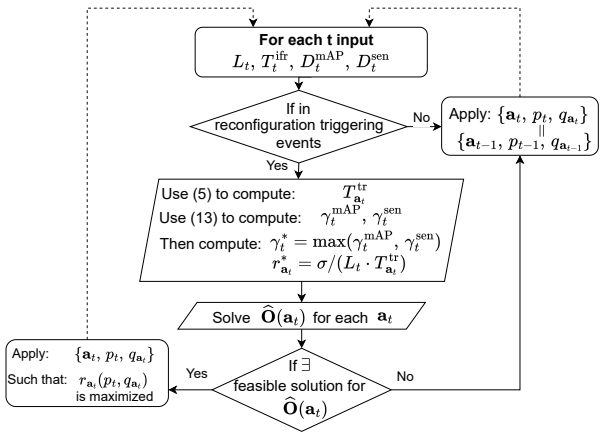

Fig. 8: Flow chart of the configuration update strategy.

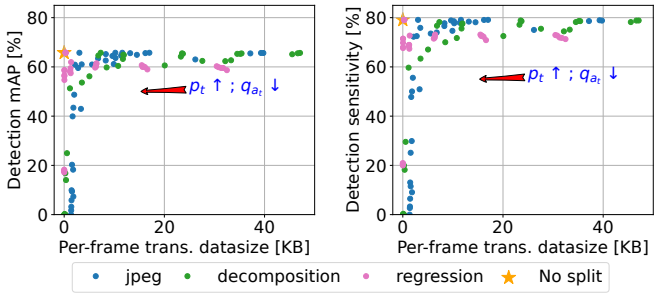

Fig. 9: DNN detection performance: (left) mAP, (right) sensitivity, as functions of the averaged per-frame transmitted data size, under different framework configurations.

include the bandwidth availability L_t, the split DNN inference request success rate, calculated as the percentage of requests that fulfill all performance requirements (i.e., the object detection performance (D_t^{mAP}, D_t^{sen}), and the inference deadline T_t^{ifr}). Moreover, we study in Sec. V-C the relative properties of CoTeD with respect to baseline approaches, namely, static tensor compression solutions, in terms of DNN object detection performance and system resource consumption.

A. *CoTeD with split DNN vs. local DNN execution*

We evaluate the object detection and inference time performance when the CoTeD framework is adopted with the DNN being split between UAV and edge server, and compare it against the case where the DNN is deployed and run entirely at the UAV. To this end, for the split deployment, we performed multiple inference iterations for all video frames, under different static CoTeD-framework configurations (i.e., pruning threshold p_t, compression technique $\mathbf{a}_t$, and compression quality $q_{\mathbf{a}_t}$) that were uniformly selected from the configuration space. For the local deployment, instead, we performed only one inference iteration across all the video frame samples, as in this case the system configuration does not change over time.

Fig. 9 presents the averaged object detection mAP and sensitivity as functions of the averaged per-frame transmitted data size when different compression techniques are adopted by CoTeD (i.e., JPEG, Decomposition, and Regression). Note that, in the case of local deployment, we do not need to transfer any data. In the split deployment, different CoTeD framework configurations lead to different volumes of data to be transmitted from the UAV to the edge server for each video frame. As expected, for each compression technique, the amount of per-frame transmitted data decreases as the pruning threshold p_t increases and/or the compression quality $q_{\mathbf{a}_t}$ decreases. When the per-frame transmitted data size is smaller than 10 KB, the averaged detection mAP and sensitivity decrease with the decrease of the per-frame transmitted data size (in which case the pruning threshold p_t increases and/or the compression quality $q_{\mathbf{a}_t}$ decreases). In this configuration range, by properly selecting the compression technique, one can trade off DNN detection performance with bandwidth consumption. Instead, when the per-frame transmitted data size exceeds 10 KB, the CoTeD's detection mAP and sensitivity always match the performance of the non-split DNN. It follows that **CoTeD can reduce up to 97% the volume of the data to be transmitted from the UAV to the edge server (with respect to the 350 KB tensor raw size), without noticeably degrading the detection performance of the non-split DNN.**

TABLE IV: Per-frame latency for inference with local DNN and with split DNN & CoTeD

Per-frame latency [ms]	**Mean & St. dev.**	**90-th perc.**
Local DNN (UAV)	381.0 ± 3.6	420.0
Split DNN Head (T^{Head}, UAV)	94.0 ± 1.4	99.0
Split DNN L-Tail (T^{Tail}, Edge)	4.5 ± 0.5	4.5
CoTeD JPEG time $T^{\text{CoteD}}_{a_t^{\text{jpeg}}}$	4.0 ± 0.5	4.9
CoTeD Decomposition time $T^{\text{CoteD}}_{a_t^{\text{decom}}}$	88.0 ± 42.0	140.0
CoTed Regression time $T^{\text{CoteD}}_{a_t^{\text{reg}}}$	108.0 ± 42.3	145.0

Next, Table IV presents the mean, standard deviation, and 90-th percentile values of the latency measurements for local DNN inference and split DNN with CoTeD. We can observe that, due to the limited computational resources at the UAV, the local DNN inference latency is very high, on average 381 ms per-frame. Conversely, under split DNN with CoTeD, the DNN processing time reduces to $T^{\text{Head}}+T^{\text{Tail}}$=98.5 ms. As defined in (4), the inference latency for the split DNN deployment should also include the delay introduced by the CoTeD framework, $T^{\text{CoTeD}}_{\mathbf{a}_t}$, and the tensor transmission time $T^{\text{tr}}_{\mathbf{a}_t}$. When JPEG compression is adopted, $T^{\text{CoTeD}}_{\mathbf{a}_t}$ is the lowest (4 ms on average), while the other two compression techniques lead to higher delays (100 ms on average) and higher variability. As for $T^{\text{tr}}_{\mathbf{a}_t}$, this highly depends on the available bandwidth (which significantly vary over time) and the framework configuration. By fixing its value to 20 ms (a fair value in the case of medium-high bandwidth availability), one can notice that the split DNN with the CoTeD approach reduces the inference latency by 40%–70%, relatively to the case of local DNN deployment. (A detailed analysis of $T^{\text{tr}}_{\mathbf{a}_t}$ is presented in Sec. V-B.)

B. *CoTeD* `Manager` *orchestration performance*

One of the essential functions of the CoTeD `Manager` is to adapt the CoTeD framework configuration in real-time to fulfill the system and application requirements. To validate the `Manager`'s orchestration performance, we designed three different application scenarios, as listed below.

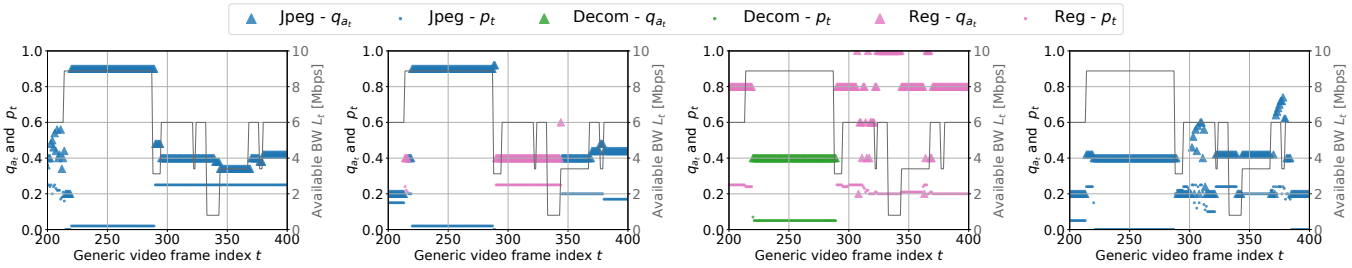

Fig. 10: Real-time CoTeD-framework configuration for the LTL (left), MTL (middle-left), MTL-HL (middle-right), and HTL-LM (right) scenarios.

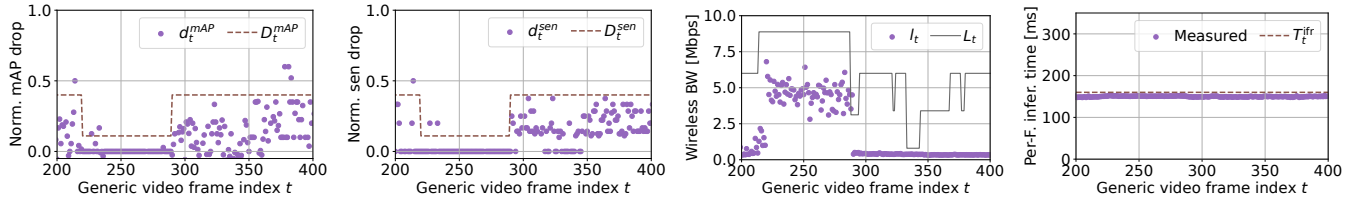

Fig. 11: LTL scenario: CoTeD `Manager` orchestration performance. (left) Normalized DNN mAP drop, (middle-left) Normalized DNN sensitivity drop, (middle-right) Used wireless bandwidth, and (right) Per-frame inference time.

• **Low target latency (LTL)**: The UAV is performing critical tasks (e.g., steering or approaching a target) and the requirement for the maximum inference latency for object detection is very strict, e.g., T_t^{ifr}=150 ms [22].

• **Medium target latency (MTL)**: The UAV is performing normal navigation tasks (e.g., hovering or cruising) and has a lower requirement on the maximum object detection inference latency, e.g., T_t^{ifr}=300 ms [22].

• **Medium target latency & High UAV load (MTL-HL)**: The UAV's GPU is experiencing a high computation load due to other onboard applications, making the JPEG compression speed slower than for the other two (much lighter) compression techniques. As before, the object detection inference deadline is set to T_t^{ifr}=300 ms [22].

• **High target latency & large DNN model (HTL-LM)**: The UAV is performing object detection tasks in a steady state (e.g., hovering in the sky) and has a low target maximum object detection inference latency, e.g., T_t^{ifr}=500 ms [22]. However, due to the higher detection quality required by the task, it is necessary to use the (larger) YOLOv3 model, and change some of the parameter settings accordingly, namely, we set: T^{Head}=280 ms, T^{Tail}=40 ms, and $T^{\text{CoteD}}_{a_t^{\text{jpeg}}}$=40 ms.

Further, we let the bandwidth available between the UAV and the edge server (L_t) vary according to a 5G bandwidth trace that we obtained using our testbed. As for the application requirements, we consider D_t^{mAP}=D_t^{sen} and let it vary according to a pre-defined sequence of values that is negatively correlated with L_t. Finally, we set T^{Head}, T^{Tail}, and $T^{\text{CoTeD}}_{\mathbf{a}_t}$ equal to their 90-th percentile value reported in Table IV.

Figures 10 (left), 10 (middle-left), 10 (middle-right), and 10 (right) show the temporal evolution of the CoTeD's configuration (presented as the continuous video frame index t from the 200th to the 400th video frame), for the LTL, MTL, MTL-HL, and HTL-LM scenarios, respectively. Specifically, Fig. 10 presents the value of the pruning threshold p_t and the normalized compression quality $q_{\mathbf{a}_t}$ (denoted with "dot" and "triangle" markers, respectively) on the left y-axis, and the value of the available bandwidth on the right y-axis. The compression techniques selected by CoTeD are denoted with different colors (blue: JPEG, green: Decomposition, and pink: Regression). Similarly, Figures 11, 12, 13, and 14 illustrate the CoTeD `Manager` orchestration performance for LTL, MTL, and MTL-HL scenarios respectively, including (a) the normalized mAP drop d_t^{mAP}, (b) sensitivity drop d_t^{sen}, (c) used bandwidth l_t, and (d) per-frame inference time on the y-axis.

LTL scenario. Fig. 10 (left) illustrates the framework configuration evolution for the LTL scenario, in which the target inference latency is very low (T_t^{ifr}=150 ms) – a value that only the JPEG compression technique can meet. The operational conditions are such that, when the available bandwidth L_t is high (e.g., between the 220th and the 280th frame), the tolerance on detection performance drop D_t^{mAP} and D_t^{sen} reduces to 0.15, as highlighted by the brown dotted line in Fig. 11 (left) and Fig. 11 (middle-left). In this case, the CoTeD `Manager` selects a low pruning threshold (p_t=0.05) and a high normalized compression quality ($q_{\mathbf{a}_t}$=0.9), which lead to more information being transmitted to the edge server and guarantees a high reconstructed tensor SNR. As a result, in this time interval, the detection performance drop (d_t^{mAP} and d_t^{sen}) decreases significantly, as shown by Fig. 11 (left) and Fig. 11 (middle-left)). In contrast, the used bandwidth increases, although it does not exceed the available bandwidth limit (Fig. 11 (middle-right)).

Different operational conditions emerge in the 300th–400th frame time interval. In this period, the available bandwidth is low, but also the tolerance on the detection performance drop (D_t^{mAP} and D_t^{sen}) increases, as, again, highlighted by the brown dotted line in Fig. 11 (left) and Fig. 11 (middle-left). Thus, the CoTeD `Manager` selects a configuration with higher pruning threshold (p_t=0.22) and lower normalized compression quality ($q_{\mathbf{a}_t}$=0.4), which result in a higher compression ratio. This, however, still guarantees an acceptable reconstructed tensor SNR. It follows that the used bandwidth decreases significantly (Fig. 11 (middle-right)) and the detection performance drop

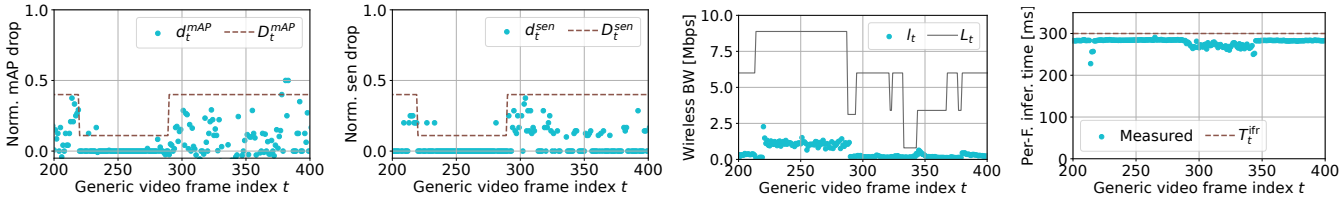

Fig. 12: MTL scenario: CoTeD `Manager` orchestration performance. (left) Normalized DNN mAP drop, (middle-left) Normalized DNN sensitivity drop, (middle-right) Used wireless bandwidth, and (right) Per-frame inference time.

increases, although remaining below the maximum tolerable value (see Fig. 11 (left) and Fig. 11 (middle-left)). Notably, even if the available bandwidth L_t varies substantially during the experiment, the actual used bandwidth always stays under the bandwidth limit L_t and the per-frame inference time never exceeds the inference deadline T_t^{ifr} (see Fig. 11 (middle-right) and Fig. 11 (right)).

Importantly, only 7 frames do not meet the object detection mAP drop tolerance D_t^{mAP} or the sensitivity drop tolerance D_t^{sen}, which is mainly due to the DNN approximations we made in the model of the detection performance drop. To summarize, when computing the inference success rate in the LTL scenario, CoTeD with split DNN is remarkably high and never drops below 96.5%.

MTL scenario. Fig. 10 (middle-left) shows the temporal evolution of the framework configuration for the MTL scenario, in which the inference deadline is less stringent (T_t^{ifr}=300 ms). In this case all three compression techniques can provide a feasible solution to the optimization problem $\widehat{\mathbf{O}}(\mathbf{a}_t)$. The CoTeD `Manager` dynamically selects the compression technique that maximizes the tensor compression ratio while fulfilling the system and application requirements.

Initially (in the 220th–280th frame interval), the operational conditions are such that the available bandwidth is high and the target detection performance drop (D_t^{mAP} and D_t^{sen}) is low (see Fig. 12 (left) and Fig. 12 (middle-left)). In this time period, the best configuration of the JPEG compression achieves the highest compression ratio and hence, it is selected by CoTeD.

Conversely, when the available bandwidth is low (in the 300th–340th frame interval) and the tolerance on detection performance drop is high (see Fig. 12 (left) and Fig. 12 (middle-left)), CoTeD chooses the optimal configuration of the Regression compression, as it yields the highest compression ratio. We remark that, in this scenario and under the considered operational conditions, Decomposition is never adopted, since it is always outperformed by either JPEG or Regression.

Similarly to the LTL scenario, very few video frames – a total of 5 frames – do not meet the target mAP drop or sensitivity drop D_t^{mAP}. In summary, CoTeD with the split DNN again achieves excellent performance, with a computed inference success rate of 97.5%.

MLT-HL scenario. The time evolution of the CoTeD configuration for the MTL-HL scenario is depicted in Fig. 10 (middle-right). We recall that in this case the UAV's GPU experiences high load and the target inference deadline is T_t^{ifr}=300 ms. Due to the high GPU load, the compression time of JPEG (which is significantly more complex than the other compression techniques) increases substantially. As a consequence, only Decomposition and Regression are feasible choices. Similar to the MTL scenario, the CoTeD `Manager` selects the compression technique that can maximize the tensor compression ratio. It thus adopts the Decomposition technique when the available bandwidth is high (e.g., in the 220th–280th frame interval), and the Regression technique when the available bandwidth is low (e.g., in the 280th–400th frame interval). From Figures 13 (middle-right)–13 (right), we can observe that the used bandwidth increases compared to the MTL scenario, which is mainly due to the limited compression ratio that Decomposition can achieve. However, during the whole experiment period, the used bandwidth and per-frame inference time always stay under the limit values. Finally, also in this case only few (namely, 8) video frames do not meet the target mAP or sensitivity drop, and CoTeD with split DNN achieves an inference success rate as high as 96%.

HLT-LM scenario. Fig. 10 (right) illustrates the temporal evolution of the framework configuration in the HLT-LM scenario, where the large YOLOv3 model is adopted for object detection and the DNN inference deadline is set to T_t^{ifr}=500 ms. As described in Sec. II, the activation tensor produced by the YOLOv3's Head model is substantially larger (5,540 KB) than that of YOLOv3-tiny (350 KB), while the time available for compression remains limited. Consequently, as shown in Fig. 10 (right), despite the larger inference deadline, only JPEG compression can satisfy the requirements (similarly to what we observed in the LTL scenario). Interestingly, in the HLT-LM case, the CoTeD `Manager` selects a lower compression quality $q_{\mathbf{a}_t}$ than in the LTL scenario, but a comparable prune threshold q_t (see also Fig. 10 (left)). This allows for a higher compression ratio, hence, a faster transmission, while preserving essential tensor information – which confirms the CoTeD `Manager`'s flexibility in handling tensors of different sizes. Figures 14 (middle-right)–14 (right) highlight that bandwidth usage grows with respect to the LTL case, mainly due to the larger YOLOv3 activation tensors. Nevertheless, throughout the experiment, the per-frame inference time consistently remains within the deadline, with only 8 frames violating the bandwidth constraint under low-bandwidth availability. Finally, only 3 video frames fail to meet the target mAP or sensitivity requirements, and overall, CoTeD achieves a success rate as high as 94.5%.

In summary, Fig. 15 presents the breakdown of the average per-frame inference latency and the corresponding average wireless bandwidth usage for the above scenarios. For comparison, we also include the case where YOLOv3-tiny (for LTL, MTL, and MTL-HL) and YOLOv3 (for HTL-LM) are split, and the activation tensor produced by the DNN head

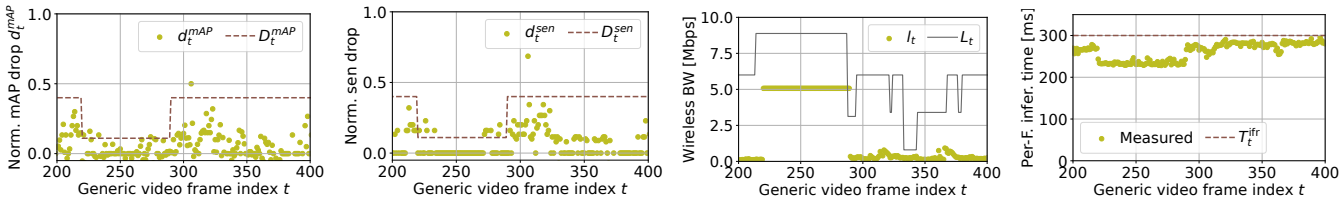

Fig. 13: MTL-HL scenario: CoTeD `Manager` orchestration performance. (left) Normalized DNN mAP drop, (middle-left) Normalized DNN sensitivity drop, (middle-right) Used wireless bandwidth, and (right) Per-frame inference time.

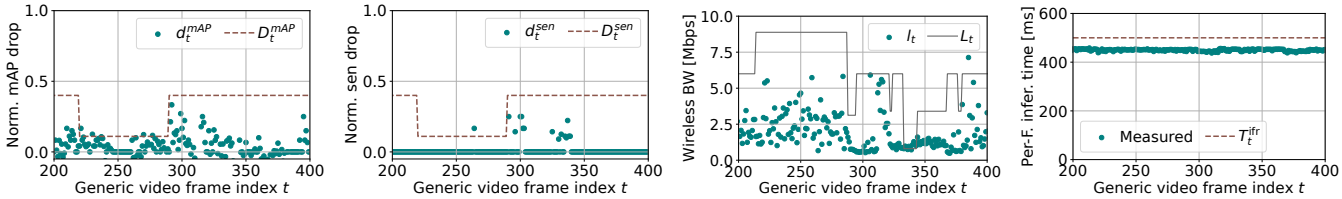

Fig. 14: HTL-LM scenario: CoTeD `Manager` orchestration performance: (left) Normalized DNN mAP drop, (middle-left) Normalized DNN sensitivity drop, (middle-right) Used wireless bandwidth, and (right) Per-frame inference time.

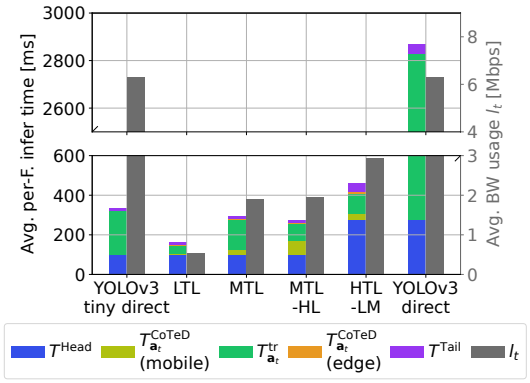

Fig. 15: Breakdown of average per-frame inference latency and average wireless bandwidth usage for CoTeD `Manager` orchestration and direct DNN splitting in target scenarios.

is directly transmitted over the radio link. We observe that, thanks to CoTeD's dynamic orchestration, the transmission latency is reduced by 82%, 34%, and 62% for the LTL, MTL, and MTL-HL scenarios compared to "YOLOv3-tiny direct" transmission, and by 96% for the HTL-LM scenario compared to "YOLOv3 direct" transmission. Meanwhile, CoTeD processing accounts for only about 10% of the average per-frame inference time (except in the special MTL-HL case, where it contributes 30%). In addition, the average wireless bandwidth usage is reduced by 92%, 70%, and 70% for the LTL, MTL, and MTL-HL scenarios compared to "YOLOv3-tiny" direct, and by 54% for the HTL-LM case compared to "YOLOv3 direct". These results demonstrate the significant advantages and flexibility provided by CoTeD.

C. CoTeD vs. baseline split DNN

We now compare CoTeD against baseline split DNN approaches, namely direct splitting (no compression), direct splitting with DNN model quantization [23], activation tensor quantization [24], and fixed-level JPEG compression [9]. For brevity, we focus on the YOLOv3-tiny model and the St. Marc dataset's test samples (in total 400 continuous video frames); the analysis with the YOLOv3 full model on the BEV and the ImangeNetVidVRD datasets indeed yielded similar results.

In the direct splitting approach, the activation output from the DNN Head model is serialized (e.g., via Pickle https://docs.python.org/3/library/pickle.html) and transmitted to the edge server, causing large transmission delays and high bandwidth usage. Direct splitting with DNN model quantization reduces both model size and activation output, but detection quality is often degraded under aggressive low-bit quantization. For a fair comparison, we quantize YOLOv3-tiny from `float32` to `float16`, which is denoted with F16MQ. Another alternative is activation tensor quantization, where we apply 8-bit quantization (`float32` to `int8`), the minimum bit-width natively supported by most ML libraries, and denote it with INT8TQ. As for JPEG compression, prior work [9] often uses it at fixed quality levels to compress the DNN Head output before transmission. Following this approach, we adopt JPEG at quality 100 and 50, denoted with JPEG-100 and JPEG-50, respectively. Table V summarizes the UAV-side memory (excluding libraries), processing, and transmission footprints of all these methods. Notice that, since CoTeD dynamically selects compression methods at runtime, in this case we provide the range of per-frame processing time and transmission size, instead of averages with standard deviations.

We observe that F16MQ achieves the lowest memory usage (44.9 MB) and the lowest average per-frame processing time (90 ms), while JPEG-50 achieves the lowest average transmission size (9 MB). Direct splitting, instead, yields the worst results. Importantly, CoTeD introduces only a 1% memory overhead on the UAV, it but reduces transmission data by more than 90% compared to direct splitting.

Then, to evaluate the performance of the above schemes in serving inference requests, we set the inference deadline to T_t^{ifr}=150 ms and vary both the available bandwidth between the UAV and edge server (L_t) and the application requirements (D_t^{mAP}, D_t^{sen}), as in Sec. V-B. We run YOLOv3-tiny with CoTeD and all baselines over 400 continuous video frames from three datasets (BEV, St. Marc, and ImageNet-VidVRD), and present the inference request success rate and per-frame

TABLE V: Memory, processing and data transmitting footprints on UAV for CoTeD and the baseline splitting approaches

Split DNN approach	Memory usage [MB]	Per-frame proc. time [ms]	Per-frame trans. data size [kB]
Direct split	89.8	110 ± 10	347 ± 0
F16MQ	44.9	90 ± 12	174 ± 0
INT8TQ	89.6	127 ± 8	87 ± 0
JPEG-100	89.6	97 ± 5	55 ± 10
JPEG-50	89.6	97 ± 4	9 ± 5
CoTeD	90.5	96–240	2–35

TABLE VI: Inference success rate and average per-frame inference time of CoTeD and baseline splitting approaches

Split DNN approach	Inference request success rate			Per-frame inference time [ms]
	BEV	St. Marc	VidVRD	
Direct split	10.5%	33%	24.5%	174 ± 59
F16MQ	76.5%	76.5%	76.5%	147 ± 29
INT8TQ	70.5%	74%	53.5%	148 ± 15
JPEG-100	76%	70%	74.5%	154 ± 73
JPEG-50	79%	38.5%	60%	118 ± 12
CoTeD	91%	96.5%	91.5%	149 ± 5

latency in Table VI. Direct splitting achieves very low success rates (10–30%), mainly due to excessive transmission delay that frequently violates T_t^{ifr}. F16MQ, INT8TQ, and JPEG-100 reduce the size of the transmitted data, but still suffer from deadline violations under low-bandwidth conditions, resulting in success rates of 50–75%. JPEG-50, thanks to its higher compression ratio, keeps inference time consistently low (118 ms on average), always meeting T_t^{ifr}. However, the severe quality loss frequently violates application requirements (D_t^{mAP}, D_t^{sen}), yielding inference request success rates of just 40–70%. Conversely, CoTeD consistently achieves success rates above 90% across all datasets, with average per-frame inference time closest to the target deadline, underscoring the superiority of CoTeD over the considered baseline approaches.

VI. Related Work

Edge-assisted split DNN inference is a crucial technology that enables the deployment of complex DNNs on resource-constrained devices such as UAVs [25]. Due to their inherent mobility and limited computational power, mobile nodes often struggle to directly execute highly accurate yet computationally demanding models, making real-world deployment impractical [26]. While higher-resolution images can improve detection quality [27], they also require greater bandwidth [28], making real-time processing infeasible in environments where responsiveness is critical. To address these challenges, splitting the computational workload between the edge server and the mobile device emerges as a promising solution, offering the potential to simultaneously mitigate both computational and bandwidth constraints [29].

A survey of the main split DNN applications is presented in [30], which also highlights as major challenges the use of early exit branches in DNN models and the need to reduce bandwidth consumption due to tensor transfer between DNN Head and Tail. For tensor compression, a common approach is to incorporate "bottleneck" layers at the end of the DNN head [7], [8]. This method is highly effective in reducing tensor size while maintaining a favorable accuracy ratio.

TABLE VII: Comparison of CoTeD vs. state-of-the-art split DNN-based schemes for activation tensor transmission

Split DNN solutions	Costs		Benefits	
	DNN model modification	On UAV tensor process	Dynamic compression	Quality guarantee
CoTeD	✗	✓	✓	✓
"Bottleneck" based e.g. [7], [8]	✓	✗	✗	✓
Static compression based e.g. [9], [32]	✗	✓	✗	✓
Model quantization based e.g. [36]	✓	✗	✗	✓
Tensor quantization based e.g. [10], [24]	✗	✓	✗	✓
Split point selection based e.g. [33], [34]	✗	✗	✗ or limited	✓

However, it introduces the additional challenge of Knowledge Distillation [31], requiring mandatory retraining of a new auto-encoder based model for each candidate split layer. An alternative approach leverages picture/video encoding-based algorithms, e.g., JPEG [9] or HEVC [32], to encode the DNN Head output tensor directly, which, however, requires a careful balance between encoding quality and bandwidth consumption. Other solutions like [10], [24] propose to quantize and compress the features that are output by the split DNN Head, which requires analyzing the value distribution of each activation tensor and, thus, may lead to high processing delay. NASC [33] tackles this challenge by jointly optimizing the model structure and split point selection, ensuring high accuracy while meeting latency constraints. It employs a one-shot NAS approach, eliminating the need for repeated model training, thus making the search process computationally efficient. Similarly, DNNSplit [34] pre-identifies a small set of candidate split points corresponding to activation tensors of different sizes, and then it dynamically selects at runtime the one that minimizes latency and cost.

Finally, we mention that an preliminary version of this work appeared in our conference paper [35], which sketched the split DNN approach using only Decomposition and a simple CoTeD `Manager` that could only adjust the pruning threshold.

Novelty. As highlighted in Table VII, unlike existing work, our CoTeD aims to perform efficient tensor transmission for split DNNs *without requiring modifications* to the pre-trained model architecture. Also, differently from the DNN split point-selection approaches, which address the issue of a time-varying radio bandwidth by changing the DNN split point, CoTeD is designed to optimize tensor transmission and, hence, meet the application requirements with very high probability, at any split layer. CoTeD does so by selecting at the UAV the relevant tensor information and applying adaptive compression to it, while at the edge server, it effectively reconstructs the original tensor ensuring high-quality inference. This approach improves the efficiency of split DNN deployment in practical UAV scenarios by maximizing communication and computational performance without incurring the additional training overhead associated with bottleneck architectures.

VII. Conclusions

We addressed the execution of DNN-based computer vision tasks in scenarios where resource-limited UAVs can leverage

the help of edge servers. We exploited a split computing approach to run the low-complexity Head DNN at the UAV and the computationally heavier DNN Tail at the edge of the network infrastructure, and tackled the problem of efficiently transmitting over the radio link the tensor from the Head to the Tail model. Our solution, called CoTeD, consists of a sequence of computational-light and dynamically configurable differential, pruning, and compression operators at the UAV, and a similar chain of operations for tensor reconstruction at the network edge. CoTeD is managed by a real-time application- and system-aware optimization algorithm that targets at fulfilling dynamic system and application requirements including the available radio bandwidth limit, DNN object detection quality and inference latency. CoTeD reduces data transmission over the radio link by up to 90% with negligible loss in object detection quality and reduces inference latency up to 70% compared to local DNN deployment on the UAV. CoTeD guarantees an inference request success rate of at least 90%, which corresponds to a 20%-80% improvement when compared to direct DNN splitting, static JPEG compressions, and DNN model quantization. An interesting direction for future work is the integration of DNN Head Distillation techniques to develop frameworks that aim at a DNN-specific optimization of the compression of the Head output tensor.

REFERENCES

[1] X. Wu, W. Li, D. Hong, R. Tao, and Q. Du, "Deep learning for unmanned aerial vehicle-based object detection and tracking: A survey," *IEEE Geoscience and Remote Sensing Magazine*, 2022.

[2] G. Tang, J. Ni, Y. Zhao, Y. Gu, and W. Cao, "A survey of object detection for uavs based on deep learning," *Remote Sensing*, 2023.

[3] Y. Matsubara, M. Levorato, and F. Restuccia, "Split computing and early exiting for deep learning applications: Survey and research challenges," *ACM Computing Surveys*, 2022.

[4] F. Malandrino, C. F. Chiasserini, and G. di Giacomo, "Efficient distributed dnns in the mobile-edge-cloud continuum," *IEEE/ACM Transactions on Networking*, 2022.

[5] L. Yang, Z. Zheng, J. Wang, S. Song, G. Huang, and F. Li, "Adadet: An adaptive object detection system based on early-exit neural networks," *IEEE Transactions on Cognitive and Developmental Systems*, 2023.

[6] V. A. Kelkar, S. Bhadra, and M. A. Anastasio, "Compressible latent-space invertible networks for generative model-constrained image reconstruction," *IEEE Transactions on Computational Imaging*, 2021.

[7] Y. Matsubara, D. Callegaro, S. Baidya, M. Levorato, and S. Singh, "Head network distillation: Splitting distilled deep neural networks for resource-constrained edge computing systems," *IEEE Access*, 2020.

[8] P. Datta, N. Ahuja, V. S. Somayazulu, and O. Tickoo, "A low-complexity approach to rate-distortion optimized variable bit-rate compression for split dnn computing," in *26th International Conference on Pattern Recognition (ICPR)*, 2022.

[9] J. Emmons, S. Fouladi, G. Ananthanarayanan, S. Venkataraman, S. Savarese, and K. Winstein, "Cracking open the dnn black-box: Video analytics with dnns across the camera-cloud boundary," in *Workshop on Hot Topics in Video Analytics and Intelligent Edges*, 2019.

[10] R. A. Cohen, H. Choi, and I. V. Bajić, "Lightweight compression of intermediate neural network features for collaborative intelligence," *IEEE Open Journal of Circuits and Systems*, 2021.

[11] P. Zhang, H. Tian, H. Luo, X. Li, and G. Nie, "A hybrid fast inference approach with distributed neural networks for edge computing enabled UAV swarm," *Physical Communication*, 2023.

[12] M. Bakirci, "A novel swarm unmanned aerial vehicle system: Incorporating autonomous flight, real-time object detection, and coordinated intelligence for enhanced performance." *Traitement du Signal*, 2023.

[13] K. Arunruangsirilert and J. Katto, "Uplinknet: Practical commercial 5G standalone (sa) uplink throughput prediction," in *IEEE International Conference on Visual Communications and Image Processing*, 2024.

[14] M. Abdi, K. F. Haque, F. Meneghello, J. Ashdown, and F. Restuccia, "Phydnns: Bringing deep neural networks to the physical layer," in *IEEE INFOCOM 2025 - IEEE Conference on Computer Communications*, 2025, pp. 1–10.

[15] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[16] S. Ding, F. Long, H. Fan, L. Liu, and Y. Wang, "A novel yolov3-tiny network for unmanned airship obstacle detection," in *IEEE 8th Data Driven Control and Learning Systems Conference*, 2019.

[17] A. Alp and S. Agrawal, "Cable suspended robots: design, planning and control," in *IEEE International Conference on Robotics and Automation*, 2002.

[18] F. Malandrino, G. d. Giacomo, A. Karamzade, M. Levorato, and C. F. Chiasserini, "Tuning DNN model compression to resource and data availability in cooperative training," *IEEE/ACM Transactions on Networking*, 2023.

[19] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM Review*, 2009.

[20] X.-Y. Liu, Z. Zhang, Z. Wang, H. Lu, X. Wang, and A. Walid, "High-performance tensor learning primitives using gpu tensor cores," *IEEE Transactions on Computers*, 2022.

[21] J. Blank and K. Deb, "pymoo: Multi-objective optimization in python," *IEEE Access*, 2020.

[22] ETSI, "5G; unmanned aerial system (uas) support in 3GPP (3GPP TS 22.125 version 17.6.0 release 17)," https://www.etsi.org/deliver/etsi_ts/122100_122199/122125/17.06.00_60/ts_122125v170600p.pdf, 2022.

[23] H. Ahn, T. Chen, N. Alnaasan, A. Shafi, M. Abduljabbar, H. Subramoni, and D. K. Panda, "Performance characterization of using quantization for dnn inference on edge devices," in *IEEE 7th International Conference on Fog and Edge Computing*, 2023.

[24] D. Carra and G. Neglia, "Dnn split computing: Quantization and run-length coding are enough," in *IEEE Global Communications Conference*, 2023.

[25] C. Deng, X. Fang, and X. Wang, "Integrated sensing, communication, and computation with adaptive dnn splitting in multi-uav networks," *IEEE Transactions on Wireless Communications*, 2024.

[26] M. Mendula, P. Bellavista, M. Levorato, and S. L. de Guevara Contreras, "Furcifer: a context adaptive middleware for real-world object detection exploiting local, edge, and split computing in the cloud continuum," in *IEEE International Conference on Pervasive Computing and Communications*, 2024.

[27] C. Borel-Donohue and S. S. Young, "Image quality and super resolution effects on object recognition using deep neural networks," in *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*, 2019.

[28] L. Liu, H. Li, and M. Gruteser, "Edge assisted real-time object detection for mobile augmented reality," in *the 25th Annual International Conference on Mobile Computing and Networking*, 2019.

[29] Y. Matsubara, M. Mendula, and M. Levorato, "A multi-task supervised compression model for split computing," in *IEEE/CVF Winter Conference on Applications of Computer Vision*, 2025.

[30] Y. Matsubara and M. Levorato, "Split computing for complex object detectors: Challenges and preliminary results," in *the 4th International Workshop on Embedded and Mobile Deep Learning*, 2020.

[31] Y. Matsubara, "torchdistill: A modular, configuration-driven framework for knowledge distillation," in *International Workshop on Reproducible Research in Pattern Recognition*, 2021.

[32] H. Choi and I. V. Bajić, "Deep feature compression for collaborative object detection," in *25th IEEE International Conference on Image Processing*, 2018.

[33] S. Shimizu, T. Nishioy, S. Saito, Y. Hirose, C. Yen-Hsiu, and S. Shirakawa, "Neural architecture search for improving latency-accuracy trade-off in split computing," in *IEEE Global Communications Conference Workshops*, 2022.

[34] J. Lee, H. Lee, and W. Choi, "Wireless channel adaptive dnn split inference for resource-constrained edge devices," *IEEE Communications Letters*, 2023.

[35] Y. Yu, M. Levorato, and C. F. Chiasserini, "Tensor compression and reconstruction in split dnn for real-time object detection at the edge," in *IEEE International Mediterranean Conference on Communications and Networking*, 2024.

[36] G. Li, L. Liu, X. Wang, X. Dong, P. Zhao, and X. Feng, "Auto-tuning neural network quantization framework for collaborative inference between the cloud and edge," in *International Conference on Artificial Neural Networks*, 2018.