SELECTIVE CLOUD OFFLOADING FOR ACCURATE AND EFFICIENT OBJECT DETECTION

DAVOOD DEHGHANI

A THESIS SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF ARTS

GRADUATE PROGRAM IN INFORMATION SYSTEMS & TECHNOLOGY
YORK UNIVERSITY
TORONTO, ONTARIO
JULY 2025

Abstract

High-accuracy object detection on resource-constrained devices is essential for many applications including autonomous systems, agriculture, 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 cost.

This thesis proposes a selective cloud offloading framework that balances prediction accuracy and processing cost. A lightweight edge model makes initial predictions using conformal prediction to quantify uncertainty. Only high-uncertainty regions are offloaded to the cloud for refinement by more powerful models. To further optimize efficiency, multiple uncertain regions are merged into a single image before offloading, reducing transmission and processing costs. The system is evaluated on real datasets, demonstrating substantial accuracy improvements with minimal additional overhead.

Acknowledgements

This document was created with support from AI tools in a limited and transparent manner. Specifically, ChatGPT was used to help identify some relevant papers for citation purposes, all of which were subsequently reviewed in full by me to ensure accuracy and relevance. Additionally, ChatGPT was used for occasional rephrasing and grammatical or language improvements. However, the final content, structure, and writing were carefully reviewed, edited, and approved by me to ensure originality and academic integrity.

I would also like to express my deepest gratitude to my supervisors, Professor Xiaohui Yu and Dr.Yueting Chen, for their invaluable guidance, support, and encouragement throughout this research. Their insights, constructive feedback, and continuous mentorship have been instrumental in shaping this thesis and my academic development.

Contents

Abstract **ii**

Acknowledgements **iii**

Table of Contents **iv**

List of Tables **vii**

List of Figures **viii**

1 Introduction **1**

1.1 Motivation 1

1.2 Background 2

1.3 Overview of the Proposed Solution 3

1.4 Contributions 4

1.5 Thesis Organization 4

2 Related Work **5**

2.1 Object Detection 5

2.2 Edge Computing 8

2.3 Cloud Offloading 10

3 Preliminaries and Problem Definition **12**

3.1 Object Detection Problem 12

3.2 Conformal Prediction 13

3.3 Object Detection with Cloud Offloading 14

4 Proposed Solution: Selective Offloading **15**

4.1 Object Detection on the Edge 15

4.2 Selective Offloading 16

4.2.1 Uncertainty Quantification Using Conformal Prediction 17

4.2.2 Offloading Decision Policy 17

4.2.3 Advantages of Conformal Prediction for Offloading 18

5 Optimization: Image Stitching **19**

5.1 Motivation 19

5.2 Approach 19

5.2.1 Constraint Handling and Packing Optimization 21

6 Experiments **25**

6.1 Experiment Settings 25

6.1.1 Datasets 25

6.1.2 Object Detection Models 26

6.1.3 Baselines 26

6.1.4 Implementation and Default Parameters 27

6.2 Evaluation Methodology 28

6.2.1 Evaluation Metrics 28

6.2.2 Evaluation Objectives 28

6.3 Main Result 28

6.4 Impact of Parameter Settings 30

6.4.1 Confidence Threshold on the Edge Model 30

6.4.2 Conformal Prediction Threshold (α) 31

6.4.3 Offloading Cost vs. Performance 31

6.4.4 Packing Granularity 32

6.5 Summary 33

7 Conclusion and Future Work **35**

7.1 Summary of Findings 35

7.2 Future Work 36
7.2.1 Advanced System Optimization 36
7.2.2 Real-World Validation and Deployment 36
7.2.3 Scalability and Interoperability 37

Bibliography **37**

List of Tables

6.1 Precision and Recall on COCO dataset 26
6.2 Default Parameters 27

List of Figures

1.1 Comparison between edge and cloud predictions. The model at the edge does not perform as well as the model on the cloud. 2

4.1 System overview: A high-level representation of the selective cloud offloading framework. 15

5.1 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. 20

6.1 Different methods on COCO dataset 29
6.2 Different methods on VOC dataset 30
6.3 Varying C_{thresh} on VOC dataset 31
6.4 α vs Offloading Cost 32
6.5 Offloading Cost vs Performance (VOC Dataset) 32
6.6 Effects of varying the maximum number of bounding boxes per packed image m on performance and cost. 33
6.7 Effects of varying the maximum number of bounding boxes per packed image m on performance compared to full edge as baseline. 34

1 Introduction

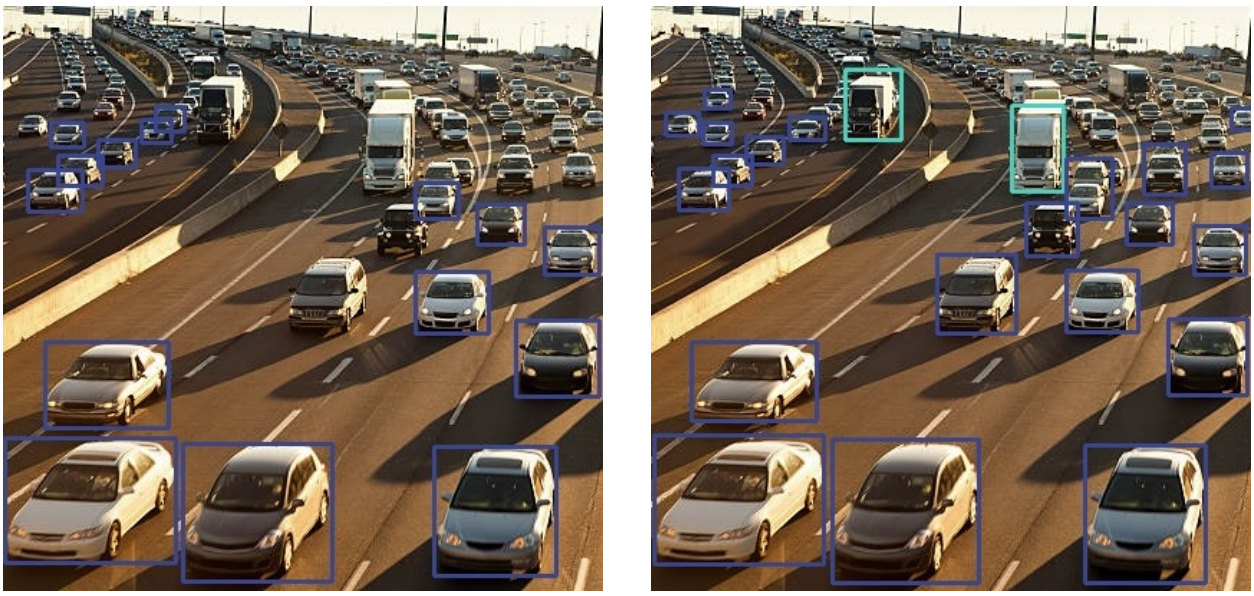
1.1 Motivation

Object detection [1] is foundational to a wide range of modern applications and continues to enable increasingly diverse use cases. It assists farmers in identifying plant diseases, pests, or assessing fruit ripeness using low-performance field devices [2, 3], and supports visually impaired individuals in recognizing objects or reading text through wearable technologies [4, 5, 6]. In healthcare, it facilitates early diagnosis through medical image analysis [7]. In autonomous driving, real-time object recognition is vital for obstacle avoidance and traffic scene understanding [8]. Smart city infrastructure and industrial automation also leverage object detection for safety monitoring, anomaly detection, and operational optimization [9]. These examples illustrate just a subset of its growing impact across domains.

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 [10], 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)

Figure 1.1: Comparison between edge and cloud predictions. The model at the edge does not perform as well as the model on the cloud.

1.2 Background

Several approaches have targeted video stream optimization, for instance, by compressing frames [11] or using local tracking with intermittent cloud offloading, sometimes guided by server context [12, 13]. 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 [14] 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.

1.3 Overview of the Proposed Solution

First, this thesis establishes an intelligent mechanism for deciding *what* to offload by accurately quantifying the uncertainty of object detections performed at the edge. The proposed system initially employs a lightweight object detection model on the resource-constrained device. Crucially, it then leverages **conformal prediction** [15, 16, 17], 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 for a reliable assessment of the likelihood that a detection is correct. By introducing a tunable threshold on this uncertainty measure (the conformal score), the 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, this work optimizes *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 address this, a **packing-based optimization strategy** is introduced. 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, the 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.

1.4 Contributions

To summarize, this thesis makes the following contributions:

1. A selective cloud offloading strategy that balances cost and accuracy by leveraging conformal prediction to assess uncertainty in object detection.

2. A packing-based optimization technique to further reduce cloud transmission costs while maintaining detection precision.

3. A user-configurable trade-off mechanism, allowing applications to adjust accuracy and cost constraints based on specific requirements.

4. An extensive experimental evaluation demonstrating that the proposed method achieves substantial accuracy improvements with significantly less data compared to existing approaches.

1.5 Thesis Organization

The remainder of this thesis is organized as follows. Chapter 3 discusses the preliminaries and defines the problem. Chapter 4 presents the architecture and components of the selective offloading solution. Chapter 5 introduces algorithms for image stitching. Chapter 6 presents experimental results. Chapter 2 discusses related work, and Chapter 7 concludes the thesis with directions for future research.

2 Related Work

This chapter reviews prior research in three key areas relevant to this thesis: object detection, edge computing, and cloud offloading. It places the proposed selective offloading framework within the context of recent advancements in each domain.

2.1 Object Detection

With the advancement of deep neural networks [18, 19], attention-based models[20], and the availability of large-scale annotated datasets[21, 22, 23], object detection has reached a level of accuracy that enables reliable recognition of objects in both images and videos. These data formats dominate today's internet traffic, making object detection a core component of many real-world applications such as agriculture [24], autonomous navigation [25, 26], surveillance [27], and even sport analytics [28]. According to recent statistics, video alone accounts for more than 70% of global internet traffic [29], highlighting the practical significance of high-performance visual recognition systems.

Several research efforts have produced highly accurate models such as Faster R-CNN [30], SSD [31], YOLOv5–v12 [32, 33, 34, 18, 35], and DETR [36], which achieve impressive results on standard benchmarks like COCO [37], Open Images [22], and Pascal VOC [38]. These models represent the current state of the art in object detection, combining advances in convolutional backbones, multi-scale feature fusion, and attention mechanisms.

Despite these gains in accuracy, deep learning models remain computationally expensive and are often unsuitable for real-time or on-device deployment without modification. For example, Faster R-CNN typically processes images at around 5 FPS on GPUs [39]. Processing high-resolution video streams or large image batches with heavyweight models introduces latency, energy consumption, and cost challenges.

To address these issues, a substantial body of research has focused on accelerating inference while preserving accuracy. Based on an internal survey of over 30 state-of-the-art papers on object detection and image and video analytics, I identified five general strategies commonly employed to speed up inference in large-scale settings:

1. *Model Architecture Optimization*: Redesigning core detector architectures for speed–accuracy trade-offs. To name a few examples, YOLOv10 [18] eliminates postprocessing (e.g., NMS), while LW-DETR [40] and RT-DETRv3 [41] provide real-time DETR alternatives with dense supervision and lightweight attention.

2. *Data Preprocessing*: Transforming or indexing visual data at ingest time to reduce computation during inference. These techniques primarily involve creating rich metadata, feature indices, or frame-level summaries to support efficient querying, as demonstrated in Tahoma [42] and VStore [43]. However, such preprocessing incurs a significant ingest-time delay before inference can begin, rendering it unsuitable for real-time or live streaming scenarios where data arrives continuously and cannot be indexed ahead of time.

3. *Filtering*: Using lightweight models or heuristic rules to eliminate uninformative frames or regions before passing them to expensive detectors. For example, NoScope[44] employs difference detectors to skip redundant frames in static camera settings.

4. *Specialized Neural Networks (SNNs)*: Training compact proxy models tailored to specific object classes or scene types. Systems like BlazeIt[45] and Focus use these to pre-filter frames or estimate query-specific statistics.

5. *Mathematical Optimization*: Applying statistical techniques such as Monte Carlo estimation or control variates to reduce sample sizes or prioritize high-value inferences, especially for aggregate or limit queries.

These categories are not mutually exclusive. Many systems adopt a hybrid approach, combining two or more of these techniques to balance speed and accuracy. Our survey provides a taxonomy of such systems and highlights where each technique is most effective, depending on the application domain and computational constraints.

Model architecture optimization is a direct and intuitive strategy for accelerating inference by redesigning object detectors from the ground up. Recent models achieve gains by removing costly postprocessing steps such as non-maximum suppression (NMS) and incorporating architecture-level improvements tailored for edge efficiency [18, 40]. However, these optimizations often reach a point of diminishing returns; despite engineering advances, they typically fall short of delivering the performance needed for real-world deployment without compromising accuracy—the very metric that made object detection interesting in recent years.

Specialized neural networks (SNNs) have emerged as an effective tool for accelerating object detection in video. These networks are trained to mimic the behavior of a larger, more accurate reference model on a limited or simplified task (e.g., binary classification for the presence of a specific object). Systems like BlazeIt [45] use SNNs both for sampling prioritization and as control variates in approximate query processing. Focus leverages SNNs to build lightweight indices during preprocessing, enabling rapid exploratory querying over large image and video datasets.

In addition, *filtering strategies* such as difference detectors (NoScope [44]) and clustering-based frame selection (Focus [46]) help eliminate redundant or low-information frames before running full object detection. These techniques are particularly useful in static-camera environments where temporal redundancy is high. However, as our survey highlights, these techniques are less effective for dynamic or mobile multimedia sources.

Recent innovations also explore dynamic model selection and adaptive cascades. For instance, Tahoma[42] builds hundreds of classifier cascades by jointly optimizing input representations (e.g., image resolution or color depth) and model architecture. Similarly, the Windowed ϵ-Greedy algorithm trains fast classifiers under short-term class skew to achieve online acceleration of object classification.

Mathematical optimization techniques offer an orthogonal direction to architectural or data-level improvements. Rather than changing the models or input data, these methods apply statistical principles to reduce computational cost, especially for aggregate and limit queries.

For instance, BlazeIt [45] uses control variates—a classical variance reduction method

from Monte Carlo simulation—to reduce the number of samples needed for answering queries with statistical guarantees. Similarly, Video Monitoring Queries [47] applies Monte Carlo estimation to spatial aggregate queries, significantly lowering variance while maintaining accuracy. SVQ++ [48] goes further by adapting the Kolmogorov–Smirnov test to dynamically trigger model adaptation in response to shifts in video content distributions.

Object detection has advanced significantly in accuracy, driven by deep models and large datasets. However, deploying these models at scale remains challenging due to high computational costs. Recent work addresses this through architectural simplifications, data filtering, lightweight proxies, and statistical optimization to balance speed and accuracy.

2.2 Edge Computing

Edge computing has emerged as a response to the growing demands of latency-sensitive, bandwidth-intensive, and privacy-aware applications, particularly in the context of the Internet of Things (IoT), autonomous vehicles, and real-time analytics [49, 50].

Unlike traditional cloud computing, which centralizes computation in large data centers, edge computing pushes computation closer to the data source (at the "edge" of the network) thereby reducing transmission latency, alleviating backbone network congestion, improving privacy, and enhancing contextual awareness [51].

Despite its advantages, edge computing faces several challenges. Resource constraints are perhaps the most cited limitation: edge nodes often have limited CPU, memory, and storage compared to cloud infrastructures [52]. Security concerns are amplified due to the distributed and heterogeneous nature of edge devices [53]. Moreover, orchestration complexity, including service placement, task scheduling, and state management across edge nodes, significantly complicates system design [54].

The rise in edge computing adoption is closely linked to the proliferation of connected devices, the exponential growth of data generated at the edge, and the limitations of centralized cloud architectures to meet real-time performance requirements. Furthermore, advancements in 5G networks and containerization technologies have enabled low-latency communication and lightweight deployment of services at the edge [55].

The literature has proposed various frameworks and paradigms to address the limitations of edge computing. Broadly, these can be categorized into:

1. *Computation Offloading:* Deciding when and what tasks to offload from the edge to more powerful nodes or the cloud [56]. For instance, MAUI [57] introduces a system that offloads fine-grained methods from smartphones to cloud servers to extend battery life. CloneCloud [58] similarly partitions mobile applications, migrating part of the execution to the cloud to improve performance without developer intervention. ThinkAir [59] extends this concept by supporting parallel execution and dynamic VM allocation, making it highly scalable for computation-heavy tasks.

2. *Resource Management:* Dynamic allocation of computing, storage, and networking resources across distributed nodes [60]. FogAtlas [61] proposes a system that utilizes software-defined networking (SDN) and network function virtualization (NFV) to intelligently place services across the fog continuum. Diro and Chilamkurti [62] present a reinforcement learning-based strategy that adapts resource allocation policies in real-time to changing workloads. Taleb et al. [63] offer a comprehensive orchestration model within 5G Multi-access Edge Computing (MEC), addressing deployment, elasticity, and mobility.

3. *Task Scheduling:* Optimizing the execution order and placement of computational tasks to minimize latency or energy consumption [64]. Xu et al. [65] introduce a deep reinforcement learning scheduler that dynamically learns optimal policies to maximize Quality of Service (QoS) under varying network conditions. Oueis et al. [66] evaluate practical scheduling algorithms that consider user mobility and limited radio resources. Wang et al. [67] design an energy-aware scheduler to optimize power consumption by adjusting task placement based on current device energy levels.

4. *Security and Privacy Models:* Employing cryptographic protocols, trust frameworks, and federated learning to ensure data confidentiality and integrity [68]. Gai et al. [69] present a privacy-preserving data aggregation model for smart metering that uses homomorphic encryption to enable secure regional statistics computation. Costan and

Devadas [70] provide an in-depth explanation of Intel SGX, which enables trusted execution environments (TEEs) for secure edge-side computations. Li et al. [71] develop FedAvg, a federated learning protocol that trains AI models on-device without centralizing raw data, preserving user privacy.

5. *Fault Tolerance and Redundancy:* Strategies for ensuring reliability in the face of node failure or communication loss [72]. Yi et al. [73] discuss proactive replication at the edge, duplicating critical tasks to backup nodes to maintain service continuity. Tang et al. [74] propose a dynamic checkpointing and rollback recovery mechanism for cloud-edge environments that minimizes state loss and recovery time. Ongaro and Ousterhout [75] present the Raft consensus algorithm, which ensures system consistency and fault tolerance among distributed edge nodes.

Among these, *computation offloading* has garnered particular attention due to its potential to mitigate the inherent limitations of edge nodes. The next section explores this solution in detail.

2.3 Cloud Offloading

Cloud offloading is a widely studied strategy within edge computing that seeks to augment the capabilities of resource-constrained edge devices by delegating intensive computation tasks to remote cloud servers. The core idea is to reduce latency, energy consumption, and processing load on end-user devices by leveraging the virtually unlimited resources of the cloud [56, 55, 10].

The motivation behind cloud offloading lies in the trade-offs between local execution and remote processing. While executing tasks locally offers low-latency data access, it is often constrained by the limited computational power and battery life of edge devices such as smartphones, IoT sensors, or autonomous drones. Offloading enables these devices to perform complex tasks—such as image processing, machine learning inference, and large-scale data analytics—without overwhelming local resources [57].

Despite its advantages, cloud offloading presents several technical and architectural challenges. Network latency and variability can negate the performance gains from offloading,

particularly for delay-sensitive applications [59]. Moreover, the overhead of determining what to offload, when to offload, and to which server to offload introduces significant scheduling complexity. Security and privacy concerns are also central: transmitting data to the cloud exposes it to interception, tampering, or misuse [58].

The literature offers various offloading strategies, each addressing different optimization goals:

- *Static Offloading:* Decisions are made at compile-time or deployment-time. CloneCloud [58] exemplifies this, partitioning applications into local and cloud-executable segments.
- *Dynamic Offloading:* Decisions are made at runtime based on contextual data such as CPU load, battery level, or network bandwidth. MAUI [57] employs a cost model to decide dynamically which methods to offload.
- *Partial vs. Full Offloading:* ThinkAir [59] allows both partial and full task migration, supporting parallel cloud execution and dynamic VM provisioning for scalability.
- *Energy-Aware Offloading:* Strategies such as those proposed in [67] minimize device energy consumption by adjusting task placement according to current battery and workload levels.
- *Multi-tier Offloading:* Recent approaches consider a hybrid edge–cloud continuum, offloading tasks first to nearby edge servers and then to the cloud if necessary [54].

The convergence of 5G, AI, and edge-cloud orchestration frameworks is reshaping the landscape of offloading. With the advent of federated learning, it is now possible to offload only model updates rather than raw data, preserving privacy while still leveraging cloud computation power [71]. Furthermore, containerized microservices and lightweight virtualization techniques have enabled more granular and efficient task migration across heterogeneous infrastructure [63].

Cloud offloading remains a cornerstone of edge computing strategies, especially in scenarios where real-time responsiveness and low device power budgets are critical. It also lays the groundwork for more complex, adaptive offloading frameworks that integrate machine learning, mobility prediction, and real-time resource orchestration.

3 Preliminaries and Problem Definition

This thesis considers the object detection problem on a set of images, X, referred to as the **Image Set**. Each element $x \in X$ represents one image. The general object detection problem is modeled in Section 3.1, followed by the specific formulation adopted in this thesis in Section 3.3.

3.1 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, ground-truth results 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. A predicted box $y_i^M \in Y^M(x)$ is considered a **match** to a ground-truth box $y_i \in Y(x)$ if the **Intersection over Union (IoU)** of the corresponding bounding boxes exceeds a predefined threshold IoU_t (typically 0.5) [76]. Specifically,

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

where Area() computes the area in pixels, and $\cap$ and $\cup$ represent the intersection and union of bounding boxes, respectively.

A prediction y_i^M is considered **correct** if there exists at least one ground-truth box B_i such that $IoU(B_i^M, B_i) \geq IoU_t$ and the predicted label matches the ground-truth label. For brevity, $Y_x^M \cap Y_x$ denotes the set of correct predictions from Y_x^M.

To evaluate the performance of a model M on an image x, two standard metrics are adopted: **Precision** and **Recall** [77].

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

For an entire image set X, define $\mathbb{Y}_X^M$ and $\mathbb{Y}_X$ as the unions of predictions and ground truth, respectively. Precision and recall are then calculated as $Prec(\mathbb{Y}_X, \mathbb{Y}_X^M)$ and $Rec(\mathbb{Y}_X, \mathbb{Y}_X^M)$.

When ground-truth data are unavailable, predictions from the cloud model—due to their typically higher quality—may serve as a proxy. In such cases, the cloud model's precision and recall are assumed to be 1.

3.2 Conformal Prediction

Conformal prediction [17, 15, 16] is a statistical framework that provides principled uncertainty estimates for machine learning predictions. It operates under the assumption of *exchangeability*, meaning that the data distribution remains invariant under permutations. This assumption allows conformal methods to offer finite-sample, distribution-free guarantees on prediction reliability.

The dataset is split into a training set and a calibration set. The model is trained on the former, while the calibration set $\mathcal{C} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$—with $x_i \in \mathbb{R}^d$ and $y_i \in \{0, 1\}$—is used to measure model uncertainty.

A nonconformity function $\alpha : \mathbb{R}^d \times \{0, 1\} \to \mathbb{R}$ quantifies how atypical a data point is. For each calibration point, compute:

$$\alpha_i = \alpha(x_i, y_i).$$

To predict the label for a new input x_{n+1}, compute the nonconformity score $\alpha_{n+1} = \alpha(x_{n+1}, y)$ for each possible label $y \in \{0, 1\}$. The associated p-value is:

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

The conformal prediction set at significance level α (confidence $1 - \alpha$) is:

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

This guarantees that the true label y_{n+1} is included in $\Gamma(x_{n+1})$ with probability at least $1 - \alpha$. The value of α allows control over prediction set size versus confidence.

3.3 Object Detection with Cloud Offloading

This thesis considers two object detection models: the **edge model** M_e and the **cloud model** M_c. The edge model operates with lower computational cost but yields lower accuracy, whereas the cloud model is more accurate but incurs higher costs in latency, computation, and data transmission.

The goal is to selectively offload detections from M_e that are likely incorrect, sending only those parts of images to M_c for refinement. This strategy is known as **selective cloud offloading** [56, 55, 10].

Let $\mathbb{Y}_X^e$ and $\mathbb{Y}_X^c$ denote predictions from M_e and M_c, respectively. Let $\mathbb{Y'}_X^e \subset \mathbb{Y}_X^e$ represent uncertain predictions selected for cloud refinement, and let $\mathbb{Y'}_X^c$ be the refined outputs. The final prediction set is:

$$\mathcal{Y}_X = (\mathbb{Y}_X^e \setminus \mathbb{Y'}_X^e) \cup \mathbb{Y'}_X^c.$$

Two extremes exist: - If no predictions are offloaded, then $\mathcal{Y}_X = \mathbb{Y}_X^e$. - If all are offloaded, then $\mathcal{Y}_X = \mathbb{Y}_X^c$.

Let $C(M_e, \mathbb{Y'}_X^c)$ denote the total offloading cost, including transmission and cloud inference. As $|\mathbb{Y'}_X^c|$ increases, so does C. To manage this trade-off, this thesis introduces a parameter $\alpha \in [0, 1]$ to control the balance between accuracy and cost.

4 Proposed Solution: Selective Offloading

This work presents 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 for further processing, the framework aims to minimize both data transmission and cloud model invocation costs, while maximizing detection accuracy.

As illustrated in Figure 4.1, the object detection pipeline is divided into two primary components: object detection on the edge and selective offloading.

4.1 Object Detection on the Edge

The first stage of the system performs object detection entirely on the edge device. This involves two steps: object localization and classification.

Object localization generates bounding boxes that may contain objects. A key challenge at the edge is low recall, where some objects may be missed due to the limited capacity of lightweight models. To mitigate this, the confidence threshold C_{thresh} of the edge model is reduced. This threshold determines the minimum certainty required for a detection to be

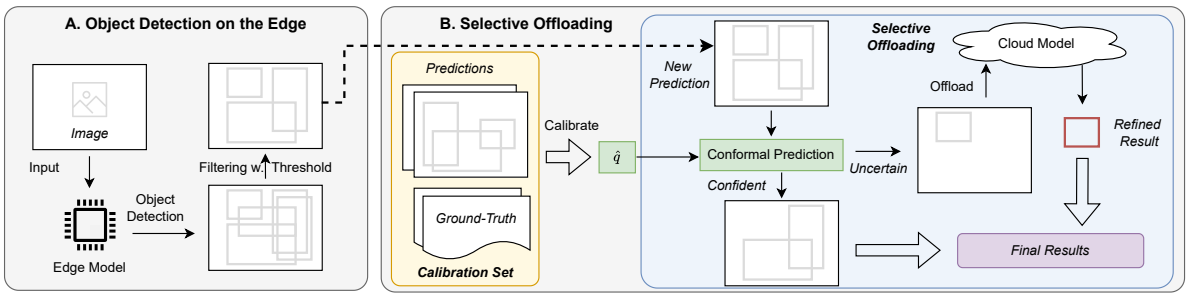


Figure 4.1: System overview: A high-level representation of the selective cloud offloading framework.

reported. Lowering it allows the model to propose more bounding boxes, improving recall 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, this is computed as:

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

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

Bounding boxes are retained if they exceed the confidence threshold:

$$B_{\text{out}} = \{b_i \in B \mid C(b_i) \geq C_{\text{thresh}}\}. \tag{4.1}$$

A lower C_{thresh} yields higher recall but may include noisy detections. These can be filtered during the subsequent classification stage, which leverages uncertainty estimation.

Once candidate object regions are localized, classification is applied to each bounding box to identify the object class. These two steps—localization and classification—can be performed separately [78] or jointly [79] depending on the detection model. Due to limitations of lightweight edge models, this system does not rely solely on edge predictions and instead introduces a selective offloading mechanism.

4.2 Selective Offloading

Given edge predictions, the selective offloading mechanism uses *uncertainty estimation* to decide whether a region should be sent to the cloud for refined detection. It comprises two main components:

(i) uncertainty quantification using conformal prediction, and

(ii) an offloading decision policy based on ambiguity.

4.2.1 Uncertainty Quantification Using Conformal Prediction

The edge model produces confidence scores $Conf(b_i)$ for each class per bounding box. These scores are derived from internal prediction logic and are often poorly calibrated—particularly under domain shift or limited model capacity. Therefore, this work employs conformal prediction to compute a statistically principled uncertainty score, using a reference calibration set.

A *nonconformity score* $\alpha_{b,c}$ is defined for each bounding box b and predicted label $c \in \mathcal{C}$:

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

where $\text{Conf}(b, c) = P_{\text{obj}}(b) \cdot P(c \mid b)$.

Calibration uses labeled detections with known outcomes. For each detection (b, y), where y is the ground-truth class:

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

A threshold $\hat{q}$ is determined as the α-quantile of these calibration scores. A prediction is considered reliable if its nonconformity score is below $\hat{q}$, which corresponds to a statistical coverage of at least $1 - \alpha$.

For a new bounding box b, a class c is included in the *prediction set* if $\alpha_{b,c} \leq \hat{q}$. If the prediction set contains multiple classes, the detection is deemed uncertain and flagged for cloud offloading.

The parameter α allows users to control the offloading behavior: lower α values increase sensitivity to uncertainty (more offloading), while higher values relax this constraint (less offloading).

4.2.2 Offloading Decision Policy

The offloading policy applies a simple rule: if the prediction set contains only one class, the detection is accepted on the edge. If it contains multiple classes, the detection is considered ambiguous and offloaded to the cloud for reclassification.

This rule ensures that cloud resources are used only for resolving uncertain cases, reducing

cost without compromising accuracy.

4.2.3 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.

5 Optimization: Image Stitching

5.1 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 implies 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, cloud costs increase significantly, even though fewer total pixels are offloaded.

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

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

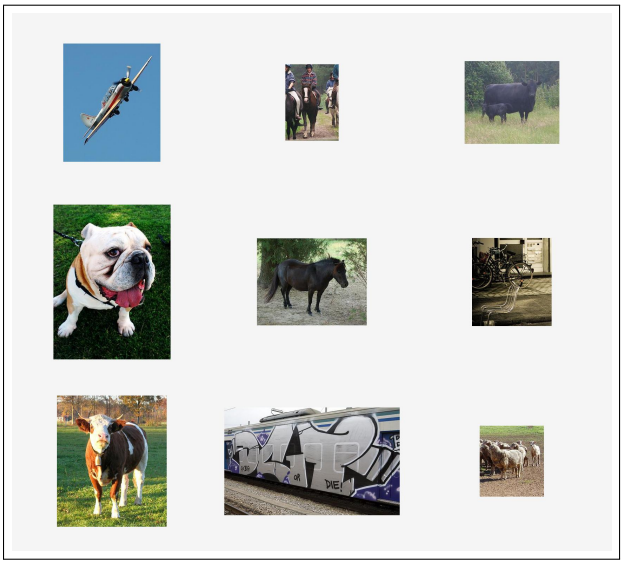
5.2 Approach

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

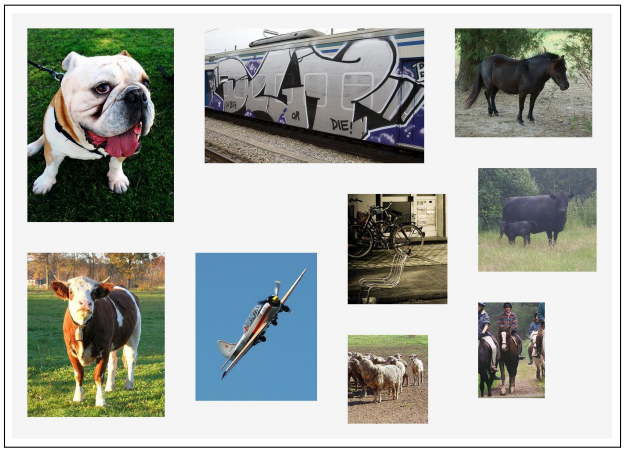
negatively impacts object detection performance. Preliminary experiments show that reducing image resolution adversely affects object detection performance, as models struggle to accurately identify and localize small objects when their features become indistinct due to downscaling.

To mitigate this issue, the approach avoids the constraints of a rigid grid layout. Instead, it aims to preserve the native resolution of bounding boxes while efficiently utilizing space within each packed image. Input images are sequentially processed, and bounding boxes are extracted one by one. Bounding boxes are collected in order until a predefined maximum number 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, the number of images offloaded is reduced 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.

Figure 5.1: 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.

5.2.1 Constraint Handling and Packing Optimization

The packing task is formalized as a constrained two-dimensional (2D) bin packing problem, a classical NP-hard optimization problem [80]. Numerous variations of this problem have been studied, and in this work, we propose a novel variant tailored to our specific requirements. 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 boxes can not be rotated
4. The packed image size, $W \times H$, is minimized,
5. The packed image dimensions align with the input size required by the cloud model

Given the fifth constraint, the packed image is assumed to be square (i.e., $W = H$), as all models require input images with a 1:1 aspect ratio.

Given its NP-hardness, the proposed optimal solutions for this problem is computationally intensive and not suitable for our setting. To address this, we propose an efficient heuristic that combines a binary search over packed image sizes with a MaxRects-based packing strategy [81] using the Best Short Side Fit (BSSF) placement heuristic.

To define the search space, our Packing-Based Image Stitching (PBIS) algorithm 1 computes the smallest possible packed image side length $S_{\min}$ as the largest width or height of any input bounding box, ensuring feasibility. The upper bound $S_{\max}$ is set as the greater of the total widths or total heights of all bounding boxes (lines 1–3):

$$S_{\min} = \max_{r_i \in \mathcal{R}}(\max(w_i, h_i), \quad S_{\max} = \max\left(\sum_{i=1}^{n} w_i,\ \sum_{i=1}^{n} h_i\right)$$

A binary search is then performed between these bounds (Lines 4–11). For each candidate size S, the algorithm invokes the TryPack subroutine to check if all boxes can be packed

without overlap (Line 6). If successful, the upper bound is reduced; otherwise, the lower bound is increased. This continues until the minimal feasible size is found, which is returned as S^* (Line 12).

The TryPack function (Lines 14–29) attempts to pack all bounding boxes into an $S \times S$ canvas. It maintains a list of free regions $\mathcal{F}$ and iteratively places the best-fitting bounding box from the unplaced list $\mathcal{U}$. For each placement (Lines 18–24), it selects the bounding box–position pair with the minimal leftover space, using a greedy strategy that minimizes packing fragmentation. If no placement is possible, it returns **false** (Line 26).

When a box is placed, it breaks the region in multiple region(splitting). Then if among all regions, a region completely contains another that region is pruned out of $\mathcal{F}$ (Line 36). Upon successful placement of all boxes, it returns **true** (Line 39).

This packing algorithm is an essential part of the selective offloading framework. By minimizing the stitched image size while respecting object boundaries, it reduces the number of required cloud API calls while keeping the image resolutions high after resizing done by the cloud model.

The proposed packing algorithm does not offer probabilistic or statistical guarantees of optimality. This is primarily due to the inherent complexity of the 2D bin packing problem, which is NP-hard. Despite extensive research over the past decades, exact methods such as Integer Linear Programming (ILP), Dantzig–Wolfe decomposition, delayed column generation, and branch-and-price remain computationally intensive—often requiring several seconds to process a single image, which is impractical for deployment on edge devices [82, 83].

Although there exist approximation algorithms with formal performance guarantees (e.g., ensuring the solution cost is within a factor α of the optimum), such as Next Fit Decreasing Height, shelf-based methods, and Harmonic algorithms, these typically exhibit high approximation ratios (e.g., α between 2 and 3) [84, 85, 86] and are still slower in practice. In contrast, the MaxRects algorithm with Best Short Side Fit (BSSF), which we adopt, performs better on average and offers a favorable balance between speed and packing efficiency.[81]

Crucially, our framework does not require exact or bounded-optimal solutions and small packing inefficiencies have minimal impact on overall system performance. Instead, fast

Algorithm 1 Packing-Based Image Stitching (PBIS)

Require: Bounding boxes $\mathcal{B} = \{b_1, \ldots, b_n\}$, where each b_i has width w_i, and height h_i
Ensure: Minimum packed image size S^* that fits all bounding boxes

$S_{\min} \leftarrow \max_i(\max(w_i, h_i))$;
$S_{\max} \leftarrow \max\left(\sum_{i=1}^{n} w_i,\ \sum_{i=1}^{n} h_i\right)$;
$low \leftarrow S_{\min},\ high \leftarrow S_{\max}$;
while $low < high$ **do**
    $mid \leftarrow \lfloor (low + high)/2 \rfloor$;
    success $\leftarrow$ TryPack($\mathcal{R}, S = mid$);
    if success then
        $high \leftarrow mid$;
    else
        $low \leftarrow mid + 1$;
    end if
end while
$S^* \leftarrow low$;
return S^*;

function TryPack($\mathcal{R}, S$)
    $\mathcal{F} \leftarrow \{(0, 0, S, S)\}$; ▷ Free space list
    $\mathcal{U} \leftarrow \mathcal{R}$; ▷ Unplaced rectangles
    while $\mathcal{U} \neq \emptyset$ **do**
        $bestScore \leftarrow \infty$, $bestRect \leftarrow$ **None**, $bestPos \leftarrow$ **None**;
        for each $r_i \in \mathcal{U}$ **do**
            for each $f_j \in \mathcal{F}$ **do**
                if r_i fits in f_j **then**
                    Compute leftover width ℓ_w and height ℓ_h;
                    $score \leftarrow \min(\ell_w, \ell_h)$;
                    if $score < bestScore$ **then**
                        $bestScore \leftarrow score$;
                        $bestRect \leftarrow r_i$, $bestPos \leftarrow$ position in f_j;
                    end if
                end if
            end for
        end for
        if $bestRect =$ **None then**
            return false; ▷ Failed to pack all rectangles
        end if
        Place $bestRect$ at $bestPos$;
        Update $\mathcal{F}$ using MAXRECTS splitting and pruning;
        Remove $bestRect$ from $\mathcal{U}$;
    end while
    return true; ▷ All rectangles successfully packed
end function

execution is essential, and our heuristic strategy fulfills that requirement effectively.

6 Experiments

This chapter presents a validation of the proposed method through extensive experiments. The experimental setting is discussed in Section 6.1, followed by an overview of the experimental methodology in Section 6.2. Experimental results are then reported in Sections 6.3 to 6.5.

6.1 Experiment Settings

6.1.1 Datasets

Three popular object detection datasets are used in the evaluation, described as follows:

COCO (Common Objects in Context)[1] [87]: 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] [38]: A classic dataset containing 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][22]: A simplified version of the Open Images V7 dataset[4] is derived by selecting a manageable subset of the data and reducing class granularity. Many classes are merged based on visual similarity, and less frequent or ambiguous classes are removed. This simplification is necessary due to the complexity and

[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`

scale of the original dataset, which proved challenging for the small YOLOv5n model. The resulting OIV7-S maintains semantic richness while enabling effective benchmarking on edge models.

6.1.2 Object Detection Models

As discussed in Section 3.3, the proposed method is model-agnostic and can be integrated with any pair of edge-cloud models, provided that the cloud model demonstrates higher precision than the edge model. To illustrate this, **YOLOv5n** is adopted as the edge model, as it is a highly compact object detector suitable for real-time deployment on edge devices with limited computational resources. In contrast, **YOLOv11x** is used as the cloud model, representing a large-scale model with significantly better performance, deployed on the cloud server to provide refined predictions. The precision and recall of both models are listed in Table 6.1.

Table 6.1: Precision and Recall on COCO dataset

Model	**Precision**	**Recall**
Edge Model	0.745	0.83
Cloud Model	0.855	0.935

6.1.3 Baselines

To assess the effectiveness of the proposed method, five main baselines are used for comparison, detailed as follows:

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

Selective Offloading Methods: Two variants of the selective offloading method are implemented: **SO-P**, which includes both the conformal prediction (Chapter 4) and the packing algorithm (Chapter 5); and **SO-NP**, which includes only the conformal prediction without the packing algorithm.

Random Offloading Methods: In this baseline, random sampling is used instead of conformal prediction to select the same number of bounding boxes as in selective offloading, and these are sent to the cloud without the packing algorithm. Similar to selective offloading, three variants of this method are included: **RO-P**, which applies random offloading with the packing algorithm; and **RO-NP**, which applies random offloading without the packing algorithm.

We did not include certain other baselines, such as Focus[46], and NoScopeP [44] because our framework is designed to be general-purpose and does not rely on video-specific optimizations. Additionally, some other works like Tahoma [42], and VStore [43] use separate stages for digestion and querying which is at odds with our unified, end-to-end design. Our framework is built to work seamlessly across both image and video, whether in real-time or offline settings. Including such baselines would compromise the generality we aim to preserve.

6.1.4 Implementation and Default Parameters

Table 6.2: 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

The algorithm is implemented in Python[5], and the default parameters are listed in Table 6.2.

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

6.2 Evaluation Methodology

6.2.1 Evaluation Metrics

To evaluate the performance of different methods, precision and recall are used, as defined in Section 3.1. 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, offloading cost is considered, as discussed in Section 3.3. A simple cost model is adopted in the experiments, which is computed as the number of images sent to the cloud (i.e., the number of cloud API calls).

6.2.2 Evaluation Objectives

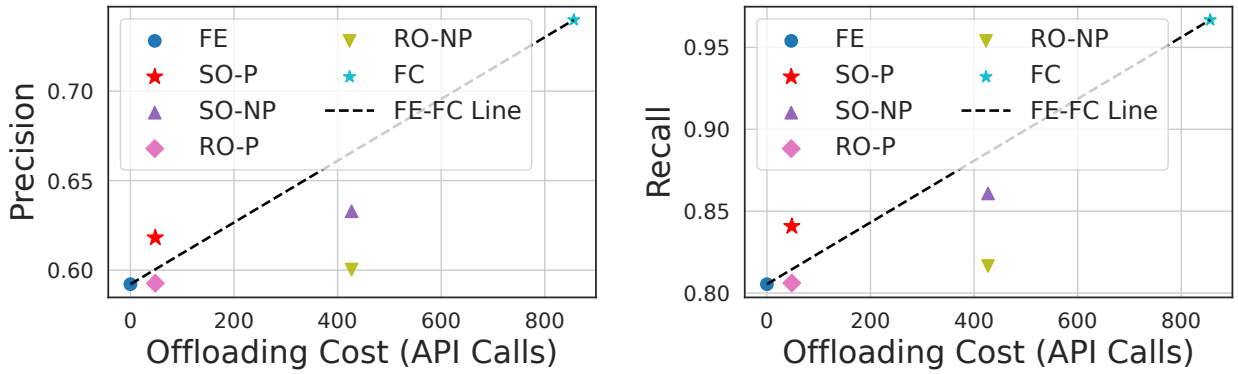
The goals of the experiments are to (a) validate that the method can significantly reduce cloud usage while maintaining high detection performance, and (b) demonstrate that the cost of invoking the cloud model can be controlled with a user-provided parameter.

6.3 Main Result

Overall Result. The results for all methods are shown in Figures 6.1 and 6.2, while the result on OIV7-S dataset shows a similar trend and is omitted for brevity. Take Figure 6.2a as an example, where the position of each method is 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). Preferred methods are 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, the proposed method falls into such an area, over-performing any existing baselines. Specifically, it achieves 0.635 precision while calling the API less than 100 times, which is over 8 times cheaper than the full cloud approach.

Figure 6.2b 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. The aim 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. The 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 the method in maintaining strong performance while significantly reducing computational cost.



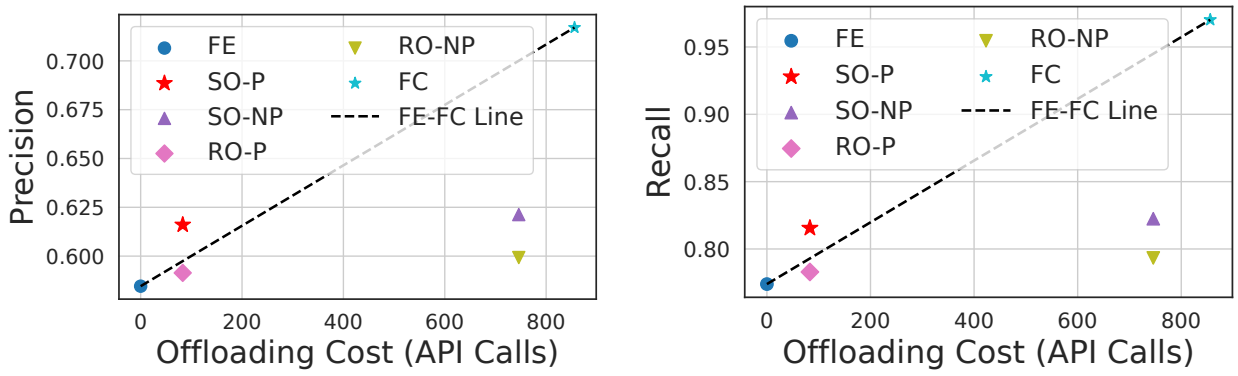
(a) Precision

(b) Recall

Figure 6.1: Different methods on COCO dataset

Varying Offloading Strategies. Selective Offloading is compared with Random Offloading. When the packing algorithm is employed, Selective Offloading (SO-P) consistently achieves higher precision than Random Offloading (RO-P), as observed in Figures 6.1a and 6.2a. Similar trends are also evident in recall, as shown in Figures 6.1b and 6.2b. Overall, Random Sampling methods provide only a minor improvement over the edge-only baseline, whereas Selective Offloading yields substantial gains. This highlights that performance improvement is not solely due to offloading, but also to the informed selection of bounding boxes.

Impact of Packing. As can be consistently observed from Figures 6.1 and 6.2, with a



(a) Precision

(b) Recall

Figure 6.2: Different methods on VOC dataset

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.

6.4 Impact of Parameter Settings

The sensitivity of the framework to several key parameters is analyzed below.

6.4.1 Confidence Threshold on the Edge Model

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 6.3 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.

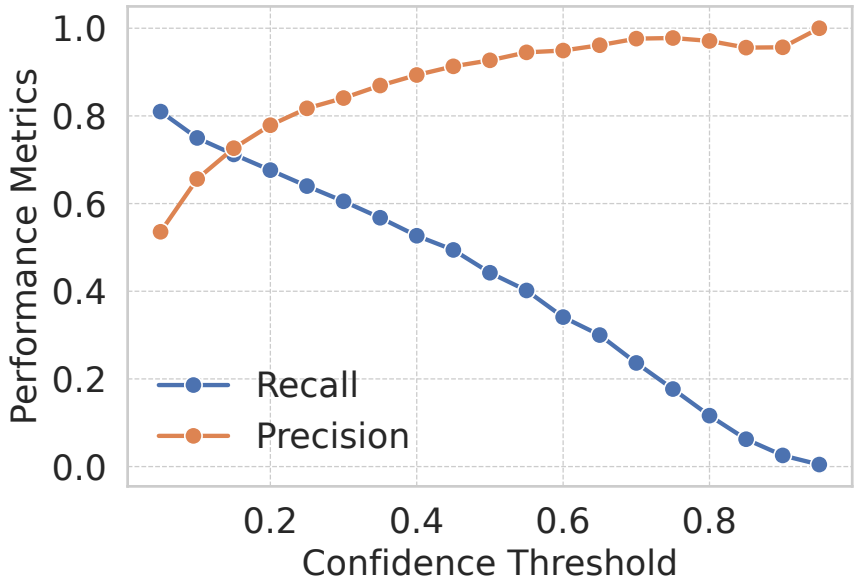


Figure 6.3: Varying C_{thresh} on VOC dataset

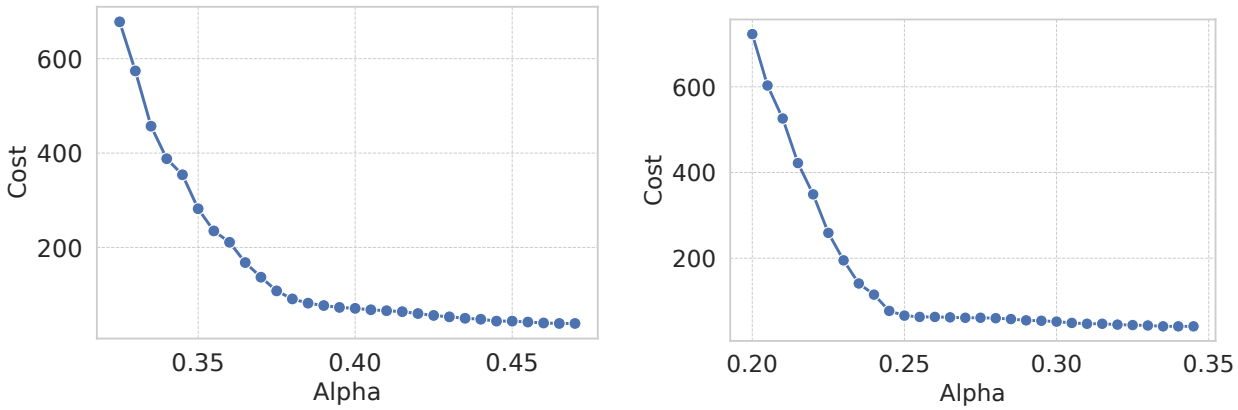
6.4.2 Conformal Prediction Threshold (α)

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.

Figure 6.4 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.

6.4.3 Offloading Cost vs. Performance

As shown in Figure 6.5, increasing the number of cloud API calls improves recall and precision. However, this improvement plateaus beyond a certain point, showing diminishing



(a) COCO (b) VOC

Figure 6.4: α vs Offloading Cost

returns. This observation supports the hypothesis that a small amount of selective offloading yields most of the precision benefits of full cloud processing.

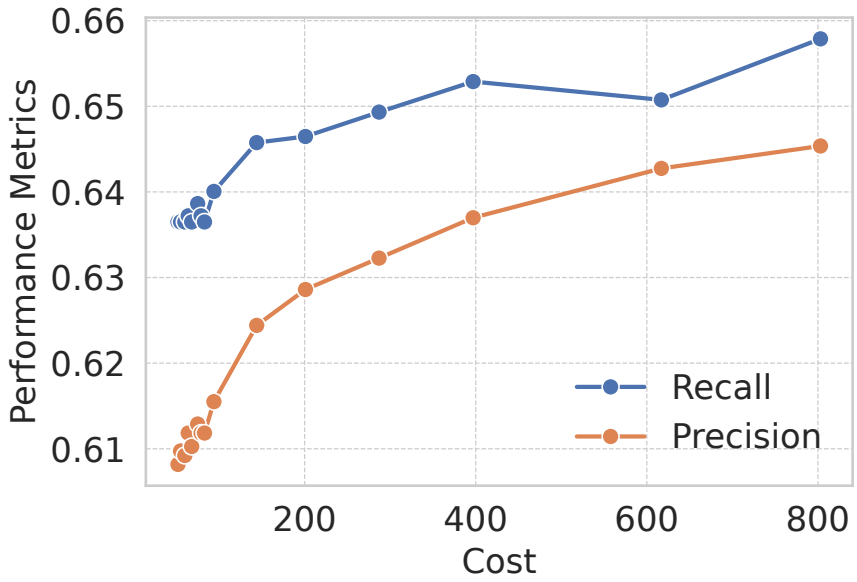


Figure 6.5: Offloading Cost vs Performance (VOC Dataset)

6.4.4 Packing Granularity

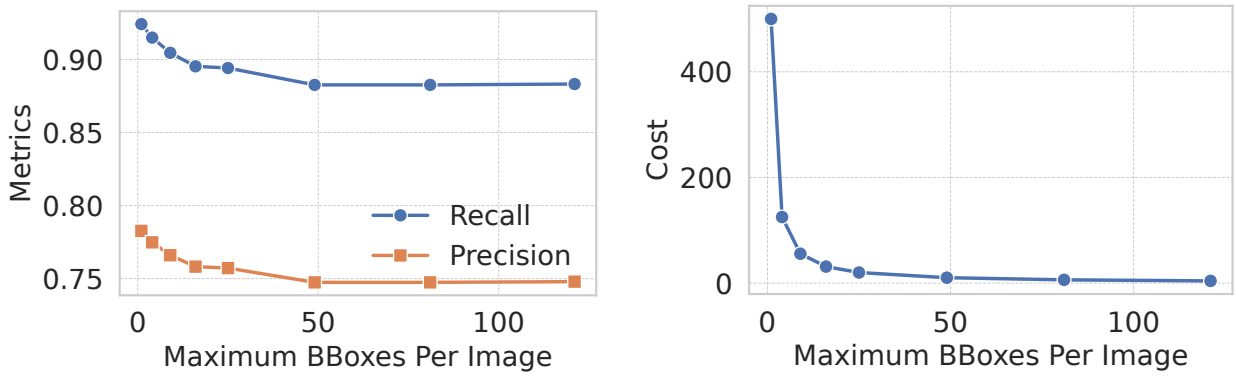
The effect of varying the maximum number of bounding boxes per packed image (m) on both detection performance and cost is shown in Figure 6.6. As the number of bounding boxes increases, both precision and recall exhibit a gradual decline due to reduced spatial

context and resolution available for each region, which negatively impacts detection quality. In contrast, the offloading cost drops significantly, as more regions are processed in a single cloud API call. 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 6.7). 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.

6.5 Summary

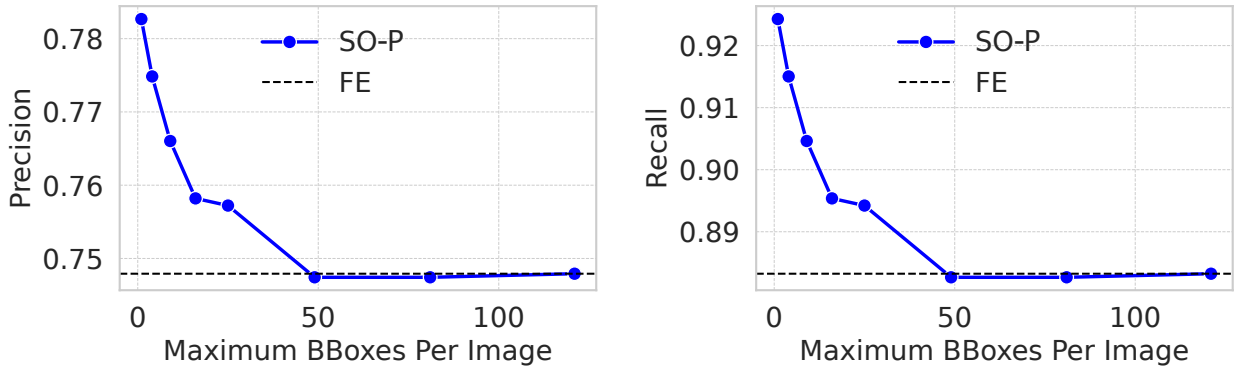
From the above experiments, several key observations can be made: (a) The proposed method achieves approximately 80–90% of the performance of full cloud inference while using only 10–20% of the offloading cost compared to the full cloud model. (b) The Selective Offloading method consistently outperforms Random Offloading across all evaluation metrics. (c) The packing strategy significantly reduces the number of API calls and enhances scalability without compromising precision and recall. (d) The combination of uncertainty estimation and packing provides a robust and cost-effective solution for edge-cloud collaborative inference.



(a) Precision & Recall

(b) Cost

Figure 6.6: Effects of varying the maximum number of bounding boxes per packed image m on performance and cost.



(a) Precision

(b) Recall

Figure 6.7: Effects of varying the maximum number of bounding boxes per packed image m on performance compared to full edge as baseline.

7 Conclusion and Future Work

7.1 Summary of Findings

In this thesis, a *selective cloud offloading framework* was proposed for accurate and cost-efficient object detection on resource-constrained edge devices. By leveraging *conformal prediction* to estimate prediction uncertainty and introducing a *packing-based optimization strategy*, the framework intelligently determines both *what* and *how* to offload to the cloud. This significantly reduces bandwidth usage and monetary costs without compromising detection accuracy, making the solution suitable for deployment in latency-sensitive and bandwidth-constrained environments such as agriculture, smart surveillance, and mobile assistive technologies.

The key contributions of this work are:

- A **selective cloud offloading strategy** based on conformal prediction to assess uncertainty in object detection, allowing for informed and cost-efficient offloading.
- A **packing-based optimization technique** that groups multiple uncertain image regions into single packed images to minimize cloud API calls and associated costs.
- A **user-configurable trade-off mechanism** enabling developers to balance accuracy and cost according to specific application requirements.
- A **comprehensive experimental evaluation** across three real-world datasets, demonstrating that the proposed method achieves ubstantial accuracy improvements with significantly less data compared to existing approaches.

7.2 Future Work

While this thesis demonstrates a promising approach for efficient and accurate edge-cloud object detection, several avenues exist to further extend and enhance the system:

7.2.1 Advanced System Optimization

- *Context-aware offloading:* Incorporating runtime environmental factors such as network bandwidth, cloud queue time, or pricing models to make dynamic, cost-effective offloading decisions.
- *Enhanced uncertainty modeling:* Combining conformal prediction with other uncertainty estimation techniques (e.g. temperature scaling) to improve robustness under domain shift and in low-data scenarios.
- *Smarter region packing:* Developing packing algorithms that consider semantic similarity or visual coherence to co-locate contextually related objects, potentially improving cloud detection performance.

7.2.2 Real-World Validation and Deployment

- *Agricultural monitoring:* Field trials for detecting diseases or pests in crops using drones or smartphones.
- *Smart surveillance:* Application in low-bandwidth security camera systems in rural or infrastructure-constrained environments.
- *Assistive technologies:* Deployment on mobile or wearable platforms for visually impaired users.

These use cases will help evaluate latency, energy consumption, and deployment complexity in practical settings.

7.2.3 Scalability and Interoperability

- *Multi-device coordination:* Enabling distributed offloading across a network of edge nodes (e.g., drones or sensors) with shared resources and collaborative bandwidth management.
- *Model-agnostic integration:* Extending compatibility with a variety of object detectors and commercial cloud APIs to enhance system generality and ease of adoption.

In summary, this thesis lays the foundation for an intelligent, adaptive object detection framework that is practical for real-world, resource-constrained environments. Continued improvements in uncertainty estimation, system-level integration, and domain-specific testing will further strengthen the viability and impact of selective cloud offloading.

Bibliography

[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)*, pp. 629–638, 2024.

[2] S. Wang and team, "Melon ripeness detection by an improved object detection algorithm (mrd-yolo)," *Plant Methods*, 2024.

[3] Y. Chan *et al.*, "Smart iot-based pest detection platform for integrated pest management," *Scientific Reports*, 2024.

[4] L. Tepelea, I. Gavrilut, and A. Gacsadi, "Smartphone application to assist visually impaired people," in *2017 14th International Conference on Engineering of Modern Electric Systems (EMES)*, pp. 228–231, 2017.

[5] 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*, pp. 444–451, 2013.

[6] P. Cappellini, C. Colombo, G. Gualdi, and S. Mattoccia, "Vision-based mobile indoor assistive navigation aid for blind people," *IEEE Transactions on Mobile Computing*, vol. 17, no. 3, pp. 622–636, 2018.

[7] K.-Y. Huang, C.-L. Chung, and J.-L. Xu, "Deep learning object detection-based early detection of lung cancer," *Frontiers in Medicine*, vol. 12, 04 2025.

[8] R.-A. Bratulescu, R.-I. Vatasoiu, G. Sucic, S.-A. Mitroi, M.-C. Vochin, and M.-A. Sachian, "Object detection in autonomous vehicles," in *2022 25th International Symposium on Wireless Personal Multimedia Communications (WPMC)*, pp. 375–380, 2022.

[9] H.-T. Duong, V.-T. Le, and V. T. Hoang, "Deep learning-based anomaly detection in video surveillance: A survey," *Sensors*, vol. 23, no. 11, 2023.

[10] 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.

[11] 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)*, pp. 3007–3012, IEEE, 2018.

[12] 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.

[13] 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)*, pp. 1–6, 2021.

[14] 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, p. 1765–1778, Dec. 2022.

[15] C. Xu, Y. N. Wu, and J. Z. Kolter, "Conformal prediction: A unified review of theory and applications," *Foundations and Trends in Machine Learning*, vol. 17, no. 1, pp. 1–114, 2024.

[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* (A. Gammerman, V. Vovk, Z. Luo, and E. Smirnov, eds.), vol. 105 of *Proceedings of Machine Learning Research*, pp. 123–141, PMLR, 09–11 Sep 2019.

[17] H. Boström, U. Johansson, and H. Linusson, "A tutorial on conformal prediction," *International Journal of Machine Learning and Cybernetics*, vol. 12, no. 1, pp. 3–22, 2021.

[18] A. Wang, H. Chen, L. Liu, K. Chen, Z. Lin, J. Han, and G. Ding, "Yolov10: Real-time end-to-end object detection," *arXiv preprint arXiv:2405.14458*, 2024. Improves efficiency and accuracy; NMS-free training; outperforms RT-DETR and YOLOv9.

[19] A. Gu and T. Dao, "Mamba: Linear-time sequence modeling with selective state spaces," *arXiv preprint arXiv:2312.XXXX*, 2023. Proposes S4-based Mamba model to handle very long sequences more efficiently than transformers.

[20] Y. Tian, Q. Ye, and D. Doermann, "Yolov12: Attention-centric real-time object detectors," *arXiv preprint arXiv:2502.12524*, 2025. Incorporates efficient attention modules into YOLO, surpassing YOLOv10/v11 and RT-DETR in both speed and accuracy.

[21] L. Cai, Z. Zhang, Y. Zhu, L. Zhang, M. Li, and X. Xue, "Bigdetection: A large-scale benchmark for improved object detector pre-training," *arXiv preprint arXiv:2203.13249*, 2022.

[22] 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.

[23] F. Zhou, P. Narayanan, L. Schmidt, and E. K. Johnson, "Counts: A large-scale dataset for fine-grained object detection and grounding under distribution shifts," *arXiv preprint arXiv:2504.10158*, 2025. First large detection dataset emphasizing fine-grained labels and OOD robustness across distribution shifts.

[24] H. Lu, B. Dong, B. Zhu, S. Ma, Z. Zhang, J. Peng, and K. Song, "A systematic review of deep learning-based object detection for crop monitoring: Pest, yield, weed, and growth applications," *The Visual Computer*, vol. 41, pp. 57–91, 2025. Comprehensive review of

DL-based object detection in precision agriculture, covering YOLOv9, v10, Transformer models, and task-specific challenges.

[25] H. Zhao, S. Zhang, X. Peng, Z. Lu, and G. Li, "Improved object detection method for autonomous driving based on detr with multi-scale feature fusion and axial attention," *Frontiers in Neurorobotics*, vol. 18, p. 1484276, 2025. Introduces axial-attention enhanced DETR and dynamic hyperparameter tuning, validated on autonomous driving datasets.

[26] W. Chen, J. Yan, W. Huang, *et al.*, "Robust object detection for autonomous driving based on semi-supervised learning," *Security and Safety*, vol. 3, p. 2024002, 2024. Uses MoCo-based pretraining, semi-supervised co-training, and BBAug for KITTI, enhancing robustness under adversarial and weather corruptions.

[27] J. Sapna P. S. Saba, and S. C., "Object detection for video surveillance using edge–cloud collaboration," in *2023 1st International Conference on Circuits, Power and Intelligent Systems (CCPIS)*, 2023. Implements a YOLOv3-based edge–cloud detection pipeline on Raspberry Pi, achieving real-time accident detection and 94

[28] J. Sapna P. S. Saba, and S. Chavan, "Object detection for video surveillance using edge–cloud collaboration," in *2023 International Conference on Circuits, Power and Intelligent Systems (CCPIS)*, 2023. Demonstrates YOLOv3 on Raspberry Pi for accident detection in smart surveillance, saving 94

[29] Cisco, "Cisco annual internet report (2018–2023) white paper." `https://www.cisco.com/c/en/us/solutions/executive-perspectives/annual-internet-report/index.html`, 2022. Accessed: 2025-06-13.

[30] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2015.

[31] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European Conference on Computer Vision (ECCV)*, 2016.

[32] G. Jocher, A. Chaurasia, TaoXie, *et al.*, "Yolov5." `https://github.com/ultralytics/yolov5`, 2020. yolov5.

[33] M. V. A. Team, "Yolov6: A single-stage object detection framework for industrial applications." `https://github.com/meituan/YOLOv6`, 2022.

[34] G. Jocher *et al.*, "Yolov8: Sota real-time object detection." `https://github.com/ultralytics/ultralytics`, 2023.

[35] R. Sapkota, R. H. Cheppally, A. Sharda, and M. Karkee, "Rf-detr object detection vs yolov12 for greenfruit detection," in *arXiv preprint arXiv:2504.13099*, 2025. Benchmarking RF-DETR and YOLOv12 on occluded agricultural data; RF-DETR excels in accuracy.

[36] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *European Conference on Computer Vision (ECCV)*, 2020.

[37] 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 *European Conference on Computer Vision (ECCV)*, pp. 740–755, 2014.

[38] 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.

[39] Anonymous, "The evolution of object detection methods," *Pattern Recognition Letters*, 2024. Reports Faster R-CNN achieves 5 FPS.

[40] Q. Chen, X. Su, X. Zhang, J. Wang, J. Chen, Y. Shen, C. Han, Z. Chen, W. Xu, F. Li, S. Zhang, K. Yao, E. Ding, G. Zhang, and J. Wang, "Lw-detr: A transformer

replacement to yolo for real-time detection," *arXiv preprint arXiv:2406.03459*, 2024. Lightweight DETR variant outperforming YOLO variants in speed and accuracy on COCO.

[41] S. Wang, C. Xia, F. Lv, and Y. Shi, "Rt-detrv3: Real-time end-to-end object detection with hierarchical dense positive supervision," in *arXiv preprint arXiv:2409.08475*, 2024. Improves real-time DETR with dense supervision; surpasses YOLOv10-X and earlier RT-DETR.

[42] S. Krishnan, E. Wang, E. Wu, K. Goldberg, J. E. Gonzalez, and I. Stoica, "The case for learned index structures for visual data," in *CIDR 2020*, 2020. Conference on Innovative Data Systems Research.

[43] Z. Zhu, Q. Zhang, W. Wu, F. Zhang, Z. Xu, L. Zhou, and B. Zang, "Vstore: A data store for analytics on large videos," in *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pp. 35–50, 2018.

[44] D. Kang, J. Emmons, F. Abuzaid, P. Bailis, and M. Zaharia, "Noscope: Optimizing neural network queries over video at scale," in *Proceedings of the VLDB Endowment*, vol. 10, pp. 1586–1597, 2017.

[45] D. Kang, J. Emmons, F. Abuzaid, P. Bailis, and M. Zaharia, "Noscope: Optimizing neural network queries over video at scale," in *Proceedings of the VLDB Endowment*, vol. 12, pp. 397–410, 2019.

[46] D. Kang, P. Bailis, and M. Zaharia, "Blazeit: Optimizing declarative aggregation and limit queries for neural network-based video analytics," in *SIGMOD '20: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pp. 515–530, 2020.

[47] K. Hsieh, G. Ananthanarayanan, P. Bodik, P. Bahl, and M. Philipose, "Focus: Querying large video datasets with low latency and low cost," in *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2020.

[48] D. Kang, P. Bailis, and M. Zaharia, "Svq++: Fast dynamic video querying with statistical guarantees," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 521–534, 2020.

[49] Y. Liu, M. Peng, G. Shou, and S. Chen, "Edge computing and cloud computing for internet of things: A review," *Future Internet*, 2020. Highlights edge computing benefits for IoT due to latency, bandwidth, and privacy constraints.

[50] S. Liu, J. Tang, and W. Shi, "Edge computing for autonomous driving: Opportunities and challenges," in *Proceedings of the IEEE*, vol. 107, pp. 1697–1716, 2019. Details real-time sensor processing and safety considerations in autonomous vehicles at the edge.

[51] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[52] P. Garcia Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Rivière, "Edge computing: A comprehensive survey of current initiatives, challenges and opportunities," *ACM Computing Surveys (CSUR)*, vol. 52, no. 1, pp. 1–34, 2019.

[53] R. Roman, J. Lopez, and M. Mambo, "Mobile edge computing, fog et al.: A survey and analysis of security threats and challenges," *Future Generation Computer Systems*, vol. 78, pp. 680–698, 2018.

[54] A. Yousefpour, G. Ishigaki, R. Gour, H. Ishii, and J. Ren, "All one needs to know about fog computing and related edge computing paradigms: A complete survey," *Journal of Systems Architecture*, vol. 98, pp. 289–330, 2019.

[55] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[56] Y. Yang, R. Deng, X. Yang, J. Ding, and S. Wang, "A survey of computation offloading for mobile-edge computing," *IEEE Access*, vol. 8, pp. 197372–197393, 2020.

[57] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: Making smartphones last longer with code offload," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pp. 49–62, 2010.

[58] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *Proceedings of the sixth conference on Computer systems*, pp. 301–314, 2011.

[59] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel execution in cloud for mobile code offloading," *INFOCOM, 2012 Proceedings IEEE*, pp. 945–953, 2012.

[60] Y. Mao, J. Zhang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[61] A. Brogi, S. Forti, and M.-H. Ibrahim, "Deploying fog computing applications: An approach based on application requirements and resource constraints," in *International Conference on Service-Oriented Computing*, pp. 285–292, Springer, 2017.

[62] A. Diro and N. Chilamkurti, "Dynamic resource management in edge computing using reinforcement learning," in *2018 14th International Conference on Network and Service Management (CNSM)*, pp. 187–193, IEEE, 2018.

[63] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On multi-access edge computing: A survey of the emerging 5g network edge cloud architecture and orchestration," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1657–1681, 2017.

[64] Q. Zhang and N. Ansari, "Dynamic service placement in geographically distributed clouds," in *IEEE ICC*, pp. 4846–4850, IEEE, 2013.

[65] X. Xu, Q. Liu, and K. Fan, "A deep reinforcement learning-based scheduling algorithm for edge services," *IEEE Access*, vol. 8, pp. 87064–87073, 2020.

[66] J. Oueis, A. Sharma, and D. Gündüz, "Performance evaluation of mobile edge computing with practical offloading scenarios," in *2015 IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 1422–1427, IEEE, 2015.

[67] S. Wang, J. Zhang, and A. Zhou, "Energy-efficient task offloading and resource allocation for mobile edge computing," *IEEE Access*, vol. 7, pp. 155760–155771, 2019.

[68] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450–465, 2018.

[69] K. Gai, M. Qiu, L. Tao, Y. Zhu, and Z. Zong, "Privacy-preserving smart metering with regional statistics and bounded error," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 5, pp. 2693–2703, 2016.

[70] V. Costan and S. Devadas, "Intel sgx explained," *IACR Cryptol. ePrint Arch.*, vol. 2016, p. 86, 2016.

[71] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, 2020.

[72] J. Li, Y. Li, M. Chen, and Z. Wang, "Efficient fault-tolerant edge computing with task replication," *IEEE Access*, vol. 6, pp. 77766–77777, 2018.

[73] S. Yi, C. Li, and Q. Li, "A survey of fog computing: Concepts, applications and issues," in *Proceedings of the 2015 Workshop on Mobile Big Data*, pp. 37–42, 2015.

[74] Y. Tang, Z. Liu, X. Zhou, and X. Fu, "Dynamic checkpointing and rollback recovery for cloud computing," *IEEE Transactions on Services Computing*, vol. 11, no. 6, pp. 965–978, 2017.

[75] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," *USENIX Annual Technical Conference*, vol. 2014, pp. 305–319, 2014.

[76] 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, p. 303–338, June 2010.

[77] R. Padilla, S. L. Netto, and E. A. B. da Silva, "A survey on performance metrics for object-detection algorithms," in *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*, pp. 237–242, IEEE, 2020.

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

[79] 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*, pp. 779–788, 2016.

[80] A. A. Pandey, "An analysis of solutions to the 2d bin packing problem and additional complexities," *International Journal of Mathematics and its Applications*, vol. 9, no. 3, pp. 111–118, 2021.

[81] J. Jylänki, "A thousand ways to pack the bin: A practical approach to two-dimensional rectangle bin packing," tech. rep., Independent Researcher, February 2010. Accessed July 25, 2025.

[82] D. Pisinger and M. Sigurd, "The two-dimensional bin packing problem with variable bin sizes and costs," *Discrete Optimization*, vol. 2, no. 2, pp. 154–167, 2005.

[83] L. Wei, M. Lai, A. Lim, and Q. Hu, "A branch-and-price algorithm for the two-dimensional vector packing problem," *European Journal of Operational Research*, vol. 281, no. 1, pp. 25–35, 2020.

[84] N. Ntene and J. van Vuuren, "A survey and comparison of guillotine heuristics for the 2d oriented offline strip packing problem," *Discrete Optimization*, vol. 6, no. 2, pp. 174–188, 2009.

[85] B. S. Baker and J. S. Schwarz, "Shelf algorithms for two-dimensional packing problems," *SIAM Journal on Computing*, vol. 12, no. 3, pp. 508–525, 1983.

[86] X. Han, F. Y. L. Chin, H.-F. Ting, and G. Zhang, "A new upper bound on 2d online bin packing," 2009.

[87] 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*, pp. 740–755, Springer, 2014.