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

Ran Xu
xu943@purdue.edu
Purdue University
West Lafayette, Indiana, USA

Jayoung Lee
lee3716@purdue.edu
Purdue University
West Lafayette, Indiana, USA

Pengcheng Wang
wang4495@purdue.edu
Purdue University
West Lafayette, Indiana, USA

Saurabh Bagchi
sbagchi@purdue.edu
Purdue University
West Lafayette, Indiana, USA

Yin Li
yin.li@wisc.edu
University of Wisconsin-Madison
Madison, Wisconsin, USA

Somali Chaterji
schaterji@purdue.edu
Purdue University
West Lafayette, Indiana, USA

Abstract

An adaptive video object detection system selects different execution paths at runtime, based on video content and available resources, so as to maximize accuracy under a target latency objective (*e.g.,* 30 frames per second). Such a system is well suited to mobile devices with limited computing resources, and often running multiple contending applications. Existing solutions suffer from two major drawbacks. *First*, collecting feature values to decide on an execution branch is expensive. *Second*, there is a switching overhead for transitioning between branches and this overhead depends on the transition pair. *LiteReconfig*, an efficient and adaptive video object detection framework, addresses these challenges. *LiteReconfig* features a cost-benefit analyzer to decide which features to use, and which execution branch to run, at inference time. Furthermore, *LiteReconfig* has a content-aware accuracy prediction model, to select an execution branch tailored for frames in a video stream. We demonstrate that *LiteReconfig* achieves significantly improved accuracy under a set of varying latency objectives than existing systems, while maintaining up to 50 fps on an NVIDIA AGX Xavier board. Our code, with DOI, is available at https://doi.org/10.5281/zenodo.6345733.

CCS Concepts: • **Computer systems organization → Embedded software**; • **Human-centered computing → Ubiquitous and mobile computing systems and tools**; • **Computing methodologies → Computer vision problems**.

EuroSys '22, April 5–8, 2022, RENNES, France

ACM ISBN 978-1-4503-9162-7/22/04.
https://doi.org/10.1145/3492321.3519577

Keywords: Video Analytics, Mobile Vision, Reconfiguration, Object Detection, Approximate Computing, Latency-Sensitive Analytics

ACM Reference Format:
Ran Xu, Jayoung Lee, Pengcheng Wang, Saurabh Bagchi, Yin Li, and Somali Chaterji. 2022. LiteReconfig: Cost and Content Aware Reconfiguration of Video Object Detection Systems for Mobile GPUs. In *Seventeenth European Conference on Computer Systems (EuroSys '22), April 5–8, 2022, RENNES, France.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3492321.3519577

1 Introduction

Video object detection on mobiles has attracted considerable attention in recent years. 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 that decides which branch to use, based on video features and the user's latency objectives. Much progress has been made in developing light-weight models and systems that are capable of running on mobile devices with moderate computation capabilities [15, 20, 56, 64, 73]. Previous work focuses on statically optimized models and systems [4, 39, 58], pushing the frontiers of accuracy and efficiency. More recently, adaptive object detection models and systems [5, 9, 10, 24, 67, 68] have emerged. These are capable of achieving different points in the accuracy-latency tradeoff space, and are thus suited to mobile devices under real-world conditions: adapting to dynamically changing content, resource availability on the device, and user's latency objectives.

Despite recent advances, no previous work considers the contending pulls of the accuracy-latency frontier of the adaptive (MBEK) vision system, on the one hand, and the latency cost of the scheduler itself, on the other. Previous work faces two fundamental challenges. *First*, the scheduler relies on

computationally light video features (*e.g.,* height, width, number of objects, or intermediate results of the execution kernel), to decide which branch to run. Such features might not be sufficiently informative. Other features or models, such as motion and appearance features of the video, can improve decision making, but are typically too heavy-weight. For example, extracting a high-dimensional Histogram of Oriented Gradient (HOG) and executing the associated models (*i.e.,* other modules of the scheduler that select the execution branch) takes 30.25 ms (Table 1) on the Jetson TX2, nearly the time of one video frame.

Second, if the conditions change frequently, the scheduler incurs high switching overhead between execution branches. Thus, a cost-aware scheduler should tamp down the frequency of reconfigurations based on the cost, which itself can vary depending on the execution branch. Prior work has not considered a cost-aware design of the scheduler and, as we show empirically, this leads to sub-optimal performance.

To address these challenges, we develop *LiteReconfig*, which is tailored to embedded boards with mobile GPUs. LiteReconfig provides a cost-benefit analysis that allows it 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 latency cost and the benefit (in terms of accuracy) of using computationally-heavy content features. By wisely enabling content features and 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 video content. Furthermore, LiteReconfig analyzes the cost-benefit overhead of switching execution branches when conditions change.

Thus, Figure 2 shows the accuracy vs. latency curve (higher is better). Here we see that the content-agnostic version is worse than the ResNet content-aware strategy. Note in contrast that the MobileNet content-aware option is worse than the content-agnostic version, indicating the need for a rational decision on *which* content features to include. Despite a seemingly higher cost than MobileNet, the ResNet features come from the object detector in the MBEK, and thus only incur minor additional extraction and model prediction costs. This makes the content-aware version with the ResNet option, the winning one. 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 the gains from the optimization. We evaluate our approach on the ImageNet VID 2015 benchmark, and compare with SSD [40] and YOLOv3 [49], which we enhance by incorporating tuning knobs to run at different points in the latency-accuracy spectrum (*e.g.,* tuning knobs such as shape of video frame and size of GoF (Group-of-Frames)). We also compare to a recent adaptive model [68] with the Faster R-CNN backbone. The evaluation uses the Jetson TX2 and Jetson AGX Xavier boards with mobile GPUs. LiteReconfig improves accuracy by 1.8% to 3.5% mean average precision (mAP) over state-of-the-art (SOTA) adaptive object detection systems, under identical latency objective. Under contention for the GPU resource, the SSD and YOLOv3 baselines completely fail to meet the latency objective. Compared to three recent accuracy-focused object detection systems, SELSA [65], MEGA [4], and REPP [53], LiteReconfig is 74.9×, 30.5×, and 20.3× faster on the Jetson TX2 board.

Contributions. We summarize our contributions as follows.

1. A cost-benefit analyzer that enables low-cost online reconfiguration. This design reduces scheduler cost and increases accuracy, since more of the latency budget can be devoted to the object detection kernel.
2. A content-aware accuracy prediction model of the execution branch, so that the scheduler selects a branch tailored to the video content. Such a model is built on computationally-heavy features and integrates well with our cost-benefit analysis.
3. Extensive experimental evaluation on two mobile GPU boards and against a set of previous approaches. It underscores two key insights (i) it is important to consider the effect of contention from co-located applications, and (ii) it is important to engineer which features to use for making the selection of the execution branch; this is especially essential for the incorporation of content-aware features, which also have a high computational overhead. The full implementation of LiteReconfig is able to satisfy even stringent latency objectives, 30 fps on the weaker TX2 board, and 50 fps on the higher performing AGX Xavier board.

2 System model, background, and requirements

We provide background on video object detection algorithms, content-aware video object detection models, and adaptive vision systems. *Readers who are knowledgeable about video object detection may skip the rest of this section.*

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 classify each instance into pre-defined categories. Convolutional neural networks (CNNs) are popular, and can be separated into two parts: a backbone network, which extracts features from images, and a detection network, which classifies object regions based on the extracted features. The detection network can be further categorized into two-stage [6, 17, 50] or single-stage [36, 40, 49, 58, 72] models.

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 can be leveraged to approximate the computations, or to enhance detection in neighboring frames [11,

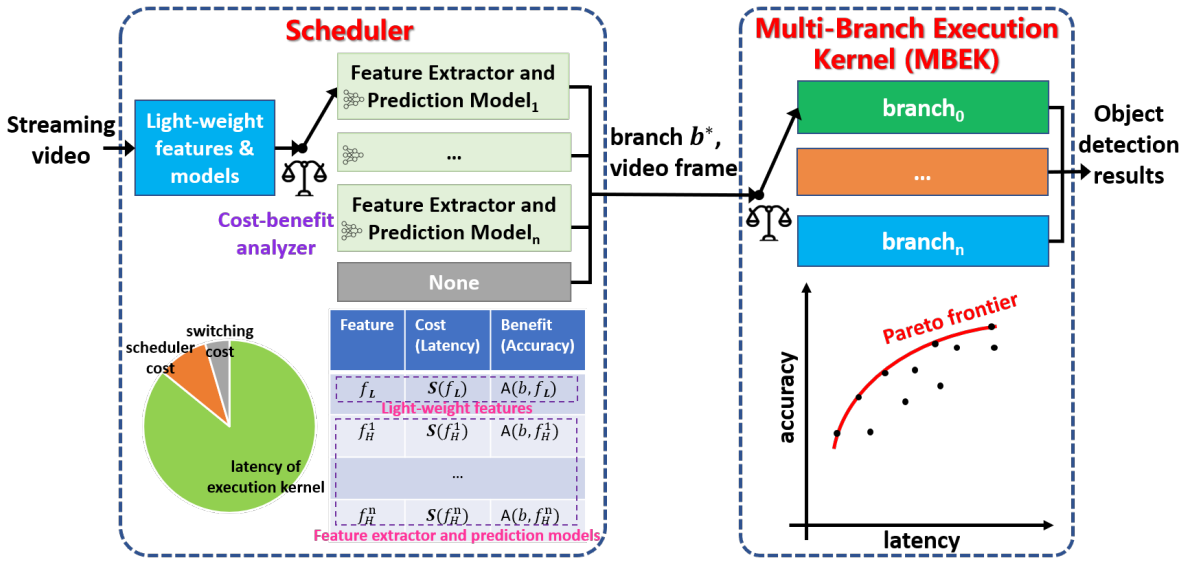


Figure 1: An illustration of our proposed cost-aware adaptive framework for video object detection. Our scheduler uses its cost-benefit analysis to decide on which features to use for making a decision and then makes a decision on which execution branch to run for detection. The multi-branch execution 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.

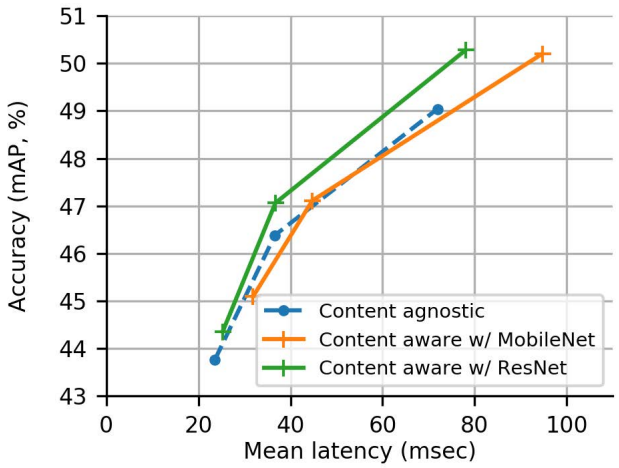


Figure 2: **Motivation of cost-benefit analysis**. We plot the accuracy vs. latency curve for three different strategies. Without a careful design, a content-aware strategy can be either better (*e.g.*, ResNet) or worse (*e.g.*, MobileNet) than a content-agnostic one. Here, the ResNet50 features come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

28, 38, 62, 69, 76, 77]. Many previous approaches optimize for accuracy, explore temporal aggregation of object features [69], using either recurrent neural networks [38], or motion analysis [62]. More practical solutions integrate object detection with visual tracking [11, 37, 68, 71], where inexpensive trackers connect costly object detection outputs. Recent papers [8, 12, 41] generally improve 1-2% mAP in accuracy.

2.2 Content-Aware Video Object Detection Models

As discussed in Sec. 2.1, videos come with inherent information within a series of contiguous frames. For example, the scale of objects, the moving speed, the complexity, etc. Therefore, some video object detection models [5, 38, 68] utilize the content information in videos, to improve latency and accuracy. We call such models *content-aware video object detection systems*. At model inference time, a content-aware system reconfigures itself based on the content information from the video stream. Instead, a content-agnostic system uses a static model variant or branch.

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 can 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

Video object detection algorithms have good accuracy and latency on server-class machines. However, existing approaches suffer when running on edge or mobile devices, particularly under a tight latency service level objective (SLO) and under varying resource contention. There has been significant work on developing continuous vision applications on mobile or resource-constrained devices — some with manual-crafted network architectures [15, 20, 73] and some

with models given by neural architecture search [35, 56, 64]. To further optimize the efficiency, additional techniques have been applied to provide adaptation to the deep models. Examples include, tuning the size of the input or other model parameters [5, 24, 67, 68] at inference time, prune a static DNN into multiple DNNs that could be dynamically selected [10], or select a different exit within a network [21, 59].

These adaptive video object detection frameworks usually feature multi-models or multi-branches as part of their design. However, 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 decision making for selecting the optimal branch needs to be cost- and content-aware, and light-weight, to minimize the overhead. This is what prior work is lacking.

2.4 Terminology in Adaptive Vision Systems

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

An Execution Branch is a distinct setting of an algorithm, typically differentiated by controlling some hyperparameters (colloquially, "knobs"), so as to finish the vision task in a distinct and fixed execution time (latency) and a consistent accuracy[1] across a dataset. Models with multiple execution branches are often considered by adaptive object detection frameworks. We adapt the design of the MBEK from ApproxDet, where an object tracking algorithm, *i.e.,* the object tracker, is paired with the object detector to greatly reduce the latency while preserving the accuracy. An execution branch might, for example, specify the choice of object detector and/or object tracker, the input shapes of video frames fed into them, the number of frames in a GoF that runs the object detector (always on the first frame of the GoF) and the object tracker (on the remaining frames of the GoF), and the number of region proposals in the object detector.

Accuracy-Latency Trade-off: The trade-off between accuracy and latency is fundamental to adaptive vision systems. If a higher accuracy is desired, one has to incur higher latency. Each execution branch has an associated accuracy and latency 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. This is because the Pareto curve presents most accurate branches for certain latency SLOs. Note that the Pareto curve is subject to change at runtime due to the dynamic content characteristics and resource contention. We will discuss in Sections 3.3 and 3.4 how our systems take content characteristics into the design of the scheduler to predict accuracy-latency values correctly.

Scheduler: The scheduler in our adaptive vision system is the key component to determine the execution branch on-the-fly, so as to achieve the optimal reachable accuracy-latency point. None of the prior work has explored the cost-aware design of the scheduler, resulting in sub-optimal performance. *Fundamentally, this is the key contribution that* LiteReconfig *makes.*

[1]Consistency means a branch with higher accuracy in the training dataset is expected to have higher accuracy in the test dataset.

3 Design

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 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.

The scheduler synthesizes the outputs from the above two parts and determines which execution branch to invoke. Specifically, it first performs cost-benefit analysis to choose a set of features, then predicts accuracy and latency of execution branches based on these selected features, and finally solves a constrained optimization problem (Equation 3) that accounts for switching cost and maximizes the benefit (the improvement of accuracy) of the object detection kernel, such that the latency stays below the SLO.

In Section 3.2-3.3, 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. We then present in Section 3.4 the cost-benefit analysis that enables the scheduler to pick the right set of features, considering their costs and benefits. And finally, we introduce the cost-benefit analysis on the switching cost to guarantee the tail latency SLO in Section 3.5.

3.2 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 object detection, while probabilistically meeting the latency guarantee. The latency guarantee is typically specified in terms of tail latency, like the

95th percentile latency, 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[2], called 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 SLO L_0, given by

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

A critical insight 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 insight 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 itself composed of three parts — (1) the 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 cost of executing corresponding models to predict the accuracy and the latency of each execution branch using 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-weight 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-weight features f_H, which may be extracted based on the cost-benefit analysis. A list of 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-weight features f_L to predict the latency of b. S_0 is the cost of the scheduler that extracts and uses the light-weight 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-weight features as one unit — in reality, the scheduler can recruit any subset of heavy-weight features.

Modeling of Accuracy: Another central component of an adaptive vision system is the accuracy prediction model. Due to the latency SLO, in much of prior work, only features that are light-weight to compute are considered for modeling the accuracy of the execution branches [10, 16, 46, 68].

Our key observation is that more expressive and 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) [45], Histogram of Oriented Gradient (HOG) [7], recent neural network based features, like MobileNetV2 [54] (details in Table 1), can provide significantly better-accuracy prediction. We call the model using such heavy-weight content features the *content-aware accuracy model*. In addition to the three external feature extractors, we also use two features from the Faster R-CNN detector in the MBEK — ResNet50 and Class Predictions on Proposal (CPoP) feature. They are less computationally costly to collect as these are obtained directly from the object detector component of the MBEK, as opposed to other features extracted on-the-fly (HoC, HOG, and MobileNetV2), and they turn out to be informative features to characterize the accuracy of each branch in the MBEK.

3.3 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 ApproxDet [68]), 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[3], 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 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-weight (f_L) or a subset of the heavy-weight features (f_H).

The accuracy prediction model $A(b, f)$ is realized with a 6-layer neural network. The first layer uses fully-connected projections to project the low-dimensional light-weight features and high-dimensional content features to the same dimension, and then concatenates them. All rest layers are fully connected with ReLU as the activation function.

[2]The window size is also a tuning knob determined by the scheduler.

[3]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	Execution time (or cost, in ms)		Description
		Extract	Predict	
Light-weight, f_L	Light, 4	0.12	3.71	Composed of height, width, number of objects, averaged size of the objects.
Heavy-weight, f_H^1	HoC, 768	14.14	4.94	Histogram of Color on red, green, blue channels.
Heavy-weight, f_H^2	HOG, 5400	25.32	4.93	Histogram of Oriented Gradients.
Heavy-weight, 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-weight, 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-weight, 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 extraction cost is the averaged execution time to extract the feature and the prediction cost is the averaged execution time to predict the accuracy of each branch given the features. The execution time 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.

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

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

To solve this optimization, we examine all branches $\{b\}$ that satisfy the latency constraint[4] and pick the branches with highest predicted accuracy $A(b, f)$. Note that the latency prediction model $L_0(b, f_L)$ incorporates light-weight features f_L but does not rely on the heavy-weight 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 from our offline dataset[5]. In the following sections, we discuss (1) the algorithm for selecting which features f to choose for scheduling (Section 3.4) and (2) the modeling of the switching cost $C(b_0, b)$ (Section 3.5).

[4]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 total cost by 5%.

[5]Our innovation lies in the design of the optimization problem. To solve it, we use standard convex solver.

3.4 Feature Selection for Scheduling

Recall that selecting the features used by the scheduler should consider the relative cost and the benefit of including various features. LiteReconfig dynamically decides which features to use during runtime, based on current video content characteristics and latency objective.

Light-weight vs. Heavy-weight Features: The light-weight features f_L can be extracted without adding cost and its corresponding content-agnostic accuracy prediction is also computationally light-weight (*e.g.,* the dimension of the image). Heavy-weight features f_H are content dependent and need processing of the video frame, including costly neural network-based processing (*e.g.,* MobileNetV2 feature of a video frame). As is well known in the literature [22, 50, 74], accuracy is enhanced with content-dependent features, such as HoC, HOG, MobileNet, and ResNet. We show empirically that this improvement happens under many scenarios, but not all. Furthermore, one has to account for the decrease in the latency budget of the execution kernel due to the overhead of the features themselves. This is the key idea behind our feature selection algorithm, which maximizes the accuracy of the selected branch in the execution kernel, with overhead considered.

Table 1 shows that HoC, HOG, and MobileNetV2 features take 14.14 ms, 25.32 ms, and 153.96 ms respectively, and the corresponding prediction models on these features take 4.94 ms, 4.93 ms, and 9.33 ms respectively. This is because these features are high-dimensional to encode. Such costs can be overwhelming especially when the continuous vision system is running under a strict latency budget, say 33.3 ms (30 fps). Supposing the scheduler is triggered at every

first frame of a GoF of size 8 (a middle-of-the-range number), the MobileNetV2 feature extraction plus prediction take 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: A key challenge is that feature selection must work *without actually extracting the heavy-weight features* or querying the corresponding models. To address this challenge, we take some pragmatic simplifications.

Let the set of all possible features $\mathcal{F}$, consisting of light-weight features f_L and a set of heavy-weight feature candidates $\mathcal{F}_\mathcal{H}$. Our algorithm will always use the light-weight features f_L and then determine which subset of heavy-weight features $f_H \in 2^{\mathcal{F}_\mathcal{H}}$ to use. It is possible that $f_H = \emptyset$. We first extract the light-weight 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-weight features is f_H^S. The optimization is given by

$$f_H^i = \underset{f_H \in \mathcal{F}_\mathcal{H} \setminus f_H^S}{\operatorname{argmax}} \max_{b \in \mathcal{B}} A(b, f_L) + Ben(f_H^S \cup f_H)$$
$$s.t.\ \ L_0(b, f_L) + S(f_L) + S(f_H^S \cup f_H) + C(b_0, b) \leq L_0.$$

$Ben(f_H^S \cup f_H)$ is the benefit (improvement in accuracy) of including additional features f_H. $S(f_L)$ is the cost to extract and use light-weight features f_L; $S(f_H^S \cup f_H)$ is the cost for heavy-weight 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-weight features in Equation 3.4. Concretely, this benefit depends on the content features and should ideally be calculated by extracting the heavy-weight 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-weight features and executing the corresponding content-aware accuracy prediction model. The benefit function $Ben(f_H^S \cup f_H)$ is collected from a *offline* dataset to reflect the accuracy improvement of the system with the heavy-weight features F against the light-weight feature f_L. To further reduce the online cost, these are all implemented using lookup tables.

3.5 Modeling Switching Cost

In contrast to prior work, LiteReconfig additionally considers the latency of switching from branches or models in its cost-benefit analysis. We observe indeed from Figure 5 that different branch transitions have different costs[6]. Considering switching from branch b_0 to b, the switching overhead is the difference between the latency of branch b in its first inference run, and the mean latency of b in the subsequent inference runs. This is estimated offline, as it is static. It depends on the implementation and the nature of execution branches, and varies with size of non-shared data structure such as disjoint parts of a TensorFlow graph. We perform a cost-benefit analysis by including the term $C(b_0, b)$, *i.e.,* the cost of switching in latency (execution time) terms, in the total cost formulation. The data is again collected from the offline training dataset.

Our model of switching cost considers only the current frame. Due to the unforeseen nature of video, we cannot forecast how long a new branch b stays optimal. Thus, the scheduler re-evaluates after every tracking-by-detection GoF. Empirically, this works better than optimizing over a look-ahead window by predicting future workload changes [29, 51] . Indeed, the latter approaches are inaccurate and have a high cost. Furthermore, re-evaluating every GoF (typically 4-20 frames) mitigates the impact of an incorrect decision.

4 Implementation

We implement LiteReconfig on top of an 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 re-implement the latency predictors in ApproxDet [68] on our embedded devices 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 from the first frame of each video snippet as the model input[7]. 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-weight 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 a Stochastic Gradient descent (SGD) optimizer, with momentum of 0.9, to train the neural network, and use ℓ_2 regularization to prevent

[6] All branches and models are loaded and preheated with several video frames in the beginning. Thus, the cost reflects the true transition time.

[7] The scheduler must predict at the first frame of a video snippet and has access only to the feature of the first frame.

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 and a more powerful NVIDIA Jetson AGX Xavier. TX2 has a 256-core NVIDIA Pascal GPU on a 8GB unified memory, whereas 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).

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.

5.1 Baselines

We consider the following baselines for evaluating LiteReconfig.

1. **ApproxDet**: is the state-of-the-art adaptive object detection framework on embedded devices [68]. ApproxDet uses Faster R-CNN [50] as its backbone object detector, and is able to adapt to latency objectives by changing the image resolution (*shape*), and the number of proposals (*nprop*) in the first RPN, and by combining different tracker types (LiteReconfig uses the same four types as ApproxDet) with the detector, coupled with varying GoF, and downsampling ratio, of the image fed into the tracker. We use ApproxDet's open-sourced implementation [66], which uses TensorFlow-gpu v1.14.0.
2. **AdaScale** by adaptively re-scaling the input image to one of a number of preset resolutions can perform inference at different latencies [5]. The primary focus of AdaScale is not efficiency, and thus its base latency values (with no contention) on embedded boards are too high, making it irrelevant to be compared with LiteReconfig under resource contention (Table 3). Therefore, we execute AdaScale, and its variants (scales of 600, 480, 360, and 240, which correspond to the number of pixels of the shortest side) on the TX2 board, without contention, and compare their mAP and latency values.
3. **YOLO+**: YOLOv3 [49] is a fast one-stage object detector. However, it is far from real-time on our embedded devices. YOLO+ is our version of YOLOv3, improved, by exposing four tuning knobs, similar to ApproxDet. The tuning knobs are: *shape* of video frame, size of GoF (*si*), type of *tracker*, and downsampling (*ds*) ratio. YOLO+ is adaptive to different latency SLOs, but not to resource contention. The YOLO implementation is YOLOv3 in PyTorch v1.4.0 from Ultralytics [61].
4. **SSD+**: SSD [40] is a one-stage object detector that combines with multiple feature extractor backbones. We use MobileNetV2 as the backbone, combined with MnasFPN [3] to further improve the object detector. SSD+ is our improved version of SSD by exposing the same tuning knobs as for YOLO, and an additional tuning knob, the confidence threshold of the detector (which controls the number of objects to be tracked). SSD+ is adaptive to different latency SLOs as well, but not to resource contention. The SSD implementation is with TensorFlow v1.14, following the official implementation from TensorFlow Object Detection API [60].
5. **EfficientDet**: EfficientDet [58] is a recent SOTA object detector, focusing on model efficiency, and consisting of 8 variants (*D0-D7*), in increasing order of computational complexity. 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, respectively, within the executable candidates.
6. **SELSA** [65], **MEGA** [4], **REPP** [53]: are 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 not suitable under resource contention on our embedded boards — their base latency is already too high and they crash or hang, when subjected to resource contention.

5.2 Evaluation Setup

Dataset and Metrics: We evaluate LiteReconfig on the ILSVRC 2015 VID dataset [52], which contains 3,862 videos in the training set, and 555 videos in the validation set. Both datasets are fully annotated, with class names, and location of each object in the video frame. We use 90% of the ILSVRC training dataset [8] to train the vision algorithms (detection backbones), including all baselines, and the remaining 10%

[8] Every 9 out of 10 videos in the alphabetic order of video names

Device and latency SLOs (ms)	GPU resource contention	Models	mAP (%)	P95 latency per-frame (ms)
TX2, 33.3/50.0/100.0	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
TX2, 33.3/50.0/100.0	50%	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	F /**43.6/47.0**	*34.0*/49.5/78.2
AGX Xavier, 20.0/33.3/50.0	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
AGX Xavier, 20.0/33.3/50.0	50%	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 SLO and thus the accuracy results are not comparable. The bold text of mAP shows the highest accuracy in each scenario and objective, while the italicized text of latency highlights that the 95% latency SLO is violated. An "F" in a latency cell means that that protocol did not execute at all.

of the training dataset to train the scheduler, *i.e.,* 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-weight features $Ben(F)$. The validation set is set aside for the evaluation only.

Our accuracy metric is the mAP on the VID validation dataset, following widely adopted protocols [4, 62, 75–77]. We report the average per-frame execution time (latency) over a GoF as the time metric. We use the 95th percentile (*P95*) latency when evaluating LiteReconfig, because our scheduler aims to guarantee a SLO violation rate < 5%. This is standard when evaluating latency-sensitive ML systems [27, 30, 33, 47]. To compare fairly against accuracy-optimized models (SELSA, REPP, EfficientDet, and AdaScale), we use the mean latency. The latency we report includes all relevant parts, such as the feature extractor, the model execution, and the scheduler.

5.3 End-to-end Evaluation

Evaluation with different SLOs and no resource contention on the TX2. We first evaluate LiteReconfig for accuracy, by varying latency objective from 33.3 ms (*tight*, for real-time 30 fps) to 50 ms (*medium*, for 20 fps) to 100 ms (*loose*, for 10 fps) per frame on the NVIDIA Jetson TX2 board, where no resource contention is injected. Table 2 summarizes the comparison, where an mAP row contains three results separated by "/", each corresponding to different latency SLOs; similarly for every *P95* latency row. A note of "F" means that this algorithm failed to meet the objective. Observe that LiteReconfig and our enhanced baselines (SSD+ and YOLO+) improve efficiency over the SOTA ApproxDet from 33.3 ms to 100 ms. LiteReconfig's improvement is higher, thanks to its superior object detector, feature extractors, and cost-benefit analyzer.

LiteReconfig achieves 3.5% mAP improvement over ApproxDet under the 100 ms objective — a significant improvement for an object detection task. LiteReconfig achieves a

Models, latency SLO	mAP (%)	Mean latency (ms)	Memory (GB)
SELSA-ResNet-101 [65], no SLO	81.5	2334	6.91
SELSA-ResNet-50, no SLO	77.31	2112	6.70
MEGA-ResNet-101 [4], no SLO	OOM	OOM	9.38
MEGA-ResNet-50, no SLO	OOM	OOM	6.42
MEGA-ResNet-50 (base), no SLO	68.11	861	3.16
REPP [53], over FGFA[76], no SLO	OOM	OOM	10.02
REPP, over SELSA	OOM	OOM	8.13
REPP, over YOLOv3 [49]	74.8	565	2.43
EfficienetDet D3	63.9	796	5.68
EfficienetDet D0	55.1	138	2.22
AdaScale-MS, no SLO	56.3	976.4	3.26
AdaScale-SS-600, no SLO	55.7	1049.4	3.20
AdaScale-SS-480, no SLO	59.0	710.5	3.18
AdaScale-SS-360, no SLO	59.4	434.0	3.18
AdaScale-SS-240, no SLO	56.5	227.9	3.18
LiteReconfig, 100 ms	50.3	72.0	3.67
LiteReconfig, 50 ms	46.5	38.4	4.09
LiteReconfig, 33.3 ms	45.4	28.2	4.12

Table 3: Performance comparison between LiteReconfig and the video object detection solutions optimized for accuracy. mAP means the mean average precision and OOM stands for out-of-memory errors.

consistent accuracy improvement over the content-agnostic variant, *i.e.,* LiteReconfig-MinCost, by 1.6%, 0.1%, and 1.3% mAP, respectively. Among all of LiteReconfig's four variants and our enhanced baselines, LiteReconfig and SSD+ achieve the highest accuracy under 33.3 ms objective, LiteReconfig-MaxContent-ResNet is the most accurate under the 50 ms objective, and LiteReconfig is again the most accurate under 100 ms objective. Finally, though LiteReconfig-MaxContent variants violate the latency objective due to costs of running feature extractors and models, *LiteReconfig is strictly always below the latency objective.*

To summarize, in addition to efficiency improvement over the SOTA, our cost and content-aware solution aces accuracy frontiers, with the latency values satisfying the objective.

Evaluation with higher GPU contention on the TX2. We extend our evaluation to a high resource-contention scenario, where 50% GPU resource is busy with other concurrent applications. Table 2 shows that under contention on TX2, our efficiency-enhanced baselines (SSD+ and YOLO+) fail to meet the latency objectives due to lack of contention-aware adaptation. LiteReconfig is 1.8% mAP more accurate than the best adaptive systems. Our full implementation is always one of best protocols among all variants and models, satisfying the latency objective (with the small exception of 34.0 ms under 33.3 ms latency objective and 39.3% mAP in this case).

Evaluation on the AGX Xavier. Finally, we extend our evaluation to a more powerful embedded device, the NVIDIA AGX Xavier. Correspondingly, we tighten the latency objective to as low as 20 ms (corresponding to 50 fps). We find that ApproxDet cannot meet any of the three latency objectives. We observe again that LiteReconfig and the MaxContent-ResNet variants both have the highest accuracy under different latency objectives and contention scenarios, and can meet the latency objectives. On the other hand, the enhanced baselines and MaxContent-MobileNet variant violate two stringent objectives.

To summarize, LiteReconfig is always the highest performing content-aware variant, being 1.0% mAP superior to LiteReconfig-MaxContent-ResNet, given no contention, and 33.3 ms SLO. It is 2.2% mAP better, given 50% GPU contention and 50.0 ms SLO on TX2. LiteReconfig is also 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), ApproxDet, SSD+, and YOLO+, in most cases.

Comparison to Recent, Accuracy-Optimized Object Detection Algorithms: Table 3 compares LiteReconfig to recent solutions optimized for accuracy, which however do not achieve real-time efficiency on the embedded device. AdaScale, one of the content-aware baselines, whose fastest variant always using the smallest scale, runs at a latency of 227.9 ms 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 lower than all accuracy-optimized baselines. In addition, EfficientDet-D0 and Efficient-D3 achieve reasonable accuracy while maintaining relatively low latency, relative to other accuracy-optimized solutions, such as SELSA [65], MEGA [4], and REPP [53]. EfficientDet-D0 runs at 138 ms per frame, at an mAP of 55.1%, with a latency close to the SLO (albeit higher), while the accuracy is higher compared to LiteReconfig. EfficientDet-D3 runs at 796 ms per frame, at an mAP of 63.9%, with the latency in an unacceptable range for on-device inferencing. Thus, with no adaptive features, both EfficientDet-D0 and EfficientDet-D3 fail to match the 100 ms latency objective.

Compared to most accurate models SELSA, MEGA and REPP, LiteReconfig is 90.3×, 36.8×, and 24.1× faster (for SLO of 33.3 ms). On the flip side, the accuracy of LiteReconfig is significantly lower. Thus, 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 could be selected. Note that we measure mAP values lower than those published by their authors (3.2% lower for SELSA, 9.2% for MEGA, and 23.8% for REPP), even though we used pre-trained models. The accuracy reduction in Table 3 results from three factors: (1) the change of the backbone feature extractors from ResNet-101 to ResNet-50 due to memory constraints on the TX2 and (2) the removal of the part of the design that refers to future frames because our problem context requires streaming and real-time processing. These

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 LiteReconfig of individual content-specific features by comparing the accuracy given different latency objectives.

algorithms benefit from using future frames to detect objects in the current frame.

5.4 Evaluation of Video Content Features

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 be achieved, with the latency objective 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 ones (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 objective. Particularly, HoC and HOG feature extractors and predictors are beneficial when the latency budget is 50.0 ms and 100.0 ms since they come with a lower cost relative to the MobileNetV2 counterpart. To precisely account for the cost and benefit of feature extractors, the selection of features comes from our constrained optimization solution and is not manually selected. 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 the overhead from the content-aware features.

5.5 Understanding Latency-Accuracy Tradeoff

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 SLO) 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 ms latencies because it cannot satisfy those SLOs. 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 objectives (50 ms and 100 ms). Finally, LiteReconfig wins over YOLO+, SSD+, and ApproxDet, due to higher latency SLO assigned to the object

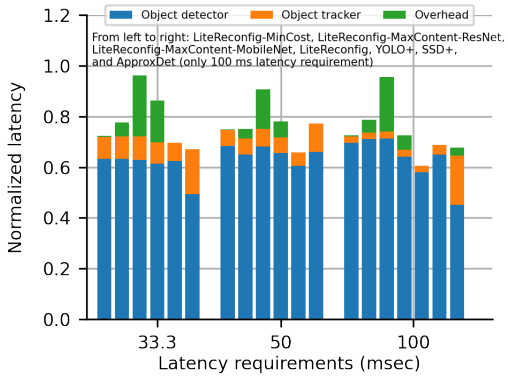


Figure 3: Percentage latency of each system component, normalized over the latency SLO, profiled on the TX2. FRCNN and YOLO cannot meet the 33.3 ms SLO and thus their bars are missing.

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 SLO is specified in terms of 95-th percentile latency (P95). Thus, LiteReconfig is using up its latency budget prudently, being conservative to avoid frequent SLO violations.

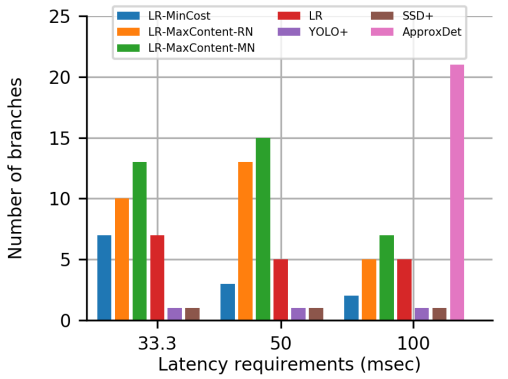
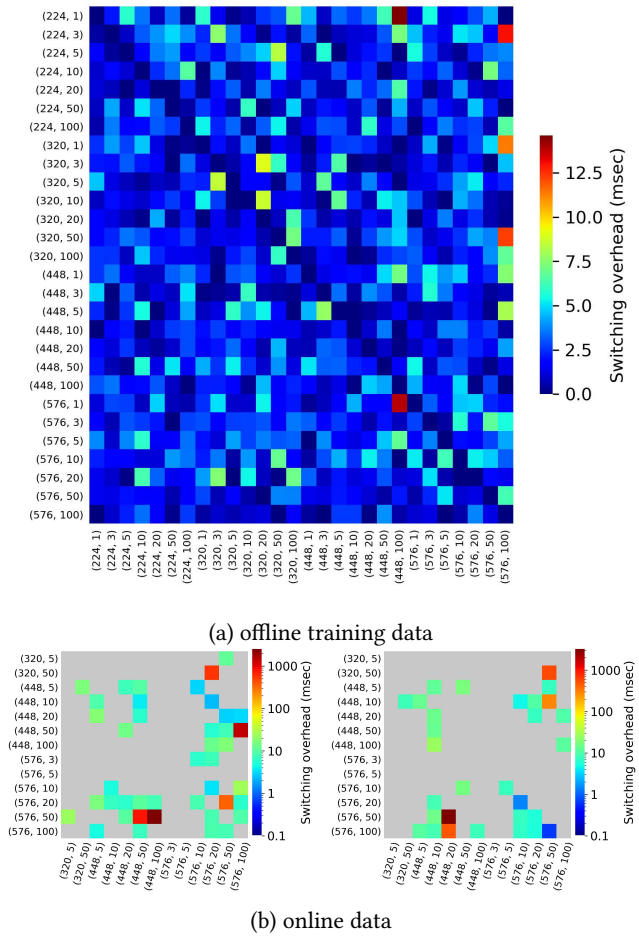


Figure 4: Branch coverage between the four variants of LiteReconfig and three other baselines. LiteReconfig is able to explore more beneficial execution branches and avoids fruitless switches between execution branches, leaving greater part of the latency budget available for the execution kernel, the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

We further investigate the branch coverage (*i.e.,* the number of distinct execution branches invoked) within the four LiteReconfig variants. Figure 4 shows that using heavy-weight features (orange and green bars) tend to explore more branches tailored to the video content, driving higher accuracy. However, using light-weight features can reduce latency even further 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 objective. ApproxDet, covering a far higher number of execution branches (100 ms latency objective), is still less accurate than any variant of LiteReconfig. To summarize the end-to-end evaluation we have described in Section 5.3 – Section 5.5, we are able to adapt at a fine granularity to content-based characteristics in the video stream by incorporating more computationally intensive, yet content-aware, features. The cost-benefit analysis of LiteReconfig enables us to squeeze out more accuracy from the content-aware features within the latency objectives. This is instantiated by exploring a greater number execution branches, which are more accurate with respect to their content characteristics.

5.6 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 ms latency SLO on the left and 50 ms latency SLO on the right. 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 ms 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 ms latency objective at the top, and 50 ms latency objective at the bottom. The online data confirms that the switching overhead is mostly less than 10 ms 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, as a testimony to this.

6 Discussion

Generalization to other domains: First, MBEK as a multi-branch solution for the object detection task in this work is a general solution for many similar tasks in other domains, like video recognition, face recognition, and so on. One may expose tuning knobs from a trained model in other domains to make it adaptive and highly efficient on mobile devices. Second, the cost-benefit analysis in the scheduler is also general because the competition of the latency budget between a feature-based scheduler and an MBEK also exists as long as the accuracy-latency trade-off is still a tuning space in the other domains. Schedulers operating on a rich feature space can generally leverage this enhanced feature space to make better decisions as to which branch to select for the MBEK. However, one must take the cost of the schedulers into consideration so as not to offset the benefits of using rich features over simple and cheap features. Thus, without such an overhead-aware scheduler, it will not be possible to leverage rich content features to improve the model's performance on the Pareto frontier.

Online drift in the data: LiteReconfig assumes the online and offline datasets are independent and identically distributed (*iid* distribution), and thus both the MBEK and the scheduler are trained offline and work well in the online phase. If an online drift exists in the data, one may have to re-train LiteReconfig, depending on the source of the drift. For example, if the object classes change, one may re-train the object detector and the accuracy predictor in the scheduler. If the compute capability or runtime environment of the devices change, one may re-train the latency predictor, switching overhead, and cost function in the scheduler.

Contention generator (CG): CG is used as a stand-in for real-world background workloads running on the mobile device. The CG is tunable between 0% and 99% GPU contention, which can represent a wide range of workload scenarios. We empirically found that beyond a contention level of 50%, the video object detection task performs equally poorly. So we evaluate two typical scenarios — 0% and 50% GPU contention.

7 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, which is categorized into two-stage or a single-stage detector depending on the architecture of the DNN. Faster R-CNN [50] is representative of two-stage detectors. Input images, represented as tensors, are passed through pre-trained CNNs, up until an intermediate layer, ending up with a convolutional feature map. Using the features that the CNN computed, Region Proposal Networks (RPN) generate regions-of-interest (RoIs) in the first stage. In the second stage, the RoI pooling layer combines the feature map and the proposals from the RPN, together, to generate positive proposal feature maps, which are then provided to the classifier network. In contrast, a single-stage object detector does not provide region proposal generation, and directly classifies a dense set of pre-defined regions. These models are optimized for lower latency, which can come at the expense of accuracy. For example, the YOLO [48] one-stage network simplifies object detection as a regression problem, by predicting bounding boxes and class probabilities directly without generating region proposals. EfficientDet [58] is a family of (nine) one-stage detector models, with various input image sizes and network depths, all of which achieve SOTA performance [18, 57, 58], measured in FLOPs[9].

However, it remains challenging to apply these image-based detectors to videos, due to motion blur, occlusion, and defocus, which frequently occur in videos. In parallel, visual tracking has evolved from light-weight trackers focusing on motion analysis and trajectory prediction [2, 13, 26] to DNN-based trackers that learn to match the appearance patterns of target objects [11, 34, 63].

These developments have led to works on video object detection that use a tracking-by-detection technique [25, 76], which has become a *de facto* standard. In this technique, an execution plan performs object detection on the current frame, and then, object tracking on the following frames, in a Group-of-Frames (GoF). Such a scheme is utilized to exploit the temporal features in the video [38, 62, 76, 77] or to efficiently reduce the overall cost of the expensive object detector by associating detected objects to the tracker [11, 37, 68, 71]. Unfortunately, the majority of previous methods are meant to run on server-class systems, and only a few solutions exist for edge devices [37, 68]. Video object detection at the edge processes the videos where they are generated, improving latency and decreasing 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 light-weight DNNs to handle resource limits, including hand-crafted network architectures [15, 20, 73], and network architectures built by neural architecture search [35, 56, 64]. 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 [5, 24, 67, 68], within an ensemble of models in a system [10], multi-exit solutions [21, 59, 67], or by generating a light network, specific to a given dataset [9]. Most of the 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 adaptive video object detection solutions exist to date [5, 68].

Some other adaptive vision systems [23, 46, 71] also leverage a multi-branch design methodology. For example, VideoStorm [71] considers two key characteristics of video analytics: resource and quality tradeoff with multi-dimensional configurations. Mainstream [23] 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 [46] 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. These methods, however, lack the ability to be fully adaptive to content and contention changes (*e.g.,* limited to a single dataset or to 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, such as what are the cost and the benefit functions, how easily can the parameters for the functions be collected, and how should the cost-benefit analyses feed into changes made by a scheduler into the system. Some prominent examples in this line of work are for clustered database servers [42, 44], for intermediate data storage for serverless jobs [31, 43], for VM allocation and consolidation [14, 19], and for VM migration [55]. In the context

[9]FLOPs denote the number of multiply-adds [18, 57, 58].

of mobile computing, such cost-benefit analysis has influenced decision making for mobile sensing [1], offloading from mobile to edge or cloud [70], 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.

8 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 under stringent latency objectives, say keeping up with 30 fps video or getting to 20 ms latency for AR/VR applications. We then present our solution, LiteReconfig, which is applicable to any adaptive vision system. LiteReconfig introduces two design innovations. First, it provides the *cost-benefit analysis* to decide on which of the rich set of features can be used to model the accuracy and the latency of different execution branches. Second, the scheduler leverages the cost-benefit analysis to achieve a superior accuracy-*vs*-latency characteristic relative to prior solutions, ApproxDet, EfficientDet, Faster R-CNN, YOLO, SELSA, MEGA, and REPP. Our evaluation provides actionable insights with broad implications — how latency-accuracy models of light-weight vision algorithms can be transferable to different content classes, such as, from fast moving to slow moving videos; how vision frameworks can be designed to better handle resource contention, much of which may be unpredictable; and what the relative utilities of different features are in guiding adaptation in such streaming video analytics systems. Much work remains to be done, such as how our design generalizes to other computer vision tasks or can provide its probabilistic guarantees for different video content.

Acknowledgments

This material is based in part upon work supported by the National Science Foundation under Grant Numbers CCF-1919197, CNS-2038986, CNS-2038566, and CNS-2146449 (NSF CAREER award). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors. The authors thank the reviewers for their enthusiastic comments and the shepherd, Marc Shapiro, for his very diligent passes with us and his valuable insights that improved our paper.

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] 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.

[4] 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.

[5] 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)*.

[6] 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.

[7] 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.

[8] Bin Dong, Fangao Zeng, Tiancai Wang, Xiangyu Zhang, and Yichen Wei. 2021. Solq: Segmenting objects by learning queries. *Advances in Neural Information Processing Systems* 34 (2021).

[9] 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)*. IEEE, 84–95.

[10] 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.

[11] 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.

[12] Golnaz Ghiasi, Yin Cui, Aravind Srinivas, Rui Qian, Tsung-Yi Lin, Ekin D Cubuk, Quoc V Le, and Barret Zoph. 2021. Simple copy-paste is a strong data augmentation method for instance segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2918–2928.

[13] 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.

[14] Ori Hadary, Luke Marshall, Ishai Menache, Abhisek Pan, Esaias E Greeff, David Dion, Star Dorminey, Shailesh Joshi, Yang Chen, Mark Russinovich, and Thomas Moscibroda. 2020. Protean: VM allocation service at scale. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. 845–861.

[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] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[19] 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.

[20] 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.

[21] 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).

[22] 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.

[23] 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)*. 29–42.

[24] 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.

[25] 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.

[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*. 1369–1378.

[35] Ji Lin, Wei-Ming Chen, Yujun Lin, Chuang Gan, Song Han, et al. 2020. Mcunet: Tiny deep learning on iot devices. *Advances in Neural Information Processing Systems* 33 (2020), 11711–11722.

[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] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. 2021. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 10012–10022.

[42] Ashraf Mahgoub, Alexander Michaelson Medoff, Rakesh Kumar, Subrata Mitra, Ana Klimovic, Somali Chaterji, and Saurabh Bagchi. 2020. OPTIMUSCLOUD: Heterogeneous configuration optimization for distributed databases in the cloud. In *2020 USENIX Annual Technical Conference (USENIX ATC)*. 189–203.

[43] Ashraf Mahgoub, Karthick Shankar, Subrata Mitra, Ana Klimovic, Somali Chaterji, and Saurabh Bagchi. 2021. SONIC: Application-aware Data Passing for Chained Serverless Applications. In *2021 USENIX Annual Technical Conference (USENIX ATC)*. 285–301.

[44] 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.

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

[46] 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.

[47] 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.

[48] 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.

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

[50] 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.

[51] 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.

[52] 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.

[53] Alberto Sabater, Luis Montesano, and Ana C Murillo. 2020. Robust and efficient post-processing for video object detection. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 10536–10542.

[54] 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.

[55] 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)*. 143–155.

[56] 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.

[57] Mingxing Tan and Quoc Le. 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*. PMLR, 6105–6114.

[58] 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.

[59] 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.

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

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

[62] 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.

[63] 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.

[64] 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.

[65] 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.

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

[67] Ran Xu, Rakesh Kumar, Pengcheng Wang, Peter Bai, Ganga Meghanath, Somali Chaterji, Subrata Mitra, and Saurabh Bagchi. 2021. ApproxNet: Content and contention-aware video object classification system for embedded clients. *ACM Transactions on Sensor Networks (TOSN)* 18, 1 (2021), 1–27.

[68] 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.

[69] 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.

[70] 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.

[71] 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)*. 377–392.

[72] 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.

[73] 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*. 6848–6856.

[74] 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.

[75] 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.

[76] 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.

[77] 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.

A Artifact Appendix

A.1 Abstract

This appendix mainly describes the artifact of LiteReconfig with four major claims that describe the key performance of LiteReconfig, the accuracy improvement of LiteReconfig over the state-of-the-art (SOTA) works, the latency improvement of LiteReconfig over the accuracy-optimized models, and the accuracy improvement of LiteReconfig over variants of LiteReconfig (ablation study). The artifact is packaged to be used easily with several commands. In particular, Section A.2 describes how to access the code and dataset, and what the hardware and software dependencies are. User needs to go through the setup steps in Section A.3 on the eligible hardware. Finally, Section A.4 explains in detail how to validate the major claims with experiments. Due to the page limit, more detailed step-by-step instructions for the experiments can be found in the "README.md" document in our packaged artifacts at https://doi.org/10.5281/zenodo.6345733.

A.2 Description & Requirements

A.2.1 How to access. The link to our artifact is: https://www.doi.org/10.5281/zenodo.6345733. Please follow the code setup instruction ("README.md") in the artifact.

A.2.2 Hardware dependencies. An NVIDIA Jetson TX2 and an NVIDIA Jetson AGX Xavier board.

A.2.3 Software dependencies. Flash the embedded boards with the default OS and CUDA library. Then, install the virtual environment tool "c4aarch64". Finally, install the Python packages through "pip install" and "conda install". The packages include numpy, numba, tensorboard, tensorflow, opencv, scikit-learn, torch, torchvision, etc. See "INSTALL.md" for more details.

A.2.4 Benchmarks. Our dataset, ILSVRC 2015 VID dataset[10], is not included in our artifacts because of external dependencies. Users may download the dataset from the link above or alternative sources.

A.3 Set-up

Please follow the library setup instruction ("INSTALL.md") in the artifact.

A.4 Evaluation workflow

Once the setup is finished, the evaluation of LiteReconfig, variants of LiteReconfig, and the baselines are generally composed of two steps — (1) running a certain protocol on a certain embedded device, given some constraint, and producing the results files, and (2) extracting the performance metrics (*i.e.,* accuracy and latency) of these protocols with post-processing scripts to understand how good or bad a protocol is. For all experiments, we provide the sample commands to run on our embedded devices (paths may vary on user's own device) and generate results files (step #1 of the workflow) and post-processing scripts to extract key performance metrics from the experiment runs (step #2 of the workflow). For artifact evaluation purposes, we also provide the results files of our evaluation so that the evaluation team can skip the time-consuming step #1 and verify step #2 with our provided files.

A.4.1 Major Claims.

- *(C1): LiteReconfig achieves 45.4% mAP accuracy at 30 fps on the NVIDIA TX2 board under no resource contention for a video object detection task. The accuracy is 46.4% mAP accuracy at 50 fps on the NVIDIA Xavier board. This is proven by experiment (E1) described in Section 5.3 whose results are reported in Table 2.*
- *(C2): LiteReconfig improves the accuracy 1.8% to 3.5% mean average precision (mAP) over the state-of-the-art (SOTA) adaptive object detection systems. This is proven by experiment (E2) described in Section 5.3 whose results are reported in Table 2.*
- *(C3): LiteReconfig is 74.9×, 30.5×, and 20.0× faster than SELSA, MEGA, and REPP on the Jetson TX2 board. This is proven by experiment (E3) described in Section 5.3 whose results are reported in Table 3.*
- *(C4): LiteReconfig is 1.0% and 2.2% mAP better than LiteReconfig-MaxContent-ResNet given (0% contention, 33.3 ms latency SLO) and (50% contention, 50.0 ms latency SLO) cases. This is proven by experiment (E4) described in Section 5.3 whose results are reported in Table 2.*

A.4.2 Experiments.

Experiment (E1) [Key accuracy and latency performance of LiteReconfig] [10 human-minutes + 4 compute-hours]: we will run LiteReconfig on two types of embedded devices and examine the key accuracy and latency performance of it. Expected accuracy and latency on TX2 are 45.4% mAP and < 33.3 ms (95 percentile), and those on Xavier are 46.4% mAP and < 20.0 ms (95 percentile) (claim C1).

On TX2, run the following commands,

```
$ conda activate ae
(ae) $ cd ~/LiteReconfig_AE
(ae) $ python LiteReconfig.py --gl 0 \
  --lat_req 33.3 --mobile_device=tx2 \
  --output=test/executor_LiteReconfig.txt
```

On AGX Xavier, run the following commands (this can be done in parallel with the ones on TX2).

```
$ conda activate ae
(ae) $ cd ~/LiteReconfig_AE
(ae) $ python LiteReconfig.py --gl 0 \
  --lat_req 20 --mobile_device=xv \
  --output=test/executor_LiteReconfig.txt
```

The results will be written to test/executor_LiteReconfig_g0_{lat33_tx2,lat20_xv}_{det,lat}.txt. We have saved a copy of these files in "offline_logs_AE/", and use "python offline_eval_exp1.py" to compute the accuracy and latency from these results files. One may replace the filenames by those in the online execution.

Experiment (E2) [Accuracy improvement at the same latency over the state-of-the-art (SOTA) work, ApproxDet] [20 human-minutes + 12 compute-hours]: we will run LiteReconfig on TX2 given 100 ms latency SLO and examine the true accuracy and latency metrics of it. Then we will compare it with ApproxDet. Expected accuracy and latency of LiteReconfig under no contention is 50.3% mAP and less than 100 ms 95 percentile latency, which is 3.5% higher than that of ApproxDet in the same condition (46.8%). Under 50% GPU contention and 100 ms SLO, the accuracy of LiteReconfig is 47.0%, which is 1.8% mAP higher than that of SmartAdapt (45.2%) (claim C2). On TX2, run the following command,

[10] the public link for the dataset is https://image-net.org/challenges/LSVRC/2015/

```
$ conda activate ae
(ae) $ cd ~/LiteReconfig_AE
(ae) $ python LiteReconfig.py --gl 0 \
  --lat_req 100 --mobile_device=tx2 \
  --output=test/executor_LiteReconfig.txt
(ae) $ python LiteReconfig_CG.py --GPU 50
(ae) $ python LiteReconfig.py --gl 50 \
  --lat_req 100 --mobile_device=tx2 \
  --output=test/executor_LiteReconfig.txt
(ae) $ python LiteReconfig_CG.py --GPU 0
```

The results will be written to test/executor_LiteReconfig_g{0, 50}_lat100_tx2_{det,lat}.txt. Similar post-processing script ("python offline_eval_exp2.py") applies.

Experiment (E3) [Latency improvement of LiteReconfig over accuracy-optimized baselines, i.e. SELSA, MEGA, and REPP] [20 human-minutes + 1 compute-hours]: we will run LiteReconfig on the TX2 and examine the latency performance of it. Expected mean latency of LiteReconfig is 28.2 ms. Those of SELSA, MEGA, and REPP are 2112 ms, 861 ms, and 565 ms. So LiteReconfig achieves 74.9×, 30.5×, and 20.0× speed up over these three baselines (claim C3). On TX2, run the following commands,

```
$ conda activate ae
(ae) $ cd ~/LiteReconfig_AE
(ae) $ python LiteReconfig.py --gl 0 \
  --lat_req 33.3 --mobile_device=tx2 \
  --output=test/executor_LiteReconfig.txt
```

The results will be written to test/executor_LiteReconfig_g0_lat33_tx2_{det,lat}.txt. Similar post-processing script ("python offline_eval_exp3.py") applies.

Experiment (E4) [Accuracy improvement at the same latency over a variant of LiteReconfig, i.e. LiteReconfig-MaxContent-ResNet] [20 human-minutes + 10 compute-hours]: we will run LiteReconfig and LiteReconfig-MaxContent-ResNet on the TX2 and examine the accuracy and latency performance of them. Expected accuracy given no contention and 33.3 ms latency SLO is 45.4% for LiteReconfig and 44.4% for LiteReconfig-MaxContent-ResNet. Expected accuracy given 50% contention and 50 ms latency SLO is 43.6% for LiteReconfig and 41.4% for LiteReconfig-MaxContent-ResNet. Thus, LiteReconfig is 1.0% and 2.2% mAP better than LiteReconfig-MaxContent-ResNet in these two cases (claim C4). On TX2, run the following commands,

```
$ conda activate ae
(ae) $ cd ~/LiteReconfig_AE
(ae) $ python LiteReconfig.py --gl 0 \
  --lat_req 33.3 --mobile_device=tx2 \
  --output=test/executor_LiteReconfig.txt
(ae) $ python LiteReconfig_MaxContent.py \
  --protocol SmartAdapt_RPN --gl 0 \
  --lat_req 33.3 --mobile_device=tx2 \
  --output=test/executor_LR_MC_ResNet.txt
(ae) $ python LiteReconfig_CG.py --GPU 50
(ae) $ python LiteReconfig.py --gl 50 \
  --lat_req 50 --mobile_device=tx2 \
  --output=test/executor_LiteReconfig.txt
(ae) $ python LiteReconfig_MaxContent.py \
  --protocol SmartAdapt_RPN --gl 50 \
  --lat_req 50 --mobile_device=tx2 \
  --output=test/executor_LR_MC_ResNet.txt
(ae) $ python LiteReconfig_CG.py --GPU 0
```

The results will be written to test/executor_{LiteReconfig, LR_MC_ResNet}_{g0_lat33,g50_lat50}_tx2_{det,lat}.txt. Similar post-processing script ("python offline_eval_exp4.py") applies.