Selective Cloud Offloading for Accurate and Efficient Object Detection

***Abstract*—High-accuracy object detection on resource-constrained devices is becoming increasingly important for applications in autonomous systems, smart surveillance, and mobile computing. However, deploying high-performance object detection models on these devices is impractical due to computational limitations, and transmitting and processing all data on a much more powerful remote server running significantly more complex and accurate models, known as full cloud offloading, incurs high latency and is constrained by network bandwidth. In this paper, we propose a selective cloud offloading framework that provides users with control over the trade-off between prediction accuracy and processing cost. Our approach employs a lightweight object detection model on the edge to make initial predictions, leveraging conformal prediction to quantify uncertainty. Only high-uncertainty regions are offloaded to the cloud, where more powerful models refine predictions, improving overall detection accuracy. To further optimize efficiency, multiple uncertain regions are combined into a single image before offloading, reducing transmission and processing costs. We present the architecture of our system and evaluate its performance on real datasets, demonstrating that it achieves cloud-level accuracy while significantly reducing offloading overhead.**

***Index Terms*—object detection, conformal prediction, cloud offloading**

I. INTRODUCTION

Object detection [1] is integral to a wide range of modern applications, from assisting farmers with identifying plant diseases, pests, or fruit ripeness using low-performance devices in the field [2], to enabling visually impaired users to recognize objects or text through affordable smartphones and wearable technology [3]–[5]. Despite its importance, achieving high-accuracy, real-time object detection on resource-constrained edge devices, such as drones and embedded systems, remains a significant challenge due to the intensive computational demands of deep neural networks (DNNs).

To overcome these limitations, cloud offloading has emerged as a promising strategy [6], wherein demanding tasks are transferred from edge devices to powerful cloud servers. For object detection, this typically means sending images or image segments to cloud data centers for analysis by more accurate, heavyweight models. However, this approach presents its own significant drawbacks. Transmitting high-volume image data can overwhelm limited network bandwidth, while the financial burden can be substantial, particularly as prominent cloud services (e.g., Amazon Rekognition, Google Cloud Vision AI) often bill per image processed. This billing model makes naive offloading of multiple image parts especially costly. Consequently, there is a critical need for an intelligent offloading strategy that judiciously balances detection accuracy with these operational costs.

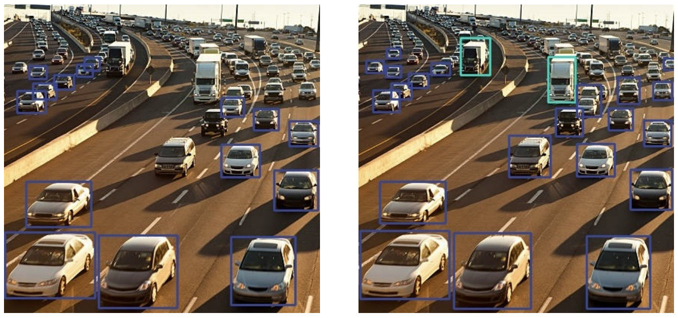

(a) Edge prediction (YOLOv5-Tiny) (b) Cloud prediction (Mask-RCNN)

Fig. 1: Comparison between edge and cloud predictions. The model at the edge does not perform as well as the model on the cloud.

Several approaches have targeted video stream optimization, for instance, by compressing frames [7] or using local tracking with intermittent cloud offloading, sometimes guided by server context [8], [9]. These video-centric techniques, however, are not directly applicable to static image processing. Their reliance on spatial and temporal relationships between frames—used for effective data compression or object tracking—is incompatible with static images, which are processed independently. For static images, some systems like EdgeDuet [10] employ heuristics, such as object size, to make offloading decisions. Unfortunately, these heuristic-based methods can lead to suboptimal outcomes because such features do not consistently correlate with true detection difficulty.

A fundamental challenge with existing offloading approaches is the trade-off between cost and system accuracy. Sending more data to the cloud improves accuracy but incurs higher bandwidth and computational expenses. Current solutions do not offer a mechanism for users to control this trade-off effectively. This underscores the need for an adaptive, efficient, and general-purpose offloading strategy that dynamically balances accuracy and cost while allowing users to configure their desired trade-off based on application requirements.

First, we establish an intelligent mechanism for deciding *what* to offload by accurately quantifying the uncertainty of object detections performed at the edge. Our system initially employs a lightweight object detection model on the resource-constrained device. Crucially, it then leverages **conformal pre-**

diction [11] to generate a mathematically rigorous uncertainty score for each detected object. Conformal prediction is a technique that provides statistically rigorous and distribution-free measures of confidence for individual predictions, allowing us to reliably assess the likelihood of a detection being correct. By introducing a tunable threshold on this uncertainty measure (the conformal score), our framework offers users explicit control over the trade-off between processing cost and detection accuracy: a lower threshold results in more offloading and higher accuracy, while a higher threshold prioritizes cost savings.

Second, having identified which detections are uncertain, we optimize *how* these detections are offloaded to the cloud. A naive approach would be to transmit each individual image region (i.e., the bounding box around a detected object) that exceeds the uncertainty threshold. However, this strategy is highly inefficient. As previously noted, cloud-based object detection services typically charge per image processed, meaning sending numerous small regions separately incurs prohibitive recognition costs. Furthermore, this leads to significant computational overhead due to repeated model initializations on the cloud server. To solve this, our second key contribution is a **packing-based optimization strategy**. This technique intelligently groups multiple uncertain image regions from the same original image into a single, composite image before offloading. By consolidating these regions and sending one packed image instead of many small ones, our method drastically reduces the number of separate requests to the cloud. This approach minimizes not only data transmission but, more importantly, the monetary and computational recognition costs, thereby creating a far more efficient and scalable offloading solution.

To summarize, we make the following contributions:

1) We propose a selective cloud offloading strategy that balances cost and accuracy by leveraging conformal prediction to assess uncertainty in object detection.
2) We introduce a packing-based optimization technique to further reduce cloud transmission costs while maintaining detection precision.
3) We provide a user-configurable trade-off mechanism, allowing applications to adjust accuracy and cost constraints based on specific requirements.
4) We conduct extensive experiments demonstrating that our method achieves cloud-comparable accuracy while offloading significantly less data compared to existing approaches.

Outline The remainder of this paper is organized as follows. Section II discusses the preliminaries and defines the problem to be solved by this paper. Section III presents the architecture and components of our selective offloading solution. Section IV further optimizes our solution by proposing algorithms for image stitching. Section V presents the results from extensive experimental evaluation. Section VI discusses related work, and Section VII concludes this paper with potential future research directions.

II. Preliminaries and Problem Definition

We consider the object detection problem on a set of images, X, which we refer to as **Image Set**. Each element $x \in X$ represents one image. We first model the object detection problem in Section II-A, and then present our problem in Section II-C.

A. Object Detection Problem

An **object detection model**, M, takes an input image x and outputs a set of predictions Y_x^M. Each prediction $y_i^M \in Y_x^M$ is a tuple (B_i^M, c_i^M, p_i^M), where B_i^M is a bounding box, c_i^M is the predicted class label, and p_i^M is the confidence score (i.e., the predicted probability for class c_i^M). In multi-class object detection, the model assigns a probability to each possible class for every detected object; here, p_i^M corresponds to the maximum probability across all classes, and c_i^M is the associated class. Each bounding box is represented as $B_i^M = (s_i^M, e_i^M, w_i^M, h_i^M)$, where (s_i^M, e_i^M) is the pixel coordinate of the top-left corner of the box, and w_i^M, h_i^M are its width and height, respectively.

In object detection, we use ground-truth results to refer to the set of bounding boxes where objects actually exist as well as their true class labels, denoted as $Y(x)$ for a given image x. For a ground-truth result $y_i \in Y(x)$ and a prediction $y_i^M \in Y^M(x)$, they are considered a **match** if the **Intersection over Union (IoU)** of the corresponding bounding boxes exceeds a predefined threshold IoU_t (typically 0.5) [12]. Let B_i^M be the bounding box from the model prediction y_i^M, and B_i be the ground-truth bounding box from y_i. Specifically, IoU is computed as

$$IoU(B_i, B_i^M) = \frac{\text{Area}(B_i \cap B_i^M)}{\text{Area}(B_i \cup B_i^M)}$$

where $B_i \cap B_i^M$ is the intersection of two bounding boxes, while $B_i \cup B_i^M$ is the union, and Area() computes the area in pixels.

We consider the prediction y_i^M as **correct** if there is at least one bounding box B_i from the ground-truth results such that $IoU(B_i^M, B_i) \geq IoU_t$ and the predicted label matches the ground-truth label. For simplicity, we use $Y_x^M \cap Y_x$ to denote the set of correct predictions from Y_x^M. The default value of IoU_t is 0.5 unless otherwise specified.

To measure the prediction quality (accuracy) for a single image x, we adopt two widely used metrics, **Precision** and **Recall** [13], defined as follows:

$$Prec(Y_x, Y_x^M) = \frac{|Y_x^M \cap Y_x|}{|Y_x^M|}; Rec(Y_x, Y_x^M) = \frac{|Y_x^M \cap Y_x|}{|Y_x|},$$

where $|Y_x|$ is the number of elements in set Y_x. Similarly, for an image set X, we use $\mathbb{Y}_X^M$ to denote the set union of all predictions and use $\mathbb{Y}_X$ to denote the ground-truth for all images, and compute precision and recall in the same manner, denoted as $Prec(\mathbb{Y}_X, \mathbb{Y}_X^M)$ and $Rec(\mathbb{Y}_X, \mathbb{Y}_X^M)$.

In the definition above, we have assumed that the ground-truth results are available. When they are unavailable, however, we can consider the output from the cloud model as a proxy of the ground truth, as that model is usually of very high quality. Apparently, in this case, both the precision and recall of the cloud model are considered to be 1.

B. Conformal Prediction

Conformal prediction [14]–[16] is a statistical framework that provides principled measures of uncertainty for predictions made by machine learning models. It operates under the assumption of exchangeability; that is, the joint distribution of the data remains invariant under permutations. This assumption allows conformal prediction to offer finite-sample, distribution-free guarantees on the reliability of its prediction sets.

In practice, the available labeled dataset is partitioned into two disjoint subsets: a training set and a calibration set. The training set is used to fit the underlying predictive model, while the calibration set, denoted as $\mathcal{C} = (x_1, y_1), \dots, (x_n, y_n)$, where each $x_i \in \mathbb{R}^d$ represents an input feature vector and $y_i \in 0, 1$ is its corresponding binary label, is utilized to assess the uncertainty of the model's predictions. The calibration set must be representative of the general data distribution to ensure the accuracy and reliability of the predictions provided by the conformal prediction model.

A nonconformity measure $\alpha : \mathbb{R}^d \times 0, 1 \to \mathbb{R}$ is defined to quantify how atypical a data point is relative to the calibration set. For each $(x_i, y_i) \in \mathcal{C}$, compute the nonconformity score $\alpha_i = \alpha(x_i, y_i)$.

To predict the label for a new input x_{n+1}, conformal prediction evaluates each possible label $y \in 0, 1$ by computing the nonconformity score $\alpha_{n+1} = \alpha(x_{n+1}, y)$. The p-value for label y is then calculated as:

$$p_y = \frac{1 + \sum_{i=1}^{n} \mathbb{I}\alpha_i \geq \alpha_{n+1}}{n+1},$$

where $\mathbb{I}\cdot$ is the indicator function. The prediction set $\Gamma(x_{n+1})$ at significance level α (with confidence level $1-\alpha$) is defined as:

$$\Gamma(x_{n+1}) = y \in 0, 1 : p_y > \alpha.$$

This set contains all labels that are not significantly nonconforming. The conformal prediction framework guarantees that the true label y_{n+1} will be included in $\Gamma(x_{n+1})$ with probability at least $1 - \alpha$, assuming exchangeability between the calibration and test data. The choice of α balances the trade-off between the size of the prediction set and the confidence in containing the true label.

C. Object Detection with Cloud Offloading

Our problem setting includes two object detection models: the **edge model**, M_e, and the **cloud model**, M_c. We assume that the edge model is capable of detecting common objects with lower accuracy compared to the cloud model. The cloud model usually provides much better accuracy, but at a higher cost in terms of resource usage, network consumption, overall latency, and monetary costs. Thus, it is usually infeasible to send all the object detection requests to the cloud. We wish to select a subset from the edge model's prediction that is likely to contain erroneous predictions and send the corresponding part of the images to the cloud model for further recognition to improve the overall precision. This step is also known as the offloading to the cloud [6], [17]. We next introduce this problem formally.

For an image set X, let $\mathbb{Y}^e_X$ denote the prediction results from the edge model M_e, and $\mathbb{Y}^c_X$ denote the results from the cloud model M_c. Assuming that the cloud model has a higher precision and recall, we aim to select a subset of predictions $\mathbb{Y}'^e_X \subset \mathbb{Y}^e_X$ that are suspected to contain erroneous predictions and send them to the cloud for further detection. We use $\mathbb{Y}'^c_X$ to denote the predictions from the cloud model. Then, our final prediction set would contain two main parts: the predictions made by the edge model solely, denoted as $\mathbb{Y}^e_X \setminus \mathbb{Y}'^e_X$, and the predictions that are refined by the cloud model, i.e., $\mathbb{Y}'^c_X$, which supersedes the predictions $\mathbb{Y}'^e_X$ made by the edge model. For simplicity, we use $\mathcal{Y}_X = \mathbb{Y}^e_X \setminus \mathbb{Y}'^e_X \cup \mathbb{Y}'^c_X$ to denote the final prediction set.

In extreme cases, if no detections are sent to the cloud, then $\mathcal{Y}_X = \mathbb{Y}^e_X$; similarly, if all detections are sent to the cloud for refinements, then $\mathcal{Y}_X = \mathbb{Y}^c_X$.

In terms of computational cost, the edge model will always be applied to each image from the image set, thus, the inference time for the edge model is constant. As more detections from the edge model are sent to the cloud, the additional cost for the cloud inference increases. Let $C(M_e, \mathbb{Y}'^c_X)$ denote the additional cost, which includes the network transmission cost and more importantly, the cost of invoking the cloud model. Clearly, as $|\mathbb{Y}'^c_X|$ increases, $C(M_e, \mathbb{Y}'^c_X)$ also increases.

In practice, we aim to provide a parameter $\alpha \in [0, 1]$ to achieve the overall accuracy-cost tradeoff.

III. Our Solution: Selective Offloading

Our system is designed to offer a principled approach to managing the trade-off between object detection performance and the cost associated with cloud offloading. By intelligently determining which parts of an image to offload to the cloud for further processing, we aim to minimize data transmission and the cost of cloud model invocation while maximizing detection accuracy.

As illustrated in Figure 2, the object detection pipeline is divided into two primary components: object detection on the edge and selective offloading.

A. Object Detection on the Edge

The first stage of our system is to perform object detection entirely on the edge. This involves two steps: object localization and classification.

Object localization focuses on generating bounding boxes that may contain objects. A key concern on the edge is low recall, where some objects may be missed due to the limited capacity of the edge model. To mitigate this, we lower the confidence threshold C_{thresh} of the edge model, which determines the minimum certainty the model must have before outputting a detection. This increases recall by allowing the

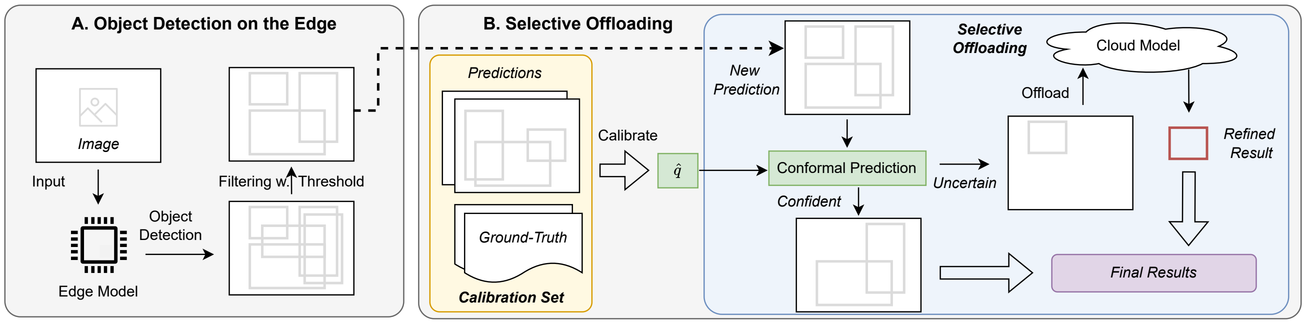


Fig. 2: System Overview: A high-level representation of our selective cloud offloading framework.

model to propose more bounding boxes, albeit at the cost of more false positives.

Each bounding box b_i is scored using a confidence function $C(b_i)$ that estimates the likelihood of the box containing an object of any class. For YOLO-style models, it is typically computed as

$$Conf(b_i) = P_{\text{obj}}(b_i) \cdot \max_{c \in \mathcal{C}} P(c \mid b_i),$$

where $P_{\text{obj}}(b_i)$ is the predicted probability that an object exists in b_i (the objectness score), and $P(c \mid b_i)$ is the predicted probability that the object belongs to class c given that an object exists in b_i.

We retain all bounding boxes whose confidence scores exceed the threshold:

$$B_{\text{out}} = \{b_i \in B \mid C(b_i) \geq C_{\text{thresh}}\}. \quad (1)$$

This filtering step is controlled by the hyperparameter C_{thresh}. A lower value yields higher recall but may include noisy detections. However, many of them can be filtered out in the subsequent classification stage, which leverages uncertainty estimation.

Once candidate object regions have been localized, classification can be applied to each bounding box to identify the object class. Note that the above two steps can be either performed separately [18] or jointly [19] depending on different object detection models. However, due to the lightweight models on the edge, instead of relying solely on the edge model's predictions, we introduce a selective offloading mechanism next.

B. Selective Offloading

With the edge prediction results, our selective offloading mechanism uses *uncertainty estimation* to determine whether a region should be sent to the cloud for refined object detection, which comprises two components: (i) uncertainty quantification using conformal prediction and (ii) a decision policy for offloading based on predicted ambiguity.

1) Uncertainty Quantification Using Conformal Prediction: The edge model produces a confidence score $Conf(b_i)$ for each class label per bounding box. It is based on the model's internal prediction (e.g., objectness multiplied by class probability) and lacks statistical calibration and can be over or under confident, especially under domain shift or limited model capacity. For this reason, we compute a statistically principled uncertainty score using conformal prediction, which assesses how atypical the model's prediction is based on a reference calibration set.

Specifically, we define a *nonconformity score* $\alpha_{b,c}$ for each bounding box b and predicted label (class) $c \in \mathcal{C}$ as:

$$\alpha_{b,c} = 1 - \text{Conf}(b, c),$$

where $\text{Conf}(b, c) = \text{Conf}(b, c) = P_{\text{obj}}(b) \cdot P(c \mid b)$ is the confidence score assigned by the model to class c for bounding box b.

To calibrate uncertainty, we use a set of labeled detections with known outcomes. For each such detection (b, y), where y is the ground-truth class, we compute:

$$\alpha_{b,y} = 1 - \text{Conf}(b, y).$$

We then determine a threshold $\hat{q}$ as the α-quantile of these calibration nonconformity scores. This defines a cutoff below which predictions are considered reliable with a statistical guarantee of coverage $1 - \alpha$.

For a new input box b, the model includes class $c \in \mathcal{C}$ in the *prediction set* if $\alpha_{b,c} \leq \hat{q}$. If this prediction set contains more than one class, the edge model is deemed uncertain. In that case, the corresponding region is flagged for offloading to the cloud.

By adjusting the significance (non-coverage) level α, users can control the offloading policy: lower values of α lead to stricter uncertainty thresholds (and more offloading), while higher values relax the threshold, reducing cloud usage.

2) Offloading Decision Policy: Based on the conformal prediction output, the system applies the following policy: If the prediction set for a box contains only one class, the detection is accepted as-is on the edge; otherwise, if the prediction set contains two or more classes, the region is considered ambiguous and offloaded to the cloud for high-accuracy reclassification.

This simple yet principled rule ensures that the cloud is used only for resolving genuinely uncertain cases, thereby minimizing cost while preserving accuracy.

3) Advantages of Conformal Prediction for Offloading: Conformal prediction offers several key advantages that make it well-suited for guiding offloading decisions in edge-cloud object detection. First, it enables *adaptive confidence estimation* by leveraging a calibration set to assess how unusual a prediction is, rather than relying on fixed softmax thresholds or uncalibrated confidence scores. This dynamic behavior ensures that the system remains responsive to variations in data and model performance.

Second, conformal prediction provides *statistical guarantees* about prediction reliability. Unlike heuristic thresholds, which offer no formal assurance of correctness, conformal scores quantify uncertainty in a principled and distribution-free manner. This ensures that only those detections likely to be incorrect are offloaded to the cloud.

Finally, conformal prediction supports *user-controlled trade-offs* between accuracy and resource usage. The offloading threshold α serves as a tunable parameter that allows users to specify their tolerance for uncertainty. Lowering this threshold favors accuracy by offloading more predictions to the cloud, while increasing it prioritizes cost savings by keeping more processing local to the edge device.

IV. Optimization: Image Stitching

A. Motivation

Offloading individual bounding boxes to the cloud can be expensive and inefficient. Cloud-based object detection services, such as Google Vision API, Amazon Rekognition, and Microsoft Azure AI Vision, charge per image rather than by image size. This means that sending multiple small images incurs significantly higher costs compared to processing a single, larger image. After initial detection by the local model, multiple uncertain bounding boxes from the same image may be flagged for offloading. If each bounding box is transmitted as a separate image, the cloud costs increase significantly, even though fewer total pixels are offloaded.

Even for organizations running their own object detection models on cloud servers, processing multiple smaller images instead of a single combined image results in much more computational overhead due to high initialization costs. Our experiments demonstrate that processing multiple small images takes significantly longer than running inference on a single, larger image.

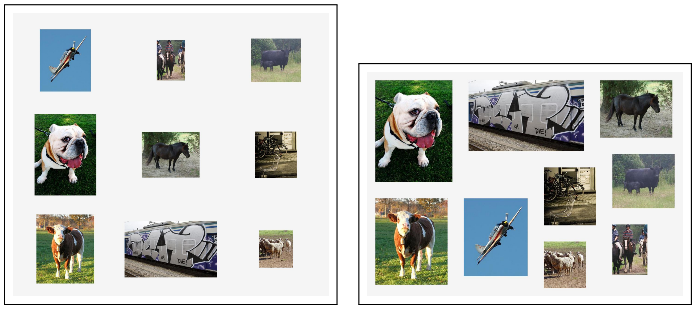
To address these issues, we introduce a Packing-Based Image Stitching (PBIS) method that combines multiple flagged regions from the same or different images into a single composite image before offloading, reducing both cloud costs and computational overhead.

B. Approach

A straightforward approach is to arrange the bounding boxes in a uniform grid. Although this simplifies the packing process, it introduces a significant drawback: the images must be resized to fit the grid structure. This resizing often results in a loss of resolution, which negatively impacts object detection performance. Our preliminary experiments show that reducing image resolution adversely impacts object detection performance, as models struggle to accurately identify and localize small objects when their features become indistinct due to downscaling.

To address this issue, our approach avoids the constraints of a rigid grid layout. Instead, we aim to preserve the native resolution of bounding boxes while efficiently utilizing space within each packed image. We sequentially process the input images and extract bounding boxes one by one. Bounding boxes are collected from each image in order until a predefined maximum number of boxes per packed image, denoted as k, is reached. The parameter k is configurable, allowing flexibility based on system requirements and resource constraints.

By maximizing the use of image real estate and preserving original resolution, we can offload fewer images while retaining object detection precision, particularly for small objects that are sensitive to resolution changes.



(a) Grid-based packing: uniform layout causes resolution loss.

(b) 2D rectangular packing: preserves detail for better detection.

Fig. 3: Comparison of packing strategies before offloading: Both images show how uncertain regions are arranged prior to being sent to the cloud. The grid-based approach resizes bounding boxes to fit into a uniform layout, causing resolution loss. In contrast, 2D rectangular packing preserves detail by optimizing space without resizing, improving the likelihood of accurate cloud detection.

1) Constraint Handling and Packing Optimization: We formalize our packing task as a constrained two-dimensional (2D) bin packing problem, a classical NP-hard optimization problem. Given a set of bounding boxes $\mathcal{B} = \{b_1, b_2, \ldots, b_n\}$, where each b_i is a rectangle defined by its width w_i and height h_i, the goal is to arrange a subset of these boxes into a single packed image of fixed dimensions $W \times H$ such that:

1) No two boxes overlap: $\forall i \neq j,\ b_i \cap b_j = \emptyset$,
2) Each box is placed entirely within the packed image: $x_i + w_i \leq W$ and $y_i + h_i \leq H$,
3) The used area is optimized by minimizing $\sum_{b_i \in \mathcal{B}} w_i h_i$.
4) The number of boxes per packed image does not exceed m.

This problem is NP-hard due to the combinatorial nature of selecting and arranging subsets of boxes under spatial constraints. To solve it efficiently in practice, we adopt a decomposition-based heuristic inspired by Dantzig-Wolfe decomposition [20].

Our approach consists of two coordinated components: *Master Problem:* Determines which bounding boxes from $\mathcal{B}$ are assigned to each packed image, subject to constraints on maximum image size and number of boxes per image, and *Subproblem:* For each image, arranges its assigned boxes using a branch-and-cut algorithm that enforces non-overlap and in-bound placement, while minimizing unused space.

By decoupling the assignment and spatial layout tasks, our framework is able to scale to large image batches while preserving high detection quality and satisfying API constraints (e.g., maximum image resolution, bounding box count per API call).

Algorithm 1 Packing-Based Image Stitching (PBIS)

```
Require: Bounding boxes $B = \{b_1, \ldots, b_n\}$, where $b_i = (w_i, h_i)$
Require: Packed image size, $W \times H$
Require: Max # of bounding boxes per packed image, $m$
1:  $\mathcal{I} \leftarrow$ empty list;
2:  Sort bounding boxes in $B$ by size and group them into bins $\{\mathcal{B}_1, \mathcal{B}_2, \ldots\}$;
3:  for each bin $\mathcal{B}_k$ do
4:      while $\mathcal{B}_k \neq \emptyset$ do
5:          Initialize empty canvas $C$ with size $W \times H$;
6:          $P \leftarrow$ empty list, $c \leftarrow 0$;
7:          while space available and $\mathcal{B}_k \neq \emptyset$ and $c < m$ do
8:              Select next box $b_i \in \mathcal{B}_k$;
9:              if $b_i$ fits without overlap then
10:                 Place $b_i$ on $C$;
11:                 Add $b_i$ to $P$;
12:                 Remove $b_i$ from $\mathcal{B}_k$;
13:                 $c \leftarrow c + 1$;
14:             end if
15:         end while
16:         Add canvas $C$ with placements $P$ to $\mathcal{I}$;
17:     end while
18: end for
19: return $\mathcal{I}$
```

The Packing-Based Image Stitching (PBIS) algorithm (Algorithm 1) is designed to reduce cloud inference cost by aggregating multiple uncertain object detections into a minimal number of packed images. It takes as input a set of bounding boxes B, a fixed packed image size $W \times H$, and a maximum number of boxes per packed image m.

The algorithm begins by grouping bounding boxes into bins based on size similarity (Line 2). For each bin (Line 3), it initializes an empty canvas and attempts to place boxes onto it (Lines 4–17). The placement loop (Line 7) continues as long as there is sufficient space on the canvas, the number of placed boxes remains under the threshold m, and the bin is not empty.

Each candidate box b_i is checked for fit (Line 9) without overlapping existing placements. If successful, it is added to the current canvas (Line 10), recorded, and removed from the bin. Once no further boxes can be placed, the canvas is finalized and appended to the output list of packed images (Line 16). This process repeats until all bins are processed. Finally, the algorithm returns the set of packed images $\mathcal{I}$ (Line 19).

Key design choices include: *Line 2:* Binning boxes by size ensures uniform scaling and reduces detection degradation. *Lines 7–15:* A greedy placement strategy respects both spatial constraints and the per-image box cap m. Apparently, Algorithm 1 has a time complexity of $O(nm + n \log n)$.

This algorithm serves as a core component of the selective offloading framework and significantly reduces the number of cloud API calls, leading to lower cloud processing costs and improved inference efficiency. Moreover, by reducing the number of individual image transmissions, our method optimizes bandwidth usage and decreases the computational overhead associated with multiple inference initializations.

V. EXPERIMENTS

In this section, we validate our proposed method through extensive experiments. We first discuss our experiment setting in Section V-A and then briefly introduce our experimental method in Section V-B, based on which we present our experiment result in Sections V-C to V-E.

A. Experiment Settings

1) Datasets: We consider three popular datasets for object detection, listed as follows:

COCO (Common Objects in Context)[1] [21]: A large-scale dataset with over 330,000 images and 80 object categories. It features diverse scenes with multiple objects, varying object sizes, and complex occlusions, making it a comprehensive testbed for detection models.

Pascal VOC[2] [22]: This classic dataset contains annotated images for 20 object classes with relatively clean backgrounds and fewer occlusions compared to COCO. It provides a controlled environment to measure model performance on simpler scenes.

Open Images V7 - Simplified (OIV7-S)[3] [23]: We derive a simplified version of the Open Images V7 dataset[4] by selecting a manageable subset of the data and reducing class granularity. Many classes were merged based on visual similarity, and less frequent or ambiguous classes were removed. This step was essential due to the complexity and scale of the original dataset, which overwhelmed the small YOLOv5n model. The simplified OIV7-S retains semantic richness while enabling effective benchmarking on edge models.

2) Object Detection Models: As discussed in Section II-C, our proposed method is model-agnostic and can be integrated with any pair of edge-cloud models, provided that the cloud model has a demonstrably higher precision than the edge model. To demonstrate this, we use **YOLOv5n** as the edge model, which is a highly compact object detector suitable for real-time deployment on edge devices with limited

[1]https://cocodataset.org/#download
[2]http://host.robots.ox.ac.uk/pascal/VOC/
[3]https://anonymous.4open.science/r/SelectiveOffloading/
[4]https://storage.googleapis.com/openimages/web/index.html

TABLE I: Precision and Recall on COCO dataset

Model	Precision	Recall
Edge Model	0.745	0.83
Cloud Model	0.855	0.935

TABLE II: Default Parameters

Parameter	Value
Confidence threshold for the edge model, C_{thresh}	0.2
Conformal prediction threshold, α	0.2
Maximum bounding boxes per packed image, m	9
Packing grid size, $W \times H$	3×3
IoU threshold for evaluation, IoU_t	0.5

computational resources; and use **YOLOv11x** as the cloud model, which is a large-scale model with significantly better performance, used on the cloud server to provide refined predictions. The precision and recall of both models are listed in Table I.

3) Baselines: To assess the effectiveness of our proposed method, we compare our method with five main baselines, detailed as follows:

Full Edge/Cloud: With the edge and cloud models, one simple approach is to apply object detections for all images either on the edge or on the cloud, denoted as Full Edge (**FE**) and Full Cloud (**FC**).

Selective Offloading Methods: We implement two variants of our selective offloading methods: **SO-P**, which includes both the conformal prediction (Section III) and the packing algorithm (Section IV); and **SO-NP**, which includes only the conformal prediction without the packing algorithm.

Random Offloading Methods: Here, random sampling is used instead of conformal prediction to select the same number of bounding boxes as in selective offloading, and send them to the cloud without the packing algorithm. Similar to selective offloading, we also introduce three variants of this method, including: **RO-P**, which applies the random offloading with the packing algorithm; and **RO-NP**, which applies random offloading without the packing algorithm.

4) Implementation and Default Parameters: We implemented our algorithm in Python [5], and the default parameters are listed in Table II.

B. Evaluation Methodology

1) Evaluation Metrics: To evaluate the performance of different methods, we use precision and recall, as defined in Section II-A. For the methods involving both edge and cloud models, the final precision and recall are computed by aggregating results from both edge and cloud detection.

Additionally, we also consider the offloading cost as discussed in Section II-C. We adopt a simple cost model in our experiments, which is computed as the number of images sent to the cloud (i.e., the number of cloud API calls).

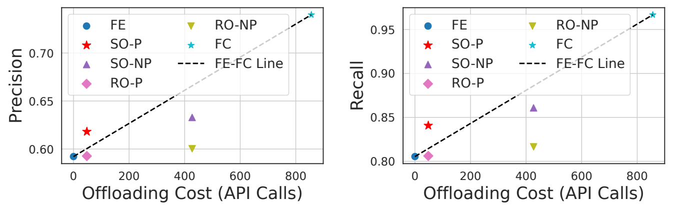
[5]The code is available at https://anonymous.4open.science/r/SelectiveOffloading/

2) Evaluation Objectives: Through the experiments, we wish to (a) validate that our method can significantly reduce cloud usage while maintaining high detection performance, and (b) demonstrate that we are able to control the cost of invoking the cloud model with a parameter provided by users.

C. Main Result

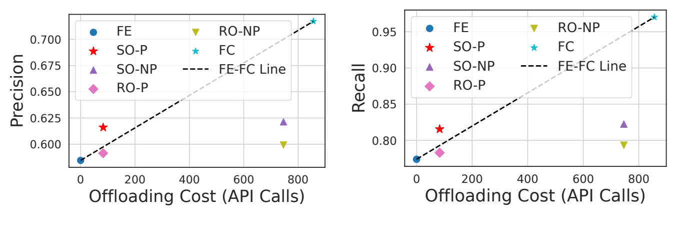
Overall Result. We first demonstrate the result for all methods in Figures 4 and 5, while the result on OIV7-S dataset shows a similar trend and is omitted for brevity. Take Figure 5a as an example, where we show the position of each method located by the recall value and offloading cost it produced. As can be observed, the full edge (FE) method is located at the bottom-left corner, meaning that it has the least precision (0.61) and the least cost (0), while the full cloud (FC) method is located at the top-right corner, depicting that it has the highest precision (0.76) yet also requires the highest cost (over 800 API calls). We prefer methods to be located on the top-left portion of the figure, which means improved precision compared with the full edge method, while having reduced cost compared with the full cloud method. As can be observed, our method falls into such an area, over-performing any existing baselines, demonstrating the effectiveness of our method. Specifically, our method achieves 0.635 precision while calling the API less than 100 times, which is over 8 times cheaper than the full cloud approach.

Figure 5b presents a similar comparison, this time focusing on recall instead of precision. As with the previous figure, the x-axis shows the offloading cost in terms of API calls, and the y-axis represents the recall achieved by each method. The full edge method is again positioned in the bottom-left corner with the lowest recall (0.79) and zero cost. In contrast, the full cloud method achieves the highest recall (0.96) but at the cost of over 800 API calls. Our goal remains to identify methods in the top-left area of the plot—those that achieve significantly higher recall than full edge while remaining much more efficient than full cloud. Our proposed method successfully occupies this desired region, attaining a recall of approximately 0.83, which is 4 out of the 17 percentage points available to gain over the full edge approach, while using fewer than 100 API calls, making it more than 8 times cheaper than the full cloud baseline. This further demonstrates the effectiveness of our approach in maintaining strong performance while significantly reducing computational cost.



(a) Precision (b) Recall

Fig. 4: Different methods on COCO dataset



(a) Precision (b) Recall

Fig. 5: Different methods on VOC dataset

Varying Offloading Strategies. We further compare our Selective Offloading results with Random Offloading. With the packing algorithm, the Selective Offloading (SO-P) always achieves higher precision compared the Random Offloading (RO-P), which can be consistently observed from Figures 4a and 5a. Similar trends can also be observed with recall from Figures 4b and 5b. In general, the Random Sampling methods show only a minor improvement over the edge-only baseline, while our Selective Offloading methods show substantial gains—proving that the value lies not just in offloading but in the selection of bounding boxes.

Impact of Packing. As can be consistently observed from Figures 4 and 5, with a similar level of precision and recall, the packing algorithm provides significant savings over the offloading costs by amortizing the cost across multiple bounding boxes, demonstrating superior performance.

D. Impact of Parameter Settings

We next analyze the sensitivity of our framework to several key parameters below.

1) Confidence Threshold on the Edge Model (C_{thresh}): The confidence threshold on the edge model, C_{thresh}, determines which detections are retained for further processing. A low threshold increases recall by allowing more detections, but often at the expense of precision due to noisy predictions. Conversely, a high threshold filters out low-confidence detections, increasing precision but reducing recall.

Figure 6 shows the impact of varying the confidence threshold on precision and recall. The x-axis represents the confidence threshold, while the y-axis shows the value of the corresponding metrics. As the threshold increases, recall drops steadily from close to 1 at very low thresholds to nearly 0 at the highest threshold. This trend indicates that lower thresholds favor recall, capturing more true positives but at the cost of lower precision. Conversely, precision consistently improves with higher thresholds, reaching 1.0 near the top end, as predictions become more conservative and selective.

This figure highlights the trade-off between precision and recall as the confidence threshold changes, and emphasizes the importance of selecting an appropriate threshold to balance these metrics based on application requirements.

2) Conformal Prediction Threshold (α): The parameter α in conformal prediction defines the statistical confidence level for detecting uncertainty. Lower values of α correspond to stricter uncertainty estimates, flagging more detections as uncertain. By choosing the right α, users can choose the trade-off between performance and cost.

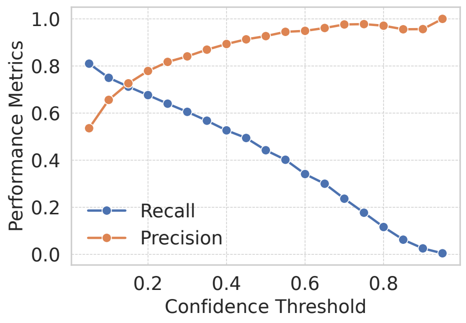
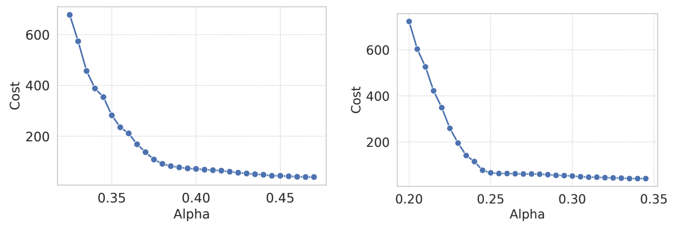


Fig. 6: Varying C_{thresh} on VOC dataset

Figure 7 shows that offloading cost shrinks significantly with increasing α, indicating a direct control mechanism over cloud usage. Empirically, $\alpha \in [0.18, 0.22]$ offers the best cost-performance tradeoff across datasets.



(a) COCO (b) VOC

Fig. 7: α vs Offloading Cost

3) Offloading Cost vs. Performance: As shown in Figure 8, increasing the number of cloud API calls improves recall and precision. However, this improvement plateaus beyond a certain point, showing diminishing returns. This validates our hypothesis that a small amount of smart offloading yields most of the precision benefits of full cloud processing.

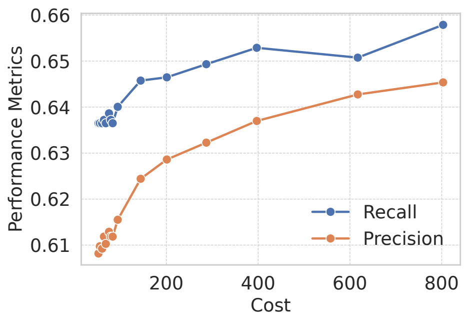


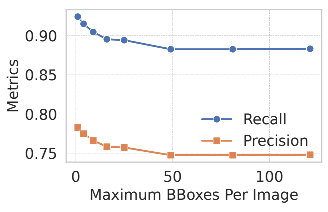
Fig. 8: Offloading Cost vs Performance (VOC Dataset)

4) Packing Granularity: We study the effect of varying the maximum number of bounding boxes per packed image (m) on both detection performance and cost, as shown in Figure 9. As the number of bounding boxes increases, both precision and recall exhibit a gradual decline due to reduced spatial context and resolution available for each region, which negatively impacts detection quality. In contrast, the offloading cost drops significantly, since more regions are processed in a single cloud API call. Importantly, after a certain point, the performance approaches that of the full edge method, as the benefits of cloud inference diminish with overly dense packing (see figure 10). The relatively shallow decline in detection quality compared to the exponential reduction in cost suggests that bounding box packing is an effective strategy for achieving cost-efficient inference.

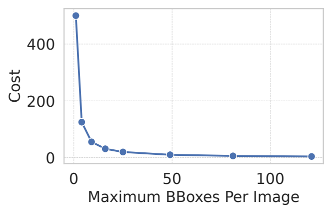
Allowing 4-9 boxes per packed image was found to be an effective compromise across all datasets.

E. Summary

From the above experiments, we have the following key observations: (a) Our method achieves 80–90% of full cloud performance while using only 10–20% offloading cost compared with the full cloud model. (b) Our Selective Offloading method consistently outperforms Random Offloading across all metrics. (c) Packing significantly reduces API calls and enhances scalability without compromising precision and recall. (d) The combination of uncertainty estimation and packing offers a robust and cost-effective solution for edge-cloud collaborative inference.

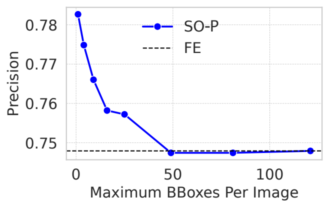


(a) Precision & Recall

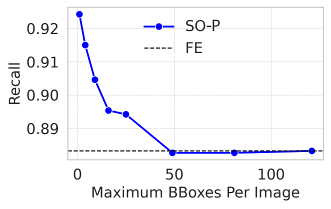


(b) Cost

Fig. 9: Effects of varying the maximum number of bounding boxes per packed image m on performance and cost.



(a) Precision



(b) Recall

Fig. 10: Effects of varying the maximum number of bounding boxes per packed image m on performance compared to full edge as baseline.

VI. Related Work

A. Edge-Based Object Detection

Object detection on edge devices has received significant attention due to the demand for real-time inference in resource-constrained environments. To address the limitations of computational capacity and memory, several lightweight models have been proposed. LES-YOLO [24] introduces a streamlined version of YOLO with a redesigned backbone and an EMS-PAN neck structure to enhance multi-scale feature fusion, achieving substantial gains in both efficiency and accuracy on UAV-based datasets. Similarly, SOD-YOLO [25] improves detection of small objects through a receptive field convolutional block attention module (RFCBAM) and a novel neck called BSSI-FPN, reducing parameters by over 80% compared to YOLOv8s.

While these models are optimized for edge inference, they often underperform in complex or cluttered scenes due to the trade-off between model size and accuracy. Thus, many real-world systems augment edge processing with cloud-based computation to mitigate these limitations.

B. Cloud Offloading for Object Detection

Cloud offloading allows high-accuracy models to be executed remotely, bypassing local hardware constraints. Full cloud offloading methods such as CODS [9] and heuristic-based frameworks like EdgeDuet [10] highlight the benefits and limitations of this approach. EdgeDuet, for instance, improves detection accuracy and latency by offloading small object detection using tile-level parallelism. However, these methods typically rely on static heuristics (e.g., object size or confidence scores), which may lead to excessive offloading or missed detections.

To address these issues, we propose an adaptive offloading strategy that uses uncertainty estimation to identify ambiguous detections at the edge, significantly improving the balance between cost and accuracy.

C. Uncertainty Estimation with Conformal Prediction

Accurate uncertainty estimation is crucial for making informed offloading decisions. Traditional approaches, such as Bayesian neural networks or ensemble methods, are often impractical for edge deployment due to their computational overhead. Conformal prediction [11], [14], [15] offers a distribution-free, statistically valid framework for uncertainty quantification. It provides reliable confidence intervals for predictions using only a calibration dataset, making it well-suited for edge environments.

Although conformal methods have been applied in classification and regression tasks [16], their use in real-time object detection and selective offloading remains underexplored. Our work integrates conformal prediction with object detection to enable dynamic offloading based on principled uncertainty thresholds.

D. System-Level Optimizations

Reducing the data transmitted to the cloud is essential for cost and latency constraints. Techniques like L-Filter [26] improve throughput by filtering out empty video frames. Our method extends this idea to static images through a novel Packing-Based Image Stitching (PBIS) algorithm, which batches uncertain regions into composite images for offloading. Inspired by the two-dimensional bin packing problem [20], this approach minimizes redundant transmissions and API calls without sacrificing resolution or detection quality.

Earlier works on video compression [7] and object tracking [8] are primarily designed for temporal data and do not generalize to independent images.

E. Summary

In summary, existing approaches to edge-cloud object detection either sacrifice accuracy for efficiency or incur excessive cloud costs. Our work integrates conformal uncertainty estimation with a packing-based optimization strategy to create a selective offloading framework that achieves near-cloud performance with significantly lower resource usage, outperforming heuristic-based methods in both accuracy and efficiency.

VII. Conclusion

This paper presents a selective cloud offloading approach for efficient and accurate object detection on resource-constrained edge devices. By leveraging conformal prediction to quantify uncertainty, our method dynamically balances accuracy and offloading costs, ensuring that only high-uncertainty regions are processed in the cloud. Additionally, our packing-based optimization strategy significantly reduces transmission overhead while maintaining high detection performance. This makes our solution well-suited for real-time applications in edge computing, autonomous systems, and smart surveillance, where efficiency and accuracy must be carefully balanced. Future work will focus on improving the accuracy estimation of both edge and cloud models, enabling better-informed offloading decisions. Additionally, we will explore novel compression techniques specifically designed for packing multiple image regions together, further reducing transmission costs while preserving detection quality.

References

[1] J. Zhao, Z. Zhang, J. Ren, H. Zhang, Z. Zhao, and M. Wang, “Dual cross-stage partial learning for detecting objects in dehazed images,” in *2024 IEEE International Conference on Data Mining (ICDM)*, 2024, pp. 629–638.

[2] M. Bhange and H. Hingoliwala, “Smart farming: Pomegranate disease detection using image processing,” *Procedia computer science*, vol. 58, pp. 280–288, 2015.

[3] L. Ţepelea, I. Gavriluţ, and A. Gacsádi, “Smartphone application to assist visually impaired people,” in *2017 14th International Conference on Engineering of Modern Electric Systems (EMES)*, 2017, pp. 228–231.

[4] R. Tapu, B. Mocanu, A. Bursuc, and T. Zaharia, “A smartphone-based obstacle detection and classification system for assisting visually impaired people,” in *Proceedings of the IEEE International Conference on Computer Vision Workshops*, 2013, pp. 444–451.

[5] M. Karthick and R. Suguna, “Obstacles detection for visually impaired people using smart phones,” *Int. J. Emerg. Technol. Comput. Sci. Electron*, vol. 13, pp. 530–535, 2015.

[6] S. Dong, J. Tang, K. Abbas, R. Hou, J. Kamruzzaman, L. Rutkowski, and R. Buyya, “Task offloading strategies for mobile edge computing: A survey,” *Computer Networks*, vol. 254, p. 110791, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128624006236

[7] L. Galteri, M. Bertini, L. Seidenari, and A. Del Bimbo, “Video compression for object detection algorithms,” in *2018 24th International Conference on Pattern Recognition (ICPR)*. IEEE, 2018, pp. 3007–3012.

[8] W. Yang, B. Liu, W. Li, and N. Yu, “Tracking assisted faster video object detection,” *2019 IEEE International Conference on Multimedia and Expo (ICME)*, pp. 1750–1755, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:199490530

[9] T. Li, X. Zhang, N. S. Nguyen, and B. Sheng, “Cods: Cloud-assisted object detection for streaming videos on edge devices,” in *2021 IEEE International Performance, Computing, and Communications Conference (IPCCC)*, 2021, pp. 1–6.

[10] Z. Yang, X. Wang, J. Wu, Y. Zhao, Q. Ma, X. Miao, L. Zhang, and Z. Zhou, “Edgeduet: Tiling small object detection for edge assisted autonomous mobile vision,” *IEEE/ACM Trans. Netw.*, vol. 31, no. 4, p. 1765–1778, Dec. 2022. [Online]. Available: https://doi.org/10.1109/TNET.2022.3223412

[11] X. Zhou, B. Chen, Y. Gui, and L. Cheng, “Conformal prediction: A data perspective,” 2024. [Online]. Available: https://arxiv.org/abs/2410.06494

[12] M. Everingham, L. Gool, C. K. Williams, J. Winn, and A. Zisserman, “The pascal visual object classes (voc) challenge,” *Int. J. Comput. Vision*, vol. 88, no. 2, p. 303–338, Jun. 2010. [Online]. Available: https://doi.org/10.1007/s11263-009-0275-4

[13] T. Silvanus, “Comparative analysis of yolov8, yolov9, and yolov10 for object detection: Performance metrics and real-world applications,” *International Journal for Research in Applied Science and Engineering Technology*, vol. 13, pp. 5045–5051, 05 2025.

[14] V. Vovk, A. Gammerman, and G. Shafer, *Algorithmic Learning in a Random World*, 01 2005.

[15] J. Lei, M. G’Sell, A. Rinaldo, R. J. Tibshirani, and L. Wasserman, “Distribution-free predictive inference for regression,” 2017. [Online]. Available: https://arxiv.org/abs/1604.04173

[16] H. Boström, U. Johansson, and A. Vesterberg, “Predicting with confidence from survival data,” in *Proceedings of the Eighth Symposium on Conformal and Probabilistic Prediction and Applications*, ser. Proceedings of Machine Learning Research, A. Gammerman, V. Vovk, Z. Luo, and E. Smirnov, Eds., vol. 105. PMLR, 09–11 Sep 2019, pp. 123–141. [Online]. Available: https://proceedings.mlr.press/v105/bostrom19a.html

[17] J. Qiu, R. Wang, B. Hu, R. Guerin, and C. Lu, “Optimizing edge offloading decisions for object detection,” 2024. [Online]. Available: https://arxiv.org/abs/2410.18919

[18] R. Girshick, “Fast r-cnn,” in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.

[19] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[20] D. Pisinger and M. Sigurd, “Using decomposition techniques and constraint programming for solving the two-dimensional bin-packing problem,” *INFORMS Journal on Computing*, vol. 19, no. 1, pp. 36–51, 2007. [Online]. Available: https://doi.org/10.1287/ijoc.1060.0181

[21] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, “Microsoft coco: Common objects in context,” in *Computer vision–ECCV 2014: 13th European conference, zurich, Switzerland, September 6-12, 2014, proceedings, part v 13*. Springer, 2014, pp. 740–755.

[22] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, “The pascal visual object classes (voc) challenge,” *International journal of computer vision*, vol. 88, no. 2, pp. 303–338, 2010.

[23] A. Kuznetsova, H. Rom, N. Alldrin, J. Uijlings, I. Krasin, J. Pont-Tuset, S. Kamali, S. Popov, M. Malloci, A. Kolesnikov *et al.*, “The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale,” *International journal of computer vision*, vol. 128, no. 7, pp. 1956–1981, 2020.

[24] H. Zhang, L. Deng, S. Lin, H. Zhang, J. Dong, D. Wan, L. Bi, and H. Liu, “Les-yolo: efficient object detection algorithm used on uav for traffic monitoring,” *Measurement Science and Technology*, vol. 36, no. 1, p. 016008, oct 2024. [Online]. Available: https://dx.doi.org/10.1088/1361-6501/ad86e2

[25] Y. Li, Q. Li, J. Pan, Y. Zhou, H. Zhu, H. Wei, and C. Liu, “Sod-yolo: Small-object-detection algorithm based on improved yolov8 for uav images,” *Remote Sensing*, vol. 16, no. 16, 2024. [Online]. Available: https://www.mdpi.com/2072-4292/16/16/3057

[26] Y. Liu and K.-D. Kang, “Filtering empty video frames for efficient real-time object detection,” *Sensors*, vol. 24, no. 10, 2024. [Online]. Available: https://www.mdpi.com/1424-8220/24/10/3025