LiteReconfig: Cost and Content Aware Reconfiguration of Video Object Detection Systems for Mobile GPUs

Anonymous Author(s)
Submission Id: <260>

Abstract

An adaptive video object detection system selects different execution paths at runtime, based on a user specified latency requirement, video content characteristics, and available resources on a platform, so as to maximize its accuracy under the target latency service level agreement (SLA). Such a system is well suited for mobile devices with limited computing resources, often times running multiple contending applications. In spite of several recent efforts, we show that existing solutions suffer from two major drawbacks when facing a tight latency requirement (*e.g.,* 30 fps). *First*, it can be very expensive to collect some feature values for a scheduler to decide on the best execution branch to run. *Second*, the system suffers from the switching overhead of transitioning from one branch to another, which is variable depending on the transition pair. This paper addresses these challenges and presents *LiteReconfig*, an efficient and adaptive video object detection framework for mobiles. Underlying *LiteReconfig* is a cost-benefit analyzer for the scheduler that decides which features to use and then which execution branch to run at inference time. *LiteReconfig* is further equipped with a content-aware accuracy prediction model to select an execution branch tailored for frames in a streaming video. With a large-scale real-world video dataset and two leading edge embedded devices with mobile GPUs, we demonstrate that *LiteReconfig* achieves significantly better accuracy under a set of varying latency requirements when compared to existing adaptive object detection systems, while running at speeds of up to 50 fps on an NVIDIA AGX Xavier board.

1 Introduction

Video object detection on mobiles has attracted considerable attention in recent years. Much progress has been made in developing light-weight models and systems that are capable of running on mobile devices with moderate computation capability [15, 19, 55, 62, 72]. The majority of previous works focused on statically optimized models and systems [5, 39, 56], pushing the frontiers of accuracy and efficiency. More recently, adaptive object detection models and systems [6, 11, 12, 23, 65, 66] have emerged. These approaches are capable of running at multiple accuracy and latency trade-offs, and thus are better suited for mobile devices running under real-world conditions. Real-world conditions include adapting to dynamically changing content characteristics, resource availability on the device, and user requirements. An adaptive vision system consists of two key components: (1) a multi-branch execution kernel **(MBEK)** with multiple execution branches each achieving an operating point in the accuracy-latency axes, and (2) a scheduler to decide which branch to use at runtime, based on video features and user requirements. Despite recent advances in adaptive object detection, no previous works consider the competing latency requirement between the two to meet an end-to-end accuracy-latency goal. Thus, maximizing the accuracy in adaptive vision systems, considering the latency cost of the scheduler itself, remains an unaddressed problem.

A major shortcoming of current adaptive vision systems is the competing demands between the MBEK and the scheduler. Importantly, previous works face two fundamental challenges. *First*, the system scheduler relies on the computationally light video features, *e.g.,* height, width, number of objects, intermediate results of the execution kernel, to make the decision on which execution branch to run. Such features might not be sufficiently informative to represent the video content. On the other hand, computationally heavier content features or models, such as motion and appearance features of the video, can improve the decision making but are typically too heavyweight to include in the decision-making process. For example, the extraction of a high-dimensional Histogram of Oriented Gradient (HOG) feature and executing the models built on such feature takes 30.25 msec (Table 1), nearly the time of one video frame, on a leading-edge mobile GPU device, the Tegra TX2. *Second*, the scheduler incurs high switching overhead due to frequent reconfiguration among execution branches if the conditions change frequently. Thus, a cost-aware scheduler needs to damp the frequency of reconfigurations based on the cost of each, which can vary depending on the execution branch. No prior work has provided a cost-aware design of the scheduler and this fundamentally leads to their sub-optimal performance.

To address these challenges, we develop *LiteReconfig*, which is tailored to embedded boards with mobile GPUs on them. At its heart, LiteReconfig provides a cost-benefit analysis to decide at any point in a video stream which execution branch to select. A schematic of LiteReconfig is shown in Figure 1. The cost-benefit analyzer factors in the cost (as latency in our use case) and the benefit (in terms of accuracy improvement) of using computationally heavy content features in its models. By wisely enabling content features and

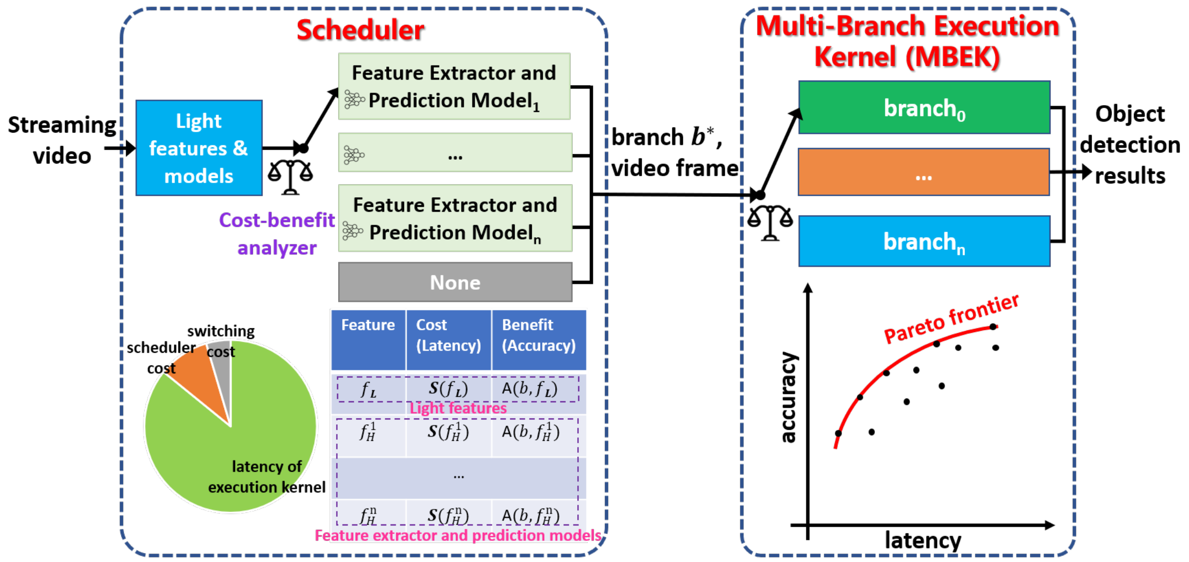

Scheduler
Streaming video
Light features & models
Cost-benefit analyzer
Feature Extractor and Prediction Model$_1$
...
Feature Extractor and Prediction Model$_n$
None
switching cost
scheduler cost
latency of execution kernel
Feature
Cost (Latency)
Benefit (Accuracy)
f_L $S(f_L)$ $A(b, f_L)$
Light features
f_H^1 $S(f_H^1)$ $A(b, f_H^1)$
...
f_H^n $S(f_H^n)$ $A(b, f_H^n)$
Feature extractor and prediction models
branch b^*, video frame
Multi-Branch Execution Kernel (MBEK)
branch$_0$
...
branch$_n$
Object detection results
Pareto frontier
accuracy
latency

Figure 1: An illustration of our proposed cost-aware adaptive framework for video object detection. Our scheduler uses its cost-benefit analysis to decide which features to use for making a decision and then makes a decision on which execution branch to run for detection. The multi-branch execution kernel (MBEK) can be provided by any adaptive vision algorithm for mobiles and we build on top of several mainstream object detection and tracking algorithms.

models, the system characterizes the accuracy of the MBEK in a *content-aware manner* so as to select a more accurate branch tailored to the content in the streaming video. Further, LiteReconfig analyzes the cost and benefit of switching to a new execution branch when runtime conditions change, versus staying with the current one. For example, the motivating Figure 2 shows the accuracy vs latency curve (higher is better). Here we see that content agnostic is worse than one of the content-aware strategies (ResNet). More subtly, there is a content-aware option (MobileNet) that is worse than content agnostic, indicating the need for a rational decision on which content features to include. Here, the ResNet50 features come from the object detector in the MBEK and only additional extraction and model prediction cost are incurred, making it a winning option. Through careful design, we ensure that the overhead of using a content feature extractor and the corresponding model is minimal so as not to erase any gain from the optimization.

We evaluate our approach on the ImageNet VID 2015 benchmark and compare with our adaptations of SSD [40] or YOLOv3 [47] where we enhance their efficiency and adaptability by incorporating some tuning knobs so to run different points in the latency-accuracy spectrum (*e.g.*, the size of the video frame). We also compare to a recent adaptive model [66] with the Faster R-CNN backbone. The evaluation is done on two embedded boards with mobile GPUs — Jetson TX2 and Jetson AGX Xavier. We show that LiteReconfig improves the accuracy by 1.8% to 3.5% mean average precision (mAP) over the state-of-the-art (SOTA) adaptive object detection systems, without affecting the latency. Under contention for the GPU resource, the SSD and YOLOv3 baselines completely fail to meet the latency requirement. Compared to three recent accuracy-focused object detection solutions, SELSA [63], MEGA [5], and REPP [51], LiteReconfig is 74.9X, 30.5X, and 20.3X faster on the Jetson TX2 board.

Contributions. We summarize our contributions as follows.

1. We develop a cost-benefit analyzer to enable low-cost online reconfiguration of the efficient and adaptive object detection framework. This optimization largely reduces the scheduler cost of the system and increases the accuracy since more of the latency budget can be used for the execution of the object detection kernel.
2. We develop a content-aware accuracy prediction model on the execution branch so that our scheduler selects a branch tailored to the content in the streaming video. Such a model is built on computationally heavy features and *cannot* be used without our cost-benefit analysis.
3. Through extensive experimental evaluation on two mobile GPU boards and against a large set of existing solutions, we provide two key insights applicable to lightweight adaptive computer vision systems (i) it is important to consider the effect of contention due to co-located applications, and (ii) it is important to engineer which features to use for making the selection of the execution branch. The full implementation of LiteReconfig is able to meet even stringent latency requirement, 30 fps on the weaker board TX2 and 50 fps on the stronger board AGX Xavier.

2 Preliminaries and Motivation

We now provide some background on video object detection algorithms, content-aware video object detection models, and adaptive vision systems.

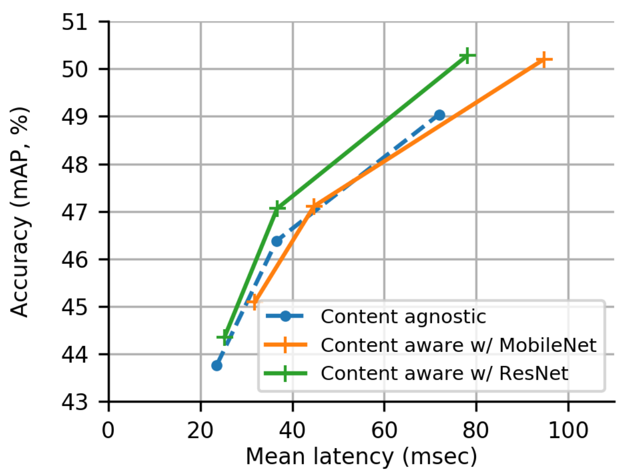


Figure 2: **Motivation of cost-benefit analysis**. We plot the accuracy vs. latency curve for three different strategies. Without a careful design, a content-aware strategy can be either better (*e.g.*, ResNet) or worse (*e.g.*, MobileNet) than a content-agnostic one. Here, the ResNet50 features come from the object detector with a lower cost than using an external MobileNet, making it a winning option.

2.1 Video Object Detection Algorithms

As a key problem in computer vision, object detection seeks to locate object instances in an image or video frame using bounding boxes and simultaneously classifying the instance into pre-defined categories. The most widely used detection models adopt convolutional neural networks (CNNs), and can be separated into two parts: a backbone network that extracts features from images and a detection network that classifies object regions based on the extracted features. *For the reader wishing to get into our contribution, the design of LiteReconfig, she may skip the rest of the background in this subsection.* This is relevant to a reader wishing to understand some pertinent details of video object detection.

The detection network can be further categorized into two-stage [9, 17, 48] or single-stage detectors [36, 40, 47, 56, 71]. One representative work of two-stage detectors is Faster R-CNN [48], which employs CNNs to extract image feature maps and feeds it into Region Proposal Networks (RPN) to generate regions-of-interest (RoIs) in the first stage. Then, in the second stage, the RoI pooling layer combines the feature maps from convolutional layers, and the proposals from the RPN, together, to generate positive proposal feature maps, which is then provided to the classifier network. In contrast, single-stage object detection solutions do not include the step of region proposal generation and directly classify a dense set of pre-defined regions. These models are optimized for efficiency. For example, YOLO [46] is a well-known one-stage network that simplifies the detection of objects as a regression problem by predicting bounding boxes and class probabilities directly without generating region proposals. The EfficientDet [56] family is a group of one-stage detectors. It includes nine models with various input image sizes and network depths, all of which achieve SOTA performance, measured in FLOPs.

While single-image object detectors can be applied to videos frame-by-frame, this method ignores the reality that adjacent frames have redundancies. This temporal continuity in videos and can be leveraged to approximate the computations or to enhance detection results in neighboring frames. This has motivated this research direction in video object detection systems [13, 28, 38, 60, 68, 75, 76]. Many previous approaches optimize for accuracy, explore temporal aggregation of object features [68], using either recurrent neural networks [38], or motion analysis [60]. More practical solutions integrate object detection with visual tracking [13, 37, 66, 70], where inexpensive trackers are used to connect costly object detection outputs. In this tracking-by-detection technique [24, 75], which has become a *de facto* strategy for real-time video analysis, an execution plan performs object detection, on the current frame, and object tracking on the following frames, in a Group-of-Frames (GoF). This combination of object detection and visual tracking into a video analytic system exposes additional design choices, such as the selection of the tracker and the triggering interval of detection, interleaved by tracking.

2.2 Content-Aware Video Object Detection Models

As has been discussed in Sec. 2.1, videos come with inherent information within a series of continuous frames. For example, the scale of objects, the moving speed of objects, the complexity of objects, etc. Based on these characteristics, some video object detection models [6, 38, 66] utilize the content information in videos, targeting improved latency and accuracy performance. We call such models as content-aware video object detection systems. During the model inference time, a content-aware video object system is able to reconfigure itself based on the content information from the streaming video while a content-agnostic system uses a static model variant or branch. AdaScale [6] is an example that makes use of the content information of videos to dynamically re-scale the images to achieve better performance. Another example is Liu *et al.*'s work [38] that employed Long Short-Term Memory (LSTM) layers to propagate the temporal context of each video frame to assist with the detection. ApproxDet [66] is a recent work that adapts in both dimensions of content and contention by changing the parameters of Faster R-CNN and object trackers, including the input image resolution, the number of object proposals, the detection interval, and the choice of trackers.

However, an efficient object detection system that is capable of reconfiguration at runtime faces two challenges: (1) Lack of content-rich features and fine-grained accuracy prediction. Insufficient feature extraction and inaccurate prediction before the reconfiguration could worsen performance. (2) Lack of cost-aware design. The system reconfiguration overhead (cost) is not considered when a decision is made.

This may degrade overall performance if the reconfiguration cost is high. To the best of our understanding, no prior video object detection work solves either of these two challenges.

2.3 Adaptive Vision Systems

Despite the fact that video object detection algorithms perform well in terms of accuracy and latency on server-class machines, existing solutions suffer when running on edge or mobile devices, particularly with a tight latency service level agreement (SLA) and under varying resource contentions. There has been significant work on developing continuous vision applications on mobile or resource-constrained devices — some with human-designed deep models [15, 19, 72] and some with models given by neural architecture search [35, 55, 62]. Examples include deep models that tune the size of the input or other model parameters [6, 23, 65, 66] at inference time, prune a static DNN into multiple DNNs that could be dynamically selected [12], or select a different exit within a network [20, 57].

These adaptive video object detection frameworks usually feature multi-models or multi-branches as part of their design. In the meantime, in addition to the multi-branch backbone, this architecture includes a scheduler component that supports switching among models or branches. For such multi-model or multi-branch frameworks, we call the various combinations of object detectors and trackers as the MBEK, and the upper layer scheduling part, scheduler. In real applications, considering the changing video content and available computational resources, the requirement for switching between execution kernels may be frequent, with a concomitant switching overhead. The uncertainty of performance after the switching makes it hard for the system to maintain consistent latency and accuracy performance at runtime. Thus, the scheduler needs to be cost- and content-aware, and lightweight, to minimize the overhead. This is where prior work is lacking. *For the reader wishing to get into our contribution, the design of LiteReconfig, she may skip the rest of the background in this subsection.* It is relevant, however, to a reader wishing to understand some further details of adaptive vision systems.

Most of existing approaches consider image or video classification tasks. Rather than predicting a single label per frame or video clip, video object detection has to examine every frame and output a varying number of object boxes per frame. Therefore, the design of an adaptive video object detection system poses additional challenges compared to an adaptive classification system. Only a few video object detection solutions exist to date [6, 66].

Some other adaptive vision systems [22, 44, 70] also leverage a multi-branch design methodology. For example, VideoStorm [70] considers two key characteristics of video analytics: resource-quality tradeoff with multi-dimensional configurations. Mainstream [22] automatically determines at deployment time the right tradeoff between more specialized DNNs to improve accuracy, and retaining more of the unspecialized base model to increase (model) sharing. The latter reduces the per-frame latency. DeepDecision [44] designs a framework that ties together front-end devices with more powerful backend servers to allow deep learning to be executed locally or remotely in the cloud/edge.

3 Design of LiteReconfig

3.1 Approach Overview

The workflow of our system LiteReconfig is presented in Figure 1. LiteReconfig uses MBEK with multiple execution branches to meet different latency-accuracy trade-offs or to handle dynamic runtime conditions such as content changes. It can execute on top of any multi-branch continuous vision algorithm that consumes streaming video frames as inputs. At a high level, our solution LiteReconfig comprises of two parts that work together as a scheduler to determine which branch of the execution kernel to execute. The *first part* models the cost and the benefit of all possible *features* used by the scheduler to decide among the execution branches and then decides which features to use in choosing the execution branch. The *second part* models the cost and the benefit of the various *execution branches* of the MBEK and chooses the optimal execution branch. A final, meta-level optimization is performed to synthesize the outputs from the above two parts and to determine which execution branch to invoke. This is done by solving a constrained optimization problem (Equation 3) that maximizes the benefit (the improvement of accuracy) of the object detection kernel, such that the latency stays below the SLA.

In what follows, we present the optimization problem of the scheduler assuming the particular features to be considered are decided. The solution of this optimization leads to our choice of the execution branch to invoke (Section 3.3). We then present the cost-benefit analysis that enables the scheduler to pick the right set of features, considering their costs and benefits (Section 3.5). And finally, we introduce the cost-benefit analysis on the switching cost to guarantee the tail latency SLA. (Section 3.6).

3.2 Terminology in Adaptive Vision Systems

We first introduce several important terms and concepts encountered in adaptive vision systems.

Execution Branch: is a distinct setting of a solution algorithm, typically differentiated by controlling some hyper-parameters (colloquially, "knobs"), so as to finish the vision task in a distinct and fixed execution time (latency) and a consistent accuracy across the dataset. We observe that models with multiple execution branches are often considered by adaptive object detection frameworks. Such solution algorithms are gradually becoming more popular in the vision community, with an early start by BranchyNet [57], and a recent example being ApproxDet [66].

Accuracy-Latency Trade-off: The trade-off between accuracy and latency is fundamental to all adaptive vision systems. If a higher accuracy is desired, then one has to incur higher latency. For each execution branch, a certain accuracy and latency is achieved (for a given content type and contention level). The set of all such accuracy-latency values is represented in a curve like that shown in Figure 1, *bottom right*. Of these, the Pareto frontier is the one to focus on as any scheduler strives to stay on it. Note that the Pareto curve is subject to change at runtime due to the dynamic content characteristics and resource contention.

3.3 Scheduler Optimization

LiteReconfig builds on an existing MBEK with a set of execution branches $\mathcal{B}$, and strives to pick the execution branch that maximizes the accuracy of the object detection while probabilistically meeting the latency guarantee. The latency guarantee is typically specified in terms of tail latency, like the 95th percentile latency, though it could be specified, and this does not intrinsically affect the algorithms in LiteReconfig. Specifically, we create a latency prediction model $L(b, f)$ and an accuracy prediction model $A(b, f)$ to predict the latency (*i.e.,* cost) and accuracy (*i.e.,* benefit) of the execution branch b, based on a set of features f, in a short look-ahead window, *e.g.,* Group-of-Frames (GoF). The choice of the optimal branch is thus determined by the solution to a constrained optimization problem that maximizes the predicted accuracy while maintaining the predicted latency within the latency SLA L_0, given by

$$\begin{aligned} b^* &= \operatorname*{argmax}_{b\in\mathcal{B}} A(b, f) \\ &s.t.\ \ L(b, f) \le L_0 \quad given\ \ f \in \mathcal{F}. \end{aligned} \tag{1}$$

A critical design choice of our latency and accuracy prediction model is that these models are not only a function of the execution branch b, but also of the content-based features, which can be included in f. This design thus allows us to choose different features f from a set of features $\mathcal{F}$ with varying computational cost at runtime, such that our scheduler can be better adapted to the video content characteristics and the computing resources available. We now present the design of our latency and accuracy models.

Modeling of Latency: To build our latency model, we start by analyzing the sources of latency of our system. The end-to-end latency is comprised of two parts — the latency of the MBEK and the execution time overhead of LiteReconfig's scheduler. The latter is again composed of three parts — (1) the scheduler cost of extracting various features, (*e.g.,* the number and sizes of objects in the video frame, the histogram of colors, the degree of motion from one frame to another) (2) the scheduler cost of executing corresponding models to predict the accuracy and the latency of each execution branch for these feature values, and (3) the switching cost from the current execution branch to a new one.

Next, we observe that the selected features f can be divided into two types: light features f_L that will always be computed, such as height and width of the input video or the number of objects in the frame and are thus available to the scheduler for "free", and heavy features f_H, which may be extracted based on the cost-benefit analysis. A detailed list of the features considered in LiteReconfig and their costs are summarized in Table 1. We consider the following latency model that consists of four terms, given by:

$$L(b, f) = L_0(b, f_L) + S_0 + S(f_H) + C_{b_0}(b), \tag{2}$$

where $L_0(b, f_L)$ is a linear regression model defined on each branch b using the light features f_L to predict the latency of b. S_0 is the cost of the scheduler that extracts and uses the light features f_L to determine the optimal branches; $S(f_H)$ is the additional cost of the scheduler that extracts and uses computationally heavy content features f_H; $C_{b_0}(b)$ is the switching cost from the current branch b_0 to the new branch b. For ease of exposition, in this formulation, we have considered all the heavy features as one unit — in reality, the scheduler can recruit any subset of heavy features.

Modeling of Accuracy: Another central component of an adaptive vision system is the accuracy prediction model. Due to the latency SLA, in much of prior work, only features that are lightweight to compute are considered for modeling the accuracy of the execution branches [12, 16, 44, 66].

Our key observation is that the more expressive, yet computationally heavy features (f_H), can significantly improve the prediction. For example, we find that the widely used computer vision features, like Histogram of Colors (HoC) [43], HOG [10], recent neural network based features, like MobileNetV2 [52] in Table 1, can help build a significantly better accuracy prediction model, which we call the *content-aware accuracy model.* In addition to the three external feature extractors, we also use two features from the Faster R-CNN detector from the MBEK — ResNet50 and Class Predictions on Proposal feature (CPoP). These are efficient to collect as these are obtained directly from the detector. These two features turn out to be informative features to characterize the accuracy of each branch in the MBEK.

3.4 Content-agnostic vs. Content-aware Accuracy Model

Instead of predicting the accuracy of an execution branch b on a representative large dataset (as one would with the content-agnostic features in [66]), we aim at predicting the accuracy of an execution branch b at a finer granularity, using a video snippet. A video snippet is a sequence of N consecutive frames, [1] starting at any point of the streaming video. In practice, since the scheduler must make a decision right on the current frame, we extract features from the first frame of the snippet and use these features to predict

[1]Too small an N will make it hard to characterize the accuracy of execution branches and too large an N will tend toward a content-agnostic system. We take $N = 100$ to balance these two goals.

Category, Notations	Feature names, Dimension	Cost (msec)		Description
		Extraction	Prediction	
Light, f_L	Light, 4	0.12	3.71	Composed of height, width, number of objects, averaged size of the objects.
Heavy, f_H^1	HoC, 768	14.14	4.94	Histogram of Color on red, green, blue channels.
Heavy, f_H^2	HOG, 5400	25.32	4.93	Histogram of Oriented Gradients.
Heavy, f_H^3	Resnet50, 1024	26.96	6.07	ResNet50 feature from the object detector in the MBEK, average pooled over height and width dimensions and only reserving the channel dimension
Heavy, f_H^4	CPoP, 31	3.62	4.84	Class Predictions on Proposal feature from the Faster R-CNN detector in the MBEK. Prediction logits on the region proposals are extracted and average pooled over all region proposals. We only reserve the class dimension (including a background class)
Heavy, f_H^5	MobileNetV2, 1280	153.96	9.33	Efficient and effective feature extractor, average pooled from the feature map before the fully-connected layer.

Table 1: List of features and their costs considered in our scheduler. The latency is evaluated on the NVIDIA Jetson TX2 board. ResNet50, CPoP, MobileNetV2 feature extractors and the prediction models use the GPU; the others are mainly on the CPU.

the accuracy of execution branches on the video snippet. Concretely, $A(b, f)$ predicts the accuracy of branch b in a short look-ahead window using input features f, where the features can include either light (f_L) or a subset of the heavy features (f_H).

The accuracy prediction model $A(b, f)$ is realized with a 6-layer neural network. The first layer is a fully-connected projection so as to project the low-dimensional light features and high-dimensional content features to the same dimension and then concatenate them. The later five layers are all fully connected layers with ReLu as the activation function.

Constrained Optimization: Given the optimization problem in Equation 1 and our latency model in Equation 2, our scheduler is tasked to select the optimal execution branch b^* based on the selected features f under the latency budget L_0, by solving the following constrained optimization problem

$$\begin{aligned} b^* &= \underset{b \in \mathcal{B}}{\operatorname{argmax}}\, A(b, f) \\ &s.t.\ \ L_0(b, f_L) + S_0 + S(f_H) + C_{b_0}(b) \le L_0 \\ &given\ \ f = [f_L, f_H] \in \mathcal{F}. \end{aligned} \tag{3}$$

To solve this optimization, we examine all branches $\{b\}$ [2] that satisfy the latency constraint and pick the branches with highest predicted accuracy $A(b, f)$. Note that the latency prediction model $L_0(b, f_L)$ incorporates light features f_L but does not rely on the heavy content features f_H. Additionally, both the accuracy prediction model $A(b, f)$ and the latency prediction model $L_0(b, f_L)$ are trained from the data on our offline dataset. Critically, our key innovation lies in the design of the optimization problem rather than in solving it (we use standard convex solver). In the following sections, we will discuss (1) our feature selection algorithm for deciding which features f to choose for scheduling (Section 3.5), and (2) the modeling of switching cost $C_{b_0}(b)$ (Section 3.6).

[2]The computational cost of the feature extractors and accuracy prediction model dominates the overhead of the scheduler. Reducing the number of examined branches does not significantly reduce the cost as the execution time of a neural network (our accuracy prediction model) is not linear in relation to the number of output neurons, i.e., the number of branches to predict on. For example, reducing the number of branches to predict on by 20% may only reduce the cost by 5%.

3.5 Feature Selection for Scheduling

An important first step for our LiteReconfig is to select proper features used by the scheduler. To repeat the motivation, existing solutions do not consider the relative cost and the benefit of including various features, rather treat them as a single monolithic bucket of features to use for latency and accuracy prediction [66]. Consequently, only light features are considered for the scheduler due to the latency budget. In contrast, LiteReconfig dynamically decides which features to use during runtime, based on current video content characteristics and latency requirement. We now present our findings on the utility of different features, and describe our model for selecting features for accuracy prediction.

Light vs. Heavy Features: The light features f_L can be extracted with no cost and the corresponding content-agnostic accuracy prediction model on it is also computationally light. On the other hand, the heavy features f_H are content dependent and need processing of the video frame and more costly downstream neural network-based processing on them. An example of the former is the dimension of the image while an example of the latter is the MobileNetV2 feature of a video frame. Naturally, accuracy is enhanced with content-dependent features, such as HoC, HOG, MobileNet, and ResNet, as is well known in the literature [21, 48, 73]. We show empirically that this improvement happens under many scenarios (but not all). Further, one has to account for the decrease in the latency budget of the execution kernel due to the overhead of the features themselves, *i.e.,* extracting the features and querying the content-dependent accuracy prediction models with the features. *This is the key idea behind our feature selection algorithm.*

Table 1 shows that the extraction of the HoC, HOG, and MobileNetV2 features takes 14.14 msec, 25.32 msec, and 153.96 msec, respectively, and the prediction models on these features take 4.94 msec, 4.93 msec, and 9.33 msec, respectively. This is because these features are high-dimensional to encode. Such costs can be overwhelming especially when the continuous vision system is running with a strict latency requirement, say 33.3 msec (30 fps). Supposing the scheduler is triggered at every first frame of a GoF, with size 8 (a middle-of-the-range number), the MobileNetV2 feature extraction plus prediction alone takes 61% of the latency budget. In several situations, this offsets its benefit in selecting a better execution branch through its content-aware accuracy prediction model.

Modeling the Cost and Benefit of Features: The key challenge of the feature selection process is that the algorithm has to make the determination *without actually extracting the heavy features* or querying the models with these heavy features. We thus must make some pragmatic simplifications.

Consider the set of all possible features $\mathcal{F}$ consists of light features f_L and a set of heavy feature candidates $\mathcal{F_H}$. Our algorithm will always use the light features f_L and then determine which subset of heavy features $f_H \in 2^{\mathcal{F_H}}$ to use. It is possible none of the heavy features is used, *i.e.,* $f_H = \phi$. We first extract the light features and run the latency prediction model $L_0(b, f_L)$ and accuracy prediction model $A(b, f_L)$. Then, we use the following nested optimization to decide f_H, one element at a time, f_H^i. Let us say at any point in the iterative process, the currently selected set of heavy features is f_H^S. The optimization is given by

$$f_H^i = \underset{f_H \in \mathcal{F_H} \setminus f_H^S}{\text{argmax}} \max_{b \in \mathcal{B}} A(b, f_L) + Ben(f_H^S \cup f_H)$$

$$s.t.\ \ L_0(b, f_L) + S(f_L) + S(f_H^S \cup f_H) + C_{b_0}(b) + M(b) \leq L_0.$$

$Ben(f_H^S \cup f_H)$ is the benefit (improvement in accuracy) of including additional features f_H. $S(f_L)$ is the cost to extract and use light features f_L; $S(f_H^S \cup f_H)$ is the cost for heavy features $f_H^S \cup f_H$; $C_{b_0}(b)$ is the switching cost from the current branch b_0 to the new branch b.

We further simplify the calculation of the benefit $Ben(f_H^S \cup f_H)$ due to the heavy features in Equation 3.5. Concretely, this benefit depends on the content features and should ideally be calculated by extracting the heavy features from the current video frame. However, doing so would be costly and would defeat the purpose of this feature selection algorithm. The key difference of this equation from Equation 3 is that we use $A(b, f_L) + Ben(f_H^S \cup f_H)$ as a proxy of $A(b, f_H^S \cup f_H)$ to avoid extracting heavy features and executing the corresponding content-aware accuracy prediction model. The benefit function $Ben(f_H^S \cup f_H)$ is collected from the *offline* dataset to reflect the accuracy improvement of the system with the heavy features F against the light feature f_L.

3.6 Modeling Switching Cost

All prior works that have introduced adaptive vision models [6, 12, 22, 66, 70] have omitted to consider the latency cost of switching from one branch to another (or, in the case of ensemble models like [16, 67] from one model to another). On the other hand, LiteReconfig takes that switching cost into account in its cost-benefit analysis. This is motivated by our observation from Figure 5 that different branch transitions have different costs. This switching cost depends on the implementation and the nature of execution branches, and varies due to the size of non-shared data structure (such as, disjoint parts of a TensorFlow graph) to be loaded. We therefore perform a cost-benefit analysis to decide whether switching to a new branch b is worthwhile from the current branch b_0 by including the term $C_{b_0}(b)$, *i.e.,* the cost of switching in latency terms, in the total cost formulation. The data is again collected from the offline training dataset.

Our model of switching cost only considers the current frame in the streaming video. Due to the unforeseen nature of the streaming video, we cannot forecast how long such a new branch b might stay optimal. Thus, the scheduler is triggered after every tracking-by-detection GoF number of frames, to deal with the dynamic nature of content characteristics. We found empirically that this strategy works better than the one employed in previous works [29, 49], which optimize over a look-ahead window by predicting future workload changes. We also found that previous approaches suffer from inaccurate predictions and a high cost of determining the schedule over a look-ahead. Further, our design of invoking the scheduler after every GoF (the size of GoF is typically 4–20) mitigates the impact of an incorrect decision.

4 Implementations and Baselines

4.1 Implementation of LiteReconfig

We implement LiteReconfig on top of a MBEK with Faster R-CNN as the detection backbone and four types of object trackers: MedianFlow, CSRT, KCF, and Optical Flow. We implement LiteReconfig in Python-3 (v3.7.3) using TensorFlow-gpu v1.14.0 (for Faster R-CNN), PyTorch v1.4.0 (for MobileNetV2 feature extractors and neural network-based accuracy prediction models), CUDA v10.0.326, and cuDNN v7.5.0. We replicate the latency predictors in ApproxDet [66], enhance it for more types of embedded devices (ApproxDet only runs on NVIDIA Jetson TX2), and use them for predicting the latency of each execution branch, *i.e.,* $L(b, f)$ (Equation 1). To train content-aware accuracy prediction model $A(b, f_H)$, we first collect the content-dependent features for each video snippet as the model input. Then, we collect the snippet-specific accuracy, *i.e.,* the mAP, under each execution branch. These mAP results are used as labels for training our content-aware accuracy prediction model in a supervised manner. We use a 6-layer neural network for each content-dependent feature. The first projection layer projects both the light feature f_L and content-dependent feature f_H into vectors of 256 neurons and then concatenates the two. The following fully-connected layers come with 256 neurons in the hidden layer and M neurons in the output layer, where M is the number of execution branches. We use MSE loss and Stochastic Gradient descent (SGD) optimizer, with momentum of 0.9, to train the neural network, and use ℓ_2 regularization to prevent overfitting. We train the neural networks for 400 epochs at maximum with batch size of 64 and observe that the models converge within 100 epochs.

We evaluate LiteReconfig on two embedded platforms: an NVIDIA Jetson TX2 [8] and a more powerful NVIDIA Jetson AGX Xavier [7]. TX2 has a 256-core NVIDIA Pascal GPU on a 8GB unified memory while AGX Xavier has a 512-core Volta GPU on a unified 32GB memory. TX2 has compute capability similar to high-end smartphones like Samsung Galaxy S20 and iPhone 12 Pro, while AGX Xavier represents the next-generation mobile SoCs. Using two different platforms allows us to evaluate the performance of LiteReconfig with different target latency ranges.

LiteReconfig Variants: We consider four variants: namely LiteReconfig-MinCost (content agnostic), LiteReconfig-MaxContent-ResNet, LiteReconfig-MaxContent-MobileNet (the two best performing content-aware models), and LiteReconfig (the full implementation with cost-benefit analysis and all content features and models).

4.2 Baselines

We consider these baselines for evaluating LiteReconfig.

1. **ApproxDet**: is the state-of-the-art adaptive object detection framework on embedded devices [66]. ApproxDet uses Faster R-CNN [48] as its backbone object detector, and is able to adapt to given latency requirements during runtime by dynamically changing the resolution of the image that is fed into the detector (*shape*), and changing the number of proposals (*nprop*) in the first RPN. The ability to adapt during runtime is enhanced by combining an object tracker with the detector, coupled with different tracker types (LiteReconfig uses the same four types as ApproxDet), different values of GoF, and the downsampling ratio of the image fed into the tracker.
   We use their open-sourced implementation [64], which uses TensorFlow-gpu v1.14.0.
2. **AdaScale**: AdaScale [6] is an adaptive object detection framework that can adaptively re-scale the input image to one of a number of preset resolutions. This feature gives AdaScale the ability to perform inference at different latencies, while maintaining optimal accuracy. Even with the adaptive features, the primary focus of AdaScale is not efficiency, and thus its base latency on embedded boards are too high to be compared with resource contention. Therefore, we execute AdaScale on TX2 without contention and only compare their mAP and latency values.
3. **YOLO+**: YOLOv3 [47] is a popular one-stage object detector with a faster speed due to its single-stage design. However, the implementation is still far from achieving real-time processing on our embedded devices. Thus, we enhance the efficiency of YOLOv3, call it YOLO+, by exposing four tuning knobs similar to the ApproxDet work – *shape* of video frame fed into YOLOv3, size of GoF (*si*), type of *tracker*, and downsampling (*ds*) ratio of the frame fed into the tracker. The YOLO implementation is YOLOv3 in PyTorch v1.4.0 from Ultralytics [59].
4. **SSD+**: SSD [40] is also a popular one-stage object detector that can be combined with multiple backbones as the feature extractor. In our work, we use the MobileNetV2 as the backbone, combined with MnasFPN [4] to further enhance the object detector. We further engineer SSD for efficiency and adaptability and name it SSD+. This is done by exposing the same tuning knobs as for YOLO. An additional tuning knob is the confidence threshold of the detector that controls the number of objects to be tracked by the tracker. The SSD implementation is with Tensorflow v1.14, following the official implementation from Tensorflow Object Detection API [58].
5. **EfficientDet**: EfficientDet [56] is one of the recent SOTA object detectors, engineered for both accuracy and efficiency, with a total of 8 model variants in its family (D0-D7), each having a different scaling factor. From our observation, heavier model variants above D3 cannot run on the Jetson TX2 board due to insufficient memory, thus we have selected D0 and D3, which are the lightest and heaviest models within the executable candidates.
6. **SELSA** [63], **MEGA** [5], **REPP** [51]: We do a more limited benchmarking of these three recent solutions with the best benchmarking results for video object detection (on server-class machines, not embedded). We use pre-trained models, trained on the same dataset as ours. This comparison is more limited since these models are not optimized for efficiency, and therefore, have unacceptable latency on embedded devices. Additionally, they cannot execute with resource contention on our embedded boards — their base latency is already too high and they crash or hang with resource contention. Therefore, we execute them on TX2, without contention, and compare mAP and latency.

5 Evaluation

We conduct extensive experiments on embedded boards with mobile GPUs to evaluate the ability of LiteReconfig to achieve high accuracy and low latency and contrast with a slew of strong baselines. Our evaluations account for dynamic conditions of changing content and contention levels.

GPU resource contention	Device and latency SLAs (msec)	Models	mAP (%)	P95 latency per-frame (msec)
0%	TX2, 33.3/50.0/100.0	SSD+	**45.5**/46.3/46.7	30.5/47.8/79.9
		YOLO+	42.1/45.8/47.3	26.0/36.3/65.3
		ApproxDet	F / F /46.8	F / F /83.9
		LiteReconfig-MinCost	43.8/46.4/49.0	26.4/41.0/77.7
		LiteReconfig-MaxContent-ResNet	44.4/**47.1**/50.3	28.5/40.8/82.3
		LiteReconfig-MaxContent-MobileNet	F / F /50.2	*35.2/50.9*/99.0
		LiteReconfig	**45.4**/46.5/**50.3**	32.2/42.1/80.5
50%	TX2, 33.3/50.0/100.0	SSD+	F / F / F	*41.6/68.3/118.7*
		YOLO+	F / F /**47.3**	*36.3/55.2*/98.8
		ApproxDet	F / F /45.2	F / F /78.7
		LiteReconfig-MinCost	39.0/42.1/46.0	25.0/41.8/84.6
		LiteReconfig-MaxContent-ResNet	**39.2**/41.4/46.6	30.9/47.5/90.5
		LiteReconfig-MaxContent-MobileNet	F / F /**47.1**	*36.2/60.0*/79.2
		LiteReconfig	39.3/**43.6/47.0**	*34.0*/49.5/78.2
0%	AGX Xavier, 20.0/33.3/50.0	SSD+	45.5/46.3/46.7	*21.1*/27.9/41.5
		YOLO+	44.2/45.7/48.8	14.4/26.2/38.1
		LiteReconfig-MinCost	45.5/47.4/49.6	16.4/25.5/38.8
		LiteReconfig-MaxContent-ResNet	**46.4/48.5/50.7**	17.0/26.9/39.3
		LiteReconfig-MaxContent-MobileNet	F / F /50.7	*20.3/35.6*/38.7
		LiteReconfig	**46.4/48.5/50.7**	18.2/28.9/41.4
50%	AGX Xavier, 20.0/33.3/50.0	SSD+	F /46.3/ F	*25.2*/33.3/*53.4*
		YOLO+	F / F / F	*30.6/59.0/93.1*
		LiteReconfig-MinCost	38.9/45.1/46.3	17.7/25.1/38.4
		LiteReconfig-MaxContent-ResNet	**39.3/45.4/46.9**	17.6/25.4/39.0
		LiteReconfig-MaxContent-MobileNet	F /44.6/46.8	*21.8*/32.8/47.0
		LiteReconfig	**39.4**/45.1/**46.9**	19.0/27.8/40.0

Table 2: Performance comparison on the ImageNet VID validation set. "F" in the mAP column indicates that the protocol fails to meet the latency SLA and thus the accuracy results are not comparable (one exception for LiteReconfig in one cell to show the complete accuracy data). The bold text of mAP shows the highest accuracy in each scenario and requirement, while the italicized text of latency highlights that the 95% latency SLA is violated. An "F" in a latency cell means that that protocol did not execute at all.

Models, latency SLA	mAP (%)	Mean latency per-frame (msec)	Memory (GB)
SELSA-ResNet-101 [63], no SLA	81.5	2334	6.91
SELSA-ResNet-50, no SLA	77.31	2112	6.70
MEGA-ResNet-101 [5], no SLA	OOM	OOM	9.38
MEGA-ResNet-50, no SLA	OOM	OOM	6.42
MEGA-ResNet-50 (base), no SLA	68.11	861	3.16
REPP [51], over FGFA[75], no SLA	OOM	OOM	10.02
REPP, over SELSA	OOM	OOM	8.13
REPP, over YOLOv3	74.8	565	2.43
EfficienetDet D3	63.9	796	5.68
EfficienetDet D0	55.1	138	2.22
AdaScale-MS, no SLA	56.3	976.4	3.26
AdaScale-SS-600, no SLA	55.7	1049.4	3.20
AdaScale-SS-480, no SLA	59.0	710.5	3.18
AdaScale-SS-360, no SLA	59.4	434.0	3.18
AdaScale-SS-240, no SLA	56.5	227.9	3.18
LiteReconfig, 100 msec	50.3	72.0	3.67
LiteReconfig, 50 msec	46.5	38.4	4.09
LiteReconfig, 33.3 msec	45.4	28.2	4.12

Table 3: Performance comparison between LiteReconfig and the video object detection solutions optimized for accuracy.

5.1 Evaluation Setup

Dataset and Metrics: We evaluate LiteReconfig on the ILSVRC 2015 VID dataset [50], which contains 3,862 videos in the training set, and 555 videos, in the validation set. Both datasets are fully annotated with classes of the objects and their localizations. We report the mAP on the VID validation dataset as the accuracy metric, following widely adopted protocols [5, 60, 74–76]. We use 90% of the ILSVRC training dataset to train the vision algorithms (detection backbones and heads), including all baselines, and the remaining 10% of the training dataset, to build the following: the latency prediction model $L(b)$, the accuracy prediction model $A(b, f)$, the switching overhead model $C_{b_0}(b)$, and the benefit of heavy features $Ben(F)$. We report violation rate of the per-frame 95th percentile (P95) latency over SLA, as the latency metric, where the scheduler of each protocol aims to guarantee a violation rate $< 5\%$. This is the standard metric used for the evaluation of latency-sensitive ML systems [27, 30, 33, 45]. Our reported latency also includes the execution times from all parts of LiteReconfig, such as, the feature extractor, the model execution, and the scheduler. The latency of object detection on a frame is much higher than that of object tracking on a frame (7X to 330X in our system). Thus, we follow the widely used "*tracking-by-detection*" technique [3, 25, 42], where the object detector is invoked once in a GoF, and the tracker is used for the rest of the frames. If the averaged latency violates the latency SLA, the entire GoF is considered to violate the SLA.

5.2 End-to-end Evaluation

We first evaluate LiteReconfig for the accuracy of the entire system by setting several latency requirements from 100 msec (loose) to 33.3 (tight) msec per frame (10 to 30 fps) on the NVIDIA Jetson TX2 board, where no resource contention is injected. Table 2 summarizes the comparisons of LiteReconfig's four variants with the baselines. We should read the mAP and P95 per-frame latency, separated by slash, and corresponding to three latency requirements from tight to loose; a note of "F" means that solution failed to meet that latency objective. First, we observe that LiteReconfig and our in-house enhanced baselines (SSD+ and YOLO+) have improved efficiency over the SOTA ApproxDet from 33.3 msec to 100 msec. Second, LiteReconfig achieves 3.5% mAP improvement over ApproxDet given the 100 msec latency requirement — a significant improvement for an object detection task. Third, LiteReconfig achieves consistent accuracy improvement over the content-agnostic variant, *i.e.*,LiteReconfig-MinCost, by 1.6%, 0.1%, and 1.3% mAP, respectively, for each latency requirement. Fourth, among all of LiteReconfig's four variants and our enhanced baselines, LiteReconfig and SSD+ achieve the highest accuracy under 33.3 msec requirement, LiteReconfig-MaxContent-ResNet is the most accurate under 50 msec requirement, and LiteReconfig is again the most accurate under 100 msec requirement. Finally, though LiteReconfig-MaxContent variants face the issue of violating the latency requirement due to their high cost of running feature extractors and models, *LiteReconfig is strictly always below the latency requirement.* To summarize, in addition to efficiency improvement over SOTA, our cost and content-aware solution scales accuracy frontiers, with latency guaranteed to meet the requirement.

Further, we extend our evaluation to a higher resource contention scenario, *i.e.*, 50% GPU resource is occupied by other concurrent applications. As can be seen in Table 2, first, under higher resource contention on TX2, our efficiency-enhanced baselines still fail to meet the latency requirement due to lack of awareness to adapt to the resource contention. Second, we are 1.8% mAP more accurate than the best adaptive systems. Third, our full implementation is always one of best protocols among all variants and models, satisfying the latency requirement (slight exception of 34.0 msec under 33.3 msec latency requirement).

Finally, we extend our evaluation to a more powerful embedded device, NVIDIA AGX Xavier. Correspondingly, we tighten the latency requirement to as low as 20 msec (corresponding to 50 fps from the earlier 30 fps). We find that ApproxDet cannot meet any of the three latency requirements. We see again LiteReconfig and the MaxContent-ResNet variants both lead the accuracy frontier under different latency requirements and resource contention scenarios and can meet the latency requirement. On the other hand, the enhanced baselines and MaxContent-MobileNet violate the latency requirements for the two stringent requirements. This shows LiteReconfig can always achieve the best content-aware variant, and clearly superior to the content-agnostic variant (0.9-1.1% higher mAP, given no contention, and 0.5-0.6% higher mAP, given 50% GPU contention).

Comparison to Recent, Accuracy-optimized Object Detection Solutions: Table 3 further compares LiteReconfig to recent solutions optimized for accuracy, which however cannot achieve real-time efficiency on the embedded device. We also add our evaluation on AdaScale, one of the content-aware baselines here since its fastest variant, the one always using the smallest scale, is running at 227.9 msec on the TX2 board and is much less efficient than LiteReconfig. Its accuracy of 55.7% to 59.4% mAP, though higher than LiteReconfig, is still the worst among all accuracy-optimized models. In addition, EfficientDet D0 and D3 achieve reasonable accuracy while maintaining relatively low latency compared with other accuracy-optimized solutions such as SELSA [63], MEGA [5], and REPP [51]. EfficientDet D0 runs at 138 msec at an mAP of 55.1% coming close to 100 msec, while the accuracy is higher compared to LiteReconfig. With no adaptive features, both EfficientDet D0 and D3 fail to match the 100 msec latency requirement.

Compared to most accurate models SELSA, MEGA and REPP, LiteReconfig is 90.3X, 36.8X, and 24.1X faster (for SLA of 33.3 msec). On the flip side, our accuracy is significantly lower. Thus, one may argue that each kind of solution has its own applicability — if real-time processing is required on these embedded boards, LiteReconfig is a good solution, while if frames can be sub-sampled for detection and high accuracy is required, then these recent solutions should be chosen. It should be noted that our experimental mAP values are lower than the published numbers in the papers (3.2% lower for SELSA, 9.2% for MEGA, and 23.8% for REPP), even though we used the author pre-trained models. We can explain the accuracy reduction in Table 3 to three factors (1) the change of the backbone feature extractors due to the memory constraint on the TX2, *e.g.*, from ResNet-101 to ResNet-50, (2) the removal of the part of the solution that refers to future frames because our problem context requires streaming and real-time processing, and (3) the usage of the same mAP calculation script across all protocols.

5.3 Evaluation of Video Content Features

Feature	33.3 ms	50.0 ms	100.0 ms
None	43.8%	46.4%	49.0%
HoC	44.4%	**47.1%**	**50.3%**
HOG	44.3%	**47.1%**	50.2%
ResNet50	44.4%	47.0%	**50.3%**
CPoP	44.8%	46.1%	**50.3%**
MobileNetV2	**45.1%**	**47.1%**	50.2%

Table 4: Effectiveness of individual content-specific features on the accuracy given different latency budgets.

Next, we analyze the benefit of applying each content feature in our content-aware design. To study this, we always extract a particular feature and use it in the corresponding prediction model and see what accuracy can we achieve, with the latency requirement applied to the MBEK only, and *ignoring the overhead of that feature.* We see that all content features achieve higher utility than the content-agnostic (labeled as "None"). The maximum accuracy improvement achieved by a single content feature (over "None") is 2.3%, 0.7%, and 1.3%, respectively, for each latency requirement. This validates our design to use cost-benefit analysis to select the best features among all (feature) options and also determine whether the benefit is enough over the content-agnostic protocol, considering its cost.

5.4 Understanding Latency-Accuracy Tradeoff

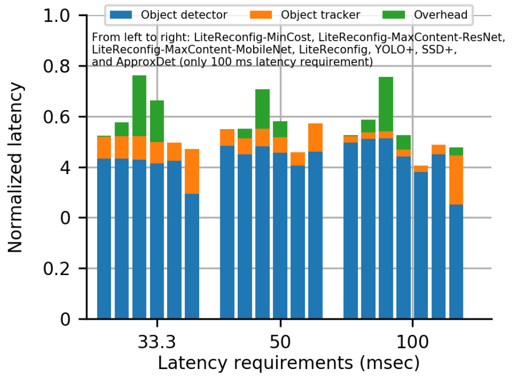


Figure 3: Percentage latency of each system component, normalized over the latency SLA, profiled on the TX2. FRCNN and YOLO cannot meet the 33.3 msec SLA and thus their bars are missing.

We examine the detailed latency breakdown of each system component in LiteReconfig to uncover the source of our benefit. Figure 3 shows the percentage latency (normalized by the latency SLA) of the object detector, the object tracker, and the cost, where the cost is either modeling (feature execution, regression models, solving optimization) or switching between execution branches. There is no bar for ApproxDet for 33.3 and 50 msec latencies because it cannot satisfy those SLAs. First, we can observe the cost of LiteReconfig is between the two LiteReconfig-MaxContent variants, owing to its cost benefit analysis on feature selection. Second, the overhead of LiteReconfig is always below 10%, and much less, for the higher latency requirements (50 ms and 100 ms). Finally, LiteReconfig wins over YOLO+, SSD+, and ApproxDet, due to higher latency SLA assigned to the object detector. Also, as we select the content-dependent optimal branches, even the latency of these branches may seem similar, but our selected branches finally lead to higher accuracy (Table 2) One may wonder why LiteReconfig does not try to use up the latency budget to get close to the 1.0 normalized latency value. This is merely an artifact of the presentation of this result — we are reporting mean latency while SLA is specified in terms of 95-th percentile latency (P95). Thus, LiteReconfig is using up its latency budget prudently, without causing too frequent SLA violations.

We further investigate the branch coverage (*i.e.,* the number of distinct execution branches invoked) within the four

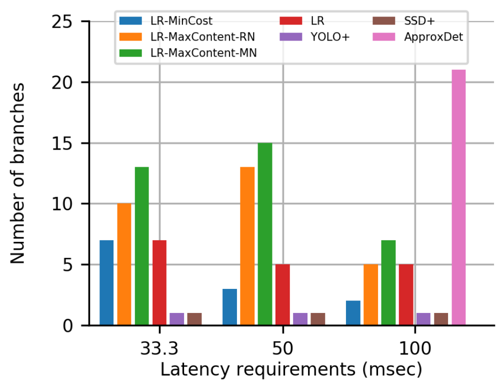
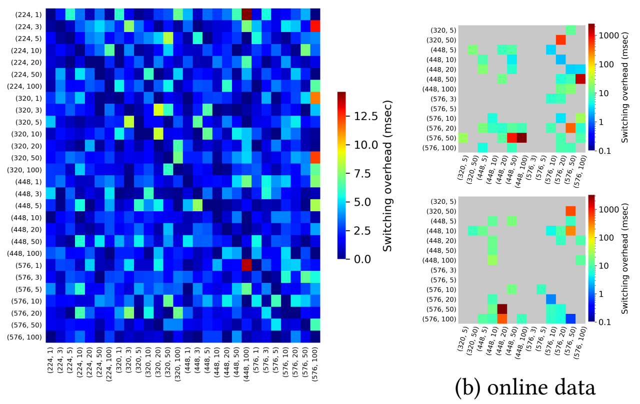


Figure 4: Branch coverage between the four variants of LiteReconfig and three other baselines. LiteReconfig is able to explore more number of beneficial execution branches and avoids fruitless switches between execution branches, leaving greater part of the latency budget available for the execution kernel, the object detector, and the tracker, leading to improved accuracy.

LiteReconfig variants. Figure 4 shows that using heavy features (orange and green bars) tend to explore more branches tailored for the video content and thus is the driving force for higher accuracy. However, using light features can reduce more latency for the MBEK. The complete LiteReconfig (red) through its cost-benefit analysis balances these two tendencies and leads to the winning overall accuracy over other variants and baselines, and probabilistically guarantees the latency requirement. ApproxDet, covering a far higher number of execution branches (100 msec latency requirement), is still less accurate than any variant of LiteReconfig.

5.5 Switching Cost and Benefit



(a) offline training data (b) online data

Figure 5: Switching overhead between execution branches in object detectors from (a) offline training data and (b) online data given 33.3 msec latency SLA on the top and 50 msec latency SLA on the bottom. The source branches are on the y-axis and the destination branches are on the x-axis, with (*shape*, *nprop*) notation.

LiteReconfig considers the switching overhead between branches in deciding whether to reconfigure its execution to a new branch — this is done through the term $C_{b_0}(b)$ in Eq. 3. In Figure 5, we show empirically the switching overhead between any two execution branches in the object detector. The *offline* training data on Figure 5(a) shows that generally the switching overhead is below 10 msec but it is higher with a light source branch, *e.g.,* *shape*=576 and *nprop*=1, or with a heavy destination branch, *e.g.,* *shape*=576 and *nprop*=100. Figure 5(b) shows the *online* switching cost with 33.3 msec latency requirement at the top, and 50 msec latency requirement at the bottom. The online data confirms that the switching overhead is mostly less than 10 msec and we also observe the non-deterministic outliers with high values, in the 1–5 sec range. These outlier results happen due to cold misses of the neural network graphs and become rarer still as the video analytics system runs for a longer period of time. Such outlier switches are impossible to model — see how the outliers do not show up at the same cells in our two independent runs.

6 Related Work

Object Detection and Tracking in Videos. Modern object detectors make use of DNNs to localize the recognized object instances within an input image. Examples include the YOLO series [46, 47], Faster-RCNN [48], EfficientDet [56], and Cascade R-CNN [53]. However, it remains challenging to apply these image-based detectors to videos, due to motion blur, occlusion, and defocus that frequently occur in videos. In parallel, visual tracking has evolved from lightweight trackers focusing on motion analysis and trajectory prediction [2, 14, 26] to DNN-based trackers that learn to match appearance patterns of target objects [13, 34, 61]. These developments have led to works on video object detection that use tracking to aggregate temporal features [38, 60, 75, 76] or to associate detected objects [13, 37, 66, 70]. Unfortunately, the majority of previous methods are meant to run on server-class systems, and only a few solutions exist for edge devices [37, 66]. Video object detection at the edge processes the videos where they are generated, and thus can improve latency and decrease network congestion.

Computer Vision on Mobiles. Resource utilization and management is one of the key factors to match the Quality of Experience (QoE) for end users in mobile devices. Many previous works focus on the design of lightweight DNNs to handle resource limits, including hand-crafted network architectures [15, 19, 72] , and network architectures built by neural architecture search [35, 55, 62]. Despite the efficiency of these models, none of these approaches is adaptive to content or contention at runtime. There have been recent works on developing adaptive computer vision algorithms and systems (which we have described earlier in Sections 1 and 2). To sum up, based on the image content or latency budget, adaptive configuration can occur within one model [6, 23, 65, 66], within an ensemble of models within a system [12], multi-exit solutions [20, 57], or by generating a light network that is specific to a given dataset [11]. These methods, however, lack the ability to be fully adaptive to content and contention change (*e.g.,* limited to a single dataset or a specific time interval) or to do a cost-benefit analysis to guide their adaptation.

Cost Benefit Analysis in Online Reconfigurable Systems. The specific context dictates many of the technical challenges in this space, like what are the cost and the benefit functions, how easily can the parameters for the functions be collected, and how should the cost-benefit analysis feed into changes made by a scheduler into the system. Some prominent examples in this line of work are for clustered database servers [41], for serverless jobs [31], for VM allocation and consolidation [18], and for VM migration [54]. In the context of mobile computing, such cost-benefit analysis has influenced decision making for mobile sensing [1], offloading from mobile to edge or cloud [69], or context-aware application scheduling on mobile devices [32]. While principles from this volume of prior work inspire our design, they do not directly solve our problem of reconfiguration of video object detection.

7 Conclusion

Several adaptive computer vision systems have been proposed that change the execution paths depending on content and runtime conditions on mobile devices. In this paper, we first uncover that these adaptive vision algorithms actually perform worse than static algorithms under a large range of conditions such as stringent latency requirements, say keeping up with 30 fps video or getting to 20 msec latency for AR applications. We then present our solution called LiteReconfig applicable to any approximate vision system, which provides the *cost-benefit analysis* of the different features that can be used to model the accuracy and the latency of the different execution branches. The scheduler leverages the cost-benefit analysis to achieve a superior accuracy-*vs*-latency characteristic than prior solutions, ApproxDet, EfficientDet, Faster R-CNN, YOLO, SELSA, MEGA, and REPP. Our evaluation provides a few insights with broad implications. How can latency-accuracy models of lightweight vision algorithms be transferable to different content classes, such as, from fast moving to slow moving video. How can vision frameworks be designed to better handle contention, much of which may be unpredictable. What are the relative utilities of different features in guiding adaptation in streaming video analytics systems. Much work remains to be done, some of which we are pursuing, such as, can lightweight prediction of the content stream enable optimization over a lookahead window of time, how do these approximations compose with approximations of other algorithmic blocks downstream (such as, object or facial recognition).

References

[1] Amin Anjomshoaa, Fábio Duarte, Daniël Rennings, Thomas J Matarazzo, Priyanka deSouza, and Carlo Ratti. 2018. City scanner: Building and scheduling a mobile sensing platform for smart city services. *IEEE Internet of Things Journal* 5, 6 (2018), 4567–4579.

[2] Boris Babenko, Ming-Hsuan Yang, and Serge Belongie. 2009. Visual tracking with online multiple instance learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 983–990.

[3] Seung-Hwan Bae and Kuk-Jin Yoon. 2014. Robust online multi-object tracking based on tracklet confidence and online discriminative appearance learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition (CVPR)*. 1218–1225.

[4] Bo Chen, Golnaz Ghiasi, Hanxiao Liu, Tsung-Yi Lin, Dmitry Kalenichenko, Hartwig Adam, and Quoc V Le. 2020. MnasFPN: Learning latency-aware pyramid architecture for object detection on mobile devices. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 13607–13616.

[5] Yihong Chen, Yue Cao, Han Hu, and Liwei Wang. 2020. Memory enhanced global-local aggregation for video object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 10337–10346.

[6] Ting-Wu Chin, Ruizhou Ding, and Diana Marculescu. 2019. AdaScale: Towards Real-time Video Object Detection Using Adaptive Scaling. In *Systems and Machine Learning Conference (SysML)*.

[7] NVIDIA Corporation. [n. d.]. NVIDIA Jetson AGX Xavier Board. https://developer.nvidia.com/embedded/jetson-agx-xavier-developer-kit.

[8] NVIDIA Corporation. [n. d.]. NVIDIA Jetson TX2 Board. https://developer.nvidia.com/embedded/jetson-tx2.

[9] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. 2016. R-FCN: Object detection via region-based fully convolutional networks. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*. 379–387.

[10] Navneet Dalal and Bill Triggs. 2005. Histograms of oriented gradients for human detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Vol. 1. IEEE, 886–893.

[11] Biyi Fang, Xiao Zeng, Faen Zhang, Hui Xu, and Mi Zhang. 2020. FlexDNN: Input-Adaptive On-Device Deep Learning for Efficient Mobile Vision. In *Proceedings of the 5th ACM/IEEE Symposium on Edge Computing (SEC)*.

[12] Biyi Fang, Xiao Zeng, and Mi Zhang. 2018. NestDNN: Resource-aware multi-tenant on-device deep learning for continuous mobile vision. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking (MobiCom)*. 115–127.

[13] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. 2017. Detect to track and track to detect. In *Proceedings of the IEEE International Conference on Computer Vision*. 3038–3046.

[14] Helmut Grabner, Michael Grabner, and Horst Bischof. 2006. Real-time tracking via on-line boosting.. In *Proceedings of the British Machine Vision Conference (BMVC)*, Vol. 1. 47–56.

[15] Kai Han, Yunhe Wang, Qi Tian, Jianyuan Guo, Chunjing Xu, and Chang Xu. 2020. GhostNet: More features from cheap operations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 1580–1589.

[16] Seungyeop Han, Haichen Shen, Matthai Philipose, Sharad Agarwal, Alec Wolman, and Arvind Krishnamurthy. 2016. Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services (Mobisys)*. 123–136.

[17] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. 2017. Mask r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2961–2969.

[18] Nguyen Trung Hieu, Mario Di Francesco, and Antti Ylä-Jääski. 2015. Virtual machine consolidation with usage prediction for energy-efficient cloud data centers. In *2015 IEEE 8th International Conference on Cloud Computing*. IEEE, 750–757.

[19] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. 2019. Searching for MobileNetV3. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 1314–1324.

[20] Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. 2016. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250* (2016).

[21] Xun Huang, Chengyao Shen, Xavier Boix, and Qi Zhao. 2015. Salicon: Reducing the semantic gap in saliency prediction by adapting deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*. 262–270.

[22] Angela H Jiang, Daniel L-K Wong, Christopher Canel, Lilia Tang, Ishan Misra, Michael Kaminsky, Michael A Kozuch, Padmanabhan Pillai, David G Andersen, and Gregory R Ganger. 2018. Mainstream: Dynamic stem-sharing for multi-tenant video processing. In *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*. 29–42.

[23] Junchen Jiang, Ganesh Ananthanarayanan, Peter Bodik, Siddhartha Sen, and Ion Stoica. 2018. Chameleon: scalable adaptation of video analytics. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. 253–266.

[24] Kinjal A Joshi and Darshak G Thakore. 2012. A survey on moving object detection and tracking in video surveillance system. *International Journal of Soft Computing and Engineering* 2, 3 (2012), 44–48.

[25] Jaeyong Ju, Daehun Kim, Bonhwa Ku, David K Han, and Hanseok Ko. 2016. Online multi-object tracking based on hierarchical association framework. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 34–42.

[26] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. 2010. Forward-backward error: Automatic detection of tracking failures. In *Proceedings of the 20th International Conference on Pattern Recognition (ICPR)*. IEEE, 2756–2759.

[27] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. 2017. NoScope: Optimizing Neural Network Queries over Video at Scale. *Proceedings of the VLDB Endowment* 10, 11 (2017).

[28] Kai Kang, Wanli Ouyang, Hongsheng Li, and Xiaogang Wang. 2016. Object detection from video tubelets with convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 817–825.

[29] Harshad Kasture and Daniel Sanchez. 2014. Ubik: efficient cache sharing with strict qos for latency-critical workloads. *ACM SIGPLAN Notices (ASPLOS)* 49, 4 (2014), 729–742.

[30] Harshad Kasture and Daniel Sanchez. 2016. Tailbench: a benchmark suite and evaluation methodology for latency-critical applications. In *2016 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 1–10.

[31] Ana Klimovic, Yawen Wang, Patrick Stuedi, Animesh Trivedi, Jonas Pfefferle, and Christos Kozyrakis. 2018. Pocket: Elastic ephemeral storage for serverless analytics. In *13th Usenix Symposium on Operating Systems Design and Implementation (OSDI)*. 427–444.

[32] Joohyun Lee, Kyunghan Lee, Euijin Jeong, Jaemin Jo, and Ness B Shroff. 2016. Context-aware application scheduling in mobile systems: What will users do and not do next?. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. 1235–1246.

[33] Yunseong Lee, Alberto Scolari, Byung-Gon Chun, Marco Domenico Santambrogio, Markus Weimer, and Matteo Interlandi. 2018. PRETZEL: Opening the black box of machine learning prediction serving systems. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 611–626.

[34] Xin Li, Chao Ma, Baoyuan Wu, Zhenyu He, and Ming-Hsuan Yang. 2019. Target-Aware Deep Tracking. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[35] Ji Lin, Wei-Ming Chen, Yujun Lin, John Cohn, Chuang Gan, and Song Han. 2020. Mcunet: Tiny deep learning on iot devices. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*. 1–15.

[36] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. 2017. Focal loss for dense object detection. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2980–2988.

[37] Luyang Liu, Hongyu Li, and Marco Gruteser. 2019. Edge assisted real-time object detection for mobile augmented reality. In *The 25th Annual International Conference on Mobile Computing and Networking (MobiCom)*. 1–16.

[38] Mason Liu and Menglong Zhu. 2018. Mobile video object detection with temporally-aware feature maps. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*. 5686–5695.

[39] Mason Liu, Menglong Zhu, Marie White, Yinxiao Li, and Dmitry Kalenichenko. 2019. Looking fast and slow: Memory-guided mobile video object detection. *arXiv preprint arXiv:1903.10172* (2019).

[40] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. 2016. SSD: Single shot multibox detector. In *Proceedings of the European Conference on Computer Vision (ECCV)*, Vol. 9907. 21–37.

[41] Ashraf Mahgoub, Paul Wood, Alexander Medoff, Subrata Mitra, Folker Meyer, Somali Chaterji, and Saurabh Bagchi. 2019. {SOPHIA}: Online reconfiguration of clustered nosql databases for time-varying workloads. In *Usenix Annual Technical Conference (Usenix ATC)*. 223–240.

[42] Matthias Muller, Adel Bibi, Silvio Giancola, Salman Alsubaihi, and Bernard Ghanem. 2018. Trackingnet: A large-scale dataset and benchmark for object tracking in the wild. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 300–317.

[43] Carol L Novak, Steven A Shafer, et al. 1992. Anatomy of a color histogram.. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Vol. 1992. 599–605.

[44] Xukan Ran, Haolianz Chen, Xiaodan Zhu, Zhenming Liu, and Jiasi Chen. 2018. Deepdecision: A mobile deep learning framework for edge video analytics. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 1421–1429.

[45] Waleed Reda, Marco Canini, Lalith Suresh, Dejan Kostić, and Sean Braithwaite. 2017. Rein: Taming tail latency in key-value stores via multiget scheduling. In *Proceedings of the Twelfth European Conference on Computer Systems*. 95–110.

[46] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 779–788.

[47] Joseph Redmon and Ali Farhadi. 2018. YOLOv3: An incremental improvement. *arXiv preprint arXiv:1804.02767* (2018).

[48] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2016. Faster R-CNN: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 39, 6 (2016), 1137–1149.

[49] Nilabja Roy, Abhishek Dubey, and Aniruddha Gokhale. 2011. Efficient autoscaling in the cloud using predictive models for workload forecasting. In *2011 IEEE 4th International Conference on Cloud Computing*. IEEE, 500–507.

[50] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)* 115, 3 (2015), 211–252.

[51] Alberto Sabater, Luis Montesano, and Ana C Murillo. 2020. Robust and efficient post-processing for video object detection. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*.

[52] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. MobileNetV2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 4510–4520.

[53] Hui Shuai, Qingshan Liu, Kaihua Zhang, Jing Yang, and Jiankang Deng. 2018. Cascaded Regional Spatio-Temporal Feature-Routing Networks for Video Object Detection. *IEEE Access* 6 (2018), 3096–3106.

[54] Rahul Singh, David Irwin, Prashant Shenoy, and Kadangode K Ramakrishnan. 2013. Yank: Enabling green data centers to pull the plug. In *Presented as part of the 10th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 13)*. 143–155.

[55] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. 2019. MNasNet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2820–2828.

[56] Mingxing Tan, Ruoming Pang, and Quoc V Le. 2020. Efficientdet: Scalable and efficient object detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 10781–10790.

[57] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. 2016. BranchyNet: Fast inference via early exiting from deep neural networks. In *Proceedings of the 23rd International Conference on Pattern Recognition (ICPR)*. IEEE, 2464–2469.

[58] Tensorflow. 2021. Official implementation of SSD-MobileNetv2-MnasFPN. https://github.com/tensorflow/models/tree/master/research/object_detection.

[59] Ultralytics. 2021. YOLOv3 in PyTorch. https://github.com/ultralytics/yolov3.

[60] Shiyao Wang, Yucong Zhou, Junjie Yan, and Zhidong Deng. 2018. Fully motion-aware network for video object detection. In *Proceedings of the European conference on computer vision (ECCV)*. 542–557.

[61] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. 2017. Simple online and realtime tracking with a deep association metric. In *2017 IEEE international conference on image processing (ICIP)*. IEEE, 3645–3649.

[62] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. 2019. FBNet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 10734–10742.

[63] Haiping Wu, Yuntao Chen, Naiyan Wang, and Zhaoxiang Zhang. 2019. Sequence level semantics aggregation for video object detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 9217–9225.

[64] Ran Xu. 2020. ApproxDet: Content and Contention-Aware Approximate Object Detection for Mobiles. https://github.com/StarsThu2016/ApproxDet.

[65] Ran Xu, Jinkyu Koo, Rakesh Kumar, Peter Bai, Subrata Mitra, Ganga Meghanath, and Saurabh Bagchi. 2019. ApproxNet: Content and Contention Aware Video Analytics System for the Edge. *arXiv preprint arXiv:1909.02068* (2019).

[66] Ran Xu, Chen-lin Zhang, Pengcheng Wang, Jayoung Lee, Subrata Mitra, Somali Chaterji, Yin Li, and Saurabh Bagchi. 2020. ApproxDet: content and contention-aware approximate object detection for mobiles. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems (SenSys)*. 449–462.

[67] Le Yang, Yizeng Han, Xi Chen, Shiji Song, Jifeng Dai, and Gao Huang. 2020. Resolution Adaptive Networks for Efficient Inference. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2369–2378.

[68] Chun-Han Yao, Chen Fang, Xiaohui Shen, Yangyue Wan, and Ming-Hsuan Yang. 2020. Video Object Detection via Object-Level Temporal Aggregation. In *European Conference on Computer Vision*. Springer, 160–177.

[69] Shuai Yu, Rami Langar, Xiaoming Fu, Li Wang, and Zhu Han. 2018. Computation offloading with data caching enhancement for mobile edge computing. *IEEE Transactions on Vehicular Technology* 67, 11 (2018), 11098–11112.

[70] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael J Freedman. 2017. Live video analytics at scale with approximation and delay-tolerance. In *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*. 377–392.

[71] Shifeng Zhang, Longyin Wen, Xiao Bian, Zhen Lei, and Stan Z Li. 2018. Single-shot refinement neural network for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 4203–4212.

[72] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. 2018. ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[73] Zhong-Qiu Zhao, Peng Zheng, Shou-tao Xu, and Xindong Wu. 2019. Object detection with deep learning: A review. *IEEE transactions on neural networks and learning systems* 30, 11 (2019), 3212–3232.

[74] Xizhou Zhu, Jifeng Dai, Lu Yuan, and Yichen Wei. 2018. Towards high performance video object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 7210–7218.

[75] Xizhou Zhu, Yujie Wang, Jifeng Dai, Lu Yuan, and Yichen Wei. 2017. Flow-guided feature aggregation for video object detection. In *Proceedings of the IEEE International Conference on Computer Vision*. 408–417.

[76] Xizhou Zhu, Yuwen Xiong, Jifeng Dai, Lu Yuan, and Yichen Wei. 2017. Deep feature flow for video recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2349–2358.