APPROXIMATION FOR STREAMING VIDEO ANALYTICS ON MOBILE DEVICES

by

Ran Xu

A Dissertation

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Doctor of Philosophy

School of Electrical and Computer Engineering

West Lafayette, Indiana

December 2021

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Saurabh Bagchi, Chair

School of Electrical and Computer Engineering

Dr. Somali Chaterji

Department of Agricultural and Biological Engineering

Dr. Yin Li

Department of Biostatistics & Medical Informatics, University of Wisconsin-Madison

Dr. Fengqing Maggie Zhu

School of Electrical and Computer Engineering

Dr. Yexiang Xue

Department of Computer Science

Approved by:

Dr. Dimitrios Peroulis

Head of the School of Electrical and Computer Engineering

ACKNOWLEDGMENTS

It has been a long journey since I started in the graduate program at Purdue University in 2016. I would like to acknowledge the people who advanced my career and supported my life. This PhD would not have been possible without them.

First and foremost, I would like to express my utmost gratitude to Prof. Saurabh Bagchi, for the unlimited guidance and support throughout the years that make me a productive researcher. His academic, emotional, and financial support is the key to my success. I will never forget the deadline days when we worked together to finalize pieces of work and joyful moments that our research papers got accepted into the premier conferences and journals. His spirit in research and life will continue to inspire me.

I would also like to express my gratitude to my mentors who advanced my research projects and academic career. My sincere appreciation first goes to Prof. Somali Chaterji, Prof. Yin Li (University of Wisconsin-Madison), and Prof. Sasa Misailovic (University of Illinois at Urbana-Champaign), whose help in my research projects was indispensable. Second, many senior researchers have given me constructive feedback and help during my PhD studies, and I would like to acknowledge them—Dr. Subrata Mitra (Adobe Research), Dr. Jinkyu Koo (while he was at DCSL, now at NVIDIA), Dr. Haoliang Wang (Adobe Research), Dr. Stefano Petrangeli (Adobe Research), Dr. Viswanathan Swaminathan (Adobe Research), Dr. Rajesh Panta (AT&T Research), Dr. Brian Amento (AT&T Research), Dr. Hal Purdy (AT&T Research), and Dr. Jay Ye (Facebook). I would also thank my PhD advisory committee members—Prof. Fengqing Zhu and Prof. Yexiang Xue, the ECE graduate program director, Matt Golden, and the associate head of graduate and professional programs, Prof. Vijay Raghunathan, for their academic advice. I would thank the National Science Foundation, which is the source of funding of my research projects. Finally, I thank Prof. Yong Li at Tsinghua University and Prof. Xinyu Zhang at University of Wisconsin-Madison for advising my earlier research.

My research projects cannot succeed without the many help from my research collaborators. I would first thank all the students who worked with me within DCSL (Dependable Computing Systems Laboratory) and ICAN (Innovatory for Cells and Neural Ma-

chines)–Jayoung Lee, Pengcheng Wang, Rakesh Kumar, Preeti Mukherjee, Akash Melachuri, Sarthak Jain, Peter Bai, Ganga Meghanath, Karthick Shankar, and Ashraf Mahgoub. I would also thank the external collaborating students at the University of Wisconsin-Madison—Chenlin Zhang and Fangzhou Mu.

Last but not least, I am deeply grateful to my beloved parents Donghong Wang and Qibing Xu. Studying abroad for years, I own them a lot for not being able to accompany and reward them. I wish this dissertation and my PhD degree would be the best gifts to them for their years of love since I was born. I would also thank my friends who got me through the tough time, accompanied me during my PhD studies, and advanced my career–Heng Zhang (being the most supportive and caring person in life), Yajie Geng, Diyu Yang, Tiantu Xu, Pei Xu, Cheng cheng, Donghao Wang, and Jiacheng Shi.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Xu, Ran PhD, Purdue University, December 2021. Approximation for Streaming Video Analytics on Mobile Devices. Major Professor: Saurabh Bagchi.

Approximate algorithms have shown great success to reduce the computation latency with minor accuracy loss. Such algorithms are especially useful on resource-constrained devices like mobile phones and embedded boards. However, current approximate algorithms for streaming video analytics cannot provide accuracy and latency guarantees. This happens in the face of changing content characteristics in the video stream and changing resource availability on the target devices.

In this dissertation, we propose the analytic foundation on how to characterize the accuracy and latency of approximate algorithms in a content and contention aware manner. We use this to enable an approximate algorithm to probabilistically meet a user-provided latency constraint or an accuracy target. We use video processing pipelines, a video object classification system, and a video object detection system as examples to demonstrate how our solution can improve the performance of streaming video analytic systems on the resource-constrained mobile and embedded devices. Our evaluation shows that our technique can be overlaid seamlessly on top of standard vision algorithms and provides superior accuracy-latency tradeoff over the start-of-the-art approaches.

1. INTRODUCTION

Approximate algorithms have enabled great potential to reduce the computation cost (latency) from its original exact version with minor accuracy loss. Such algorithms are especially useful for computationally heavy applications, on resource-constrained devices, and with stringent latency constraints. For example, state-of-the-art object detection algorithms for streaming videos are typically considered too heavy to run on mobile devices and meet the 30 fps latency requirement. The video object classification algorithms, though lighter than the detection ones, can also suffer the degraded performance due to varying content characteristics and dynamic resource contention[1]. Thus, this dissertation mainly combines approximate algorithms and computationally heavy video analytic algorithms so as to achieve high accuracy within stringent latency requirements on mobile devices. To achieve this goal, both advanced analytical models and elegant systems design are required.

Designing approximate streaming video analytic systems is a challenging problem. Typically, a successful system faces three hard sub-problems in sequential–(1) how to apply proper approximate algorithms to the video analytic algorithms to largely reduce the computation latency with minor accuracy loss, (2) how to guarantee the accuracy and latency performance of such approximate video analytic algorithms given varying content characteristics and dynamic resource contention, and (3) how to design an efficient system so that the overhead of additional components due to approximation, modeling, and scheduling does not offset the benefits. Thus, these problems can be summarized as follows,

Problem Statement: *Approximate streaming video analytic systems require content and contention aware characterization to probabilistically meet accuracy and latency guarantees on resource-constrained mobile devices.*

To address these problems, a single analytic model is typically not sufficient. This dissertation mainly purposes many well-designed components to jointly work together, where each component aims to finish a specific task and all components together make the system outstanding. On three streaming video analytic systems–video processing pipelines, video object classification systems, and video object detection systems, our proposed design

[1] ↑We refer "resource contention" as the competition of computational resource, *e.g.* CPU, memory, and GPU, between concurrent applications on the devices.

achieves better performance than the previous state-of-the-art algorithms on the widely used datasets.

The dissertation is mainly composed of five work which either have been peer reviewed and published or are under review in the premier systems conferences, the systems journal, and the computer vision conference. Our VideoChef on video processing pipelines appeared at the 2018 USENIX Annual Technical Conference (USENIX ATC). Our ApproxNet on the video object classification system appeared at the ACM Transactions on Sensor Networks (TOSN). Our ApproxDet on the video object detection system appeared at the 18th ACM Conference on Embedded Networked Sensor Systems (SenSys). The research contributions of these work are summarized as follows:

1. **Efficient Approximation for Streaming Video Processing Pipelines**:

   Many video streaming applications require low-latency processing on resource-constrained devices. To meet the latency and resource constraints, developers must often approximate filter computations. A key challenge to successfully tuning approximations is finding the optimal configuration, which may change across and within the input videos because it is content-dependent. Searching through the entire search space for every frame in the video stream is infeasible, while tuning the pipeline off-line, on a set of training videos, yields suboptimal results. We present VideoChef, a system for approximate optimization of video pipelines. VideoChef finds the optimal configurations of approximate filters at runtime, by leveraging the previously proposed concept of canary inputs—using small inputs to tune the accuracy of the computations and transferring the approximate configurations to full inputs. VideoChef is the first system to show that canary inputs can be used for complex streaming applications. The two key innovations of VideoChef are (1) an accurate error mapping from the approximate processing with downsampled inputs to that with full inputs and (2) a directed search that balances the cost of each search step with the estimated reduction in the run time. We evaluate our approach on 106 videos obtained from YouTube, on a set of 9 video processing pipelines with a total of 10 distinct filters. Our results show significant performance improvement over the baseline and the previous approach that uses canary inputs. We also perform a user study that shows that the videos produced by VideoChef are often acceptable to human subjects. In Chapter 2, I will present the design, implementation and evaluations of VideoChef in details.

2. **Content and Contention-Aware Video Object Classification System for Embedded Clients**:

   Videos take a lot of time to transport over the network, hence running analytics on the live video on embedded or mobile devices has become an important system driver. Considering that such devices, *e.g.* surveillance cameras or AR/VR gadgets, are resource constrained, creating lightweight deep neural networks (DNNs) for embedded devices is crucial. None of the current approximation techniques for object classification DNNs can adapt to changing runtime conditions, *e.g.* changes in resource availability on the device, the content characteristics, or requirements from the user. We introduce ApproxNet, a video object classification system for embedded or mobile clients. It enables novel dynamic approximation techniques to achieve desired inference latency and accuracy trade-off under changing runtime conditions. It achieves this by enabling two approximation knobs within a single DNN model, rather than creating and maintaining an ensemble of models (*e.g.* MCDNN [MobiSys-16]. We show that ApproxNet can adapt seamlessly at runtime to these changes, provides low and stable latency for the image and video frame classification problems, and show the improvement in accuracy and latency over ResNet [CVPR-16], MCDNN [MobiSys-16], MobileNets [Google-17], NestDNN [MobiCom-18], and MSDNet [ICLR-18]. In Chapter 3, I will present the design, implementation and evaluations of ApproxNet in details.

3. **Content and Contention-Aware Approximate Object Detection for Mobiles**:

   Advanced video analytic systems, including scene classification and object detection, have seen widespread success in various domains such as smart cities and autonomous systems. With an evolution of heterogeneous client devices, there is incentive to move these heavy video analytics workloads from the cloud to mobile devices for low latency and real-time processing and to preserve user privacy. However, most video analytic systems are heavyweight and are trained offline with some *pre-defined* latency or accuracy requirements. This makes them unable to adapt at runtime in the face of three types of dynamism — the input video characteristics change, the amount of compute resources available on the node changes due to co-located applications, and the user's latency-accuracy requirements change. We introduce ApproxDet, an *adaptive* video object detection framework for mobile devices to meet accuracy-latency requirements in the face of changing content and resource contention scenarios. To achieve this, we introduce a multi-branch object detection kernel, which incorporates a data-driven modeling approach on the performance metrics, and a latency SLA-driven scheduler to pick the best execution branch at runtime. We evaluate ApproxDet on a large benchmark video dataset and compare quantitatively to AdaScale and YOLOv3. We find that ApproxDet is able to adapt to a wide variety of contention and content characteristics and outshines all baselines, *e.g.* it achieves 52% lower latency and 11.1% higher accuracy over YOLOv3. In Chapter 4, I will present the design, implementation and evaluations of ApproxDet in details.

4. **Cost and Content Aware Reconfiguration of Video Object Detection Systems for Mobile GPUs**:

   An adaptive video object detection system selects different execution paths at runtime, based on a user specified latency requirement, video content characteristics, and available resources on a platform, so as to maximize its accuracy under the target latency service level agreement (SLA). Such a system is well suited for mobile devices with limited computing resources, often times running multiple contending applications. In spite of several recent efforts, we show that existing solutions suffer from two major drawbacks when facing a tight latency requirement (*e.g.* 30 fps).*First*, it can be very expensive to collect some feature values for a scheduler to decide on the best execution branch to run. *Second*, the system suffers from the switching overhead of transitioning from one branch to another, which is variable depending on the transition pair. We address these challenges and present LiteReconfig, an efficient and adaptive video object detection framework for mobiles. Underlying LiteReconfig is a cost-benefit analyzer for the scheduler that decides which features to use and then which execution branch to run at inference time. LiteReconfig is further equipped with a content-aware accuracy prediction model to select an execution branch tailored for frames in a streaming video. With a large-scale real-world video dataset and two leading edge embedded devices with mobile GPUs, we demonstrate that LiteReconfig achieves significantly better accuracy under a set of varying latency requirements when compared to existing adaptive object detection systems, while running at speeds of up to 50 fps on an NVIDIA AGX Xavier board. In Chapter 5, I will present the design, implementation and evaluations of LiteReconfig in details.

5. **Multi-branch Object Detection Framework for Videos on Mobiles**:

   Several recent works seek to create lightweight deep networks for object detection on mobiles and, in particular, for video object detection. Many existing detectors intrinsically support adaptive inference, and offer a multi-branch object detection framework (MBODF). Here, an MBODF is referred to as a solution that has many execution branches and one can dynamically choose from among them at inference time to satisfy varying latency requirements (*e.g.* by varying resolution of the input frame). In this paper, we ask, and answer, the wide-ranging question across all MBODFs: How to expose the right set of execution branches and then how to schedule the optimal one at inference time? In addition, we uncover the importance of making a content-aware decision on the branch to run, as the optimal one is conditioned on the video content. Finally, we explore a content-aware scheduler, an Oracle one, and then a practical one, leveraging various lightweight feature extractors. Our evaluation shows that layered on Faster R-CNN-based MBODF, compared to 7 baselines, our method SmartAdapt achieves a higher Pareto optimal curve in the accuracy-vs-latency space for ILSVRC VID dataset. In Chapter 6, I will present the design, implementation and evaluations of SmartAdapt in details.

2. VIDEOCHEF: EFFICIENT APPROXIMATION FOR STREAMING VIDEO PROCESSING PIPELINES

2.1 Introduction

Video processing has brought many emerging applications such as augmented reality, virtual reality, and motion tracking. These applications implement complex video pipelines for video editing, scene understanding, object recognition and object classification [1], [2]. They often consume significant computational resources, but also require short response time and low energy consumption. Often, the applications need to run on the local machines instead of the cloud, due to latency [1], bandwidth [3], or privacy constraints [4].

To enable low-latency and low-energy video processing, we leverage the the fact that most stages in the video pipeline are inherently *approximate* because human perception is tolerant to modest differences in images and many end goals of video processing require only estimates (*e.g.*, detecting object movement or counting the number of objects in a scene [5]). Many *domain-specific* algorithms have exposed algorithmic knobs, that can e.g., subsample the input images or replace expensive computations with lower-accuracy but faster alternatives [6]–[9]. To complement domain-specific approximations, researchers have proposed various generic *system-level techniques* that expose additional knobs for optimizing performance and energy of applications while trading-off accuracy of the results. The techniques span compilers [10]–[15], systems [16]–[20], and architectures [14], [21]–[24].

Content-dependent Approximation. A fundamental challenge of uncovering the full power of both generic and domain specific approximations is finding the configurations of these approximations that provide maximum savings, while providing acceptable results for *each given input*. This challenge has two main parts.

First, the optimal approximation setting is dependent on the *content of the video*, not just on the algorithms being used in the processing pipeline. Often individual videos, or parts of the same video should have different approximation settings, requiring the program to make the decisions at runtime. Second, the optimization needs to explore a large number of approximate configurations before selecting the optimal one for the given input, requiring the optimizer to construct off-line models. Systems like Green [16] and Paraprox [14] dynamically adapt the computation using runtime checks of the intermediate results, while Capri [25] selects approximation level from the input features at the program start. However, the systems rely on extensive off-line training to map the approximation levels to accuracy and performance.

To relax the dependency on off-line training, Laurenzano *et al.* [26] propose Input Responsive Approximation (IRA) for runtime recalibration with no offline training. IRA creates *canary inputs*, smaller representations of the full inputs (obtained via subsampling), and then re-runs the computation on the canary input with different approximation settings, until it finds the most efficient setting that maintains the accuracy requirement (on the canary). While the concept is promising, the application of IRA to video processing pipelines is limited:

- IRA has been applied to individual computational kernels (in contrast to full pipelines). It is unclear how to capture the interactions between the stages of the pipeline, how often to calibrate, and what are the optimal canary sizes.

- IRA uses the approximation settings derived from the canary input to the full input, assuming that the errors for the full and correlated inputs will be *identical*. However, this assumption is often incorrect (98% of cases, Figure 2.2) and leads to missed speedup opportunities.

- IRA's greedy search may introduce additional overheads and may not find good approximation settings efficiently because it has no notion of what are the appropriate points in the stream to search.

Our Solution: VideoChef. We present VideoChef, a fast and efficient processing pipeline for streaming videos. VideoChef can optimize the performance subject to accuracy constraints for the system-level and domain-specific approximations of all kernels in the video processing pipeline. Figure 2.1 presents VideoChef's end-to-end workflow:

- Like IRA, VideoChef uses small-sized canary input to guide the on-line search for approximation setting. However, unlike IRA, VideoChef is tailored for optimization of the whole video processing pipelines, not just individual kernels.

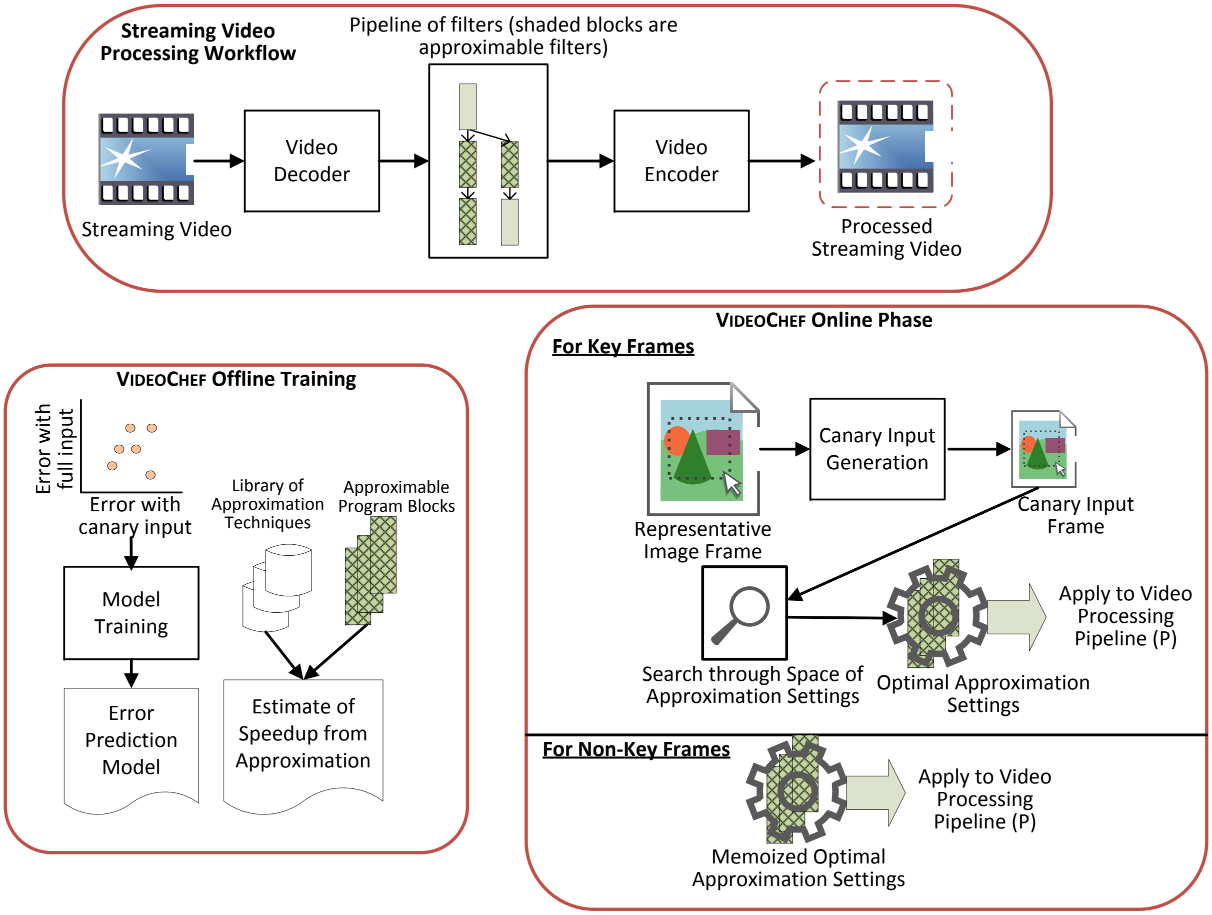

Streaming Video Processing Workflow
Pipeline of filters (shaded blocks are approximable filters)
Streaming Video
Video Decoder
Video Encoder
Processed Streaming Video
VIDEOCHEF Offline Training
Error with full input
Error with canary input
Model Training
Error Prediction Model
Library of Approximation Techniques
Approximable Program Blocks
Estimate of Speedup from Approximation
VIDEOCHEF Online Phase
For Key Frames
Representative Image Frame
Canary Input Generation
Canary Input Frame
Search through Space of Approximation Settings
Optimal Approximation Settings
Apply to Video Processing Pipeline (P)
For Non-Key Frames
Memoized Optimal Approximation Settings
Apply to Video Processing Pipeline (P)

Figure 2.1. End-to-end flow of approximate video processing with VIDEOCHEF. The video processing pipeline comprises multiple filters, which can be approximated to save computation at the expense of tolerable video quality. The offline and the online components of VIDEOCHEF work together to determine the best approximation setting for each approximable filter block.

- In contrast to IRA, VIDEOCHEF presents a finely tunable prediction model for mapping the error from the canary input to that with the original input. This prediction model is trained offline and hence does not generate any additional runtime overhead. At the same time, it is much more lightweight than the full off-line training employed by other approaches.

- At runtime, VIDEOCHEF performs an efficient search through the space of approximation settings and ensures that the cost of the search does *not* overwhelm the benefit of approximating the computation.

We evaluate VideoChef with three error models and two search strategies, applied to a corpus of 106 YouTube videos from 8 content categories, which span the range of video features (e.g., color and motion). We analyze 10 filters arranged in 9 pipelines of size 3. We find that VideoChef is able to reach within 20% of the theoretical best performance possible and outperforms IRA's performance by 14.6% averaged across all videos and saves on an average 39.1% over the exact computation given a relatively restrict quality requirement. While given a more loose quality requirement, VideoChef is able to reach within 26.6% of the theoretical best and also achieve higher performance gain – 53.4% and 61.5% over IRA and exact computation, respectively.

While we have framed this discussion in terms of video processing, the novel contributions outlined below apply to other low-latency streaming applications, with the fundamental requirement that the characteristics change to some extent from one segment of the stream to another, for instance, online video gaming, augmented reality and virtual reality applications.

Contributions. We make the following contributions:

1. We present VideoChef, a system for performance and accuracy optimization of video streaming pipelines. It consists of off-line and on-line components, that together adapt the application's approximation level to the desired output quality.

2. We build a predictive model to accurately estimate the quality degradation in the full output from the error generated when using the canary input. This enables more aggressive approximation setting the approximation algorithm that has tunable knobs.

3. We propose an efficient and incremental search technique for the optimal approximation setting that takes hints from the video encoding parameters to reduce the overhead of the search process.

4. We demonstrate the benefits of VideoChef through (1) quantitative evaluation on various real-world video contents and filters and (2) a user study.

2.2 Background and Motivation

Error Metric: At a high-level, a video is composed of a sequence of image frames. To quantify the error in the output or the processed video due to approximation, we measure the Peak Signal-to-Noise Ratio (PSNR) of the output video. PSNR is the average of the PSNRs of the individual frames in the output video. Suppose that a video consists of K frames where each frame has $M \times N$ pixels. Let $Y_k(\mathrm{i}, \mathrm{j})$ be the value of the pixel at (i, j) position on the k-th frame of the processed video without any use of approximation, and $Z_k(\mathrm{i}, \mathrm{j})$ be the value of the pixel when approximation was applied. Then, the PSNR of the approximate output is computed as follows:

$$PSNR = \frac{1}{K} \sum_{k=0}^{K-1} 20 \times log_{10} \frac{MaxValue}{\sqrt{MSE(Z_k, Y_k)}} \tag{2.1}$$

where $MaxValue$ is the maximum possible pixel value present in the frame, and $MSE(Z_k, Y_k)$ is the mean square error between Z_k and Y_k, i.e., $\sum_{\mathrm{i}} \sum_{\mathrm{j}} (Z_k(\mathrm{i}, \mathrm{j}) - Y_k(\mathrm{i}, \mathrm{j}))^2$, as a result of approximation. Thus, lower the PSNR, the higher the error in the output video.

Isn't the problem solved by IRA and Capri? IRA (Input Responsive Approximation) [26] and Capri [25] attempted to address the problem of selecting optimal approximation level for individual inputs.

IRA [26] solely relies on canary inputs to search for best approximation settings. Thus, it implicitly assumes that the magnitude of error corresponding to a particular approximation setting on the *canary inputs* is identical to the error with the same approximation settings on the *full-sized inputs*. But, Figure 2.2 shows our experiment with 424 real images and 216 different approximation settings. We found that for the same approximation settings, the PSNR of the full-sized inputs can be significantly different from the PSNR of the canary inputs. Most of the points (about 98%) are above the diagonal, indicating that the error on the full input is lower than that with the canary input for the same approximation level.

We attribute the difference in the approximation to the higher variations between neighboring pixel values for canary inputs. Therefore, for the same approximation settings, the approximate processing on canary inputs gives lower PSNR. We found that on an average,

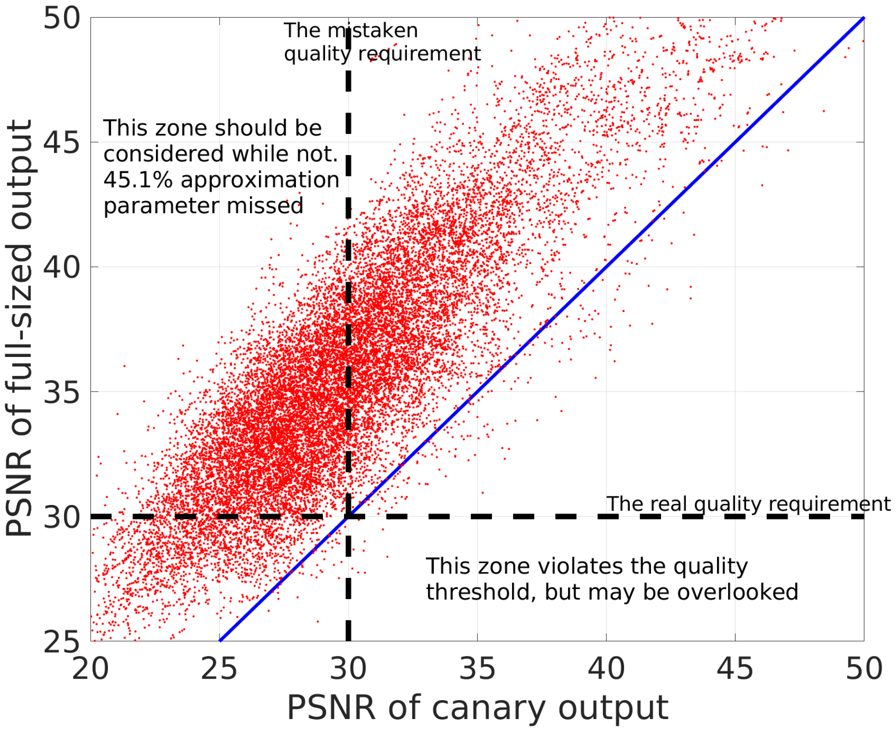


Figure 2.2. The PSNR of full-sized output versus the PSNR of canary output, for the I-frames of 106 videos on one of our application Boxblur-Vignette-Dilation video filter pipeline. The PSNR of full output is higher for over 98% approximation settings and 45.1% of the approximation settings lie in such a zone that is ignored by IRA approach but actually satisfies the quality requirement.

the PSNR of a full-sized output is 5.36 dB higher than the PSNR of canary output. Therefore, IRA misses an opportunity for more aggressive optimization that can fit within the user-specified quality threshold.

Capri [25] rigorously addresses the problem of selecting the best approximation settings to minimize the computational cost, while meeting the error bound. But Capri also fails in the video processing setting because it does not recalibrate itself with the stream and thus cannot change its approximation settings when the characteristics of the stream change. Further, it (1) relies on prior enumeration of all possible inputs, which is impossible in this target domain, and (2) performs the selection of approximation settings completely offline, which reduces the cost of the optimization but makes it non-responsive to changes in the input data.

2.3 Solution Overview

Figure 2.1 shows the end-to-end workflow of streaming video processing, with approximation.

Approximation. Under normal processing, a video decoder converts the video into its constituent frames. Then a sequence of "filters" (synonymously, processing steps or pipeline stages) is applied to each frame. Examples include blurring filter (*e.g.*, at the TSA airport checkpoint scanners) and edge detection filter (*e.g.*, for counting people in a scene). Finally, the transformed frames are optionally put together by a video encoder. To make such processing fast and resource efficient, VideoChef intelligently uses selective approximation (Section 2.3.1) during the computation of the filters. The user sets the quality constraint on the output video quality. An example specification is that the PSNR of the output video should be above 30 dB.

Accuracy Calibration with Canary Inputs. For each representative frame (called "key frame" here), VideoChef determines a *canary input* (Section 2.3.2), which summarizes the full frame such that the dissimilarity between the full and the canary frame remains below a threshold. With the canary input, VideoChef occasionally recalibrates the approximation levels of the filters. It determines when to call the search algorithm using domain specific knowledge about the frames and scenes (Section 2.3.3). For this, we extract hints from the video decoder which lets VideoChef determine the key frames. This amortizes the cost of the search across many frames of the video, with 80-120 frames being a typical range for MPEG-4 videos. In the absence of such a video decoder, we have a variant, which triggers the search upon a scene change detection.

Online Search for Optimal Tradeoffs. VideoChef searches for the approximation setting of each filter that gives the lowest execution time subject to a threshold for the output quality (Section 2.3.4). Since the search for approximation is done with the *canary input*, the error of approximate computation is different from the error of the computation on the full input. VideoChef introduces a method to accurately map between these two errors (Section 2.3.5). In performing this estimation, we consider multiple variants of VideoChef, depending on what features are available to the predictor, such as, some categorization of the video frame according to its image properties. Through this procedure, we aim to maximally leverage the approximation potential in the application and give flexible approximation choices.

2.3.1 Approximation techniques

The computations involved in filtering operations can be approximated by VideoChef in various ways as long as each of the underlying approximation techniques exposes knobs that can be tuned to control the approximation levels (ALs). For example, in the three popular program transformation-based approximation techniques, the variable `approx_level` is a tuning knob that controls the levels of approximation. A higher value implies more aggressive approximation, leading most often to higher speedup but also higher error. These transformations are performed automatically by a compiler (LLVM in our case).

Loop perforation: In loop perforation [11], [12], the computation is reduced by skipping some iterations, as shown below.

```
for (i = 0; i < n; i = i + approx_level)
  result = compute_result();
```

Loop truncation: In loop truncation [11], [12], the last few iterations of the computation are dropped as shown in the following example:

```
for (i = 0; i < (n - approx_level); i++)
  result = compute_result();
```

Loop memoization: In this technique [13], [14], for some iterations in a loop we compute the result and cache it. For other iterations we use the previously cached results.

```
for (i = 0; i < n; i++)
  if (i % approx_level == 0)
    cached_result = result = compute_result();
  else
    result = cached_result;
```

2.3.2 Canary Inputs

To reduce the search overhead for finding the best approximation level within each frame of the video, we generate canary inputs for the frame following the work in [26]. A good canary input should meet two requirements: (1) it should be *close enough* to the original input so that the AL found by the canary is the same as the AL computed from the original; (2) it should be small enough that the search process using the canary input is efficient. We first define the dissimilarity metric to compare the canary sample video and the full-sized video and then show how to choose the appropriate canary input.

Metrics of Dissimilarity. We define two metrics of dissimilarity. Let a full-sized video have K frames and $M \times N$ pixels in each frame, and each pixel has the property $X(\mathrm{i}, \mathrm{j})$. A canary video has K frames with $m \times n$ pixels, and the same property $Y(\mathrm{i}, \mathrm{j})$. The property could be one component in the YUV colorspace of an image, where the Y component determines the brightness of the color (known as "luminance") while the U and V components determine the color itself (known as "chroma") and each ranges from 0 to 255. The "dissimilarity metric for mean" (SMM), is defined as follows (following [26]):

$$mFull = \frac{1}{M \times N \times K} \sum_{\mathrm{i}=0}^{K-1} \sum_{\mathrm{i}=0}^{M-1} \sum_{\mathrm{j}=0}^{N-1} X(\mathrm{i}, \mathrm{j}) \tag{2.2}$$

$$mSmall = \frac{1}{m \times n \times K} \sum_{\mathrm{i}=0}^{K-1} \sum_{\mathrm{i}=0}^{m-1} \sum_{\mathrm{j}=0}^{n-1} Y(\mathrm{i}, \mathrm{j}) \tag{2.3}$$

$$SMM = \frac{|mSmall - mFull|}{mFull} \tag{2.4}$$

When a pixel has a vector of values, such as the YUV colorspace which has 3 values for the 3 components, then the SMM metric is combined across the different elements of the vector. The combination could be a simple average or a weighted average; we use the latter due to the higher weight of the Y-channel in the YUV colorspace. Similarly, we define the "dissimilarity metric of standard deviation" (SMSD) to capture the dissimilarity in the Standard Deviation between the full input and the canary input.

Generating Candidate Canary Videos. Given a frame of the video from which to generate the canary video, we resize the frame to a fraction $1/N$ of its original size to create the canary video. Typical sizes that we find useful in our target domain are 1/16, 1/32, 1/64, 1/128, 1/256 of the original size. Since the frame is a 2-D matrix of pixels, to resize it to $1/N$ of its original size, we shrink the width and height each to $1/\sqrt{N}$ of the full size by sub-sampling 1 pixel out of every $\sqrt{N}$ pixels.

Reducing an input size causes at least proportional reduction in the amount of work inside the filter. Many filters that are finding increasing use are super-linear, where the benefit of using a small canary frame is even more significant. Two popular examples are determining optical flow to measure motion [27] and morphological filter [28], where the value of each pixel in the output image is based on a comparison of the corresponding pixel in the input image with its neighbors. We compute the similarity between the full-sized video frame and the canary video frame according to the metrics SMM and SMSD. We set the maximum dissimilarity metric we can tolerate as a threshold parameter—we find 10% is a practically useful threshold for both SMM and SMSD. Among all the qualified canary inputs, we select the smallest one as our final choice.

2.3.3 Identifying Key Video Frames

Searching for the best AL for each approximable program block is computationally expensive. Conceptually, we would want to repeat the search when the characteristic of the video changes significantly so that the optimal approximation setting is expected to be different. In practical terms, we want to perform such change point detection without having to parse the content of the video. Video encoders already provide hints when the content of the scene has changed significantly.

We make the observation that videos have temporal locality, and many frames in the *same group* will have the same approximation setting. Therefore, we can perform a single search once per a single group of frames. We leverage domain-specific knowledge about videos to automatically select the group boundaries in two ways:

Scene Change Detector. Our first observation is to recalibrate the approximation at the beginning of different scenes. This approach is general and works for any video format. There are mainly 2 classes of scene change detectors, namely, pixel-based and histogram-based. The pixel-based methods are highly sensitive to camera and object motion. Histogram-based methods are good for detecting abrupt scene changes. To keep our overhead low (since the detection algorithm runs on every frame), we limit ourselves to detecting only abrupt scene changes and use canary frames for this detection.

We implement a histogram-based scene change detector using only the Y-channel of the canary frames [29]. We experimentally found the Y-channel information was sufficient to detect abrupt scene changes and we were more concerned about overhead of scene change detector than its accuracy. The algorithm detects a scene change whenever the sum of the absolute difference across all the bins of histograms of two consecutive frames is greater than some predefined threshold (20% of the total pixels in our evaluation). Our experiments show that the optimal configuration (found through offline brute-force search) changes dramatically at scene change boundaries but stay relatively stable within a group of frames.

I-frame Selection for MPEG videos. The second solution takes advantage of I-frames, present in the popular H.264 encoder (which the MPEG-4 and many other video formats follow). It defines three main types of frames: $I-$, $P-$, and $B-frames$ [30]. An I-frame uses intra-prediction meaning the predicted pixels within this frame are formed using only samples from the same frame. The P- and the B-frames use inter-prediction meaning the predicted pixel within this frame depends on samples from the same frame *as well as* samples from other frames around it (the distinction between P- and B-frames is not relevant for our discussion).

When to insert an I-frame (also called a “reference frame”) depends on the exact coding scheme being used, but in all such coding schemes that we are aware of, a big difference in the frame triggers the insertion of a new I-frame, since inter-coding will give almost as long a code as intra-coding. Further, because an I-frame does not have dependencies on other frames, this makes it easier to reconstruct and perform the (exact or approximate) computation. We see empirically that for a wide range of videos used in our evaluation, the average spacing between adjacent I-frames is 137 frames. Although specific to only some video formats, it results in a low sampling rate and consequently the low search overhead, while triggering search at a suitable granularity.

2.3.4 Search with Canary Inputs

An approximable program block exposes one or more approximation knobs. The `approx_level` variable mentioned with the loop-based approximation techniques in Section 2.3.1 is an example of such a knob. In our notation, we use AL 1 to denote the exact computation. The higher the AL is, the less accurate the computation is and the higher the speedup is. Now for a pipeline of cascaded filters, each having one or more approximation knobs we have a *vector* of approximation settings per frame. We define a *setting* in the processing pipeline as the combination of ALs for each of the approximable program blocks in the video processing pipeline. For example, with an n-stage processing pipeline and each stage being approximable and having exactly one approximation knob, the setting will be $\vec{A} = \{a_1, a_2, \cdots, a_n\}$, where a_i denotes the AL of knob i.

To find the best approximation setting, we follow a searching algorithm outlined as follows:

1. **Start searching** at a particular setting, typically $(1, 1, \cdots, 1)$, corresponding to no approximation.

2. **Select a group of candidate settings** by the Candidate Selection Algorithm. The selection algorithm simply selects the next set of settings to try out in the search process. The greedy algorithm works as follows: given the current setting $\vec{A^{(0)}}$, we assume that we can reach the best AL by looking at each step 1 AL further in each approximable block. So the candidates are $\{\vec{A^{(\mathrm{i})}}\}$, with $\mathrm{i} = 1 \cdots n$, for n approximable blocks and $\vec{A^{(\mathrm{j})}} = \{a_1^{(0)}, \cdots a_{\mathrm{j}-1}^{(0)}, a_\mathrm{j}^{(0)} + 1, a_{\mathrm{j}+1}^{(0)}, \cdots, a_n^{(0)}\}$.

3. **Decide whether to continue search**, i.e., whether it is worthwhile to try any of these candidate settings. We use the Approximation Payoff Estimation Algorithm. If not, return the current setting. This algorithm estimates whether the saving due to the

more aggressive approximation can compensate for the time of the additional search step.

4. **Try each worthwhile candidate setting** from the set computed by the previous step. Use the ALs in candidate settings to run approximate computation on canary video and compute the error metric for each candidate setting. Then map the error metric to that with the full sized outputs.

5. **Check for exit or iterate** – if error metrics of all the full video outputs exceed the error boundary, return the current setting. Otherwise, the candidate setting which gives the lowest error becomes the next setting, go to (2) and iterate.

Approximation Payoff Estimation Algorithm: The goal of this algorithm is to estimate the benefit of executing the application with the new AL searched for versus the cost of searching with the new AL, all for the key frame under question. Let the current setting be represented by $\vec{A^{(0)}} = \{a_1^{(0)}, a_2^{(0)}, \cdots, a_n^{(0)}\}$. Recollect that this is the set of ALs for each of the approximable blocks in the application. Let the execution time of the application at setting $\vec{A^{(\mathrm{i})}}$ and with canary downsampling C_d be given by $g(A^{(\mathrm{i})}, C_d)$, where $C_d = 1$ denotes the execution time with the full input.

This algorithm works in a breadth-first fashion and attempts to prune some of the paths where exploring higher degrees of approximation for a particular knob cannot speedup the execution further and may lead to slowdown due to associated overheads. From the current setting of $\vec{A^{(0)}}$, let the next possible settings of exploration be $\vec{A^{(1)}}, \vec{A^{(2)}}, \cdots, \vec{A^{(N)}}$. For example, with greedy search, with n approximable blocks, there will be n possible next settings. The maximum possible benefit by exploring all the candidate next settings is calculated as:

$$B = \max_{\mathrm{i}=1}^{N} [g(A^{(0)}, 1) - g(A^{(\mathrm{i})}, 1)] \tag{2.5}$$

This benefit B simply means the maximum reduction in execution time across all the possible candidate settings, when run with the full input. However, to realize this gain, we have to pay the cost of searching, which can be expressed in terms of the overhead as follows:

$$O = \sum_{\mathrm{i}=1}^{N} g(A^{(\mathrm{i})}, C_d) \tag{2.6}$$

This overhead O is simply the cost of executing the application with the next step ALs, but with the canary input (and hence the downsampling ratio C_d). The decision for VideoChef becomes simple: if $B > O$, then continue the search, else stop and return the current setting.

2.3.5 Error mapping model

We have to develop an error mapping model to characterize the relation between error in the canary output and error in the full output, for the same approximation levels. This is important because we have seen empirically (Figure 2.2) that the canary errors are higher than full frame errors for most points. We propose three different mapping models to use according to different amounts of knowledge in the model.

Model-C: Suppose we know the error metric of a canary output C. The error metric of a full-sized output F is estimated by a quadratic regression model as follows,

$$F = w_0 + w_1 \times C + w_2 \times C^2 \tag{2.7}$$

Offline, we calculate the ground truth of the pairs (C, F) for every possible AL $\vec{A}$ and for all the videos in the training set. In practice, we find that sub-sampling the space of possible ALs still provides accurate enough training, with a sub-sampling rate of 10% being adequate. Let us say that the error bound specified by the user is E_B. Then clearly we want $F < E_B$. However, due to the possible inaccuracy of the error mapping model, we want to explore a larger space so that we are not missing out on opportunities for approximating. Therefore, while training the model, Model-C, we explore all the points where $F \leq E_B + \Delta$, where Δ is a user configurable parameter for how far outside the tolerable region we want to explore in the model. Then we solve the unknown coefficients w_0, w_1, and w_2 in the model. We find empirically that for a large set of videos, this model reaches its limits with the quadratic regression function.

Model-CA: Now, suppose VideoChef has additional knowledge of what ALs were in effect. Given the error metric of a canary output C, which is computed approximately with ALs $\vec{A} = \{a_1, a_2, \cdots, a_n\}$, we construct the input vector $\vec{I} = (1, C, \vec{A})$. The first element of this vector is the constant 1 and allows for a constant term in our equation for F. Then the error metric of a full-sized output F is estimated by a regression model as follows,

$$F = \vec{I} \cdot w \tag{2.8}$$

where w is a $(n + 2) \times 1$ coefficient vector. The goal of the training is to estimate the matrix w. The elements of the matrix w provide the weights to multiply the different input components—the error in canary output (C), and the different ALs. We use similar training method offline as for Model-C.

Model-CAD: Many of approximation techniques on image processing reduce computation load by skipping a fraction of rows of images. Thus, the difference over rows is often related with approximation quality. Inspired by this characteristic, we consider a new feature vector $\vec{D} = (d_1, d_2, d_3)$, where each of d_k's represents a feature extracted from one of Y, U, and V channels of an image. The feature d_k is referred to as a row-difference feature and is defined as the mean of absolute difference in pixel values of the same column between consecutive rows in each channel. Averaging over rows and columns, we use only one representative number as d_k for each channel.

Considering an input vector $\vec{I} = (1, C, \vec{A}, \vec{D})$, the error metric of a full-sized output F is estimated by a linear regression model as:

$$F = \vec{I} \cdot w, \tag{2.9}$$

where w is a $(n + 5) \times 1$ coefficient vector. In the experiment results, we will see that Model-CAD outperforms the other models.

Non-linear models. We have also tested complex non-linear models to predict F, using artificial neural networks with all pixel information as input. However, considering the run-time complexity [31], we could not observe any significant benefit of the non-linear models over the linear models mentioned earlier. Thus, we do not report their results in the evaluation.

2.4 Implementation and Dataset

We use loop perforation and memoization [11], [12] to approximately filter the frames in the video. The implementation of VideoChef is comprised of an offline and an online component. The offline component uses a set of training videos (50% of videos described under the dataset below) and creates models for the error mapping and for the cost and the benefit of each step of the search. This last model is actually implemented as a lookup table, due to the space being only piece-wise continuous. During runtime, VideoChef queries these models, using linear interpolation if needed, and performs an efficient search to identify the optimal ALs and runs each of the three filters in any pipeline with their optimal values.

VideoChef API. Our compiler pass identifies the approximable blocks using program annotations and then performs the relevant transformations to insert the approximation knobs to be tuned (such as `approx_level` in Section 2.3.1). We have provided support for similar annotations in other domain specific languages that we have built in the past and that helps to reduce the programmer burden [32]. The user can then use the following API calls to enable VideoChef in the video pipeline:

- *setCalibrationFrequency(f=”I-frame”)* : This will set how frequently VideoChef will search for the best approximation settings. The default value is VideoChef will trigger a search for every I-frame. If f=”x”, then VideoChef will search every x-th frame.

- *setQualityThreshold(b=”30”)* : This will set the (lower) PSNR threshold that the approximated pipeline must deliver. Default is 30 dB. VideoChef exposes to the user approximate versions of many filters from the FFmpeg library, with names like *deflate_approx*. The developer of VideoChef can register a callback with the video decoder using the call *void notifyIFrame(void *)*.

Table 2.1. Summary of the analyzed pipelines. We denote the approximation applied to each filter: Loop Perforation (LP) or Memoization (M)

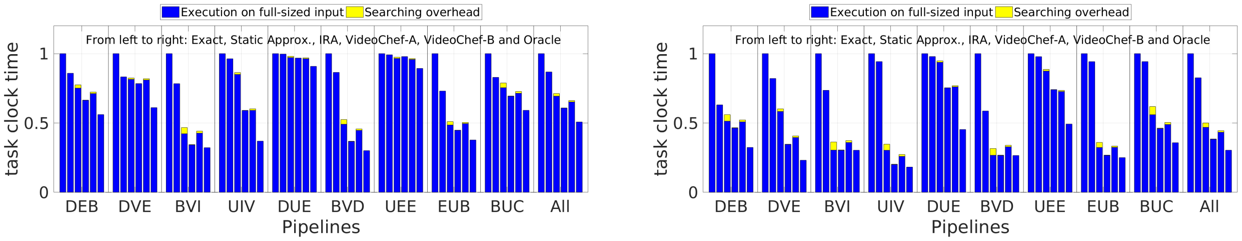
Name	Description, labeled with Approx. type	Approximation Type	Approximation Levels
DEB	Deflate(LP)-Emboss(LP)-Boxblur(M)	Loop perforation(LP) & Memoization(M)	1-6, 1-6, 1-6
DVE	Deflate(LP)-Vignette(LP)-Emboss(LP)	Loop perforation	1-6, 1-6, 1-6
BVI	Boxblur(M)-Vignette(LP)-Inflate(LP)	Loop perforation & Memoization	1-6, 1-6, 1-6
UIV	Unsharp(LP)-Inflate(LP)-Vignette(LP)	Loop perforation	1-6, 1-6, 1-6
DUE	Dilation(LP)-Unsharp(LP)-Emboss(LP)	Loop perforation	1-6, 1-6, 1-6
BVD	Boxblur(M)-Vignette(LP)-Dilation(LP)	Loop perforation & Memoization	1-6, 1-6, 1-6
UEE	Unsharp(LP)-Erosion(LP)-Emboss(LP)	Loop perforation	1-6, 1-6, 1-6
EUB	Erosion(LP)-Unsharp(LP)-Boxblur(M)	Loop perforation & Memoization	1-6, 1-6, 1-6
BUC	Boxblur(M)-Unsharp(LP)-Colorbalance(LP)	Loop perforation & Memoization	1-6, 1-6, 1-6

Video Dataset. We used 106 YouTube MPEG-4 videos for our evaluation. We used `libvideo`, a lightweight .NET library [33], to download the videos. The videos were collected from 8 different categories to cover a spectrum of different motion and color artifacts in the frames: Lectures, Ads, Car Races, Entertainment, Movie trailers, Nature, News, and Sports. At the first step, a single seed video was downloaded from each category, then we downloaded all YouTube's recommendations to the seed video, which turned out to belong to the same category as that of the seed video. Once the set of videos was collected, we randomly sub-sampled a 20 second clip from each video, being motivated by a desire to bound the experiment time. For each category, we collected approximately 25 videos and filtered out those with low resolution (since the quality threshold was likely already breached with the original video).

2.5 Evaluation

We describe our benchmarks first and then the four experiments to evaluate the macro properties of VideoChef and then its various components.

Benchmarks. We construct our benchmark by including different video processing pipelines. Each video processing pipeline consists of 3 consecutive filters, which are selected from a pool of 10 video filters from the FFmpeg library. These filters are modified to support approximation with tuning knobs. To execute on these filter pipelines, one needs to provide a video input and a quality threshold. Finally, the output is also a video, together with a quality metric with respect to each frame. We have a total of 9 different filter pipelines.



(a) Quality threshold = 30dB

(b) Quality threshold = 20dB

Figure 2.3. Mean execution times over all frames of all videos. Geometric means of the speedups are on the right.

Quality Metric. We use PSNR (Equation 2.1) as the quality metric for the videos produced by the approximate pipelines. We present the results for two acceptable PSNR thresholds. The threshold of 30 dB is considered a typical lower bound for lossy image and video compression [34], [35]. The threshold of 20 dB is considered the lower bound for lossy wireless transmission [36].

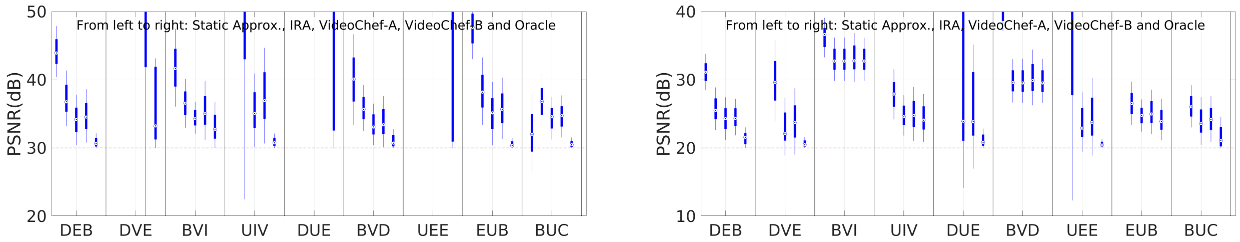
Evaluation Metrics. We define improvement as decrease in execution time, expressed as a percentage of the competitive protocol. We define the speedup of our approach as $Speedup = \frac{Speed\ of\ our\ protocol}{Speed\ of\ compared\ protocol} - 1$

Setup. We split the input videos into three groups: training, validation and test, with a share of 50%, 25%, and 25% of the videoset. The experiments are done on an x86 server with a six-core Intel(R) Xeon CPU, 16 GB RAM, and Ubuntu Linux kernel 4.4. We used FFmpeg library version 3.0 (compiled with gcc 5.4.0).

Canary Input Selection. We fixed the canary size to be 1/64 of the original size because our preliminary experiments showed that it is a good parameter to guarantee a dissimilarity metric value of 10% or less. Thus, the overhead of appropriate canary selection is not included in the results.

2.5.1 Performance and Quality Comparison for End-to-End Workflow

Figure 2.3 presents the results of the end-to-end workflow for the nine different video processing pipelines over all videos from the test set. Each plot presents the speedup relative to the exact pipeline for the following configurations (from left to right):



(a) Quality threshold = 30dB

(b) Quality threshold = 20dB

Figure 2.4. Quality of each frame across different video filter pipelines.

- Exact computation, with default parameters.

- Best static approximation, created by setting the AL that is just over the error threshold for *all* the frames in training videos.

- IRA extended with a simple searching policy that has a fixed interval of 10 frames. This number is chosen according to SAGE [23], which gives an analytic bound for a video processing setting.

- VideoChef version A – with I-frames detection.

- VideoChef version B – with scene change detection.

- Oracle version uses exhaustive search but does not incur search overhead. This sets the upper bound of the performance.

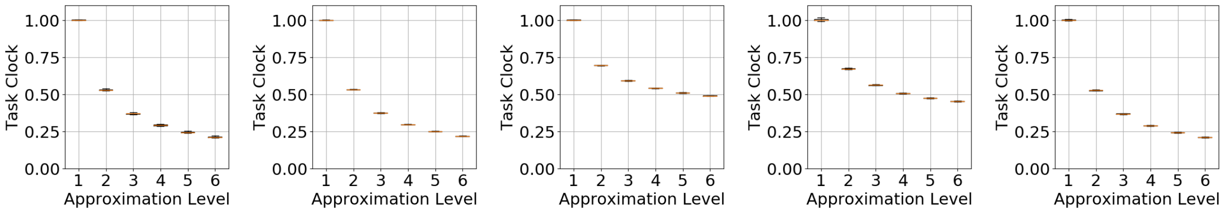
For both VideoChef versions, we used the CAD error model with 3dB margin, as the result of the analysis in Section 2.5.3.

Performance for 30db Threshold. Figure 2.3(a) shows that VideoChef version A reduces the execution time by 39.1% over exact computation and is within 20% of the Oracle. It outperforms both static approximation and IRA, by respectively 29.9% and 14.6% in the aggregate. The advantage exists for all the video filter pipelines with the greatest savings relative to IRA being in Unsharp-Inflate-Vignette (UIV) pipeline. We are 39.2%, 36.8% and 29.5% better than exact computation, static approximation, and IRA, respectively. The search overhead for VideoChef (both versions A and B) is small – the yellow portions of the bars are almost not visible – and yet it finds more aggressive approximations than the competitive approaches (static or IRA) (the blue portions of the bars are shorter). The IRA approach, due to its assumption that the error in the canary output is identical to the error in the full output, cannot use aggressive ALs and thus cannot achieve the full speedup available through approximation. Within the two variants of VideoChef, scene change detector (version B) is slower than an I-frame lookup (version A).

Performance for 20db Threshold. We also evaluate on VideoChef on a different quality thresholds 20dB. Given a larger error budget, Figure 2.3 shows that VideoChef is able to achieve more performance gain over exact computation (1.6x speedup). We also outperform static approximation and IRA by 53.4% and 23.1% and within 26.6% from the Oracle results. Notice that the pipelines where we achieve the maximum performance gain over IRA changes from UIV to DVE.

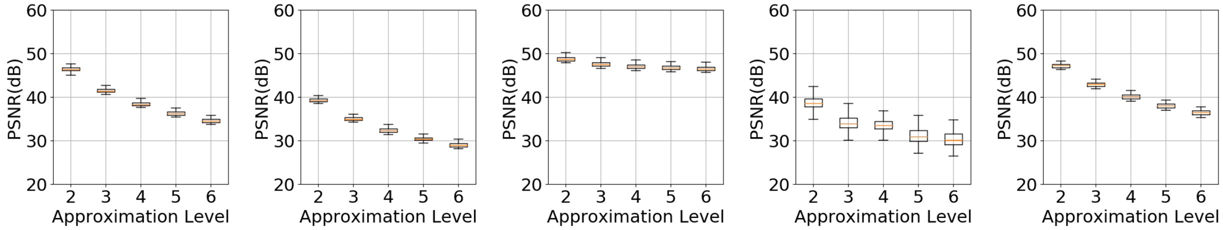
Quality for 30db Threshold. Figure 2.4(a) shows that IRA and static approximation both achieve much higher quality than what the user specified (30 dB), an undesirable outcome here since this comes at the expense of higher execution time. VideoChef on the other hand tracks the Oracle quality quite closely, which in turn meets the user requirement. It does however, drop below the threshold on some inputs, albeit by small amounts. This indicates that a future design feature should compensate for the tendency of VideoChef to sometime drop below the target video quality, say by adding a penalty function when the AL brings it close to the boundary. Further, a carefully designed margin in the searching algorithm can reduce the violation in quality requirement but still achieve speedup. The careful reader would have noticed that for some pipelines, some protocol results are missing here. This happens because no approximation is possible for some pipelines and there is no error introduced and hence, PSNR is not defined.

We also use the percentage of frames that violate the quality threshold to characterize the robustness of each protocol. The violation rate of static approximation, IRA, VideoChef version A and B are 3.27%, 0.64%, 6.6% and 4.79%. Although the two versions of VideoChef have higher violation rates, they are still within a typical user acceptable threshold (5%). We consider the violation may due to two factors – (1) Inaccurate error prediction in the key frame. (2) The quality of non-key frames degrade and drop below the threshold before a fresh key frame is identified and a search triggered. According to our



(a) Deflate filter (b) Emboss filter (c) Boxblur filter (d) Histeq filter (e) Vignette filter

Figure 2.5. The normalized execution time for each filter as the Approximation Level (AL) is varied, across all 106 videos in our dataset. The number of CPU cycles is normalized by the measure for exact computation. As the AL increases, the execution time decreases and this happens consistently across all videos and filters.



(a) Deflate filter (b) Emboss filter (c) Boxblur filter (d) Histeq filter (e) Vignette filter

Figure 2.6. The video quality for each filter as the Approximation Level (AL) is varied, across all 106 videos in our dataset. The effect depends on the video content and the filter being used.

modeling in Section 2.5.3, the violation due to the first factor is limited to at most 5%, while the second error may be inevitable as long as we do not search for every frame. Considering the trade-off between searching overhead and better error control, VideoChef is able to largely reduce the searching overhead and still maintain good quality.

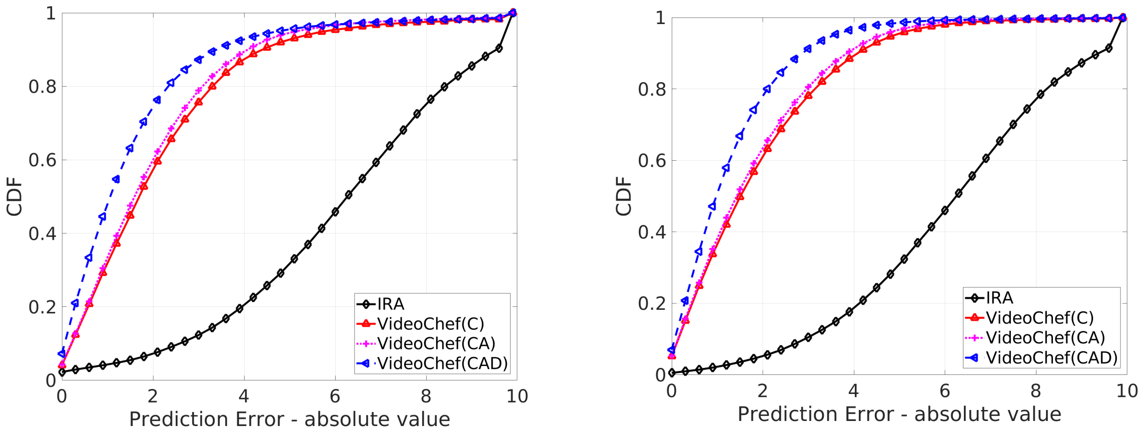
Quality for 20db Threshold. Figure 2.4(b) shows the quality measurement of different protocols across all the pipelines. The mean violation rate averaged across all pipelines of static approximation, IRA, VideoChef version A and B are 0%, 0.23%, 7.18% and 3.93%. In the two quality threshold case, we see the advantage of scene change detection as an add-on in VideoChef version B to decrease the violation rate because it can accurately detect the frame which differs largely from the previous and trigger a required search for optimal approximation levels.

2.5.2 Speedup and Video Quality versus Approximation Levels

This experiment studies (1) how the execution time of each filter varies with the AL setting for that filter and (2) how the video quality varies with the AL setting. This result is dependent on the approximation technique but is independent of the VideoChef configuration used to decide on the AL. We show the results with all the videos in our dataset and 5 out of 10 representative filters in Figure 2.5 (number of executed instructions) and Figure 2.6 (video quality). When showing the result for a specific filter, we only execute on this filter and not the 3-stage pipeline. Here the results have higher variability due to the content-dependent effect. For the execution time, we normalize by the measure for exact computation.

Execution Time. Figure 2.5 shows that as the AL becomes higher, *i.e.*, the approximation becomes more aggressive, the execution time decreases. But the rate of decrease slows down as the AL becomes higher and the behaviors among the different filters in our evaluation are comparable. Note that this is a box plot, but there is little variation across the different videos and hence each AL gives very tight result. This is expected because the amount of processing done in the filter, whether with exact or approximate computation, is *not* content dependent, but the effect of the approximation *is* content dependent.

Quality. Figure 2.6 shows the effect of AL on the video quality when the full frame is used. The quality degrades as the approximation gets more aggressive, but the nature of the decrease is not uniform across all the filters. Even within each filter, the effect on quality depends on the exact video frame, as implied by the vertical data spread for any given AL. We identify two forms of unpredictability of how AL correlates with video quality: with the content (which video frame is being approximated) and with the filter. Due to these two factors, we do not try to come up with a closed form curve for doing the prediction, rather, we do the actual computation with the canary input for a given AL setting, compute the PSNR, and then map it to the PSNR with the full input (Section 2.3.5). Contrast this to the execution time where we create a lookup table through training, which is content independent, and just look it up during the online search (Section 2.3.4). The variability due to video content in the PSNR plot validates our rationale for doing the approximation



(a) Validation set

(b) Test set

Figure 2.7. Results of the error modeling in VideoChef mapping error in canary output to error in full output. The CAD model with characteristics of the frame performs best, though it is only slightly better than the CA models.

in a content-dependent manner. The rationale is shared with [26], but it sets our work apart from the approximation techniques that select the approximation configuration in a content-independent manner.

2.5.3 Evaluation of Error Mapping Models

In this experiment, we evaluate the quality of the various error mapping models in VideoChef. We trained the model on the training video set. Table 2.2 and Figure 2.3 present the performance of our model on the validation videos. Figure 2.7 shows that even a simple model C can greatly reduce the prediction error relative to IRA. Also, as we increase the level of knowledge, the model achieves higher prediction accuracy and model-CAD performs the best due to its good use of the feature extraction from the frames. We can see that with our CAD model, we can successfully control the error within 2dB in 80% of the cases and within 3dB in 90% of the cases. Given these results, we set up a 3dB margin when mapping from the canary error to the full error.

The results on the test videos (Section 2.5.1) show that the cases when we violate the quality requirement are within 10%.

Table 2.2. F-1 measure of different error mapping models averaged over all pipelines. We regard IRA as a pass through error mapping.

Models	IRA	C	CA	CAD
30dB threshold	0.8650	0.9576	0.9594	0.9686
20dB threshold	0.8007	0.9679	0.9660	0.9759

2.5.4 User Perception Study

To evaluate if the protocols cause any perceptual difference, we conduct a small user study with 16 participants. Users were recruited by emailing students of certain ECE classes. We picked 16 videos, 2 from each YouTube content category, randomly picked from our dataset. We processed each video (a snippet of 20 seconds from each, as in the rest of the evaluation) using the Oracle approach and using VideoChef for pipeline DBE.

This pipeline was chosen because its result in the rest of the evaluation is representative and it produces videos which are still visually pleasing. In the experiment, we showed the two versions of each of the 16 videos concurrently, processed using the Oracle and VideoChef tools, without letting the participant know which window corresponded to which tool. All participants watched the videos independently. The participants were asked to rate the videos in four categories: Same, Little difference, Large difference, and Total difference. We gave guidance to the participants for the four categories as difference $\in$ [0%,5%),[5%,20%),[20%,50%), and $\geq$ 50%.

We show the results in Table 2.3. The percentage figure is the percentage of the total number of videos shown, which is 16 × 16 (number of videos × number of users). We conclude that 58.59% of the videos got no difference rating between the Oracle and the VideoChef processed videos, while 34.77% got a little difference rating. Although 6.64% of videos got large difference rating, none of the videos got total difference rating. This validates that qualitatively human perception is not seeing significant difference in video quality due to approximate processing using VideoChef.

Table 2.3. Results of the user studies with 16 videos processed using Oracle and VideoChef.

Degree of difference	Percentage
No difference	58.59%
Little difference	34.77%
Large difference	6.64%
Total difference	0

2.6 Related Work

Approximate Tradeoffs in Computations and Data. Researchers presented various techniques for changing computations at the system level to trade accuracy for performance, *e.g.*, in hardware [21], [24], [37]–[39], runtime systems [16], [18], and compilers [10]–[12], [40], [41]. A key challenge of approximate computing is finding good tradeoffs between accuracy and performance. For this, researchers have looked at both off-line autotuning [11], [12], [22], [42] and on-line dynamic adaptation [16], [18], [19], [23], [43]. In image processing, various techniques exist for synthesizing approximate filter versions, e.g., using genetic programming [6]–[8]. Recently, Lou et al. [9] present "image perforation", an adaptive version of loop perforation tailored for individual image filters. Researchers also proposed storing multimedia data in approximate memories, including standard [10], [44], solid-state [45], and multi-level cell memories specialized for video encodings [46]. We consider such storage approaches complementary to our computation-based technique for video encoding.

Input-Aware Approximation. Several techniques provide input-aware approximations to monitor output quality and control the aggressiveness of the approximation during execution. Green [16] was an early approach that applied dynamic quality monitoring to adjust the level of approximation, based on a user-defined quality function. More recently, input-aware approximation identifies classes of similar inputs and applies different approximations for each input class [47]. Opprox [20] learns the control-flow of the input-optimized program and then selects in which phase to approximate as well as how much to approximate. In contrast to our work, all these approaches use off-line models for prediction of input quality and do not craft the smaller inputs at run-time. Ringenburg *et. al.* [48] proposed online monitoring mechanisms, where a random subset of approximate outputs is compared with a precise output on a sampling basis, or the output of the current execution is predicted from past executions with similar inputs. Raha *et al.* [49] present a precise analysis of accuracy for a commonly used reduce-and-rank computational pattern. Rumba [43] and Topaz [50] detect outliers in intermediate computation results. In contrast to IRA [26] and our VideoChef, these approaches do not use canary inputs to guide the optimization and monitoring and therefore grapple with the overhead issue.

2.7 Conclusion

Fast and resource efficient processing of videos is required in many scenarios. We built a resource efficient and input-aware approximate video processing pipeline called VideoChef. VideoChef controls the approximation in each frame (using the properties of the frame) to meet the user's accuracy requirement. In particular, VideoChef uses a canary-input based approach for fast searching, as proposed in prior work, but overcomes some fundamental challenges by innovating a machine-learning based accurate error estimation technique and an input-aware search technique that finds best approximation settings. We show that VideoChef can provide significant speedup in 9 different video processing pipelines while satisfying user's quality requirements.

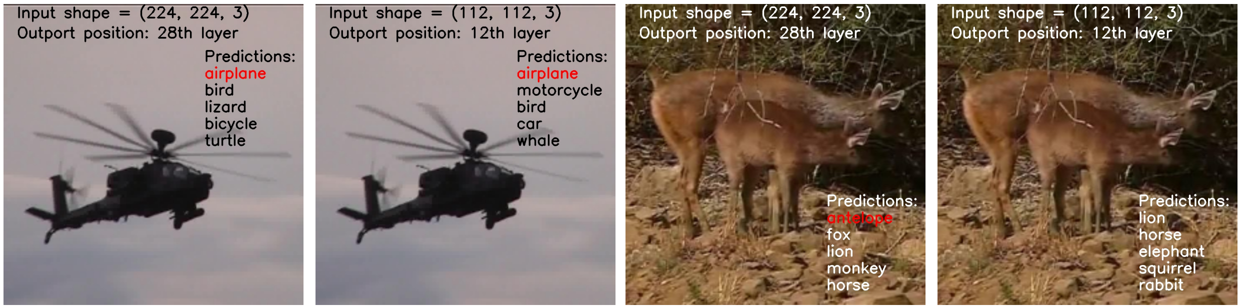
3. APPROXNET: CONTENT AND CONTENTION-AWARE VIDEO OBJECT CLASSIFICATION SYSTEM FOR EMBEDDED CLIENTS

3.1 Introduction

There is an increasing number of scenarios where various kinds of analytics are required to be run on live video streams, on resource-constrained mobile and embedded devices. For example, in a smart city traffic system, vehicles are redirected by detecting congestion from the live video feeds from traffic cameras while in Augmented Reality (AR)/Virtual Reality (VR) systems, scenes are rendered based on the recognition of objects, faces or actions in the video. These applications require low latency for event classification or identification based on the content in the video frames. Most of these videos are captured at end-client devices such as IoT devices, surveillance cameras, or head-mounted AR/VR systems. Video transport over wireless network is slow and these applications often must operate under intermittent network connectivity. Hence such systems must be able to run video analytics in-place, on these resource-constrained client devices[1] to meet the low latency requirements for the applications.

State-of-the-art is too heavy for embedded devices: Most of the video analytics queries involve performing an inference over DNNs (mostly convolutional neural networks, *aka* CNNs) with a variety of functional architectures for performing the intended tasks like classification [51]–[54], object detection [55]–[58], face [59]–[62] or action recognition [63]–[66] etc. With advancements in deep learning and emergence of complex architectures, DNN-based models have become *deeper* and *wider*. Correspondingly their memory footprints and their inference latency have become significant. For example, DeepMon [67] runs the VGG-16 model at approximately 1-2 frames-per-second (fps) on a Samsung Galaxy S7. ResNet [53], with its 101-layer version, has a memory footprint of 2.8 GB and takes 101 ms to perform inference on a single video frame on the NVIDIA Jetson TX2. MCDNN,

[1] ↑For end client devices, we will use the term “mobile devices” and “embedded devices” interchangeably. The common characteristic is that they are computationally constrained. While the exact specifications vary across these classes of devices, both are constrained enough that they cannot run streaming video analytics without approximation techniques.


Input shape = (224, 224, 3)
Outport position: 28th layer
Predictions:
airplane
bird
lizard
bicycle
turtle
Input shape = (112, 112, 3)
Outport position: 12th layer
Predictions:
airplane
motorcycle
bird
car
whale
Input shape = (224, 224, 3)
Outport position: 28th layer
Predictions:
antelope
fox
lion
monkey
horse
Input shape = (112, 112, 3)
Outport position: 12th layer
Predictions:
lion
horse
elephant
squirrel
rabbit

(a) Simple video frame

(b) Complex video frame

Figure 3.1. Examples of using a heavy DNN (on the left) and a light DNN (on the right) for simple and complex video frames in a video frame classification task. The light DNN downsamples an input video frame to half the default input shape and gets prediction labels at an earlier layer. The classification is correct for the simple video frame (red label denotes the correct answer) but not for the complex video frame.

Mainstream, VideoStorm and Liu *et al.* [2], [68]–[70] require either the cloud or the edge servers to achieve satisfactory performance. Thus, on-device inference with a low and stable inference latency (*i.e.* 30 fps) remains a challenging task.

Content and contention aware systems: Content characteristics of the video stream is one of the key runtime conditions of the systems with respect to approximate video analytics. This can be leveraged to achieve the desired latency-accuracy tradeoff in the systems. For example, as shown in Figure 3.1, if the frame is very simple, we can downsample it to half of the original dimensions and use a shallow or small DNN model to make an accurate prediction. If the frame is complex, the same shallow model might result in a wrong prediction and would need a larger DNN model.

Resource contention is another important runtime condition that the video analytics system should be aware of. In several scenarios, the mobile devices support multiple different applications, executing concurrently. For example, while an AR application is running, a voice assistant might kick in if it detects background conversation, or a spam filter might become active, if emails are received. All these applications share common resources on the device, such as, CPU, GPU, memory, and memory bandwidth, and thus lead to *resource contention* [71]–[73] as these devices do not have advanced resource isolation mechanisms. It is currently an unsolved problem how video analytics systems running on the mobile devices

Table 3.1. ApproxNet's main features and comparison to existing systems.

Solution	Single model	Considers switching overhead	Focused on video	Handles runtime conditions	Open-sourced	Replicable in our datasets
MCDNN [MobiSys'16]	❌	🟡	❌	❌	🟡	✅
MobileNets [ArXiv'17]	❌	❌	❌	❌	✅	✅
MSDNet [ICLR'18]	✅	❌	❌	❌	✅	✅
BranchyNet [ICPR'16]	✅	❌	❌	❌	✅	❌
NestDNN [MobiCom'18]	✅	🟡	❌	🟡	❌	❌
ApproxNet	✅	✅	✅	✅	✅	✅

✅Supported 🟡Partially Supported ❌Not Supported

Notes for partially support:

1. MCDNN and NestDNN only consider the switching overhead in memory size, but in the execution latency.
2. NestDNN handles multiple concurrent DNN applications with joint optimization goals.
3. The core models in MCDNN are open-sourced while the scheduling components are not.

can maintain a low inference latency under such variable resource availability and changing content characteristics, so as to deliver satisfactory user experience.

Single-model vs. multi-model adaptive systems: How do we architect the system to operate under such varied runtime conditions? Multi-model designs came first in the evolution of systems in this space. These created systems with an ensemble of multiple models, satisfying varying latency-accuracy conditions, and some scheduling policy to choose among the models. MCDNN [68], being one of most representative works, and well-known DNNs like ResNet, MobileNets [53], [74] all fall into this category. On the other hand, the single-model designs, which emerged after the multi-model designs, feature one model with tuning knobs inside so as to achieve different latency-accuracy goals. These typically have lower switching overheads from one execution path to another, compared to the multi-branch models. MSDNet [75], BranchyNet [76], NestDNN [77] are representative works in this single-model category. However, none of these systems can adapt to runtime conditions, primarily, changes in content characteristics and contention levels on the device.

Our solution: ApproxNet. We present ApproxNet, our content and contention aware object classification system over streaming videos, geared toward GPU-enabled mobile/embedded devices. We introduce a novel workflow with a set of integrated techniques to solve the three main challenges as mentioned above: (1) on-device real-time video analytics, (2) content and contention aware runtime calibration, (3) a single-model design. The fundamental idea behind ApproxNet is to perform approximate computing with tuning knobs that are changed automatically and seamlessly within the same video stream. These knobs trade off the accuracy of the inferencing for reducing inference latency and thus match the frame rate of the video stream or the user's requirement on either a latency target or an accuracy target. The optimal configuration of the knobs is set, particularly considering the resource contention and complexity of the video frames, because these runtime conditions affects the accuracy and latency of the model much.

In Table 3.1, we compare ApproxNet with various representative prior works in this field. First of all, none of these systems [68], [74]–[77] is able to adapt to dynamic runtime conditions (changes in content characteristics and contention levels) as we can. Second, although most systems are able to run at variable operation points of performance, MCDNN [68] and MobileNets [74] use a multi-model approach and incur high switching penalty. For those that works with a single model, namely, MSDNet [75], BranchyNet [76], and NestDNN [77], they do not consider switching overheads in their models (except partially for NestDNN, which considers switching cost in memory size), do not focus on video content, and do not show how their models can adapt to the changing runtime conditions (except partially for NestDNN, which considers joint optimization of multiple DNN workloads). For evaluation, we mainly compare to MCDNN, ResNet, and MobileNets, as the representatives of multi-model approaches, and MSDNet, as the single-model approach. We cannot compare to BranchyNet as it is not designed and evaluated for video analytics and thus not suitable for our datasets. BranchyNet paper evaluates it on small image dataset: MNIST and CIFAR. We cannot compare to NestDNN since it's models or source-code and architecture and hyperparameter details are publicly available and we need those for replicating the experiments.

To summarize, we make the following contributions in this work:

1. We develop an end-to-end, approximate video object classification system, ApproxNet, that can handle dynamically changing workload contention and video content characteristics on resource-constrained embedded devices. It achieves this through

performing system context-aware and content-aware approximations with the offline profiling and online lightweight sensing and scheduling techniques.

2. We design a novel workflow with a set of integrated techniques including the adaptive DNN that allows runtime accuracy and latency tuning *within a single model*. Our design is in contrast to ensemble systems like MCDNN that are composed of multiple independent model variants capable of satisfying different requirements. Our single-model design avoids high switching latency when conditions change and reduces RAM and storage usage.

3. We design ApproxNet to make use of video features, rather than treating video as a sequence of image frames. Such characteristics that we leverage include the temporal continuity in content characteristics between adjacent frames. We empirically show that on a large-scale video object classification dataset, popular in the vision community, ApproxNet achieves a superior accuracy-latency tradeoff than the three state-of-the-art solutions on mobile devices, MobileNets, MCDNN, and MSDNet (Figures 3.9 and 3.10).

The rest of the chapter is organized as follows. Section 3.2 gives the relevant background. Section 3.3 gives our high-level solution overview. Section 3.4 gives the detailed design. Section 3.5 evaluates our end-to-end system. Section 3.6 discusses the details about training the DNN. Section 3.7 highlights the related works. Finally, Section 3.8 gives concluding remarks.

3.2 Background and Motivation

3.2.1 DNNs for Streaming Video Analytics

DNNs have become a core element of various video processing tasks such as frame classification, human action recognition, object detection, face recognition, and so on. Though accurate, DNNs are computationally expensive, requiring significant CPU and memory resources. As a result, these DNNs are often too slow when running on mobile devices and become the latency bottleneck in video analytics systems. Huynh *et al.* [67] experimented with VGG [51] of 16 layers on the Samsung Galaxy S7 and noted that classification on a single image takes as long as 644 ms, leading to less than 2 fps for continuous classification. Motivated by the observation, we explore in ApproxNet how we can make DNN-based video analytics pipelines more efficient through content-aware approximate computation *within the neural network*.

ResNet: Deep DNNs are typically hard to train due to the vanishing gradient problem [53]. ResNet solved this problem by introducing a short cut identity connection in between layers, which helps achieve at least the same accuracy upon further increasing the number of network layers. The unit of such connected layers is called a ResNet block. We leverage this key idea of a deeper model producing no higher error than its shallower counterpart, for the construction of an architecture that provides more accurate execution as it proceeds deeper into the DNN.

Spatial Pyramid Pooling (SPP) [78]: Popular DNN models, including ResNet, consist of convolutional and max-pooling (CONV) layers and fully-connected (FC) layers and the shape of an input image is fixed. Changing the input shape in a CNN typically requires redesigning the architecture of the CNN. SPP is a special layer that eliminates this constraint. The SPP layer is added at the end of CONV layers, providing the following FC layers with a fixed-dimensioned feature representation by pooling the CONV layer output with bins whose shapes are proportional to the input shape. We use SPP layers to change input shape as an approximation knob.

Trading-off accuracy for inference latency: DNNs can have several variants due to different configurations, and these variants yield different accuracies and latencies. But these variants are trained and inferenced independently and cannot be switched efficiently at inference time to meet differing accuracy or latency requirements. For example, MCDNN [68] sets up an ensemble of (up to 68) model variants to satisfy different latency/accuracy/cost requirements. MSDNet [75] enables five early exits in a *single* model but does not evaluate on streaming video with any variable content or contention situations. Hence, we set ourselves to design a single-model DNN that is capable of handling the accuracy-latency trade-off at inference time and guarantees our video analytics system's performance under variable content and runtime conditions.

3.2.2 Content-aware Approximate Computing

IRA [26] and VideoChef [79] first introduced the notion of content-aware approximation and applied the idea, respectively to image and video processing pipelines. These works for the first time showed how to tune approximation knobs as content characteristics change, *e.g.* the video scene became more complex. In particular, IRA performs approximation targeting individual images, while VideoChef exploits temporal similarity among frames in a video to further optimize accuracy-latency trade-off. However, these works do not perform approximation for ML-based inferencing, which comprises the dominant form of video analytics. In contrast, we apply approximation to the DNN model itself with the intuition that depending on complexity of the video frame, we want to feed input of a different shape and output at a different depth of layers to achieve the target accuracy.

3.2.3 Contention-aware Scheduling

Managing the resource contention of multiple jobs on high-performance clusters is a very active area of work. Bubble-Up [80], Bubble-Flux [81], and Pythia [82] develop characterization methodologies to predict the performance degradation of latency-sensitive applications due to shared resources in the memory subsystem. SMiTe [83] and Paragon [84] further extend such concurrent resource contention scenario to SMT processors and thousands of different unknown applications, respectively. On the other hand, we apply contention-aware approximation to the DNN model on the embedded and mobile devices, and consider the three major sources of contention – CPU, GPU, and memory bandwidth.

3.3 Overview

Here we give a high-level overview of ApproxNet. In Section 3.4, we provide details of each component.

3.3.1 Design Principles and Motivation

We set four design requirements for streaming video analytics on the embedded devices motivated by real-world scenarios and needs. *First*, the application should adapt to changing input characteristics, such as, the complexity of the video frames because the accuracy of the DNN may vary based on the content characteristics. We find such changes happen often enough within a single video stream and without any clear predictive pattern. *Second*, the application should adapt to the resource contention due to the shared CPU, GPU, memory, or memory bandwidth with other concurrent applications on the same device. Such contention can happen frequently with co-location due to limited resources and the lack of clean resource isolation on these hardware platforms. Again, we find that such changes can happen without a clear predictive pattern. *Third*, the application should support different target accuracies or latencies at runtime with little transition overhead. For example, the application may require low latency when a time-critical query, such as detection of a miscreant, needs to be executed and have no such constraint for other queries on the steam. Thus, the aggregate model must be able to make efficient transitions in the tradeoff space of accuracy and latency, and less obviously throughput, optionally using edge or cloud servers. *Fourth*, the application must provide real-time processing speed (30 fps) while running on the mobile/embedded device. To see three instances where these four requirements come together, consider mobile VR/AR games like Pokemon Go (some game consoles support multitasking, accuracy requirements may change with the context of the game), autonomous vehicles (feeds from multiple cameras are processed on the same hardware platform resulting in contention, emergency situations require lower latency than benign conditions such as for fuel efficiency) and autonomous drones (same arguments as for autonomous vehicles).

A *non-requirement* in our work is that multiple concurrent applications consuming the same video stream be jointly optimized. MCDNN [68], NestDNN [77], and Mainstream [69] bring significant design sophistication to handle the concurrency aspect. However, we are only interested in optimizing a single video analytics application.

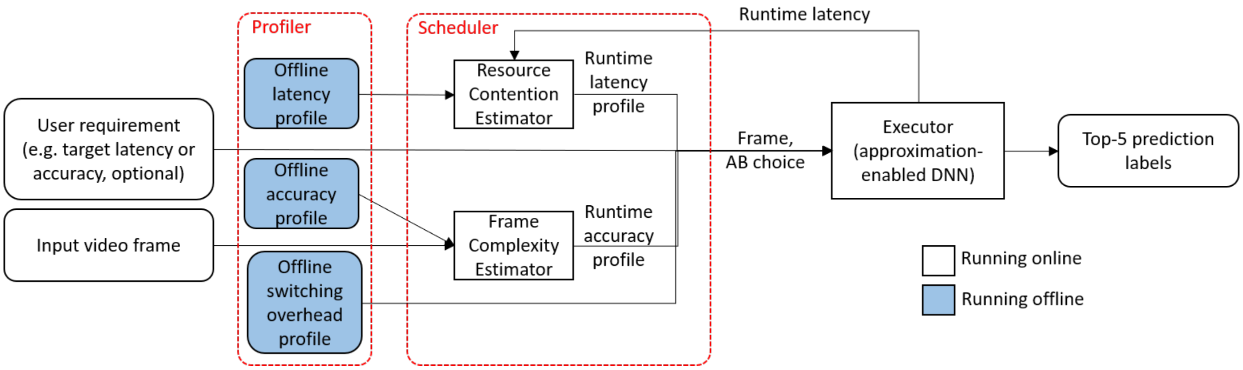


Figure 3.2. Workflow of ApproxNet. The input is a video frame and an optional user requirement, and the outputs are prediction labels of the video frame. Note that the shaded profiles are collected offline to alleviate the online scheduler overhead.

3.3.2 Design Intuition and Workflow

To address these challenges in our design, we propose a novel workflow with a set of integrated techniques to achieve a content and contention-aware video object classification system. We show the overall structure with three major functional units: *executor*, *profiler*, and *scheduler* in Figure 3.2. ApproxNet takes a video frame and optional user requirement for target accuracy or latency as an input, and produces top-5 prediction labels of the object classes as outputs.

The executor (Section 3.4.1) is an approximation-enabled, single-model DNN. Specifically, the single-model design largely reduces the switching overhead so as to support the adaptive system. On the other hand, the multiple **approximation branches (ABs)**, each with variable latency and accuracy specs, are the key to support dynamic content and contention conditions. ApproxNet is designed to provide real-time processing speed (30 fps) on our target device (NVIDIA Jetson TX2). Compared to the previous single-model designs like MSDNet and BranchyNet, ApproxNet provides novelty in enabling both depth and shape as the approximation knob for run-time calibration.

The scheduler is the key component to react to the dynamic content characteristics and resource contention. Specifically, it selects an AB to execute by combining the precise accuracy estimation of each AB due to changing content characteristics via a **Frame Com-**

plexity Estimator (**FCE**, Section 3.4.2), the precise latency estimation of each AB due to resource contention via a **Resource Contention Estimator** (**RCE**, Section 3.4.3), and the switching overhead among ABs (Section 3.4.4). It finally reaches a decision on which AB to use based on the user's latency or accuracy requirement and its internal accuracy, latency, and overhead estimation (Section 3.4.5).

Finally, to achieve our goal of real-time processing, low switching overhead, and improved performance under dynamic conditions, we design an offline profiler (Section 3.4.4). We collect three profiles offline — first, the accuracy profile for each AB on video frames of different complexity categories; second, the inference latency profile for each AB under variable resource contention, and third, the switching overhead between any two ABs.

Video-specific design. We incorporate these video-specific designs in ApproxNet, which is orthogonal to the existing techniques presented in the prior works on video analytics, e.g. frame sampling [2], [69] and edge device offloading [70].

1. The FCE uses a Scene Change Detector (SCD) as a preprocessing module to further alleviate its online cost. This optimization is beneficial because it reduces the frequency with which the FCE is invoked (only when the SCD flags a scene change). This relies on the intuition that discontinuous jumps in frame complexity are uncommon in a video stream.

2. The scheduler decides whether to switch to a new AB or stay depending on how long it predicts the change in the video stream to last and the cost of switching.

3. We drive our decisions about the approximation knobs by the goal of keeping up with the streaming video rate (30 fps). We achieve this under most scenarios when evaluated with a comprehensive video dataset (the ILSVRC VID dataset).

3.4 Design and Implementation

3.4.1 Approximation-enabled DNN

ApproxNet's key enabler, an approximation-enabled DNN, is designed to support multiple accuracy and latency requirements at runtime *using a single DNN model*. To enable

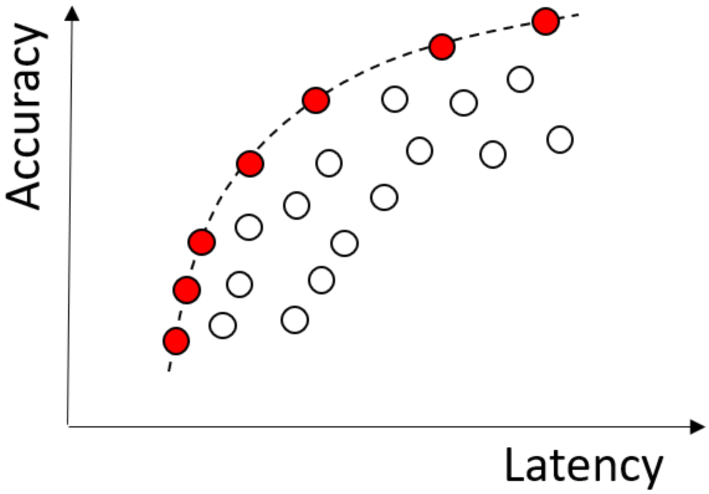
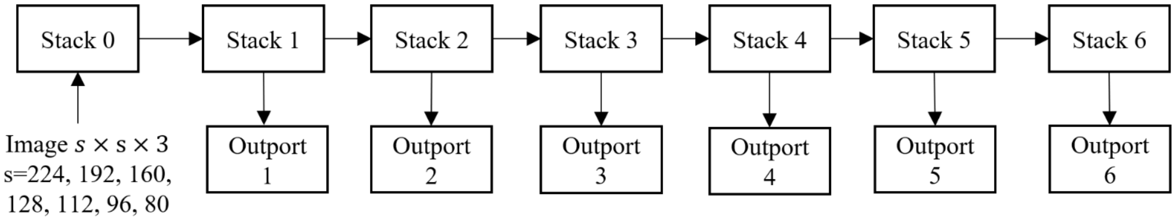


Figure 3.3. A Pareto frontier for trading-off accuracy and latency in a particular frame complexity category and at a particular contention level.

this, we design a DNN that can be approximated using two approximation knobs. The DNN can take an input video frame in different shapes, which we call *input shapes*, our first approximation knob and it can produce a classification output at multiple positions in the intervening layers, which we call *outports*, our second approximation knob. There are doubtless other approximation knobs, *e.g.* model quantization, frame sampling, and others depending on specific DNN models. These can be incorporated into ApproxNet and they will all fit within our novelty of the one-model DNN to achieve real-time on-device, adaptive inference. The appropriate setting of the tuning knobs can be determined on the device (as is done in our considered usage scenario) or, in case this computation is too heavyweight, this can be determined remotely and sent to the device through a reprogramming mechanism such as [85].

Combining these two approximation knobs, ApproxNet creates various **approximation branches (ABs)**, which trade off between accuracy and latency, and can be used to meet a particular user requirement. This tradeoff space defines a set of Pareto optimal frontiers, as shown in Figure 3.3. Here, the scatter points represent the accuracy and latency achieved by all ABs. A Pareto frontier defines the ABs which are either superior in accuracy or latency against all other branches.



Stack 0	Stack 1	Stack 2	Stack 3	Stack 4	Stack 5	Stack 6
Conv, 7x7, 64 stride 2 Relu	ResNetBlock 3x3, 128 stride 2	ResNetBlock 3x3, 128 stride 1	ResNetBlock 3x3, 256 stride 2	ResNetBlock 3x3, 256 stride 1	ResNetBlock 3x3, 256 stride 1	ResNetBlock 3x3, 512 stride 2
Max pool 3x3 stride 2	ResNetBlock 3x3, 128 stride 1	ResNetBlock 3x3, 128 stride 1	ResNetBlock 3x3, 256 stride 1	ResNetBlock 3x3, 256 stride 1	ResNetBlock 3x3, 256 stride 1	ResNetBlock 3x3, 512 stride 1
ResNetBlock*3 3x3, 64 stride 1						ResNetBlock 3x3, 512 stride 1

Figure 3.4. The architecture of the approximation-enabled DNN in ApproxNet.

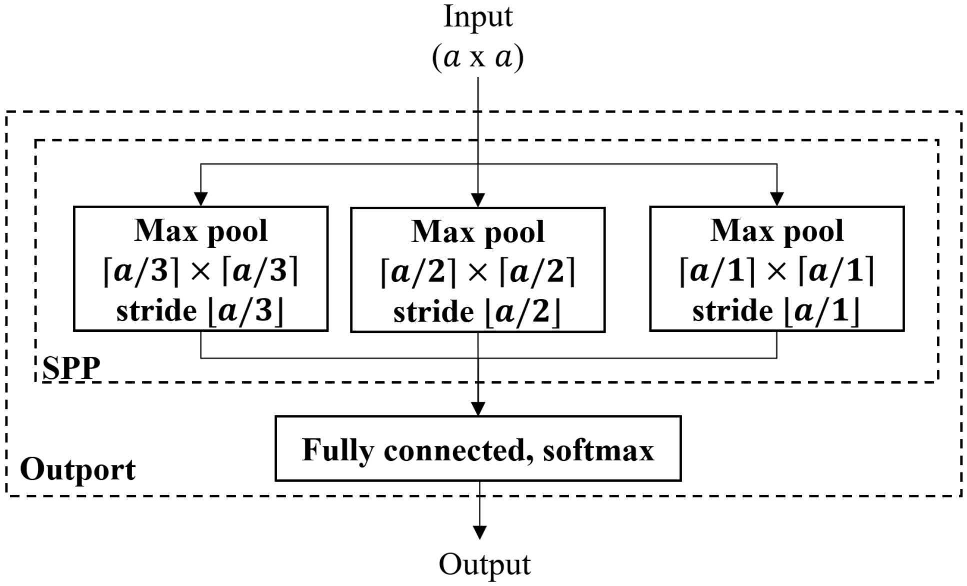

Input
(a x a)
Max pool
$\lceil a/3 \rceil \times \lceil a/3 \rceil$
stride $\lfloor a/3 \rfloor$
Max pool
$\lceil a/2 \rceil \times \lceil a/2 \rceil$
stride $\lfloor a/2 \rfloor$
Max pool
$\lceil a/1 \rceil \times \lceil a/1 \rceil$
stride $\lfloor a/1 \rfloor$
SPP
Fully connected, softmax
Outport
Output

Figure 3.5. The outport of the approximation-enabled DNN.

We describe our design using ResNet as the base DNN, though our design is applicable to any other mainstream CNN consisting of convolutional (CONV) layers and fully-connected (FC) layers such as VGG [51], DenseNet [54] and so on. Figure 3.4 shows the design of our DNN using ResNet-34 as the base model. This enables 7 input shapes ($s \times s \times 3$ for

$s = 224, 192, 160, 128, 112, 96, 80$) and 6 outports (after 11, 15, 19, 23, 27, and 33 layers). We adapt the design of ResNet in terms of the stride, shape, number of channels, use of convolutional layer or maxpool, and connection of the layers. In addition, we create *stacks*, with stacks numbering 0 through 6 and each stack having 4 or 6 ResNet layers and a variable number of blocks from the original ResNet design (Table 3.2). We then design an outport (Figure 3.5), and connect with stacks 1 to 6, whereby we can get prediction labels by executing only the stacks (i.e., the constituent layers) till that stack. The use of 6 outports is a pragmatic system choice—too small a number does not provide enough granularity to approximate in a content and contention-aware manner and too many leads to a high training burden. Further, to allow the approximation knob of downsampling the input frame to the DNN, we use the SPP layer at each outport to pool the feature maps of different shapes (due to different input shapes) into one unified shape and then connect with an FC layer. The SPP layer performs max-pooling on its input by three different levels $l = 1, 2, 3$ with window size $\lceil a/l \rceil$ and stride $\lfloor a/l \rfloor$, where a is the shape of the input to the SPP layer. Note that our choice of the 3-level pyramid pooling is a typical practice for using the SPP layer [78]. In general, a higher value of l requires a larger value of a on the input of each outport, thereby reducing the number of possible ABs. On the other hand, a smaller value of l results in coarser representations of spatial features and thus reduces accuracy. To support the case $l = 3$ in the SPP, we require that the input shape of an outport be no less than 7 pixels in width and height, *i.e.*, $a \geq 7$. This results in ruling out some input shapes as in Table 3.2. Our model has 30 configuration settings in total, instead of 7×6 (number of input shapes × number of outports) because too small input shapes cannot be used when the outport is deep.

To train ApproxNet towards finding the optimal parameter set θ, we consider the softmax loss $L_{s,\mathrm{i}}(\theta)$ defined for the input shape $s \times s \times 3$ and the outport i. The total loss function $L(\theta)$ that we minimize to train ApproxNet is a weighted average of $L_{s,\mathrm{i}}(\theta)$ for all s and i, defined as

$$L(\theta) = \sum_{\forall \mathrm{i}} \frac{1}{n_{\mathrm{i}}} \sum_{\forall s} L_{s,\mathrm{i}}(\theta) \tag{3.1}$$

Table 3.2. The list of the total 30 ABs supported for a baseline DNN of ResNet-34, given by the combination of the input shape and the outport from which the result is taken. "–" denotes the undefined settings.

Input shape	Outport 1	Outport 2	Outport 3	Outport 4	Outport 5	Outport 6
224x224x3	28x28x64	28x28x64	14x14x64	14x14x64	14x14x64	7x7x64
192x192x3	24x24x64	24x24x64	12x12x64	12x12x64	12x12x64	–
160x160x3	20x20x64	20x20x64	10x10x64	10x10x64	10x10x64	–
128x128x3	16x16x64	16x16x64	8x8x64	8x8x64	8x8x64	–
112x112x3	14x14x64	14x14x64	7x7x64	7x7x64	7x7x64	–
96x96x3	12x12x64	12x12x64	–	–	–	–
80x80x3	10x10x64	10x10x64	–	–	–	–

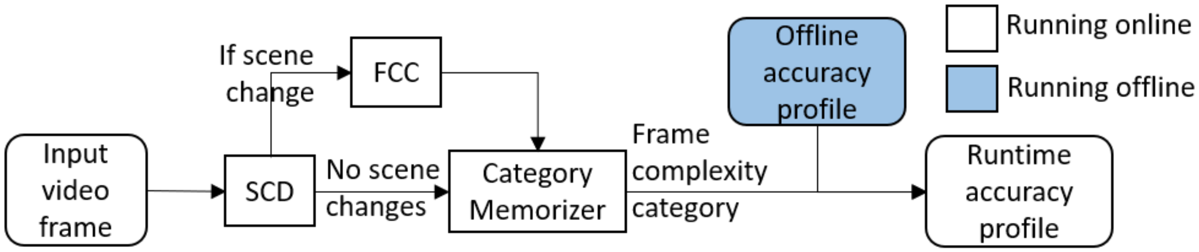


Figure 3.6. Workflow of the Frame Complexity Estimator.

where the value of n_i is the factor that normalizes the loss at an outport by dividing by the number of shapes that are supported at that port i. This makes each outport equally important in the total loss function. For mini-batch, we use 64 frames for each of the 7 different shapes. To train on a particular dataset or generalize to other architectures, we discuss more details in Section 3.6.

3.4.2 Frame Complexity Estimator (FCE)

The design goal of the Frame Complexity Estimator (FCE), which executes online, is to estimate the expected accuracy of each AB in a content-aware manner. It is composed of a Frame Complexity Categorizer (FCC) and a Scene Change Detector (SCD) and it makes use of information collected by the offline profiler (described in Section 3.4.4). The workflow of the FCE is shown in Figure 3.6.

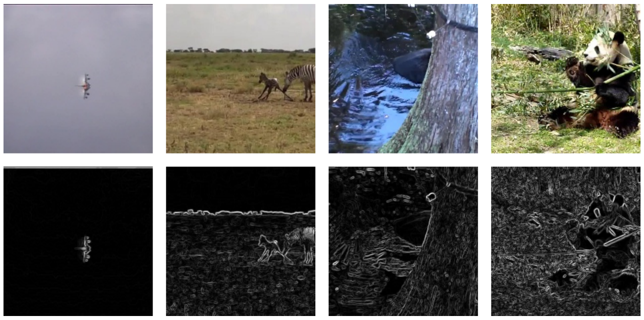


Figure 3.7. Sample frames (first row) and edge maps (second row), going from left to right as simple to complex. Normalized mean edge values from left to right: 0.03, 0.24, 0.50, and 0.99 with corresponding frame complexity categories: 1, 3, 6, and 7

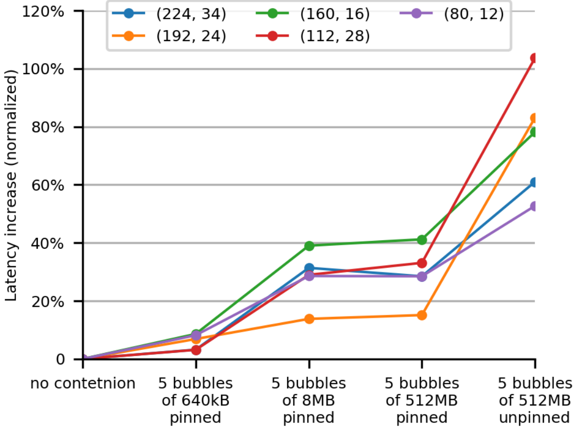


Figure 3.8. Latency increase of several ABs in ApproxNet under resource contention with respect to those under no contention. The input shape and outport depth of the branches are labeled.

Frame Complexity Categorizer (FCC). FCC determines how hard it is for ApproxNet to classify a frame of the video. Various methods have been used in the literature to calculate frame complexity such as edge information-based methods [86], [87], compression information-based methods [86] and entropy-based methods [88]. In this work, we use mean edge value as the feature of the frame complexity category, since it can be calculated with very low computation overhead (3.9 ms per frame on average in our implementation). Although some counterexamples may show the edge value is not relevant, we show empirically that with this feature, the FCE is able to predict well the accuracy of each AB with respect to a large dataset.

To expand, we extract an edge map by converting a color frame to a gray-scale frame, applying the Scharr operator [89] in both horizontal and vertical directions, and then calculating the L2 norm in both directions. We then compute the mean edge value of the edge map and use a pre-trained set of boundaries to quantize it into several frame complexity categories. The number and boundaries of categories is discussed in Section 3.4.4. Figure 3.7 shows examples of frames and their edge maps from a few different complexity categories.

Scene Change Detector (SCD). The Scene Change Detector is designed to further reduce the online overhead of FCC by determining if the content in a frame is significantly different from that in a prior frame in which case the FCC will be invoked. SCD tracks a histogram of pixel values, and declares a scene change when the mean of the absolute difference across all bins of the histograms of two consecutive frames is greater than a certain threshold (45% of the total pixels in our design). To bound the execution time of SCD we use only the R-channel and downsample the shape of the frame to 112 × 112. We empirically find that such optimizations do not reduce the accuracy of detecting new scenes but do reduce the SCD overhead, to only 1.3 ms per frame.

3.4.3 Resource Contention Estimator (RCE)

The design goal of the Resource Contention Estimator (RCE), which also executes online, is to estimate the expected latency of each AB in a contention-aware manner. Under resource contention, each AB is affected differently and we call the latency increase pattern the *latency sensitivity*. As shown in Figure 3.8, five approximation branches have different ratios of latency increase under a certain amount of CPU and memory bandwidth contention.

Ideally we would use a sample classification task to probe the system and observe its latency under the current contention level C. The use of such micro-benchmarks is commonly done in datacenter environments [82], [90]. However, we do not need the additional probing since the inference latencies of the latest video frames form a natural observation of the contention level of the system. Thus we use the averaged inference latency $\overline{L_B}$ of the current AB B across the latest N frames. We then check the latency sensitivity $L_{B,C}$ of branch B (offline profile as discussed in Section 3.4.4) and get an estimated contention level $\hat{C}$ with the nearest neighbor principle,

$$\hat{C} = \arg\min_{C} abs(L_{B,C} - \overline{L_B}) \tag{3.2}$$

By default, we use $N = 30$. This will lead to an average over last one second when frame-per-second is 30. Smaller N can make ApproxNet adapt faster to the resource contention, while larger N make it more robust to the noise. Due to the limited observation data (one data point per frame), we cannot adapt to resource contention that is changing faster than the frame rate.

Specifically in this work, we consider CPU, GPU and memory contention among tasks executing on the device (our SoC board shares the memory between the CPU and the GPU), but our design is agnostic to what causes the contention. Our methodology considers the resource contention as a black-box model because we position ourselves as an application-level design instead of knowing the execution details of all other applications. We want to deal with the effect of contention, rather than mitigating it by modifying the source of the contention.

3.4.4 Offline Profiler

Per-AB Content-Aware Accuracy Profile. The boundaries of frame complexity categories are determined based on the criteria that all frames within a category should have an identical Pareto frontier curve (Figure 3.3) and frames in different categories should have distinct curves. We start with considering the whole set of frames as belonging to a single category and split the range of mean edge values into two in an iterative binary manner, till the above condition is satisfied. In our video datasets, we derive 7 frame complexity categories with 1 being the simplest and 7 the most complex. To speedup the online estimation of the accuracy on any candidate approximation branch, we create the offline accuracy profile $A_{B,F}$ given any frame complexity categories F and any ABs B, after the 7 frame complexity categories are derived.

Per-AB Contention-Aware Latency Profile. ApproxNet is able to select ABs at runtime in the face of resource contention. Therefore, we perform offline profiling of the inference latency of each AB under different levels of contention. To study the resource contention, we develop our synthetic contention generator (CG) with tunable "contention levels" to simulate resource contention and help our ApproxNet profile and learn to react under such scenarios in real-life. Specifically, we run each AB in ApproxNet with the CG in varying contention levels to collect its contention-aware latency profile. To reduce the profiling cost, we quantize the contention to 10 levels for GPU and 20 levels for CPU and memory and then create the offline latency profile $L_{B,C}$ for each AB B under each contention level C. Note that contention increases latency of the DNN but does not affect its accuracy. Thus, offline profiling for accuracy and latency can be done independently and parallelly and profiling overhead can be reduced.

Switching Overhead Profile. Since we find that the overhead of switching between some pairs of ABs is non-negligible, we profile the overhead of switching latency between any pairs of approximation branches offline. This cost is used in our optimization calculation to select the best AB.

3.4.5 Scheduler

The main job of the scheduler in ApproxNet is to select an AB to execute. The scheduler accepts user requirement on either the minimum accuracy, the maximum latency per frame. The scheduler requests from the FCE a runtime accuracy profile (B is the variable for the AB and $\hat{F}$ is the frame category of the input video frame) $A_{B,\hat{F}} \forall B$. It then requests from the RCE a runtime latency profile ($\hat{C}$ is the current contention level) $L_{B,\hat{C}} \forall B$. Given a target accuracy or latency requirement, we can easily select the AB to use from drawing the Pareto frontier for the current $(\hat{F}, \hat{C})$. If no Pareto frontier point satisfies the user requirement, ApproxNet picks the AB that achieves metric value closest to the user requirement. If the user does not set any requirement, ApproxNet sets a latency requirement to the frame interval of the incoming video stream. One subtlety arises due to the cost of switching from one AB to another. This cost has to be considered by the scheduler to avoid too frequent switches without benefit to outweigh the cost.

To rigorously formulate the problem, we denote the set of ABs as $\mathcal{B} = \{B_1, ...B_N\}$ and the optimal AB the scheduler has to determine as B_{opt}. We denote the accuracy of branch B on a video frame with frame complexity F as $A_{B,F}$, the estimated latency of branch B under contention level C as $L_{B,C}$, the one-time switch latency from branch B_p to B_{opt} as $L_{B_p \rightarrow B_{opt}}$, and the expected time window over which this AB can be used as W (in the unit of frames). For W, we use the average number of frames for which the latest ABs can stay unchanged and this term introduces hysteresis to the system so that the AB does not switch back and forth frequently. The constant system overhead per frame (due to SCD, FCC, and resizing the frame) is L_0. Thus, the optimal branch B_{opt}, given the latency requirement L_τ, is:

$$B_{opt} = \arg\max_{B \in \mathcal{B}} A_{B,F}, \ s.t. \ L_{B,C} + \frac{1}{W} L_{B_p \to B} + L_0 \leq L_\tau \tag{3.3}$$

When the accuracy requirement A_τ is given,

$$B_{opt} = \arg\min_{B \in \mathcal{B}} [L_{B,C} + \frac{1}{W} L_{B_p \to B} + L_0], s.t. \ A_{B,F} \geq A_\tau \tag{3.4}$$

3.5 Evaluation

3.5.1 Evaluation Platforms

We evaluate ApproxNet by running it on an NVIDIA Jetson TX2 [91], which includes 256 NVIDIA Pascal CUDA cores, a dual-core Denver CPU, a quad-core ARM CPU on a 8GB unified memory [92] between CPU and GPU. The specification of this board is close to what is available in today's high-end smart phones such as Samsung Galaxy S20 and Apple iPhone 12. We train the approximation-enabled DNN on a server with NVIDIA Tesla K40c GPU with 12GB dedicated memory and an octa-core Intel i7-2600 CPU with 24GB RAM. For both the embedded device and the training server, we install Ubuntu 16.04 and TensorFlow v1.14.

3.5.2 Datasets, Task, and Metrics

ImageNet VID dataset: We evaluate ApproxNet on the video object classification task using ILSVRC 2015 VID dataset [93]. Although the dataset is initially used for object detection, we convert the dataset so that the task is to classify the frame into one of the ground truth object categories. If multiple objects exist, the classification is considered correct if matched with any one of the ground truth classes and this rule applies to both ApproxNet and baselines. According to our analysis, 89% of the video frames are single-object-class frames and thus the accuracy is still meaningful under such conversion.

For the purpose of training, ILSVRC 2015 VID training set contains too many redundant video frames, leading to an over-fitting issue. To alleviate this problem, we follow the best practice in [94] such that the VID training dataset is sub-sampled every 180 frames and the resulting subset is mixed with ILSVRC 2014 detection (DET) training dataset to construct a new dataset with DET:VID=2:1. We use 90% of this video dataset to train ApproxNet's DNN model and keep aside another 10% as validation set to fine-tune ApproxNet (offline profiling). To evaluate ApproxNet's system performance, we use ILSVRC 2015 VID validation set – we refer to this as the "test set".

ImageNet IMG dataset: We also use ILSVRC 2012 image classification dataset [95] to evaluate the accuracy-latency trade-off of our single DNN. We use 10% of the ILSVRC training set as our training set, first 50% of the validation set as our validation set to fine-tune ApproxNet, and the remaining 50% of the validation set as our test set. The choices made for training-validation-test in both the datasets follows common practice and there is no overlap between the three. **Throughout the evaluation, we use ImageNet VID dataset by default, unless we explicitly mention the use of the ImageNet IMG dataset.**

Metrics: We use latency and top-5 accuracy as the two metrics. The latency includes the overheads of the respective solutions, including the switching overhead, the execution time of FCE, RCE and scheduler.

3.5.3 Baselines

We start with the evaluation on the static models without the ability to adapt. This is because we want to reveal the relative accuracy-latency trade-offs in the traditional settings, compared to the single approximation branches in ApproxNet. The baselines for this static experiment include model variants, which are designed for different accuracy and latency goals, *i.e.* ResNet [53] MobileNets [74] and MSDNets [75] for which we use 5 execution branches in the single model that can provide different accuracy and latency goals. We use the ILSVRC IMG dataset to evaluate these static models, since this dataset is larger and with more classes.

We then proceed with the evaluation on the streaming videos, under varying resource contention. This brings two additional aspects for evaluation – (1) how the video frames are being processed in a timely and streaming manner as frames in this case cannot be batched like images, and (2) how the technique can meet the latency budget in the presence of resource contention from other application that can raise the processing latency. The baselines we use are: MCDNN [68] as a representative of the multi-model approach, and MSDNets, a representative of the single-model approach (with multiple execution branches). We also compare the switching overhead of our single-model design with the multi-capacity models in NestDNN [77]. Unfortunately, we were not able to use BranchyNet [76], because their DNN is not designed for large images in the ImageNet dataset. BranchyNet was evaluated on MNIST and CIFAR datasets in their paper and it does not provide any guidance on the parameter settings for training and makes it impossible to use on different datasets.

The details of each baseline are as follows,

ResNet: ResNet is the base DNN architecture of many state-of-the-art image and video object classification tasks, with superior accuracy to other architectures. While it was originally meant for server-class platforms, as resources on mobile devices increase, ResNet is also being used on such devices [96]–[98]. We use ResNet of 18 layers (ResNet-18) and of 34 layers (ResNet-34) as base models. We modify the last FC layer to classify into 30 labels in the VID dataset and fine-tune the whole model. ResNet-34 plays a role as the reference providing the upper bound of the target accuracy. ResNet architectures with more than 34 layers ([53] has considered up to 152 layers) become impractical as they are too slow to run on the resource-constrained mobile devices and their memory consumption is too large for the memory on the board.

MobileNets: This refers to 20 model variants (trained by the original authors) specifically designed for mobile devices ($\alpha = 1, 0.75, 0.5, 0.35, shape = 224, 192, 160, 128, 96$).

MSDNets: This refers to the 5 static execution branches to meet the different latency budgets in their anytime evaluation scenario. We have enhanced MSDNets with a scheduler to dynamically choose the static branches for dynamic runtime conditions. The former is compared with static models in the IMG dataset and the latter is compared with adaptive systems in the VID dataset. For the sake of simplicity, we reuse the same term (MSDNets) to refer to both.

NestDNN: This solution provides multi-capacity models with ResNet-34 architecture by varying the number of filters in each convolutional layer. We compose 9 descendant models, where (1) the seed model, or the smallest model, reduces the number of filters of all convolutional layers uniformly by 50%, (2) the largest descendant model is exactly ResNet of 34 layers, and (3) the other descendant models reduce the number of filters of all convolutional layers uniformly by a ratio equally distributed between 50% and 100%. We only compare the switching overhead of NestDNN inside its descendant models with ApproxNet because NestDNN is not open-sourced and the paper does not provide enough details about the training process or the architecture.

MCDNN: We change the base model in MCDNN from VGG to the more recent ResNet for a fairer comparison. This system chooses between MCDNN-18 and MCDNN-34 depending on the accuracy requirement. MCDNN-18 uses two models: a specialized ResNet-18 followed by the generic ResNet-18. The specialized ResNet-18 is the same as the ResNet-18 except the last layer, which is modified to classify the most frequent N classes only. This is MCDNN's key novelty that most inputs belong to the top N classes, which can be handled by a reduced-complexity DNN. If the top-1 prediction label of the specialized model in MCDNN is not among the top N frequent classes, then the generic model processes the input again and outputs its final predictions. Otherwise, MCDNN uses the top-5 prediction labels of the specialized model as its final predictions. We set $N = 20$ that covers 80% of training video frames in the VID dataset.

3.5.4 Typical Usage Scenarios

We use a few usage scenarios to compare the protocols, although ApproxNet can support finer-grained user requirements in latency or accuracy.

- **High accuracy, High latency (HH)** refers to the scenario where ApproxNet has less than 10% (relative) accuracy loss from ResNet-34, our most accurate single model baseline. Accordingly, the runtime latency is also high to achieve such accuracy.

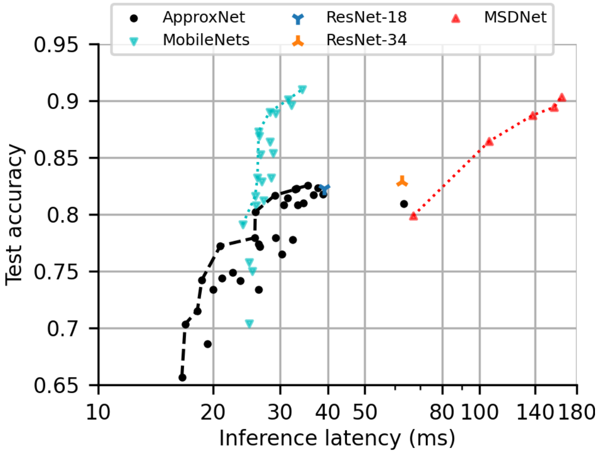


Figure 3.9. Pareto frontier for test accuracy and inference latency on the ImageNet IMG dataset for ApproxNet compared to ResNet and MobileNets, the latter being specialized for mobile devices.

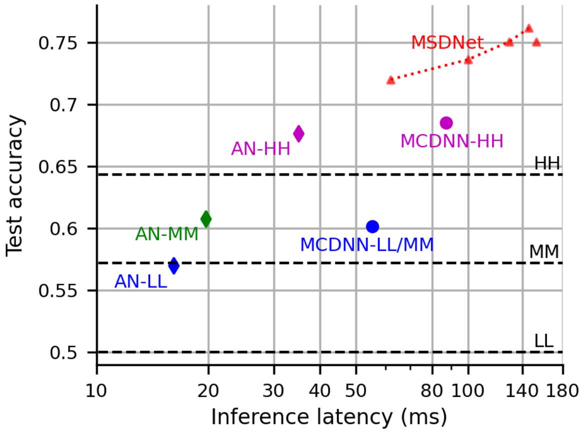


Figure 3.10. Comparison of system performance in typical usage scenarios. ApproxNet is able to meet the accuracy requirement for all three scenarios. User requirements are shown in dashed lines.

- **Medium accuracy, Medium latency (MM)** has an accuracy loss less than 20% from our base model ResNet-34.

- **Low accuracy, Low latency (LL)** can tolerate an accuracy loss of up to 30% with a speed up in its inferencing.

- **Real time (RT)** scenario, by default, means the processing pipeline should keep up with 30 fps speed, *i.e.* maximum 33.33 ms latency. This is selected if no requirement is specified.

3.5.5 Accuracy-latency Trade-off of the Static Models

We first evaluate ApproxNet on ILSVRC IMG dataset on the accuracy-latency trade-off of each AB in our single DNN as shown in Figure 3.9. Our AB with higher latency (satisfying 30 fps speed though higher) has close accuracy to ResNet-18 and ResNet-34 but much lower latency than ResNet-34. Meanwhile, our AB with reduced latency (25 ms to 30 ms) has close accuracy to MobileNets. And finally, our AB is superior to all baselines

Table 3.3. Averaged accuracy and latency performance of ABs on the Pareto frontier in ApproxNet and those of the baselines on validation set of the VID dataset. Note that the accuracy on the validation dataset can be higher due to its similarity with the training dataset. Thus the validation accuracy is only used to construct the look up table in ApproxNet and baselines and does not reflect the true performance.

(a) Averaged accuracy and latency performance in ApproxNet.

Usage Scenario (rate)	Shape	Layers	Latency	Accuracy
HH (32 fps)	128x128x3	24	31.42 ms	82.12%
	160x160x3	20	31.33 ms	80.81%
	128x128x3	20	27.95 ms	79.35%
	112x112x3	20	26.84 ms	78.28%
MM (56 fps)	128x128x3	12	17.97 ms	70.23%
	112x112x3	12	17.70 ms	68.53%
	96x96x3	12	16.78 ms	67.98%
LL (62 fps)	80x80x3	12	16.14 ms	66.39%

(b) Lookup table in MCDNN's scheduler.

Scenario (rate)	Shape	Layers	Latency	Accuracy
HH (11 fps)	224x224x3	34	88.11 ms	77.71%
MM/LL (17 fps)	224x224x3	18	57.83 ms	71.40%

(c) Lookup table in MSDNet's scheduler.

Scenario (rate)	Shape	Layers	Latency	Accuracy
HH (5.2 fps)	224x224x3	191	153 ms	96.79%
MM/LL (16 fps)	224x224x3	63	62 ms	95.98%

(d) Reference performance of single model variants or execution branches.

Model name (rate)	Shape	Layers	Latency	Accuracy
ResNet-34 (16 fps)	224x224x3	34	64.44 ms	85.86%
ResNet-18 (22 fps)	224x224x3	18	45.22 ms	84.59%
MSDNet-branch5 (5.2 fps)	224x224x3	191	153 ms	96.79%
MSDNet-branch4 (5.6 fps)	224x224x3	180	146 ms	96.55%
MSDNet-branch3 (7.8 fps)	224x224x3	154	129 ms	96.70%
MSDNet-branch2 (10 fps)	224x224x3	115	100 ms	96.89%
MSDNet-branch1 (16 fps)	224x224x3	63	62 ms	95.98%

in achieving extremely low latency (< 20 ms). However, the single execution branches in MSDNet are much slower than ApproxNet and other baselines. The latency ranges from 62 ms to 153 ms, which cannot meet the real-time processing requirement and will be even worse in face of the resource contention. MobileNets can keep up with the frame rate but it lacks the configurability in the latency dimensional that ApproxNet has. Although MobileNets does win on the IMG dataset at higher accuracy, it needs an ensemble of models (like MCDNN) when it comes to the video where content characteristics, user requirement, and runtime resource contention change.

3.5.6 Adaptability to Changing User Requirements

We then, from now on, switch to the ILSVRC VID dataset and show how ApproxNet can meet different user requirements for accuracy and latency. We list the averaged accuracy and latency of Pareto frontier branches in Table 3.3(a), which can serve as a lookup table in the simplest scenario, *i.e.* without considering frame complexity categories and resource contention. ApproxNet, provides content-aware approximation and thus keeps a lookup table for each frame complexity category, and to be responsive to resource contention, updates the latency in the lookup table based on observed contention.

We perform our evaluation on the entire test set, but without the baseline protocols incurring any switching penalty. Figure 3.10 compares the accuracy and latency performance between ApproxNet and baselines in three typical usage scenarios "HH", "MM", and "LL" (AN denotes ApproxNet). In this experiment, ApproxNet uses the content-aware lookup table for each frame complexity category and chooses the best AB at runtime to meet the user accuracy requirement. MCDNN and MSDNet use similar lookup tables (Table 3.3(b) and (c)) to select among model variants or execution branches to satisfy the user requirement. We can observe that "AN-HH" achieves the accuracy of 67.7% at a latency of 35.0 ms, compared to "MCDNN-HH" that has an accuracy of 68.5% at the latency of 87.4 ms. Thus, MCDNN-HH is 2.5X slower while achieving 1.1% accuracy gain over ApproxNet. On the other hand, MSDNet is more accurate and slower than all ApproxNet's branches. The lightest branch and heaviest branch achieve 4.3% and 7.3% higher accuracy respectively, and incur 1.8X and 4.4X higher latency respectively. In "LL" and "MM" usage scenarios, MCDNN-LL/MM is 2.8-3.3X slower than ApproxNet, while gaining in accuracy 3% or less. MSDNets, on the other hand, is running with much higher latency (62 ms to 146 ms) and higher accuracy (72.0% to 76.2%). Thus, compared to these baseline models, ApproxNet wins by providing lower latency, satisfying the real-time requirement, and flexibility in achieving various points in the (accuracy, latency) space.

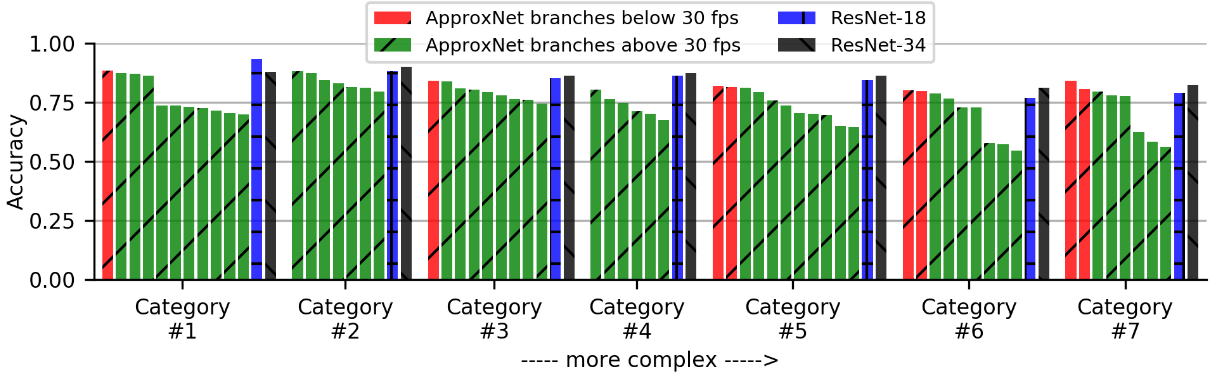


Figure 3.11. Content-specific accuracy of Pareto frontier branches. Branches that fulfill real-time processing (30 fps) requirement are labeled in green. Note that both ResNet-18 and ResNet-34 models, though with the higher accuracy, cannot meet the 30 fps latency requirement.

3.5.7 Adaptability to Changing Content Characteristics & User Requirements

We now show how ApproxNet can adapt to changing content characteristics and user requirements within the same video stream. The video stream, typically at 30 fps, may contain content of various complexities and this can change quickly and arbitrarily. Our study with the FCC on the VID dataset has shown that in 97.3% cases the frame complexity category of the video will change within every 100 frames. Thus, dynamically adjusting the AB with frame complexity category is beneficial to the end-to-end system. We see in Figure 3.11 that ApproxNet with various ABs can satisfy different (accuracy, latency) requirements for each frame complexity category. According to user's accuracy or latency requirement, ApproxNet's scheduler picks the appropriate AB. The majority of the branches satisfy the real-time processing requirement of 30 fps and can also support high accuracy quite close to the ResNet-34.

In Figure 3.12, we show how ApproxNet adapts for a particular representative video from the test dataset. Here, we assume the user requirement changes every 100 frames between "HH", "MM", and "LL". This is a synthetic setting to observe how models perform at the time of switching. We assume a uniformly distributed model selection among 20 model variants for MCDNN's scheduler (in [68], the MCDNN catalog uses 68 model variants) while the embedded device can only cache two models in the RAM (more detailed memory results

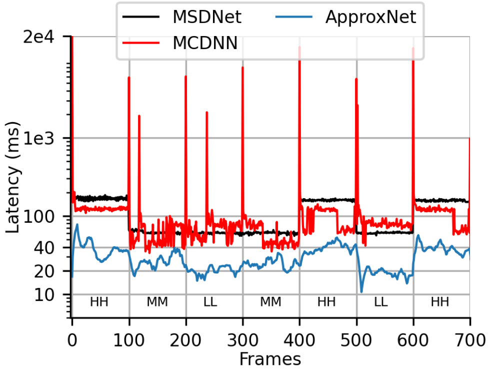
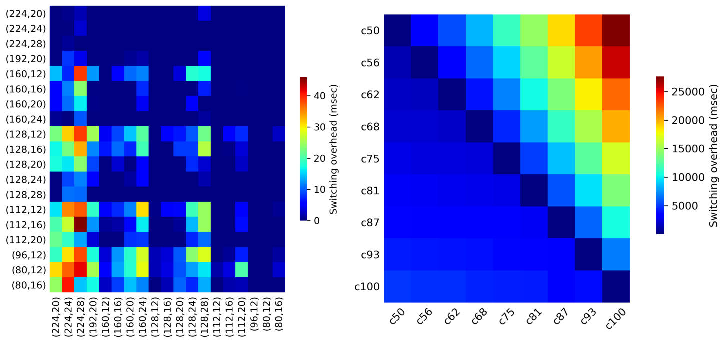


Figure 3.12. Latency performance comparison with changing user requirements throughout video stream.

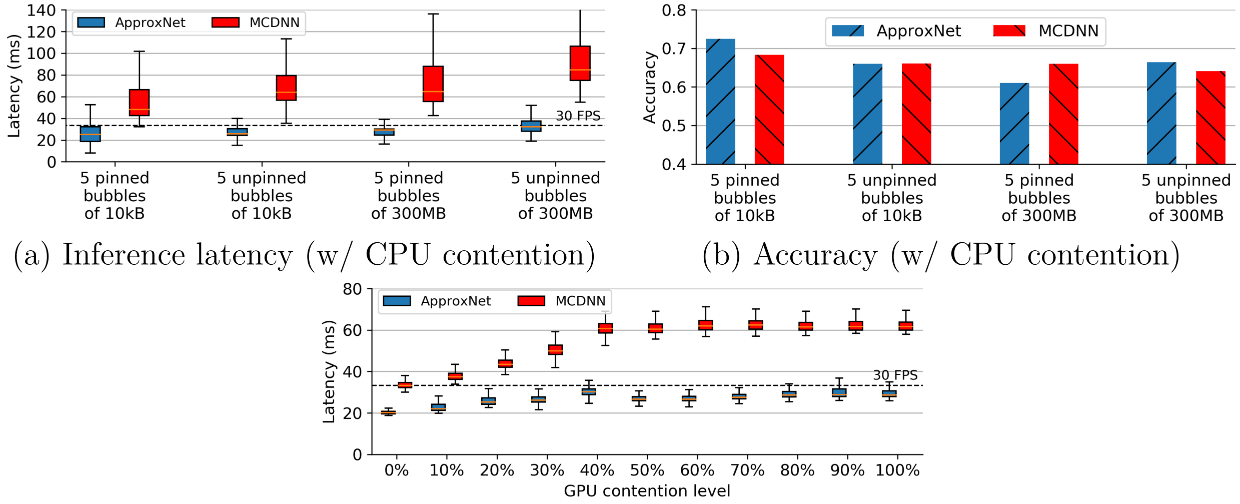


(a) ApproxNet (b) NestDNN

Figure 3.13. Transition latency overhead across (a) ABs in ApproxNet and (b) descendant models in NestDNN. "from" branch on Y-axis and "to" branch on X-axis. Inside brackets: (input shape, outport depth). Latency unit is millisecond.

in Section 3.5.9). In this case, MCDNN has a high probability to load a new model variant into RAM from Flash, whenever the user requirement changes. This results in a huge latency spike, typically from 5 to 20 seconds at each switch. It is notable that for some cases, there are also small spikes in MCDNN following the larger spikes because the generic model is invoked due to the specialized model's prediction of "infrequent" class. On the other hand, ApproxNet and MSDNets incur little overhead in switching between any two branches, because they are all available within the same single-model DNN. Similar to the results before, ApproxNet wins over MSDNets at lower latency to meet the real-time processing requirement even though its accuracy is slightly lower.

To see in further detail the behavior of ApproxNet, we profile the mean transition time of all Pareto frontier branches under no contention as shown in Figure 3.13 (a). Most of the transition overheads are extremely low, while only a few transitions are above 30 ms. Our optimization algorithm (Equations 3.3 and 3.4) filters out such expensive transitions if they happen too frequently. In Figure 3.13 (b), we further show the transition time between the descendant models in NestDNN, which uses a multi-capacity model with varying number of filters in the convolutional layers. The notation c50 stands for the descendant model with



(a) Inference latency (w/ CPU contention)

(b) Accuracy (w/ CPU contention)

(c) Inference latency (w/ GPU contention)

Figure 3.14. Comparison of ApproxNet vs MCDNN under resource contention. (a) and (b) inference latency and accuracy on the whole test dataset. (c) inference latency on a test video.

50% capacity of the largest variant, and so on. We can observe that the lowest switching cost of NestDNN is still more than an order of magnitude higher than the highest switching cost of ApproxNet. The highest switching cost of NestDNN compared to the highest of ApproxNet is more than three orders of magnitude—the highest cost is important when trying to guarantee latencies with the worst-case latency spikes. We can observe that the transition overhead can be up to 25 seconds from the smallest model to the largest model, and is generally proportional to the amount of data that is loaded into the memory. This is because NestDNN only keeps a single model, the one in use in memory and loads/unloads all others when switching. In summary, the benefit of ApproxNet comes from the fact that (1) it can accommodate multiple (accuracy, latency) points within one model through its two approximation knobs while MCDNN has to switch between model variants, and (2) switching between ABs does not load large amount of data and computational graphs.

3.5.8 Adaptability to Resource Contention

We evaluate in Figure 3.14, the ability of ApproxNet to adapt to resource contention on the device, both CPU and GPU contention. First, we evaluate this ability by running a *bubble application* [80], [82] on the CPU that creates stress of different magnitudes on the (shared) memory subsystem while the video analytics DNN is running on the GPU. We generate bubbles, of two different memory sizes 10 KB (low contention) and 300 MB (high contention). The bubbles can be "unpinned" meaning they can run on any of the cores or they can be "pinned" in which case they run on a total of 5 CPU cores leaving the 6th one for dedicated use by the video analytics application. The unpinned configuration causes higher contention. We introduce contention in phases—low pinned, low unpinned, high pinned, high unpinned.

As shown in Figure 3.14(a), MCDNN with its fastest model variant MCDNN-18, runs between 40ms and 100 ms depending on the contention level and has no adaptation. For ApproxNet, on the other hand, our mean latency under low contention (10 KB, pinned) is 25.66 ms, and it increases a little to 34.23 ms when the contention becomes high (300 MB, unpinned). We also show the accuracy comparison in Figure 3.14(b), where we are slightly better than MCDNN under low contention and high contention (2% to 4%) but slightly worse (within 4%) for intermediate contention (300 MB, pinned).

To further evaluate ApproxNet with regard to GPU contention, we run a synthetic matrix manipulation application concurrently with ApproxNet. The contention level is varied in a controlled manner through the synthetic application, from 0% to 100% in steps of 10%, where the control is the size of the matrix, and equivalently, the number of GPU threads dedicated to the synthetic application. The contention value is the GPU utilization when the synthetic application runs alone as measured through `tegrastats`. For baseline, we use the MCDNN-18 model again since among the MCDNN ensemble, it comes closest to the video frame rate (33.3 ms latency). As shown in Figure 3.14(c), without the ability to sense the GPU contention and react to it, the latency of MCDNN increases by 85.6% and is far beyond the real-time latency threshold. The latency of ApproxNet also increases with gradually increasing contention, 20.3 ms at no contention to 30.77 ms at 30% contention. However, when we further raise the contention level to 50% or above, ApproxNet's scheduler senses the contention and switches to a lighter-weight approximation branch such that the latency remains within 33.3 ms. The accuracy of MCDNN and ApproxNet were identical for this sample execution. Thus, this experiment bears out the claim that ApproxNet can respond

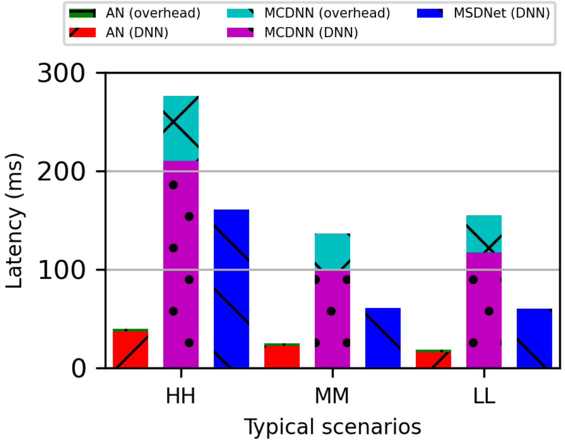


Figure 3.15. System overhead in ApproxNet and MCDNN.

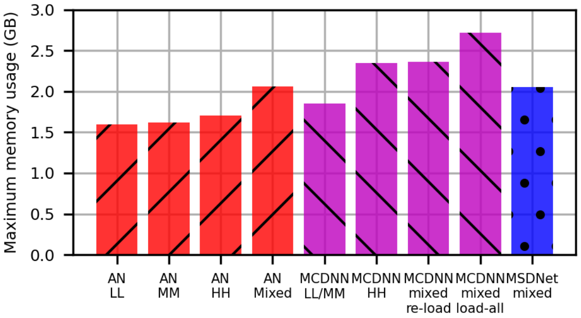


Figure 3.16. Memory consumption of solutions in different usage scenarios (unit of GB).

to contention gracefully by recreating the Pareto curve for the current contention level and picking the appropriate AB.

3.5.9 Solution Overheads

With the same experiment as in Section 3.5.7, we compare the overheads of ApproxNet, MCDNN, and MSDNets in Figure 3.15. For ApproxNet, we measure the overhead of all the steps outside of the core DNN, *i.e.* frame resizing, FCE, RCE, and scheduler. For MCDNN, the dominant overhead is the model switching and loading. The model switching overhead of MCDNN is measured at each switching point and averaged across all frames in each scenario. We see that ApproxNet, including overheads, is 7.0X to 8.2X faster than MCDNN and 2.4X to 4.1X faster than MSDNets. Further, we can observe that in “MM” and “LL” scenarios, ApproxNet’s averaged latency is less than 30 ms and thus ApproxNet can achieve real-time processing of 30 fps videos. As mentioned before, MCDNN may be forced to reload the appropriate models whenever the user requirement changes. So, in the best case for MCDNN the requirement never changes or it has all its models cached in RAM. ApproxNet is still 5.1X to 6.3X faster.

Figure 3.16 compares the peak memory consumption of ApproxNet and MCDNN in typical usage scenarios. ApproxNet-mixed, MCDNN-mixed, and MSDNet-mixed are the cases where the experiment cycles through the three usage scenarios. We test MCDNN-mixed with two model caching strategies: (1) the model variants are loaded from Flash when they get triggered (named "re-load"), simulating the minimum RAM usage (2) the model variants are all loaded into the RAM at the beginning (named "load-all"), assuming the RAM is large enough. We see that ApproxNet in going from "LL" to "HH" requirement consumes 1.6 GB to 1.7 GB memory and is lower than MCDNN (1.9 GB and 2.4 GB). MCDNN's cascade DNN design (specialized model followed by generic model) is the root cause that it consumes about 15% more memory than our model even though they only keep one model variant in the RAM and it consumes 32% more memory if it loads two. For the mixed scenario, we can set an upper bound on the ApproxNet memory consumption—it never exceeds 2.1 GB no matter how we switch among ABs at runtime, an important property for proving operational correctness in mobile or embedded environments. Further, ApproxNet, with tens of ABs available, offers more choices than MCDNN and MSDNet, and MCDNN cannot accommodate more than two models in the available RAM.

Storage is a lesser concern but it does affect the pushing out of updated models from the server to the mobile device, a common use case. ApproxNet's storage cost is only 88.8 MB while MCDNN with 2 models takes 260 MB and MSDNet with 5 execution branches takes 177 MB. A primary reason is the duplication in MCDNN of the specialized and the generic models which have identical architecture except for the last FC layer. Thus, ApproxNet is well suited to the mobile usage scenario due to its low RAM and storage usage.

3.5.10 Ablation Study with FCE

To study the necessity of FCE, we conduct an ablation study of ApproxNet with a content agnostic scheduler. Different from the content-aware scheduler, this scheduler picks the same AB for all video frames regardless of the content (the contention level is held unchanged). The test dataset obviously has variations in the content characteristics. With the same experiment as in Section 3.5.6, we compare the accuracy and latency performance be-

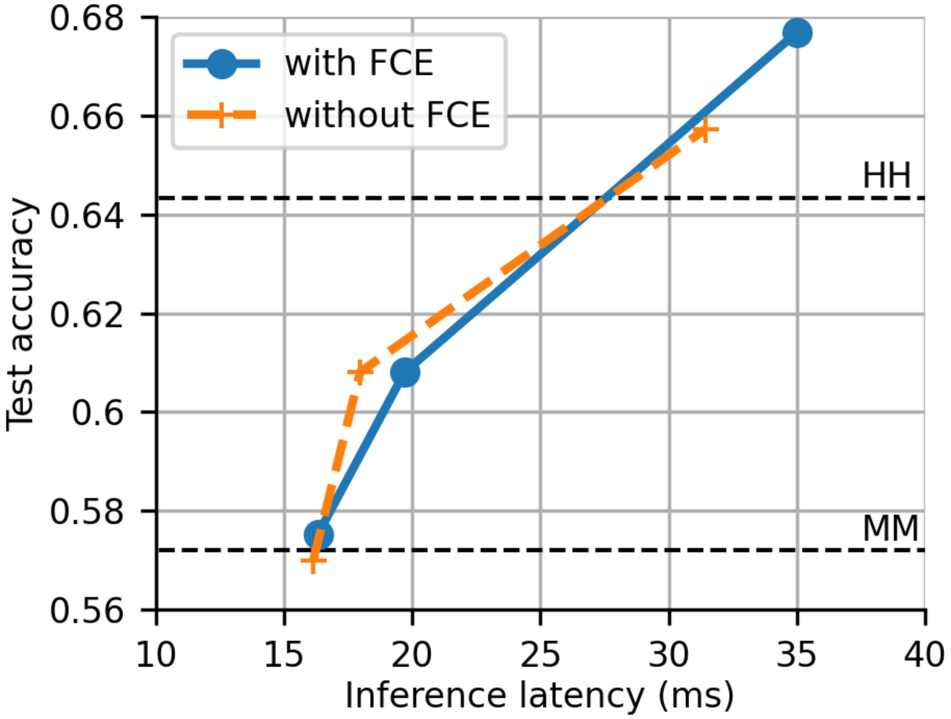
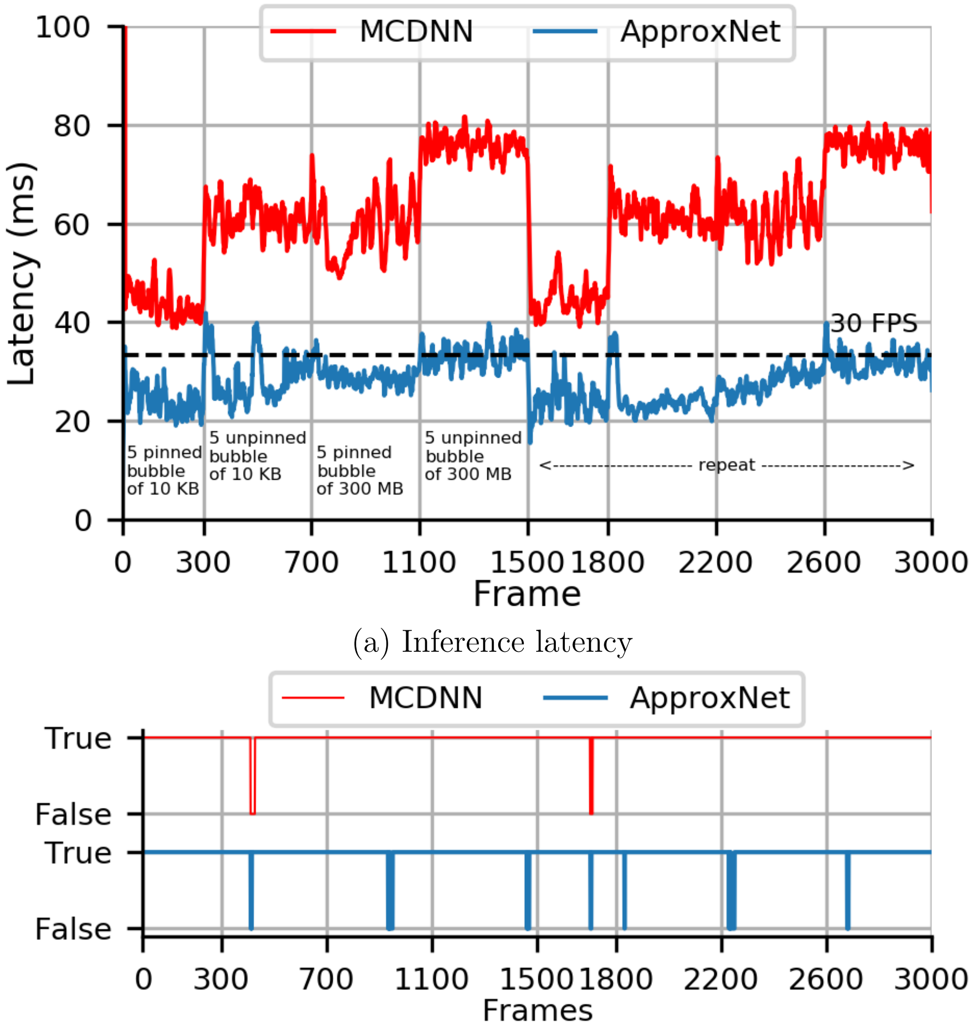


Figure 3.17. Comparison of system performance in typical usage scenarios between ApproxNet with FCE and ApproxNet without FCE.

tween ApproxNet with FCE and that without FCE in three typical usage scenarios “HH”, “MM”, and “LL”. Figure 3.17 shows that ApproxNet with FCE is able to improve the accuracy by 2.0% under a “HH” scenario with an additional 3.57 ms latency cost. ApproxNet with FCE under “MM” and “LL” scenarios is either slightly slower or more accurate than that without FCE. To summarize, ApproxNet’s FCE is more beneficial when the accuracy goal is higher, and however the latency budget is also higher.

3.5.11 Case Study with YouTube Video

As a case study, we evaluate ApproxNet on a randomly picked YouTube video [99], to see how it adapts to different resource contention scenarios at runtime (Figure 3.18). The video is a car racing match with changing scenes and objects, and thus, we want to evaluate the object classification performance. The interested reader may see a demo of ApproxNet and MCDNN on this and other videos at https://approxnet.github.io/. Similar to the control



(a) Inference latency

(b) Accuracy

Figure 3.18. Case study: performance comparison of ApproxNet vs MCDNN under resource contention for a Youtube video.

setup in Section 3.5.8, we test ApproxNet and MCDNN for four different contention levels. Each phase is 300-400 frames and the latency requirement is 33 ms to keep up with the 30 fps video. We see ApproxNet adapts to the resource contention well—it switches to a lightweight AB while still keeping high accuracy, comparable to MCDNN (seen on the demo site). Further, ApproxNet is always faster than MCDNN, while MCDNN, with a latency of 40-80 ms and even without switching overhead, has degraded performance under resource contention, and has to drop approximately every two out of three frames. As for the accuracy, there are only some occasional false classifications in ApproxNet (in total: 51 out of 3,000 frames, or 1.7%). MCDNN, in this case, has slightly better accuracy (24 false classifications in 3,000 frames, or 0.8%). We believe commonly used post-processing algorithms [94], [100] can easily remove these occasional classification errors and both approaches can achieve very low inaccuracy.

3.6 Discussion

Training the approximation-enabled DNN of ApproxNet may take longer than conventional DNNs, since at each iteration of training, different outports and input shapes try to minimize their own softmax loss, and thus, they may adjust internal weights of the DNN in conflicting ways. In our experiments with the VID dataset, we observe that our training time is around 3 days on our evaluation edge server, described in Section 3.5.1, compared to 1 day to train a baseline ResNet-34 model. However, training being an offline process, the time is of less concern. However, training can be sped up by using one of various actively researched techniques for optimizing training, such as [101].

Generalizing the approximation-enabled DNN to other architectures. The shape and depth knobs are general to all CNN-based architectures. Theoretically, we can attach an outport (composed of SPP to adjust the shape and fully-connected layer to generate classification) to any layers. The legitimate input shape can be tricky and dependent on the specific architecture. Considering the training cost and exploration space, we select 7 shapes in multiples of 16 and 6 outports, in mostly equally spaced positions of the layers. More input shapes and outports enable finer-grained accuracy-latency trade-offs and the granularity of such a trade-off space depends on the design goal.

3.7 Related Work

System-wise optimization: There have been many optimization attempts to improve the efficiency of video analytics or other ML pipelines by building low power hardware and software accelerators for DNNs [102]–[109] or improving application performance using database optimization, either on-premise [110] or on cloud-hosted instances [111]. These are orthogonal and ApproxNet can also benefit from these optimizations. VideoStorm [2], Chameleon [112], and Focus [113] exploited various configurations and DNN models to handle video analytics queries in a situation-tailored manner. ExCamera [1] and Sonic [114] enabled low-latency video processing on the cloud using serverless architecture (AWS Lambda [115]). Mainstream [69] proposed to share weights of DNNs across applications. These are all server-side solutions, requiring multiple models to be loaded simultaneously, which is challenging with resource constraints. NoScope [116] targeted to reduce the computation cost of video analytics queries on servers by leveraging a specialized model trained on a small subset of videos. Thus, its applicability is limited to a small subset of videos that the model has seen. Closing-the-loop [117] uses genetic algorithms to efficiently search for video editing parameters with lower computation cost. VideoChef [79] attempted to reduce the processing cost of video pipelines by dynamically changing approximation knobs of preprocessing filters in a content-aware manner. In contrast, ApproxNet, and concurrently developed ApproxDet [118] (for video-specific object detection), approximate in the core DNN, which have a significantly different and computationally heavier program structure than filters. Due to this larger overhead of approximation in the core DNN, ApproxNet's adaptive tuning is challenging. Thus, we plan on using either distributed learning [119] or a reinforcement learning-based scheduler for refining this adaptive feature [120].

DNN optimizations: Many solutions have been proposed to reduce computation cost of a DNN by controlling the precision of edge weights [121]–[124] and restructuring or compressing a DNN model [74], [125]–[130]. These are orthogonal to our work and ApproxNet's one-model approximation-enabled DNN can be further optimized by adopting such methods. There are several works that also present similar approximation knobs (input shape, outport depth). BranchyNet, CDL, and MSDNet [75], [76], [131] propose early-exit branches in DNNs. However, BranchyNet and CDL only validate on small datasets like MNIST [132] and CIFAR-10 [133] and have not shown practical techniques to selectively select these early-exit branches in an end-to-end pipeline. Such an adaptive system, in order to be useful (especially on embedded devices), needs to be responsive to resource contention, content characteristics, and users' requirement, *e.g.* end-to-end latency SLA. MSDNet targets a very simple image classification task without a strong use case and does not show a data-driven manner of using the early exits. It would have been helpful to demonstrate the system's end-to-end latency on either a server-class or embedded device. BlockDrop [134] trains a policy network to determine whether to skip the execution of several residual blocks at inference time. However, its speedup is marginal and it cannot be applied directly to mobile devices for real-time classification.

3.8 Conclusion

There is a push to support streaming video analytics close to the source of the video, such as, IoT devices, surveillance cameras, or AR/VR gadgets. However, state-of-the-art heavy DNNs cannot run on such resource-constrained devices. Further, the runtime conditions for the DNN's execution may change due to changes in the resource availability on the device, the content characteristics, or user's requirements. Although several works create lightweight DNNs for resource-constrained clients, none of these can adapt to changing runtime conditions. We introduced ApproxNet, a video analytics system for embedded or mobile clients. It enables novel dynamic approximation techniques to achieve desired inference latency and accuracy trade-off under changing runtime conditions. It achieves this by enabling two approximation knobs within a single DNN model, rather than creating and maintaining an ensemble of models. It then estimates the effect on latency and accuracy due to changing content characteristics and changing levels of contention. We show that ApproxNet can adapt seamlessly at runtime to such changes to provide low and stable latency for object classification on a video stream. We quantitatively compare its performance to ResNet, MCDNN, MobileNets, NestDNN, and MSDNet, five state-of-the-art object classification DNNs.

4. APPROXDET: CONTENT AND CONTENTION-AWARE APPROXIMATE OBJECT DETECTION FOR MOBILES

4.1 Introduction

Mobile devices with integrated cameras have seen tremendous success in various domains. Equipped with increasingly powerful System-on-Chips (SoCs), mobile augmented reality (AR) devices such as the Microsoft Hololens and Magic Leap One, along with top-of-the-line smartphones like iPhone 11 Pro and Samsung Galaxy S20, are opening up a plethora of new continuous mobile vision applications that were previously deemed impossible. These applications range from detection of objects around the environment for immersive experience in AR games such as Pokemon-Go [135], to recognition of road signs for providing directions in real-time [136], to identification of people for interactive photo editing [137], and to Manchester City's AR-driven stadium tour. A fundamental vision task that all of these applications must perform, is object detection on the live video stream that the camera is capturing. To maintain the immersive experience of the user (*e.g.* for AR games) or to give usable output on time (*e.g.* for road sign recognition), such tasks must be performed in near real-time with very low latency.

Computer vision and computer systems research working together has made significant progress in lightweight object detection applicable to mobile settings for *still images* in recent years [55], [98], [138], [139], thanks to development of efficient deep neural networks (DNNs) [74], [97], [121], [140]. However, directly applying image-based object detectors to video streams suffers, especially in mobile settings [141]. *First*, applying a detector on all video frames introduces excessive computational cost and would often violate the latency requirements of our target continuous vision applications. *Second*, image-based object detectors are not cognizant of the significant temporal continuity that exists in successive video frames (*e.g.* a static scene with a slowly moving object), unable to map this to the latency budget. To overcome these algorithmic challenges, the computer vision community has proposed some DNN models [94], [142], [143] for video object detection and tracking. Several lightweight DNN models [70], [144] that are suitable for mobile devices were developed.

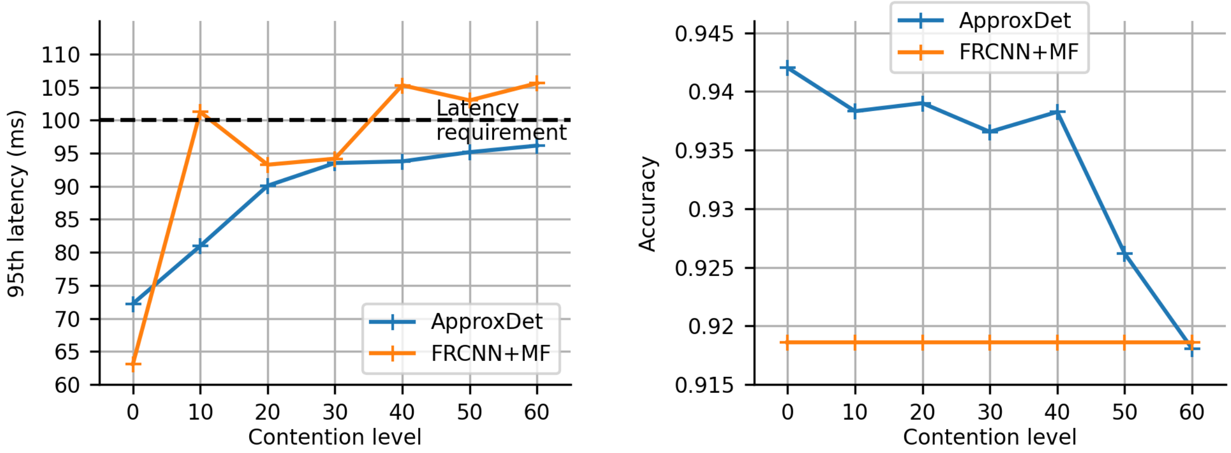


Figure 4.1. ApproxDet: The first system of mobile video object detection that takes both video content-awareness *and* resource contention-awareness within its ambit. Compared to a widely used object detector [57] optimized for a target latency requirement (orange curve), ApproxDet (blue curve) keeps its runtime latency (left) below the requirement and achieves better accuracy (right).

Despite these efforts, we argue that the system challenges of video object detection for continuous vision applications on resource-constrained devices remain largely unsolved [145]–[147]. A major shortcoming is that none of the existing approaches can adapt to runtime condition changes, such as *the content characteristics of the input videos*, and *the level of contention on the edge device*. Modern mobile devices[1] come with increasingly powerful SoCs having multiple heterogeneous processing units, and no longer process just a single application at a time. For example, both iOS and Android support multiple background tasks [148]–[150], such as an always-on personal assistant, e.g., Siri running a DNN for speech recognition (GPU contention), or a firewall constantly inspecting packets (memory-bandwidth contention). These tasks can run simultaneously with a continuous vision application that requires a video object detector, leading to unpredictable resource contention on mobile devices similar to a traditional server setting [80], [82], [151], [152]. Such concurrent appli-

[1] ↑We use "mobile devices" for the target platform, though without loss of generality, it applies to mobile *and* embedded platforms. The commonality is that both are using increasingly powerful SoCs, but are still resource constrained, relative to the servers where streaming video analytics are typically run. Further, both are used to run co-located applications (mobiles more so than embedded), which can interfere with the video analytics.

cations or background tasks can compete with object detection, drastically increasing the object detector's latency. Consider the example of a widely used DNN-based object detector: Faster R-CNN (FRCNN) [57], integrated with MedianFlow (MF) object tracking [153] and optimized for a latency requirement of 100 milliseconds (ms)[2]. The orange curve in Figure 4.1 shows the latency (left) and accuracy (right) of this [FRCNN + MF] processing an input video at different GPU contention levels on an embedded device (NVIDIA TX2). Without contention, the detector has a latency of $\approx$ 64 ms. However, as the GPU contention level increases, we observe a drastic increase in detection latency. While the accuracy remains the same, the latency of the detector fluctuates significantly and violates our 100 ms latency requirement. Different from server-class devices [82], our target mobile devices have limited ability to isolate co-located applications from interference, stemming from the paucity of VM-like isolation mechanisms.

To address this issue, we propose **ApproxDet**, a novel system that takes both video content-awareness *and* resource contention-awareness within its ambit. In contrast to the static FRCNN+MF baseline in Figure 4.1, ApproxDet manages to keep a latency below the requirement with increased level of contention while achieving a better accuracy. To this end, ApproxDet uses a *single* model with multiple approximation knobs that are dynamically tuned at runtime to stay on the Pareto optimal frontier (of the latency-accuracy curve in this case). This approach is in line with recent work on runtime-determined approximations in stateful distributed applications by us [110], [111], [154], [155] and others [156], [157]. We refer to the execution branch with a particular configuration of the approximation knob as an ***approximation branch (AB)***. This overall functionality is supported by *three core technical contributions* of this work. *First*, ApproxDet models the impacts of the contention level to the latency of the ABs. *Second*, our model combines an offline trained latency prediction model and an online contention sensor to precisely predict the latency of each AB in our system. Thus ApproxDet can adapt to resource contention at a given latency budget at runtime, an ability especially critical for the deployment on edge devices as their resources are limited and shared. *Third*, ApproxDet further considers how the video

[2] ↑We pick a combination of detector and tracker configurations that satisfies the latency requirement in 95% of the video frames from a validation dataset.

content influences both accuracy and latency. ApproxDet leverages video characteristics such as the object motion (fast vs. slow) and the sizes and the number of objects, to better predict the accuracy and latency of the ABs, and to select the best AB with reduced latency and increased accuracy.

Figure 4.2 presents an overview of ApproxDet. The object detection pipeline comprises of an object detector DNN based on our modified version of FRCNN [57], and a video object tracker that can be selected from among a set of choices. Both the detector and tracker are *jointly* approximated to achieve the required point in the accuracy-latency trade-off curve. Importantly, with the joint modeling of resource contention and content characteristics, ApproxDet can dynamically tune the approximation knobs, including the interval of performing object detection on video frames, the input shape of the frames and the number of proposals in the object detector DNN, the choice of object trackers, and the down-sampling ratio of the tracker. This means, in response to a resource contention from the GPU, ApproxDet can move to a more aggressive approximation setting of the detector DNN to bring down the latency since the detector DNN is more sensitive to the GPU contention. Moreover, in case of a content change in the video frame, *e.g.* a rapidly moving object, ApproxDet is able to switch over to a different AB where detector DNN is triggered more frequently to mitigate the tracker failure due to the fast-moving objects.

To our best knowledge, ApproxDet is the first system that accounts for *the joint adaptation to video content and resource contention* for mobile object detection. Distinct from prior work that optimizes multiple concurrent DNN applications [68], [69], [77], our system treats the contention as a black box. Our principle is that we neither know the context nor have control over the contention in real-world systems, as these video-analytic systems are typically user-space processes without any OS privilege. To estimate the contention, ApproxDet uses the current observed latency to map to the contention level, and adapts to use the AB that can satisfy the latency requirement from the user. Our evaluation bears out that this design choice is particularly effective for handling varied forms of contention under one simple algorithm. Table 4.1 further contrasts the features of ApproxDet with existing works.

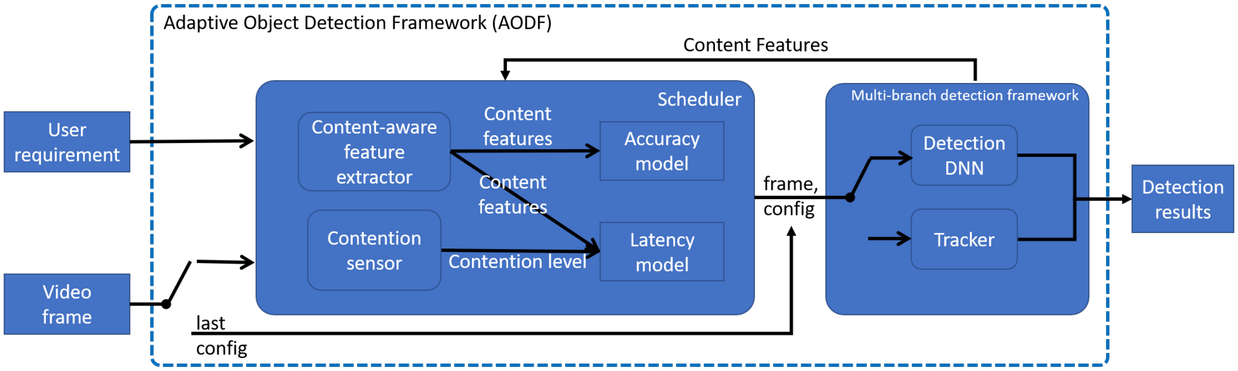


Figure 4.2. The workflow of the adaptive object detection framework used in our ApproxDet.

Table 4.1. A comparison of the key features of our ApproxDet solution to previous approaches. ApproxDet provides the most flexible framework for adaptive video object detection. “Single/multi model”: if a method uses shared execution branch for different control parameters. “Switching cost considered”: a technique takes into account switching cost while making its decision.

Solution	Single (S)/ Multi (M) Model	Tuning Knobs	Switching cost considered	Dynamic Scenario	Open Source	Mobile/ Server (M/S)	Video/ Image
ApproxDet	S	si, shape, nprop, tracker, and ds	✓	✓	✓	M	VID
Faster R-CNN	S	shape, nprop	✗	✗	✓	S	VID
YOLO, SSD	S	shape	✗	✗	✓	S	VID
AdaScale	S	scale	✗	◐	✓	S	VID
MCDNN [MobiSys16]	M	models	✗	✗	✗	M+S	IMG
NestDNN [MobiCom18]	S	# filters	◐	✗	✗	M	IMG
RANet [CVPR20]	M	resource budget	✗	✗	✓	S	IMG

✓Supported ◐Partially Supported ✗ Not Supported

To evaluate our model, we conduct extensive experiments on ImageNet Video Object Detection (VID) dataset [93] and compare our ApproxDet to a number of baselines, including AdaScale [158], Faster R-CNN [57], Faster R-CNN with tracking [153], and YOLOv3 [138]. Our results suggest that ApproxDet is able to adapt to a wide variety of contention and content characteristics and achieves 52% lower latency and 11.1% higher accuracy over the latest YOLOv3 optimized for efficiency and accuracy, and outshining all other baselines.

In summary, our work makes the following contributions:

1. We show that resource contention in mobile/embedded devices can significantly degrade the latency requirements of continuous vision applications.

2. We propose ApproxDet, an adaptive object detection framework that takes the run-time content characteristics *and* resource availability into consideration to dynamically optimize for the best accuracy-vs-latency tradeoff. This optimization is done using a single model with different approximation branches, rather than using an ensemble of models, leading to reduced switching overhead and the memory footprint.

3. ApproxDet makes use of video-specific features and does not simply consider a video as a set of discrete image frames. For example, it uses past video content characteristics (*e.g.* size of the objects) to guide its choice of the optimal approximation branch.

4.2 Background

This section introduces the background of ApproxDet, including object detection and tracking models used in ApproxDet and the context of approximate computing on edge devices.

4.2.1 Object Detection

Given an input image or video frame, an object detector aims at locating tight bounding boxes of object instances from target categories. In terms of network architecture, a CNN-based object detector can be divided into the backbone part that extracts image features, and the detection part that classifies object regions based on the extracted features. The detection part can be further divided into two-stage [57], [159] and single-stage detectors [55], [138], [160]. Two-stage detectors usually make use of Region Proposal Networks (RPN) for generating regions-of-interest (RoIs), which are further refined through the detection head and thus more accurate.

Our work builds on Faster-RCNN [57] — an accurate and flexible framework for object detection and a canonical example of a two-stage object detector. An input image or video frame is first resized to a specific *input shape* and fed into a DNN, where image features are extracted. Based on the features, a RPN identifies a *pre-defined number* of candidate object regions, known as region proposals. Image features are further aggregated within the proposed regions, followed by another DNN to classify the proposals into either background or one of target object categories and to refine the location of the proposals. Our key observation is that the input shape and the number of proposals have significant impact to the accuracy and latency. Therefore, we propose to expose *input shape* and *number of region proposals* as tuning knobs in ApproxDet.

Another line of research develops single-stage object detection [55], [138]. Without using region proposals, these models are optimized for efficiency and oftentimes less flexible. We consider one of the representative single-stage detectors as a baseline [138] in our experiments. YOLO simplifies object detection as a regression problem by directly predicting bounding boxes and class probabilities without the generation of region proposals.

4.2.2 Object Tracking

Object tracking seeks to locate moving objects over time within a video. We focused on motion-based visual tracking due to its simplicity and efficiency. A motion-based tracker assumes the initial position of each object is given in a starting frame, and makes use of local motion cues to predict the object's position in the next batch of frames. Our system considers a set of existing motion-based object trackers — MedianFlow [153], KCF [161], and CSRT [162]. The key difference lies in the extraction of motion cues, via *e.g.* optical flow or correlation filters, leading to varying accuracy and efficiency under different application scenarios. We thus propose to enable the adaptive *choice of the trackers* as one of our tuning knobs.

Another important factor of object tracking performance is the input resolution to a motion-based tracker. A downsampled version of the input image allows to better capture large motion and thus to track fast-moving objects, while a high-resolution input image facilitates the accurate tracking of objects that move slowly. Therefore, we further expose *the downsampling ratio of the input image* as another tuning knob for tracking.

4.2.3 Approximate Computing and Adaptation

Many computations are inherently approximate—they trade off quality of results for lower execution time or lower energy. Approximate computing has emerged as an area that exposes additional sources of approximation at the computer system level, including resource-constrained mobile and embedded platforms. One challenge in approximate computing is that the accuracy and performance of applying approximate techniques to a specific application and input sets are hard to predict and control [20], [40], [47]. This may lead to missed optimization opportunities, unacceptable quality outputs, and even incorrect executions. The two fundamental causes is that approximation techniques are not content-aware and contention-aware. Some recent work has started to address these issues. For example, Input Responsive Approximation (IRA) [26] and VideoChef [79] have brought in content-aware approximation for image processing and video processing pipelines respectively.

To the best of our knowledge, there is no solution that makes video object detection systems on mobile platforms adaptive to resource contention. Recently, Min *et al.* [163] assess the runtime quality of sensing models in the multi-mobile-device environment so that the best device is selected as a function of model accuracy. However, we have not seen similar work on the video object detection task. There are several works that provide tunable knobs to trade off accuracy-versus-latency, primarily in the image processing context [75], [77], [164], and some in video processing context [158]. It is conceivable that these knobs can be reconfigured to an optimal setting continuously as contention varies. However, there are key systems challenges that have to be solved before that end goal can be achieved. Such challenges include how to sense contention, how to change the knob in response to a specific level of contention, and how to optimize for the switching overhead from one approximation level to another.

4.3 Overview

Figure 4.2 presents the overall workflow of our system ApproxDet. Our system consists of two modules — a scheduler and a multi-branch object detection framework. The detection framework takes a video frame and a configuration as an input and produces the detection results while the scheduler decides which configuration the detection framework should use.

The detection framework includes two kernels: a detection kernel and a tracking kernel. This follows the common practice for video object detection that combines the heavy-weight detection and the light-weight tracker [70], [165]. At a high-level, the detection framework exposes five tuning knobs. With each tuning knob varying in a dynamic range, we construct a multi-dimensional configuration space and call the execution path of each configuration **an approximation branch (AB)**. The accuracy and the latency (execution time) are different for each AB and the values depend upon the video content characteristics (*e.g.* still versus fast-moving) and the compute resources available (*e.g.* lightly-loaded versus heavily-loaded mobile). To efficiently select an AB at runtime according to the given (and possibly changing) user requirement, the scheduler estimates the current latency and accuracy of each branch. The scheduler then selects the most accurate/fastest branch according to the specific user requirement.

We train an accuracy model and a latency model offline to support such estimation online. To better predict such online performance metric, we build two lightweight online modules – (1) a *content-aware feature extractor*, which extracts the height, width, tracks the object information of the last frame, and calculates the object movements of the past few frames, and (2) a *contention sensor*, which senses the current resource contention level. The scheduler is designed to run occasionally to re-calibrate the best approximation branch based on a learnable interval called "scheduler interval", which represents the number of frames that the configuration of the detection framework can be maintained.

4.4 Design Elements of ApproxDet

4.4.1 Multi-branch Object Detection Framework

To support the runtime adaptive object detection framework on videos, we first design a multi-branch object detection framework, with light switching overheads among branches for mapping to runtime changes. Different from object detection on still images, videos have temporal similarities and an object tracker is used to reduce the runtime cost with minor

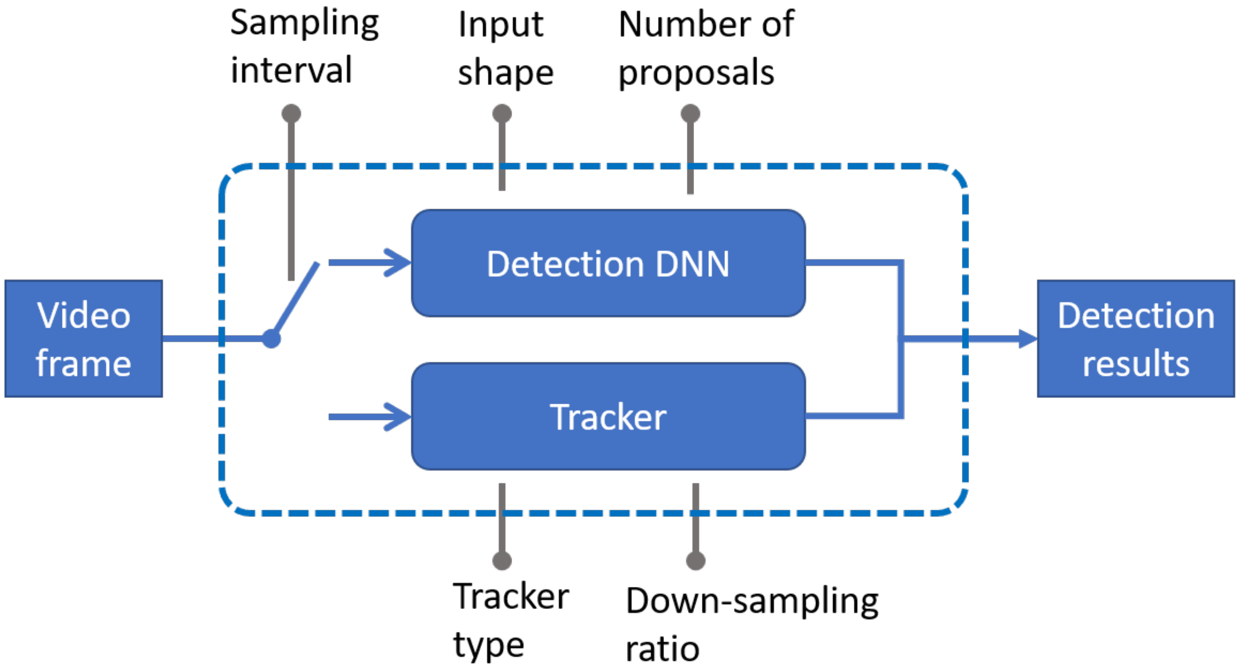


Figure 4.3. Our multi-branch object detection framework ApproxDet with the five runtime tuning knobs.

accuracy drop. In ApproxDet, the detection DNN produces initial bounding boxes for each object in the input image, while the tracker tracks objects between successive frames.

The overwhelming majority of work on lightweight object detection is for images, e.g., YOLOv3 [138] and SSD [55], thus being agnostic to video characteristics inherent to the temporal relation between image frames. This in turn influences ApproxDet's design decisions. For the detection DNN, we choose the popular Faster-RCNN with ResNet-50 as the backbone [57]. It shows state-of-the-art performance with medium speed when compared with other detection models. For the tracker part, we experiment with a set of 4 trackers — MedianFlow [153], KCF [161], CSRT [162], and Dense Optical Flow [166]. These trackers are open-sourced in OpenCV with reasonable performance. We then expose *five tuning knobs* for this object detection framework that our scheduler controls programmatically at runtime to achieve the right accuracy-latency tradeoff. We introduce them below and illustrate them in Figure 4.3.

- Sampling interval (si): For every si frame, we run the heavyweight object detection DNN on the first frame and light-weight object tracker on the rest of the frames.
- Input shape ($shape$): The resized shape of the video frame that is fed into the detection DNN.
- Number of proposals ($nprop$): The number of proposals generated from the Region Proposal Networks (RPN) in our detection DNN.
- Tracker type ($tracker$): Type of object tracker.
- Down-sampling ratio (ds): The downsampling ratio of the frame used by the object tracker.

Generally, we have empirically observed that smaller si, larger $shape$, more $nprop$, and smaller ds will raise the accuracy and vice-versa. We will discuss the specifics of the knobs in Section 4.5.1.

4.4.2 Content Feature Extraction

As multi-branch object detection framework is designed, an important prerequisite is to *precisely estimate the accuracy and computation time (latency) of each approximation branch*. To start with, the content feature has great impact on both the accuracy and latency of each AB based on the following two observations – (1) tracker latency is affected by the number and area of the objects because tracker algorithms take the bounding boxes of the detection frames as inputs and calculate features inside each box; (2) both detection and tracker accuracy are affected by the content in the video. For example, detection DNNs perform consistently poorly with small objects on MS COCO dataset [167], including Faster-RCNN [57], SSD [55], and YOLO [56]. Moreover, both detection DNN and tracker find it harder to deal with fast-moving objects. Some previous works [136] mention that movement between frames can be used as a feature to trigger the heavy detection process. This implies that for video object detection systems, we need to extract these content features to improve

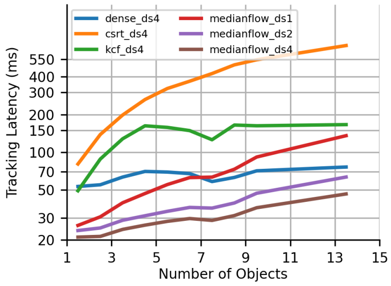


Figure 4.4. The latency curve of object trackers with different numbers of the objects on the validation dataset.

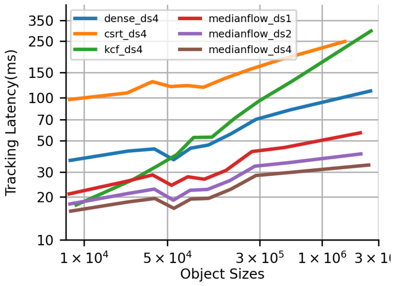


Figure 4.5. The latency curve of object trackers with different sizes of the objects on the validation dataset.

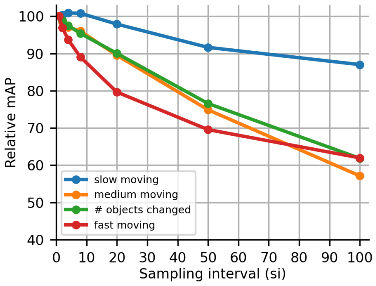


Figure 4.6. Detection plus MedianFlow tracker vis-à-vis detection-only branch on slow/medium/fast subsets.

the accuracy and latency of our models. In this work, we mainly consider two types of content features, described next.

Object Basic Features: We use *the number of objects* and *the summed area of the objects* as features for modeling the tracker latency. The intuition is that some light-weight trackers' latency increases proportionally with the number of objects and the area of the objects since each object is tracked independently, and the larger the area, the more tracking-related features need computation.

We empirically verify that the latency of the object trackers is affected by both the number and sizes of the objects, as shown in Figure 4.4 and 4.5. Specifically, we use 10% of the ImageNet video object detection (VID) training dataset to generate the latency data samples. The detailed data split can be found in Section 4.6.2.

Object Movement Features: We use *the recent movement of objects* as a feature for modeling the framework accuracy. More rigorously, we define the movement as the Euclidean distance of the objects' centers and we take the mean movement of all the objects in the recent frames. The intuition is that the faster the objects move in the video frame, the lower the accuracy, especially for the execution branches with higher sampling interval. We use the same data split as in Figure 4.4 and 4.5 to generate our accuracy data. We empirically show this in Figure 4.6, where we divide the validation dataset into three subsets — videos with slow, medium, and fast moving objects and show the accuracy reduction (compared to the detection-only branch) as we increase the sampling interval of the object detection kernel. For the detection kernel, we choose 100-proposal, 576-shape branch. The results show that the accuracy of high si branches ($si = 100$) does not drop significantly ($\approx 10\%$) on slow moving videos but reduces ($> 30\%$) on fast moving videos.

4.4.3 Latency Modeling

Latency prediction models aim to predict the frame-wise latency of each AB for future frames. Denote L_{fr} as the per-frame latency of our adaptive object detection framework. L_{fr} is a function of the DNN based detection latency L_{DNN} and the tracking latency $L_{tracker}$. If object detection DNN runs every si frames (sampling interval), the latency L_{fr} is given by

$$L_{fr} = \frac{L_{DNN}}{si} + L_{tracker} \tag{4.1}$$

We now describe the models of the detection latency L_{DNN} and the tracking latency $L_{tracker}$, respectively.

Latency Prediction for Object Detection DNN: The latency of the object detection DNN L_{DNN} is jointly determined by the two detector tuning knobs – the input image size $shape$ and the number of proposals $nprop$. Moreover, considering the input shape of frames may vary in different videos, we add the $height$ and $width$ of the input image as additional features. These features could be ignored if the video source is a video camera (which outputs fixed sized frames). Besides the input shape of video frames, system $contention$ (CPU/GPU usage and memory bandwidth, as detailed in Section 4.4.5) will also affect the DNN latency. Thus, the latency equation of the DNN is given by

$$L_{DNN} = f_{DNN}(nprop, shape, height, width, contention) \tag{4.2}$$

We fit a quadratic regression model for f_{DNN} to characterize the latency of the detection DNN. Once trained, the regression model is evaluated on a subset of the test set (sparsely sampled), where the mean squared error (MSE) between the prediction $\hat{L}_{DNN}$ and the ground-truth L_{DNN} latency are reported.

Latency Prediction for Object Trackers: As discussed in Section 4.4.2, the number of objects and average sizes of objects play a major role for the tracking latency. We further construct a model $f_{tracker}$ to characterize the latency of the object tracker under the system contention. Similar to the detection latency model, we also add the $height$ and $width$ of the input image as additional features. Thus, $f_{tracker}$ is given by:

$$L_{tracker} = f_{tracker}(height, width, n\_obj, avg\_size, contention) \tag{4.3}$$

We fit quadratic regression models to the ground-truth $L_{tracker}$. Moreover, since the model depends on n_obj and avg_size of the previous frame, we use the previous frame's n_obj and avg_size to train $L_{tracker}$. After the training process, we compute the predicted $\hat{L}_{tracker}$ and measure the MSE across a subset of the test set.

4.4.4 Accuracy Modeling

Accuracy prediction models aim to predict the expectation of the accuracy of each AB for near future frames. The accuracy of an object detector is usually defined by the metric mean average precision (mAP). However, we find that predicting the absolute mAPs given a test video is difficult. To address this issue, we propose to convert the absolute mAP metric into a relative percentage metric. More precisely, a base branch is identified in the detection framework using the detection-only branch ($si = 1$) with $nprop = 100$ and $shape = 576$. This base branch sets the performance upper-bound for all approximation branches (62.3% mAP on the validation set). The mAP of each AB is normalized to its percentage value by dividing its mAP by the base branch's mAP.

Different from the latency models, the factors on the accuracy are coupled all together (*i.e.* no distinction between detection DNN and tracking). Thus, we have a single unified model, given by:

$$A = f_A(si, shape, nprop, tracker, ds, movement) \tag{4.4}$$

Table 4.2. Applications running in the 3D contention space.

Real Apps	CPU	MB (MB/s)	GPU
Anomaly detection	99.80%	500	0%
Faster R-CNN	69.75%	1000	99%
YOLOv3	65.85%	800	98.50%

where *tracker* is the tracker type, *ds* is the downsampling ratio of the input to the tracker, and *movement* is the object movement features extracted from the video content. A decision tree model f_A was learned to predict the accuracy A, trained with the MSE loss across the whole training dataset.

4.4.5 Synthetic Contention Generator

Synthetic Contention Generator (CG) is designed as a stand-in for any resource contention on the device that may affect ApproxDet. A detection framework may suffer from unpredictable levels of resource contention when it is running on mobile platforms due to the instantiation of other co-located applications, for which we will not have information. We focus on three important types of resources on mobile platforms — CPU, memory bandwidth (MB), and GPU. We control CPU contention by the number of CPU cores our CG occupies. We control MB contention by the amount of memory-to-cache bandwidth that it consumes. The code is modified from the widely used STREAM benchmark [168], [169] that is meant to measure the MB capacity of a chip. For the GPU contention, we control the number of GPU cores that are utilized. The three-dimensional CG is orthogonal, which means we can tune each dimension without affecting the other dimensions. The CG is representative because we executed and mapped the contention caused by some widely-used applications in the 3D contention space (Table 4.2). The first one is an anomaly detection program that uses Robust Random Cut Forest (RRCF) [170] to detect anomalies from a local temperature and humidity sensor data. We also used our two object detection DNNs, namely Faster R-CNN and YOLOv3, for checking how much contention they can generate.

Table 4.3. Cost of profiling.

Task	Cost
Framework accuracy	2,414 hr · core (20% of the configurations)
Detection latency Tracker latency	7 hr · machine w/ 15 out 1 million sampling 1 hr · machine w/ 169 out 1 million sampling

4.4.6 Profiling Cost and Sub-sampling

The cost of collecting ground truth data with design features for performance prediction models is significant without proper sampling techniques. We measure our profiling cost for the accuracy, detection latency, and tracker latency models in Table 4.3.

To efficiently collect the profiling data, we use the master and worker model, where the master node manages a list of configurations of the detection framework and distributes the profiling work, while workers run the particular configuration to collect the training data for the modeling. As the feature space is huge, we sparsely sample the multi-dimensional space of ("number of proposals", "resized shape", "sampling interval", "tracker", "down-sampling ratio of the tracker'). We finally use 20% of the configurations to train our accuracy model.

Similar sub-subsampling techniques are used for the latency models as well, and we sample data points on videos of various height and width, various numbers of objects and object sizes, under discrete 3D contention levels. We finally use 15 out of a million feature points (defined in Section 4.5.2 and 4.5.1) to train our detection latency model and 169 out of a million feature points to train our tracker latency model.

4.4.7 Scheduler

The scheduler is the core component of ApproxDet that makes the decision at runtime on which AB should be used to run the inference on the input video frames. Formally, the scheduler maximizes the estimated detection accuracy of ApproxDet given a latency requirement L_{req}. This is done by identifying a feasible set of branches that satisfy the target latency requirements, and choosing the most accurate branch. In case of an empty feasible set, the fastest branch is returned. Thus, we formulate the optimal AB b_{opt} as follows,

$$b_{opt} = \begin{cases} \arg\max_{b \in \hat{B}}(A_b), & \text{if } \hat{B} \neq \emptyset, \\ \arg\min_{b \in \hat{\mathcal{B}}}(L_{est,b}) & \text{otherwise} \end{cases} \tag{4.5}$$

where $\hat{\mathcal{B}}$ is all ABs considered, $\hat{B}$ is the feasible set, *i.e.* $\hat{B} = \{b \in \hat{\mathcal{B}}\}$ iff $L_{est,b} < L_{req}$, A_b and $L_{est,b}$ are the estimated accuracy and latency of the AB respectively. The search space $\hat{\mathcal{B}}$, composed of five orthogonal knobs, has millions of states. To reduce the scheduler overhead, we use a sampling technique in Section 4.5.1 and design light-weight online feature extractors.

To further reduce the scheduler overhead and enhance our system robustness, we restrict the scheduler to make decision at least every sw frames. The motivation of introducing sw is to prevent the scheduler to make very frequent decisions. Specifically, we set $sw = max(8, s\text{i})$. The scheduler will thus make a decision at least every 8 frames. When the scheduler chooses a branch with a long si, it will make a following decision every si frames. In addition to the latency of the detection and tracking kernels, we add switching overhead L_{sw} and the scheduler overhead L_{sc} into the overall latency estimation of an AB b, *i.e.* $L_{est,b} = L_{b,fr} + (L_{sw} + L_{sc})/sw$. We also design the light-weight online feature extractors so that we can adapt seamlessly to the content and contention changes.

Online Content feature Extractor. The online content feature extractor maintains the content features of the video by extracting *height*, *width* from current frame, memorizing *n_ob*j, *avg_s*ize of last frame and *movement* from past frames. It is lightweight in terms of the compute load it puts on the target platform and this is desirable since we have to extract the features at runtime on the target board for feeding into our models.

Online Contention Sensor. The online contention sensor is designed to sense the contention level in the system so that we can refer to the modeling and make the right prediction on the latency of each AB. Although one can theoretically get the ground truth of the resource contention by probing the system and directly measuring CPU, memory bandwidth and GPU usage by other processes, it is not practical. As a normal application in the user space, it is difficult for ApproxDet to collect the exact resource information from other processes. The hardware is also lacking sufficient support for such fine-grained measurement on mobile or embedded devices [171]. In contrast, the offline latency log under various contention levels and the online latency log of the current branch in the past few runs are a natural observation of the contention level. Thus, we proposed the log-based contention sensor.

The log-based contention sensor tries to find a contention level where the offline latency log matches the averaged online latency most closely. We use the nearest-neighbor principle to search for such contention levels in our pre-defined orthogonal 3D contention space. As multiple contention levels may cause the same impact on the latency of a given AB, we call it a cluster of contention levels and we pick one level out of it as the representative. In comparison to some previous work in the systems community [77], our contention sensor is lightweight, efficient, and does not require additional privileges at system level, making it a more practical offering in real-world systems.

4.5 Implementation

We implement ApproxDet in Python 3 and C with tensorflow-gpu, CUDA, and cuDNN libraries. For detection kernel, we choose Faster R-CNN due to its high accuracy and moderate computational burden. For tracking kernel, we implement four variants and introduce the details in Section 4.5.1.

4.5.1 Configuration of the Tuning Knobs

Our five tuning knobs include the sampling interval (si), the input image size ($shape$) to the detection DNN, the number of proposals ($nprop$) in the detection DNN, the type of object tracker ($tracker$) and the downsampling ratio of the input to the tracker (ds). We now describe the implementation details of these knobs, including their data types and value ranges.

Sampling Interval (si). si defines the interval of running the object detector. The object tracker runs on the following $si - 1$ frames. For example, our system runs object detection on every frame when $si = 1$. To reduce the search space of si, we constrain si in a preset set—$\{1, 2, 4, 8, 20, 50, 100\}$. These pre-defined si are chosen empirically to cover common video object detection scenarios. With the max value of $si = 100$, the detector runs at a large interval of 3-4 seconds and the tracker runs in-between.

Input Video Frame Shape to Detector ($shape$). $shape$ defines the shortest side of the input video frame to the object detector. The value of $shape$ must be a multiple of 16 to make the precise alignment of the image pixels and the feature map [57]. We set the $shape$ range from 224 to 576, since smaller shape than 224 significantly reduces the accuracy and larger shape than 576 will result in heavy computational burden and does not improve the accuracy based on results on the validation set.

Number of Proposals ($nprop$). $nprop$ controls the number of candidate regions considered for classification in the object detector. We limit the value of $nprop$ (integer) between 1 and 100. With $nprop = 1$, only the top ranked proposal from RPN is used for detection. Increasing $nprop$ will boost the detector's performance yet with increased computational cost and runtime.

Type of Trackers ($tracker$). $tracker$ defines which tracker to use from MedianFlow [153], KCF [161], CSRT [162], and dense optical flow trackers [166]. These trackers are selected based on their efficiency and accuracy. Different trackers have varying performance under different scenarios. For example, CSRT tracker is most accurate among these trackers, but is also most time consuming. MedianFlow tracker is fast and accurate when a object move slowly in the video, yet have poor performance for a fast moving object. We use the implementation from OpenCV for all trackers.

Downsampling Ratio for the Tracker (ds). ds controls the input image size to the tracker. The value of ds is limited to 1, 2, and 4, *i.e.* no downsampling, dowsampling by a factor of 2 and 4, respectively. A larger ds reduces the computational cost, and favors the tracking of fast moving objects. A smaller ds increase the latency, yet provide more accurate tracking of slowly moving objects.

4.5.2 3D Contention Generator (CG)

Our 3D CG is lightweight in code size. It is configured to generate contention in CPU, memory bandwidth (MB), and GPU (as introduced in Section 4.4.5). For MB CG, we modify STREAM by having the code write continuously to a 152 MB memory space and controlling the interval of array elements to operate on the array data so as to control the MB occupied. We add a feedback loop to dynamically adjust the number of elements in the array to be skipped for the write operation, thus maintaining the MB contention at the experimentally given level. To increase the maximum contention that can be generated by the MB CG, we spread out the MB CG among the CPU cores on which contention is to be generated and in aggregate can generate an intense bandwidth contention of up to 18 GB/s when 6 CPU can be used. The maximum input for the MB CG depends on both the CPU and MB part, the more CPU we can occupy, the higher maximum MB contention we can achieve. For the GPU CG, we fixed the working frequency of TX2 board at 1300 MHz. Then our GPU CG performs add operation on a certain size of arrays by using a CUDA program. By changing the size of the arrays and the input to the CUDA kernel functions, we control the number of GPU cores that are kept busy. We generate contention from 1% to 99%, as measured by `tegrastats`. For the experiments, we use 11 discrete levels 1%, 99%, and 9 levels in increments of 10% starting at 10%. If there is no MB or GPU contention specified and we need CPU contention on a certain number of CPU cores, we use a MB CG with minimum input 1MB/s to serve as the CPU CG and pin it to the experimentally given number of cores. For the experiments, we use 6 discrete levels from 100% to 600% in increments of 100%.

4.5.3 Training of Latency and Accuracy Models

The latency and accuracy model in ApproxDet is trained using a subset of the validation set of ImageNet VID. Specifically, we sparsely sample the 5-D feature space $(si, shape, nprop, tracker, ds)$ and run ApproxDet using the sampled configuration on a subset of videos randomly sampled from the validation set. Standard gradient descent is then used for fitting the regression. And CART is used for the decision tree. To train content-aware accuracy model, during the training phase, the *movement* feature as the average motion across all objects and all frames is required in each video snippet. During the test phase, since movement of the objects are not available, we use average across all past frames as a substitute.

4.6 Evaluation

4.6.1 Evaluation Platform

We evaluate ApproxDet on an NVIDIA Jetson TX2 board [91], which includes 256 NVIDIA Pascal CUDA cores, a dual-core Denver CPU, a quad-core ARM CPU on a 8GB unified memory between CPU and GPU. The specification of this board is comparable to what is available in today's high-end smartphones such as Samsung Galaxy S20 and Apple iPhone 11 Pro. We train our neural network models on a server with NVIDIA Tesla K40c GPU with 12GB dedicated memory and an octa-core Intel i7-2600 CPU with 24GB RAM. For both the TX2 and the edge server, we install Ubuntu OS and Tensorflow v1.14, Pytorch v1.1, and MXNet v1.4.1.

4.6.2 Datasets, Task, and Metrics

We evaluate ApproxDet on the object detection task using ILSVRC 2015 VID dataset [93]. For training, due to the redundant video dataset and limited resources, we follow the practice in [94] such that the VID training dataset is sub-sampled every 100 frames. We use 90% of this video dataset as training set to train ApproxDet's DNN model and keep aside another 10% as validation set to fine-tune ApproxDet (modeling). To evaluate ApproxDet's system performance, we use ILSVRC 2015 VID validation set – we refer to this as the "test set" throughout the chapter. Due to the limit of time and computational resources, we use 10% of test set. For all our baselines, we follow the data split of ApproxDet's DNN model to train and test.

We use latency and mean average precision (mAP) as the two metrics. We define the latency as the short-window averaged latency among one detection frame and its following tracking frames. The definition applies to ApproxDet's baselines using different trackers.

The latency also includes the overheads of the respective solutions, e.g., the switching overhead, the execution time of the online feature extractor, the online contention sensor, and the scheduler. Detection DNNs usually use a low confidence threshold, e.g., 0.001 [55], [57] to achieve higher mAP. However, low confidence threshold will lead to many false positive outputs, which is not practical in real-world systems. To simulate the environment of real-world systems, we use a high confidence threshold (0.3) to reduce the number of output objects from detection DNNs, and make it the same for all baseline detection DNNs. Note that we always *use the ground truth annotations* available in the datatset to examine the *true* accuracy and never use other detection results as pseudo ground truth (as in [113]).

4.6.3 Baselines

In this section, we will introduce baseline models used in this work.

Faster R-CNN (FRCNN) [57] is a popular two-stage detector in the computer vision community. We vary the *nprop* and *shap*e of ApproxDet's detection DNN to create different FRCNN baseline detectors. From this, we get a total of 28 (4*shap*e × 7*nprop*) FRCNN baseline models. In these baseline models, we follow the multi-model design on the detection model, where model variants in different sizes (number of proposals and resized shape) achieve different latency and accuracy specifications. We also profile the latency of these model variants and evaluate FRCNN with our scheduler defined in Section 4.4.7.

FRCNN+MedianFlow (FRCNN+MF): Since FRCNN only uses detection, we want to enhance this baseline for a fair comparison. For the enhanced baseline, we follow the mainstream "detection plus tracking" design in lightweight object detection tasks. We pick an unmodified detection variant with the highest accuracy ($nprop = 100$, *shap*e $= 576$) and a fast object tracker—MedianFlow (without downsampling technique). Thus, the enhanced baseline models include FRCNN plus MedianFlow tracker with varying *s*i. We also profile the latency of these model variants (each *s*i) and evaluate FRCNN+MF with our scheduler defined in Section 4.4.7. To reduce the scheduler cost, we perform the same sampling strategy in Section 4.5.1. Though we can pick different models based on the latency budget, these models are static and cannot respond to contention and context changes.

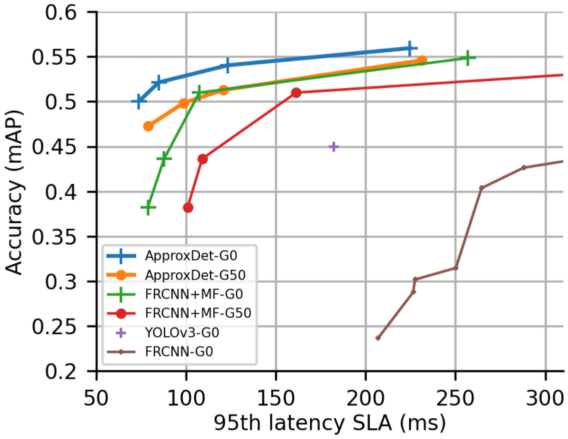


Figure 4.7. Accuracy of the models vs 95-th latency SLA. G50 represents the 50% GPU contention and G0 denotes no contention. (zoomed in)

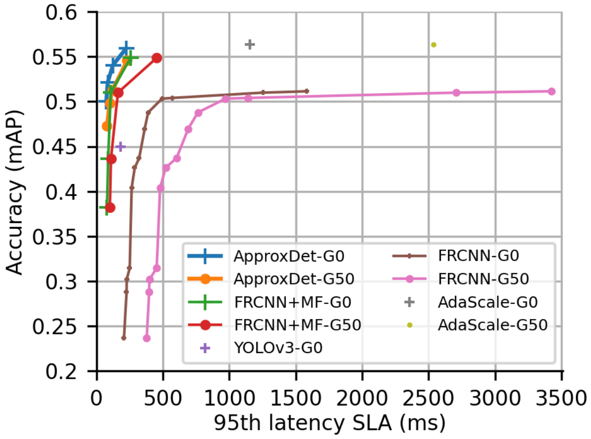


Figure 4.8. Accuracy of the models vs 95-th latency SLA. G50 represents the 50% GPU contention and G0 denotes no contention. (zoomed out)

AdaScale: Among latest methods, we choose AdaScale [158] as it dynamically adjusts the input scale to improve accuracy *and* running speed simultaneously. AdaScale is thus most relevant to our work, with similar tuning knobs for resource-constrained devices. For conducting a fair comparison between AdaScale and our proposed ApproxDet, we both use the pretrained models on ImageNet and train on the same ImageNet VID dataset.

YOLOv3: YOLOv3 is a modern one-stage detector with a fast running speed. It is widely used in many mobile applications. We use the same training/testing set and scheduler as ApproxDet to train and evaluate YOLOv3.

4.6.4 End-to-End Evaluation on Budgeted Latency

We first examine the end-to-end performance of ApproxDet vs. various baselines. Figure 4.7 shows a micro-view of the accuracy and 95-th percentile latency [3] under no contention (G0) and injected 50% GPU contention (G50) from the CG. The reason why our evaluation injects GPU contention only, is that the inference time of the DNN has the most

[3] ↑We choose the 95-th percentile latency service level agreement (SLA) because it is a promise to the users that in most cases the latency is below the number and shows much stronger latency guarantee than either mean or median latency.

impact on the detection latency and such inferences mostly run on the device's GPU (when it is available). Hence, the impact of GPU contention would focus on the most interesting scenarios showing how resilient ApproxDet is with respect to changes in the dynamics within the mobile device.

The results show the superior accuracy-vs-latency trade-off (blue vs. green vs. purple) over a FRCNN+MF and YOLOv3 under no contention where particularly, ApproxDet's accuracy is 11.8%, 9.5% higher than FRCNN+MF given 80ms and 90ms latency SLA and also 9.1% higher than YOLOv3 given 170ms latency SLA (exactly following YOLOv3's posterior 95-th latency). The accuracy gain over FRCNN+MF reduces as latency SLA grows larger since both ApproxDet and FRCNN+MF will merge into detection-only (or detection in every alternate frame) branch.

Furthermore, when 50% GPU contention is injected, ApproxDet with strong adaptability on sensing and reacting to the contention, the accuracy drops by around 2% and preserves the 95th latency SLA with minor changes (max 16%). However, the best baseline FRCNN+MF cannot adapt to the contention and the latency increases significantly (25% - 50%). Figure 4.8 shows the performance of FRCNN and AdaScale. Both of their latency increase by 75% to 120% with increased contention levels. This is significantly worse than ApproxDet. Further, FRCNN is inferior to FRCNN+MF in terms of accuracy-latency tradeoff. This is because adding the light-weight MedianFlow tracker largely reduces the latency while preserving most of the accuracy.

Although accuracy vs latency SLA shows the posterior performance metric of each model, we also want to examine under a certain latency requirement, how each solution chooses a model variant to execute and what the accuracy (Figure 4.9) and the latency violation rates (Figure 4.10) are. We evaluate at 80ms, 100ms, 150ms, and 300ms. Latency requirement that is smaller than 80ms is not chosen because no branch will be returned from both baselines and larger latency requirement is not meaningful as discussed before. Figure 4.9 and 4.10 show that under no contention scenario (G0), ApproxDet and FRCNN+MF are equally good at controlling low latency violations (ApproxDet is slightly smaller), while the accuracy of ApproxDet is 11.8%, 8.5%, 3.0%, and 1.1% higher than FRCNN+MF. Even

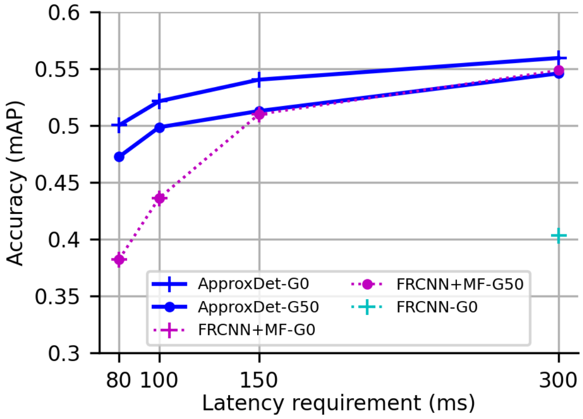


Figure 4.9. Accuracy (mAP) vs. latency requirement. FRCNN can not fulfill latency SLA < 300 ms. The accuracy of the baseline (FRCNN+MF) remains the same for G0 and G50, as it cannot adapt to contention and the execution is the same.

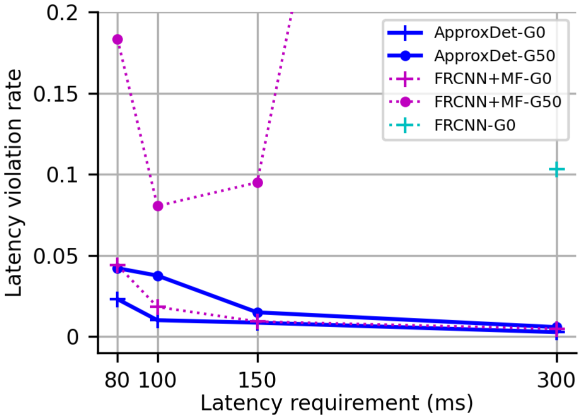


Figure 4.10. Latency violation rate vs. latency requirement. The baseline FRCNN+MF has significantly higher violation rates with increased contention levels, even when using a very conservative P95 scheduler.

under great reduction to 80ms latency requirement, ApproxDet is still able to maintain the accuracy above 50%.

Then, as 50% GPU contention is injected, the accuracy of ApproxDet is slightly reduced (by 2%) and is as good as FRCNN+MF under 150ms and 300ms latency requirements. However, ApproxDet is still able to control within the 5% violation rate while FRCNN+MF has a violation rate of 18.3%, 8.0%, 9.5%, and 100%. To summarize, ApproxDet is best at reducing the latency requirement to as low as 80ms with slightly reduced accuracy and is able to adapt well to the contention without violating the latency requirements.

4.6.5 Case Studies on Changing Conditions

Although the macro benchmark shows the overall performance of ApproxDet on a whole dataset, it is mostly the static behavior of the models since the latency requirement and resource contention levels do not change. To examine how ApproxDet adapts to the

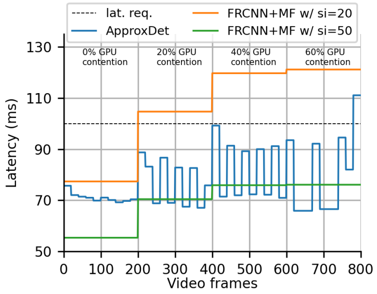


Figure 4.11. Comparison of models' latency under changing contention.

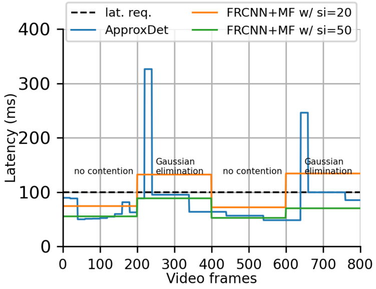


Figure 4.12. Comparison of models' latency without and with real app.

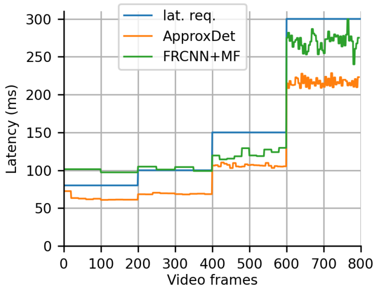


Figure 4.13. Comparison of models' latency under changing latency budget.

runtime changing conditions, we set up these case studies by injecting changing contention levels and latency requirements.

Changing Contention Levels. We first examine the performance of ApproxDet given a fixed latency requirement of 100 milliseconds on one test video. We then manually inject the GPU contention through CG and gradually tune up the contention level by 20% for every 200 frames. Figure 4.11 shows that ApproxDet with quick sensing and adaptation, is always able to meet the latency requirements while a FRCNN+MF baseline will either exceed the latency requirement by 20% ($si = 20$) or stay too conservative ($si = 50$) and suffers from low accuracy (there is a 7% mAP difference between the two branches on the test dataset).

In addition, we also pick a real application—Gaussian Elimination from the Rodinia Benchmark Suite [172], a GPU-intensive linear algebra routine widely used by many applications. We examine the performance of ApproxDet and baselines without and with the background app. Figure 4.12 shows that ApproxDet can adapt to the resource contention produced by the background app. The latency goes up dramatically when the app starts running and quickly drops as ApproxDet senses such contention and schedule a more efficient AB. A repeated experiment during the 600—800 frames has further confirmed our adaptability. In contrast, the baseline FRCNN+MF with different si is either too conservative or too aggressive under varying contention levels in terms of latency. When the contention app starts running, there is a larger latency spike of ApproxDet than FRCNN+MF. This is because our system schedules the AB with smaller si as long as the latency of such AB is below the latency requirement. However, since the object detector takes a larger portion in latency and is more sensitive to GPU contention, the latency increase is much higher when the system has not responded yet.

Changing Latency Requirements. We then examine the performance of ApproxDet given more relaxed latency requirements of 80ms, 100ms, 150ms, and 300ms per frame in four equally chunked phases of 200 frames. Figure 4.13 shows that we can always keep up with the latency requirement and always run below the latency requirement while FRCNN+MF, although is choosing a branch that satisfy 95% of the video frames in the validation dataset, still violates the latency requirement by 20% under 80ms requirement and is slightly above the threshold given 100ms latency requirement.

4.6.6 Performance Prediction Models

Accuracy Prediction Model. We set up the **oracle** method as using the ground truth validation data to predict on the test dataset. According to our belief that the accuracy reduction should be same in validation dataset and test dataset, the oracle approach should achieve zero MSE and if it is not zero, it represents the gap of accuracy reduction between the two datasets.

Figure 4.14 shows the training curve of the accuracy prediction model. We use different amounts of data to train the model and examine the mean squared error (MSE) on the rest of validation dataset for cross validation and the whole test dataset to report final performance. We can see that with only 20% of the training data, we are able to predict on the test dataset with 74.58 MSE. We can further reduce the MSE to 71.67 if 95% of the training data is available, while the oracle predicts with a comparable 71.65 MSE.

Tracker Latency Prediction Model. We set up the baseline approach, which always predicts a constant latency for each tracker using the averaged latency across all frames under specific contention levels. As seen from Table 4.4, we can largely reduce the prediction root

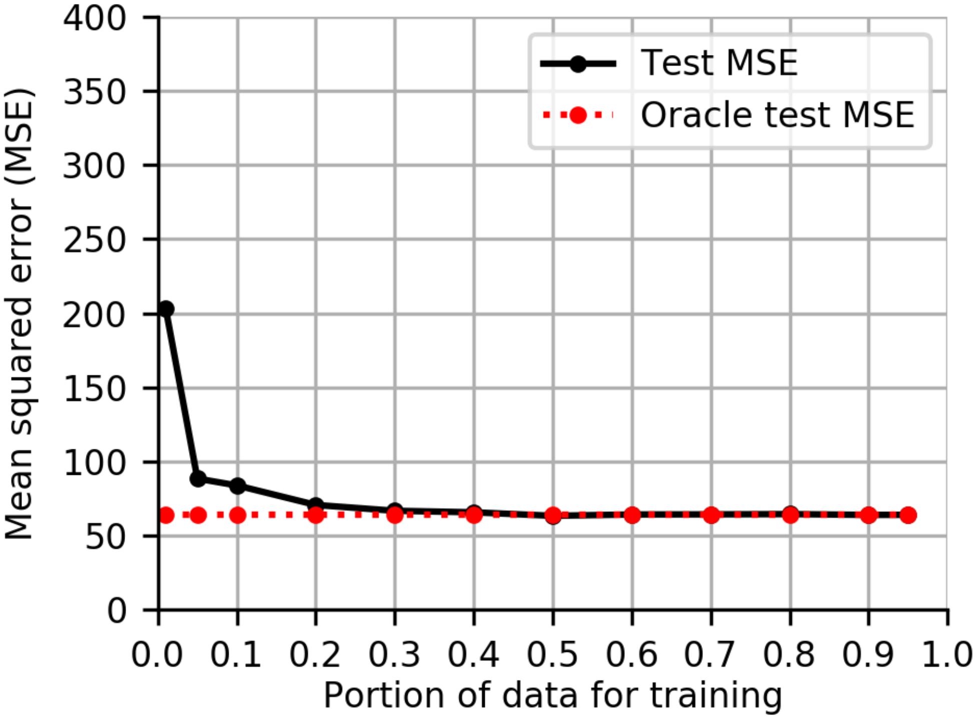


Figure 4.14. MSE of the accuracy prediction model on the validation set, with varying amount of training data.

Table 4.4. Precision of the tracker latency prediction models on the validation dataset. Please note that we only show the results of tracking frames on test datasets. In this table, "no" means no contention, "g50" means 50% GPU contention, and "c6m3600" means contention with 6 CPU cores and 3600 memory bandwidth. RMSE means root squared mean squared error. We use "our model/baseline" formats to show the result.

	RMSE (no)	RMSE (g50)	RMSE (c6m3600)
Medianflow_ds1	11.98/17.86	6.37/19.83	14.81/44.41
Medianflow_ds2	6.80/12.37	3.14/12.90	9.91/26.96
Medianflow_ds4	7.32/12.30	3.72/12.43	11.14/26.18
KCF_ds4	32.23/41.24	23.81/43.38	41.46/81.37
CSRT_ds4	46.66/92.54	44.17/102.98	78.97/179.77
Dense_ds4	14.15/24.45	6.45/26.32	11.75/53.26

MSE on the test dataset. Some trackers return high prediction RMSE (CSRT for example). The reason is some trackers may have unstable latency under high contention levels. We leave further exploration of this as our future work.

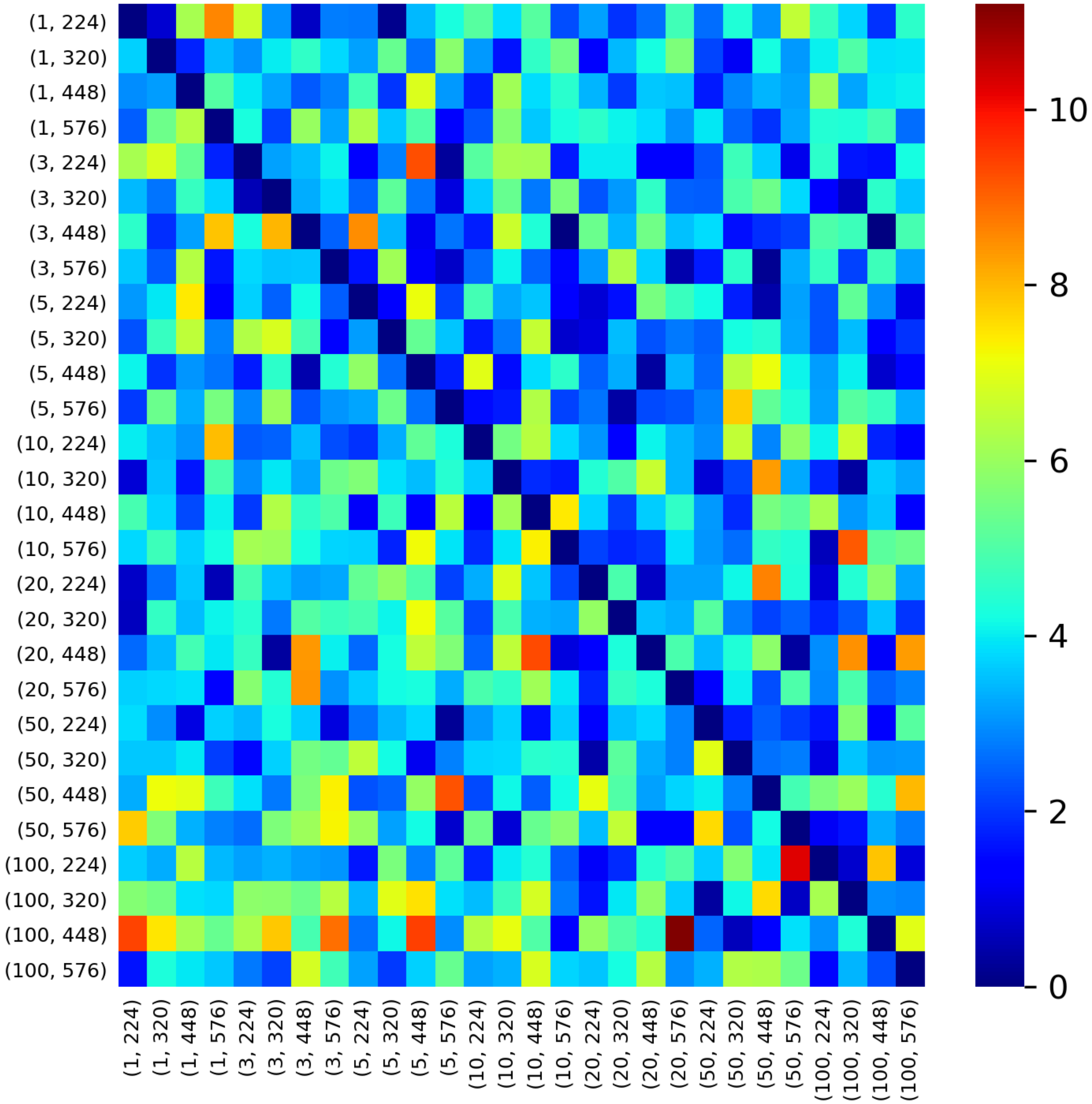


Figure 4.15. Heatmap of switching latency among ABs with varying "numbers of proposals" and "input shapes".

4.6.7 Overhead: Switching, Scheduler, and Online Components

Figure 4.15 shows a heatmap of the switching latency from any approximation branch of the detection framework to another, especially inside the detection DNN. This is an average of 10 such switches. The switching latency is defined as the difference of the execution time of

Table 4.5. Latency (as percentage of total latency) of different parts in ApproxDet system, measured with zero contention.

User requirement	Scheduler overhead (%)	Detection latency (%)	Tracking latency (%)
80ms	0.82%	59.65%	39.53%
100ms	0.62%	66.63%	32.75%
150ms	0.52%	69.10%	30.38%

the first frame in the new branches over that of following frames. We can find that switching latency is bounded by 12 ms.

The scheduler overhead comes from the accuracy and latency prediction models, as well as the contention sensor. Generally, the overhead of the scheduler is 11.09 ms per execution. Since we will only execute the scheduler once during all si frames, the average overhead of the scheduler is under 1 ms in most cases. This is supported by the profiling results in Table 4.5. The overhead of the scheduler occupies less than 1% of the total latency, suggesting that our scheduler is sufficiently fast for an online system.

4.7 Discussion and Future Work

Relation to the OS and Standing on the Contention. ApproxDet is positioned as a user-level application and does not change the OS's orchestration. Thus, we design ApproxDet to treat contention in a black-box manner. The advantage of such standing makes ApproxDet easier to implement, free from the OS restrictions, and reduce the complexity of contention scenarios by observing the impact of contention on ApproxDet's latency. However due to the black-box method, ApproxDet has the limitation of not knowing the exact contention details to adapt accordingly. Different contention scenarios may have the same impact on ApproxDet's latency and however ApproxDet will not be able to differentiate them, leading to potential sub-optimal adaptation. Future work may add OS privilege, observe the true contention levels from the OS, and even control the contention as well. The marginal benefit over the cost and lost features is yet to be revealed.

Contention Scenarios. We evaluate ApproxDet mainly with GPU contention from the synthetic CG because the object detector mainly runs on the GPU and the detection latency is much higher than the tracker latency. ApproxDet has the limitation on the contention source and scenarios, though we also include a case study with a real App. Future work may evaluate with various mobile background workloads from the real trace data and study the propagation effect of ApproxDet to the OS or other Apps.

Generalize to Other Detection Methods. Our adaptive detection framework allows any architectures of detection DNNs beyond FRCNN as long as we can expose the tuning knobs from them. The choice of the detection method can be another approximate knob.

Features of Performance Prediction Models. In ApproxDet, we consider the content features motivated by Figure 4.4, 4.5 and 4.6 with clear individual impact. We have considered the low-level content feature like the edges in the video frame but it does not have clear impact and thus has been excluded. More features can be introduced to improve the accuracy and latency prediction models, like object type, shape deformation, context, etc.

Reinforcement Learning (RL) for the Scheduler. RL based models can be an alternative method for implementing the scheduler. RL models learn to make schedule decisions based on training data and can potentially outperform the rule based policies.

Using Neural Networks to Infer Configurations. We leverage the sampling techniques to reduce the searching cost of configurations. Neural networks can be used to improve the searching efficiency if we disable the sampling and allow more flexible choices.

Evaluation Dataset and Devices. We plan to further evaluate ApproxDet on data-sets with high resolution videos (*e.g.* 1080p and 2K videos) and other mobile platforms.

4.8 Related Work

Object Detection. Object detection is a well established topic in computer vision and has made recent progress, thanks to DNNs. DNN-based object detectors can be summarized into two categories: single-stage detectors, such as YOLO series [56], [138], SSD [55], RetinaNet [160], and two-stage detectors, such as Faster-RCNN [57], R-FCN [159], Cascade R-CNN [173]. Single-stage detectors classify a dense set of grids, while two-stage detectors focus on a sparse set of region proposals oftentimes producing a separate region proposal network (RPN). Two-stage detectors are usually more accurate with reduced efficiency. While these detectors operate on single images, several recent works seek to extend them to the task of video object detection [94], [142]–[144]. A key idea behind these methods is to explore the temporal continuity of videos, where motion tracking is used to enhance the video object detection performance. We used this similar idea for our system, ApproxDet.

Efficient DNNs for Mobile Applications. There is an emerging interest to develop efficient DNNs for mobile vision applications. An important line of research is to identify lightweight network architectures with low computational cost. Examples include manually designed MobileNet family [74], [174], SqueezeNet [127], and ShuffleNet [97], as well as the more recent MNasNet [175], MobileNetV3 [176], FBNet [177], and EfficientNet [178] produced by neural architecture search [179]. Other techniques include the removal of redundant parameters (network pruning) [140], [180]–[182], the quantization of parameters (network quantization) [121], [124], [183]. While these architectures and techniques are primarily designed for image classification, they can be used as a building block for efficient object detectors [98], [139]. *In spite of the efficiency, none of these approaches can adapt to contention and content during runtime.*

Adaptive Inference for DNNs. The early work on anytime prediction [184] presents the first set of methods that are adaptive to a computing budget at runtime. This idea was recently revisited in the computer vision community using DNNs. For example, at inference time, a DNN can choose to drop certain operations [134], [185], or to select one of the many exits [75], [164] or branches [76], based on the image content or a given latency budget. Unfortunately, none of these approaches is designed for object detection. Besides, they do not consider the modeling of resource contention. In parallel to these developments, several recent work in the systems community also seek to build adaptive inference systems for DNNs. For example, NestDNN [77] uses network pruning to convert a static DNN into multiple DNNs, and dynamically selects from these to fit the resource requirement for image classification. AdaScale [158] learns to adaptively change the input shape of an object detection DNN, in order to achieve a latency-accuracy tradeoff. In comparison to NestDNN, our system ApproxDet uses a single adaptive DNN model for object detection. The use of ensemble models can be resource intensive [119] and is thus not suitable for many devices in our target class. In contrast to AdaScale, ApproxDet considers a joint adaptation of video content and resource contention.

Approximate Video Analytics. Our work shares similar ideas to several recent works in video analytics in the systems community. For example, server-side solutions like VideoStorm [2], Chameleon [112], and Focus [113] exploit various configurations and DNN models to optimize video analytics queries, but they also require loading of multiple models at the same time, which are challenging in resource-constrained mobile devices. If memory constraints prevent multiple models being co-resident on a device, it is conceivable to send them over the network on demand, using efficient wireless reprogramming [85], but the times involved are such that the prediction of which model will be required will have to be done far in advance. Liu *et al.* [70] explore the offloading of object detection to an edge device in combination with fast on-device tracking for mobile AR. ExCamera [1] enables low-latency video processing on the cloud using serverless architecture. VideoChef [79] uses approximation knobs of traditional video preprocessing filters in a content-aware manner. It cannot handle object detection and is not applicable to DNN-based video processing. A recent work in this space called MARLIN [186] shows that for AR applications, instead of continuously running the detection DNN, they can decide when to run their specially designed lightweight trackers. We take the idea of running the tracker at a configurable interval si, but we use traditional trackers.

4.9 Conclusion

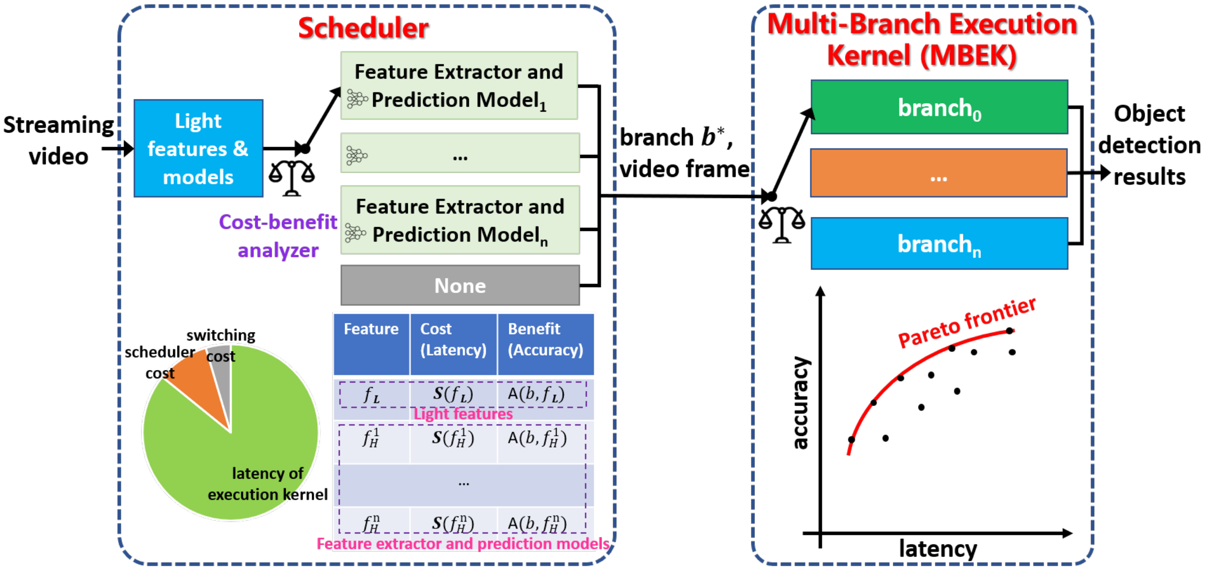
Here, we present ApproxDet, a single model adaptive system for video object detection in a video content-aware *and* contention-aware fashion, focusing on resource-constrained mobile/embedded devices. Case studies have shown that ApproxDet can adjust to different scenarios with best accuracy and least latency over previous models. Further, we contrast ApproxDet with multiple baselines, including AdaScale, a content-aware adaptive server-based video object detection system, and YOLOv3, a single-stage objection system, with high benchmarked efficiency on the ImageNet VID dataset. We find that ApproxDet is 52.9% lower latency and 11.1% higher accuracy over YOLOv3, outperforms all baselines, and has significantly lower switching overhead due to its single model design. Results on ImageNet VID dataset demonstrates the efficiency and effectiveness of ApproxDet, highlighting its promise in enabling latency-sensitive applications, pushing the frontiers of ever-evolving AR/MR (augmented/mixed reality) experiences, instantiated on embedded platforms.

5. LITEREConfig: COST AND CONTENT AWARE RECONFIGURATION OF VIDEO OBJECT DETECTION SYSTEMS FOR MOBILE GPUS

5.1 Introduction

Video object detection on mobiles has attracted considerable attention in recent years. Much progress has been made in developing light-weight models and systems that are capable of running on mobile devices with moderate computation capability [97], [175]–[177], [187]. The majority of previous works focused on statically optimized models and systems [139], [188], [189], pushing the frontiers of accuracy and efficiency. More recently, adaptive object detection models and systems [77], [112], [118], [155], [158], [190] have emerged. These approaches are capable of running at multiple accuracy and latency trade-offs, and thus are better suited for mobile devices running under real-world conditions. Real-world conditions include adapting to dynamically changing content characteristics, resource availability on the device, and user requirements. An adaptive vision system consists of two key components: (1) a multi-branch execution kernel **(MBEK)** with multiple execution branches each achieving an operating point in the accuracy-latency axes, and (2) a scheduler to decide which branch to use at runtime, based on video features and user requirements. Despite recent advances in adaptive object detection, no previous works consider the competing latency requirement between the two to meet an end-to-end accuracy-latency goal. Thus, maximizing the accuracy in adaptive vision systems, considering the latency cost of the scheduler itself, remains an unaddressed problem.

A major shortcoming of current adaptive vision systems is the competing demands between the MBEK and the scheduler. Importantly, previous works face two fundamental challenges. *First*, the system scheduler relies on the computationally light video features, *e.g.* height, width, number of objects, intermediate results of the execution kernel, to make the decision on which execution branch to run. Such features might not be sufficiently informative to represent the video content. On the other hand, computationally heavier content features or models, such as motion and appearance features of the video, can improve the


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

Feature	Cost (Latency)	Benefit (Accuracy)
f_L	$S(f_L)$	$A(b, f_L)$
f_H^1	$S(f_H^1)$	$A(b, f_H^1)$
...		
f_H^n	$S(f_H^n)$	$A(b, f_H^n)$

Figure 5.1. An illustration of our proposed cost-aware adaptive framework for video object detection. Our scheduler uses its cost-benefit analysis to decide which features to use for making a decision and then makes a decision on which execution branch to run for detection. The multi-branch execution kernel (MBEK) can be provided by any adaptive vision algorithm for mobiles and we build on top of several mainstream object detection and tracking algorithms.

decision making but are typically too heavyweight to include in the decision-making process. For example, the extraction of a high-dimensional Histogram of Oriented Gradient (HOG) feature and executing the models built on such feature takes 30.25 msec (Table 5.1), nearly the time of one video frame, on a leading-edge mobile GPU device, the Tegra TX2. *Second*, the scheduler incurs high switching overhead due to frequent reconfiguration among execution branches if the conditions change frequently. Thus, a cost-aware scheduler needs to damp the frequency of reconfigurations based on the cost of each, which can vary depending on the execution branch. No prior work has provided a cost-aware design of the scheduler and this fundamentally leads to their sub-optimal performance.

To address these challenges, we develop *LiteReconfig*, which is tailored to embedded boards with mobile GPUs on them. At its heart, LiteReconfig provides a cost-benefit analysis to decide at any point in a video stream which execution branch to select. A schematic of LiteReconfig is shown in Figure 5.1. The cost-benefit analyzer factors in the cost (as latency in our use case) and the benefit (in terms of accuracy improvement) of using

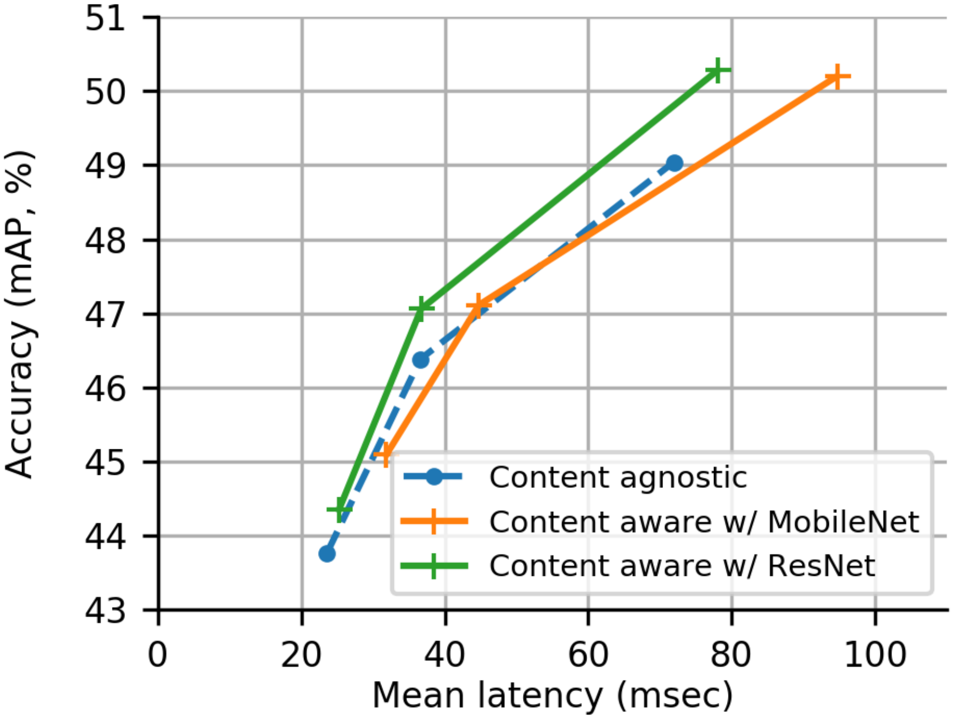


Figure 5.2. Motivation of cost-benefit analysis. We plot the accuracy vs. latency curve for three different strategies. Without a careful design, a content-aware strategy can be either better (*e.g.* ResNet) or worse (*e.g.* MobileNet) than a content-agnostic one. Here, the ResNet50 features come from the object detector with a lower cost than using an external MobileNet, making it a winning option.

computationally heavy content features in its models. 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 content in the streaming video. Further, LiteReconfig analyzes the cost and benefit of switching to a new execution branch when runtime conditions change, versus staying with the current one. For example, the motivating Figure 5.2 shows the accuracy vs latency curve (higher is better). Here we see that content agnostic is worse than one of the content-aware strategies (ResNet). More subtly, there is a content-aware option (MobileNet) that is worse than content agnostic, indicating the need for a rational decision on which content features to include. Here, the ResNet50 features come from the object detector in the MBEK and only additional extraction and model prediction cost are incurred, making it a winning option. Through careful design, we ensure that the overhead of using a content feature extractor and the corresponding model is minimal so as not to erase any gain from the optimization.

We evaluate our approach on the ImageNet VID 2015 benchmark and compare with our adaptations of SSD [55] or YOLOv3 [138] where we enhance their efficiency and adaptability by incorporating some tuning knobs so to run different points in the latency-accuracy spectrum (*e.g.* the size of the video frame). We also compare to a recent adaptive model [118] with the Faster R-CNN backbone. The evaluation is done on two embedded boards with mobile GPUs — Jetson TX2 and Jetson AGX Xavier. We show that LiteReconfig improves the accuracy by 1.8% to 3.5% mean average precision (mAP) over the state-of-the-art (SOTA) adaptive object detection systems, without affecting the latency. Under contention for the GPU resource, the SSD and YOLOv3 baselines completely fail to meet the latency requirement. Compared to three recent accuracy-focused object detection solutions, SELSA [191], MEGA [189], and REPP [192], LiteReconfig is 74.9X, 30.5X, and 20.3X faster on the Jetson TX2 board.

Contributions. We summarize our contributions as follows.

1. We develop a cost-benefit analyzer to enable low-cost online reconfiguration of the efficient and adaptive object detection framework. This optimization largely reduces the scheduler cost of the system and increases the accuracy since more of the latency budget can be used for the execution of the object detection kernel.

2. We develop a content-aware accuracy prediction model on the execution branch so that our scheduler selects a branch tailored to the content in the streaming video. Such a model is built on computationally heavy features and *cannot* be used without our cost-benefit analysis.

3. Through extensive experimental evaluation on two mobile GPU boards and against a large set of existing solutions, we provide two key insights applicable to lightweight adaptive computer vision systems (i) it is important to consider the effect of contention due to co-located applications, and (ii) it is important to engineer which features to use for making the selection of the execution branch. The full implementation of

LiteReconfig is able to meet even stringent latency requirement, 30 fps on the weaker board TX2 and 50 fps on the stronger board AGX Xavier.

5.2 Preliminaries and Motivation

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

5.2.1 Video Object Detection Algorithms

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

The detection network can be further categorized into two-stage [57], [159], [193] or single-stage detectors [55], [138], [139], [160], [194]. One representative work of two-stage detectors is Faster R-CNN [57], which employs CNNs to extract image feature maps and feeds it into Region Proposal Networks (RPN) to generate regions-of-interest (RoIs) in the first stage. Then, in the second stage, the RoI pooling layer combines the feature maps from convolutional layers, and the proposals from the RPN, together, to generate positive proposal feature maps, which is then provided to the classifier network. In contrast, single-stage object detection solutions do not include the step of region proposal generation and directly classify a dense set of pre-defined regions. These models are optimized for efficiency. For example, YOLO [56] is a well-known one-stage network that simplifies the detection of objects as a regression problem by predicting bounding boxes and class probabilities directly without generating region proposals. The EfficientDet [139] family is a group of one-stage detectors. It includes nine models with various input image sizes and network depths, all of which achieve SOTA performance, measured in FLOPs.

While single-image object detectors can be applied to videos frame-by-frame, this method ignores the reality that adjacent frames have redundancies. This temporal continuity in videos and can be leveraged to approximate the computations or to enhance detection results in neighboring frames. This has motivated this research direction in video object detection systems [142], [143], [165], [195]–[198]. Many previous approaches optimize for accuracy, explore temporal aggregation of object features [196], using either recurrent neural networks [195], or motion analysis [198]. More practical solutions integrate object detection with visual tracking [2], [70], [118], [143], where inexpensive trackers are used to connect costly object detection outputs. In this tracking-by-detection technique [142], [199], which has become a *de facto* strategy for real-time video analysis, an execution plan performs object detection, on the current frame, and object tracking on the following frames, in a Group-of-Frames (GoF). This combination of object detection and visual tracking into a video analytic system exposes additional design choices, such as the selection of the tracker and the triggering interval of detection, interleaved by tracking.

5.2.2 Content-Aware Video Object Detection Models

As has been discussed in Section 5.2.1, videos come with inherent information within a series of continuous frames. For example, the scale of objects, the moving speed of objects, the complexity of objects, etc. Based on these characteristics, some video object detection models [118], [158], [195] utilize the content information in videos, targeting improved latency and accuracy performance. We call such models as content-aware video object detection systems. During the model inference time, a content-aware video object system is able to reconfigure itself based on the content information from the streaming video while a content-agnostic system uses a static model variant or branch. AdaScale [158] is an example that makes use of the content information of videos to dynamically re-scale the images to achieve better performance. Another example is Liu *et al.*'s work [195] that employed Long Short-Term Memory (LSTM) layers to propagate the temporal context of each video frame to assist with the detection. ApproxDet [118] is a recent work that adapts in both dimensions of content and contention by changing the parameters of Faster R-CNN and object trackers, including the input image resolution, the number of object proposals, the detection interval, and the choice of trackers.

However, an efficient object detection system that is capable of reconfiguration at runtime faces two challenges: (1) Lack of content-rich features and fine-grained accuracy prediction. Insufficient feature extraction and inaccurate prediction before the reconfiguration could worsen performance. (2) Lack of cost-aware design. The system reconfiguration overhead (cost) is not considered when a decision is made. This may degrade overall performance if the reconfiguration cost is high. To the best of our understanding, no prior video object detection work solves either of these two challenges.

5.2.3 Adaptive Vision Systems

Despite the fact that video object detection algorithms perform well in terms of accuracy and latency on server-class machines, existing solutions suffer when running on edge or mobile devices, particularly with a tight latency service level agreement (SLA) and under varying resource contentions. There has been significant work on developing continuous vision applications on mobile or resource-constrained devices — some with human-designed deep models [97], [176], [187] and some with models given by neural architecture search [175], [177], [200]. Examples include deep models that tune the size of the input or other model parameters [112], [118], [155], [158] at inference time, prune a static DNN into multiple DNNs that could be dynamically selected [77], or select a different exit within a network [76], [182].

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

Most of existing approaches consider image or video classification tasks. Rather than predicting a single label per frame or video clip, video object detection has to examine every frame and output a varying number of object boxes per frame. Therefore, the design of an adaptive video object detection system poses additional challenges compared to an adaptive classification system. Only a few video object detection solutions exist to date [118], [158].

Some other adaptive vision systems [2], [69], [201] also leverage a multi-branch design methodology. For example, VideoStorm [2] considers two key characteristics of video analytics: resource-quality tradeoff with multi-dimensional configurations. Mainstream [69] 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 [201] 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.

5.3 Design of LiteReconfig

5.3.1 Approach Overview

The workflow of our system LiteReconfig is presented in Figure 5.1. LiteReconfig uses MBEK with multiple execution branches to meet different latency-accuracy trade-offs or to handle dynamic runtime conditions such as content changes. It can execute on top of any multi-branch continuous vision algorithm that consumes streaming video frames as inputs. At a high level, our solution LiteReconfig comprises of two parts that work together as a scheduler to determine which branch of the execution kernel to execute. The *first part* models the cost and the benefit of all possible *features* used by the scheduler to decide among the execution branches and then decides which features to use in choosing the execution branch. The *second part* models the cost and the benefit of the various *execution branches* of the MBEK and chooses the optimal execution branch. A final, meta-level optimization is performed to synthesize the outputs from the above two parts and to determine which execution branch to invoke. This is done by solving a constrained optimization problem (Equation 5.3) that maximizes the benefit (the improvement of accuracy) of the object detection kernel, such that the latency stays below the SLA.

In what follows, we present the optimization problem of the scheduler assuming the particular features to be considered are decided. The solution of this optimization leads to our choice of the execution branch to invoke (Section 5.3.3). We then present the cost-benefit analysis that enables the scheduler to pick the right set of features, considering their costs and benefits (Section 5.3.5). And finally, we introduce the cost-benefit analysis on the switching cost to guarantee the tail latency SLA. (Section 5.3.6).

5.3.2 Terminology in Adaptive Vision Systems

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

Execution Branch: is a distinct setting of a solution algorithm, typically differentiated by controlling some hyperparameters (colloquially, “knobs”), so as to finish the vision task in a distinct and fixed execution time (latency) and a consistent accuracy across the dataset. We observe that models with multiple execution branches are often considered by adaptive object detection frameworks. Such solution algorithms are gradually becoming more popular in the vision community, with an early start by BranchyNet [76], and a recent example being ApproxDet [118].

Accuracy-Latency Trade-off: The trade-off between accuracy and latency is fundamental to all adaptive vision systems. If a higher accuracy is desired, then one has to incur higher latency. For each execution branch, a certain accuracy and latency is achieved (for a given content type and contention level). The set of all such accuracy-latency values is represented in a curve like that shown in Figure 5.1, *bottom right*. Of these, the Pareto frontier is the one to focus on as any scheduler strives to stay on it. Note that the Pareto curve is subject to change at runtime due to the dynamic content characteristics and resource contention.

5.3.3 Scheduler Optimization

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

$$\begin{aligned} b^* &= \arg\max_{b \in \mathcal{B}} A(b, f) \\ &s.t.\ \ L(b, f) \leq L_0 \quad given \ \ f \in \mathcal{F}. \end{aligned} \tag{5.1}$$

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

Modeling of Latency: To build our latency model, we start by analyzing the sources of latency of our system. The end-to-end latency is comprised of two parts — the latency of the MBEK and the execution time overhead of LiteReconfig's scheduler. The latter is again composed of three parts — (1) the scheduler cost of extracting various features, (*e.g.*

the number and sizes of objects in the video frame, the histogram of colors, the degree of motion from one frame to another), (2) the scheduler cost of executing corresponding models to predict the accuracy and the latency of each execution branch for these feature values, and (3) the switching cost from the current execution branch to a new one.

Next, we observe that the selected features f can be divided into two types: light features f_L that will always be computed, such as height and width of the input video or the number of objects in the frame and are thus available to the scheduler for "free", and heavy features f_H, which may be extracted based on the cost-benefit analysis. A detailed list of the features considered in LiteReconfig and their costs are summarized in Table 5.1. We consider the following latency model that consists of four terms, given by:

$$L(b, f) = L_0(b, f_L) + S_0 + S(f_H) + C_{b_0}(b), \tag{5.2}$$

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

Modeling of Accuracy: Another central component of an adaptive vision system is the accuracy prediction model. Due to the latency SLA, in much of prior work, only features that are lightweight to compute are considered for modeling the accuracy of the execution branches [68], [77], [118], [201].

Our key observation is that the more expressive, yet computationally heavy features (f_H), can significantly improve the prediction. For example, we find that the widely used computer vision features, like Histogram of Colors (HoC) [202], HOG [203], recent neural network based features, like MobileNetV2 [174] in Table 5.1, can help build a significantly better accuracy prediction model, which we call the *content-aware accuracy model*. In addition to the three external feature extractors, we also use two features from the Faster R-CNN detector from the

Table 5.1. List of features and their costs considered in our scheduler. The latency is evaluated on the NVIDIA Jetson TX2 board. ResNet50, CPoP, MobileNetV2 feature extractors and the prediction models use the GPU; the others are mainly on the CPU.

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

MBEK — ResNet50 and Class Predictions on Proposal feature (CPoP). These are efficient to collect as these are obtained directly from the detector. These two features turn out to be informative features to characterize the accuracy of each branch in the MBEK.

5.3.4 Content-agnostic vs. Content-aware Accuracy Model

Instead of predicting the accuracy of an execution branch b on a representative large dataset (as one would with the content-agnostic features in [118]), we aim at predicting the accuracy of an execution branch b at a finer granularity, using a video snippet. A video snippet is a sequence of N consecutive frames,[1] starting at any point of the streaming video. In practice, since the scheduler must make a decision right on the current frame, we extract features from the first frame of the snippet and use these features to predict the accuracy of

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

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

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

Constrained Optimization: Given the optimization problem in Equation 5.1 and our latency model in Equation 5.2, 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

$$\begin{aligned} b^* &= \arg\max_{b \in \mathcal{B}} A(b, f) \\ &s.t.\;\; L_0(b, f_L) + S_0 + S(f_H) + C_{b_0}(b) \leq L_0 \\ &given\;\;\; f = [f_L, f_H] \in \mathcal{F}. \end{aligned} \tag{5.3}$$

To solve this optimization, we examine all branches $\{b\}$[2] that satisfy the latency constraint and pick the branches with highest predicted accuracy $A(b, f)$. Note that the latency prediction model $L_0(b, f_L)$ incorporates light features f_L but does not rely on the heavy content features f_H. Additionally, both the accuracy prediction model $A(b, f)$ and the latency prediction model $L_0(b, f_L)$ are trained from the data on our offline dataset. Critically, our key innovation lies in the design of the optimization problem rather than in solving it (we use standard convex solver). In the following sections, we will discuss (1) our feature selection algorithm for deciding which features f to choose for scheduling (Section 5.3.5), and (2) the modeling of switching cost $C_{b_0}(b)$ (Section 5.3.6).

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

5.3.5 Feature Selection for Scheduling

An important first step for our LiteReconfig is to select proper features used by the scheduler. To repeat the motivation, existing solutions do not consider the relative cost and the benefit of including various features, rather treat them as a single monolithic bucket of features to use for latency and accuracy prediction [118]. Consequently, only light features are considered for the scheduler due to the latency budget. In contrast, LiteReconfig dynamically decides which features to use during runtime, based on current video content characteristics and latency requirement. We now present our findings on the utility of different features, and describe our model for selecting features for accuracy prediction.

Light vs. Heavy Features: The light features f_L can be extracted with no cost and the corresponding content-agnostic accuracy prediction model on it is also computationally light. On the other hand, the heavy features f_H are content dependent and need processing of the video frame and more costly downstream neural network-based processing on them. An example of the former is the dimension of the image while an example of the latter is the MobileNetV2 feature of a video frame. Naturally, accuracy is enhanced with content-dependent features, such as HoC, HOG, MobileNet, and ResNet, as is well known in the literature [57], [141], [204]. We show empirically that this improvement happens under many scenarios (but not all). Further, one has to account for the decrease in the latency budget of the execution kernel due to the overhead of the features themselves, *i.e.* extracting the features and querying the content-dependent accuracy prediction models with the features. *This is the key idea behind our feature selection algorithm.*

Table 5.1 shows that the extraction of the HoC, HOG, and MobileNetV2 features takes 14.14 msec, 25.32 msec, and 153.96 msec, respectively, and the prediction models on these features take 4.94 msec, 4.93 msec, and 9.33 msec, respectively. This is because these features are high-dimensional to encode. Such costs can be overwhelming especially when the continuous vision system is running with a strict latency requirement, say 33.3 msec (30 fps). Supposing the scheduler is triggered at every first frame of a GoF, with size 8 (a middle-of-the-range number), the MobileNetV2 feature extraction plus prediction alone takes 61% of the latency budget. In several situations, this offsets its benefit in selecting a better execution branch through its content-aware accuracy prediction model.

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

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

$$\begin{aligned} f_H^{\mathrm{i}} = \underset{f_H \in \mathcal{F_H} \setminus f_H^S}{\arg\max} \max_{b \in \mathcal{B}} & A(b, f_L) + Ben(f_H^S \cup f_H) \\ s.t. \;\; & L_0(b, f_L) + S(f_L) + S(f_H^S \cup f_H) + C_{b_0}(b) + M(b) \leq L_0. \end{aligned}$$

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

We further simplify the calculation of the benefit $Ben(f_H^S \cup f_H)$ due to the heavy features in Equation 5.3.5. Concretely, this benefit depends on the content features and should ideally be calculated by extracting the heavy features from the current video frame. However, doing so would be costly and would defeat the purpose of this feature selection algorithm. The key difference of this equation from Equation 5.3 is that we use $A(b, f_L) + Ben(f_H^S \cup f_H)$ as a proxy of $A(b, f_H^S \cup f_H)$ to avoid extracting heavy features and executing the corresponding content-aware accuracy prediction model. The benefit function $Ben(f_H^S \cup f_H)$ is collected from the *offline* dataset to reflect the accuracy improvement of the system with the heavy features F against the light feature f_L.

5.3.6 Modeling Switching Cost

All prior works that have introduced adaptive vision models [2], [69], [77], [118], [158] have omitted to consider the latency cost of switching from one branch to another (or, in the case of ensemble models like [68], [164] from one model to another). On the other hand, LiteReconfig takes that switching cost into account in its cost-benefit analysis. This is motivated by our observation from Figure 5.5 that different branch transitions have different costs. This switching cost depends on the implementation and the nature of execution branches, and varies due to the size of non-shared data structure (such as, disjoint parts of a TensorFlow graph) to be loaded. We therefore perform a cost-benefit analysis to decide whether switching to a new branch b is worthwhile from the current branch b_0 by including the term $C_{b_0}(b)$, *i.e.* the cost of switching in latency terms, in the total cost formulation. The data is again collected from the offline training dataset.

Our model of switching cost only considers the current frame in the streaming video. Due to the unforeseen nature of the streaming video, we cannot forecast how long such a new branch b might stay optimal. Thus, the scheduler is triggered after every tracking-by-detection GoF number of frames, to deal with the dynamic nature of content characteristics. We found empirically that this strategy works better than the one employed in previous works [205], [206], which optimize over a look-ahead window by predicting future workload changes. We also found that previous approaches suffer from inaccurate predictions and a high cost of determining the schedule over a look-ahead. Further, our design of invoking the scheduler after every GoF (the size of GoF is typically 4–20) mitigates the impact of an incorrect decision.

5.4 Implementations and Baselines

5.4.1 Implementation of LiteReconfig

We implement LiteReconfig on top of a MBEK with Faster R-CNN as the detection backbone and four types of object trackers: MedianFlow, CSRT, KCF, and Optical Flow. We implement LiteReconfig in Python-3 (v3.7.3) using TensorFlow-gpu v1.14.0 (for Faster

R-CNN), PyTorch v1.4.0 (for MobileNetV2 feature extractors and neural network-based accuracy prediction models), CUDA v10.0.326, and cuDNN v7.5.0. We replicate the latency predictors in ApproxDet [118], enhance it for more types of embedded devices (ApproxDet only runs on NVIDIA Jetson TX2), and use them for predicting the latency of each execution branch, *i.e.* $L(b, f)$ (Equation 5.1). To train content-aware accuracy prediction model $A(b, f_H)$, we first collect the content-dependent features for each video snippet as the model input. Then, we collect the snippet-specific accuracy, *i.e.* the mAP, under each execution branch. These mAP results are used as labels for training our content-aware accuracy prediction model in a supervised manner. We use a 6-layer neural network for each content-dependent feature. The first projection layer projects both the light feature f_L and content-dependent feature f_H into vectors of 256 neurons and then concatenates the two. The following fully-connected layers come with 256 neurons in the hidden layer and M neurons in the output layer, where M is the number of execution branches. We use MSE loss and Stochastic Gradient descent (SGD) optimizer, with momentum of 0.9, to train the neural network, and use ℓ_2 regularization to prevent overfitting. We train the neural networks for 400 epochs at maximum with batch size of 64 and observe that the models converge within 100 epochs.

We evaluate LiteReconfig on two embedded platforms: an NVIDIA Jetson TX2 [207] and a more powerful NVIDIA Jetson AGX Xavier [208]. TX2 has a 256-core NVIDIA Pascal GPU on a 8GB unified memory while AGX Xavier has a 512-core Volta GPU on a unified 32GB memory. TX2 has compute capability similar to high-end smartphones like Samsung Galaxy S20 and iPhone 12 Pro, while AGX Xavier represents the next-generation mobile SoCs. Using two different platforms allows us to evaluate the performance of LiteReconfig with different target latency ranges.

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

5.4.2 Baselines

We consider these baselines for evaluating LiteReconfig.

1. **ApproxDet**: is the state-of-the-art adaptive object detection framework on embedded devices [118]. ApproxDet uses Faster R-CNN [57] as its backbone object detector, and is able to adapt to given latency requirements during runtime by dynamically changing the resolution of the image that is fed into the detector (*shape*), and changing the number of proposals (*nprop*) in the first RPN. The ability to adapt during runtime is enhanced by combining an object tracker with the detector, coupled with different tracker types (LiteReconfig uses the same four types as ApproxDet), different values of GoF, and the downsampling ratio of the image fed into the tracker. We use their open-sourced implementation [209], which uses TensorFlow-gpu v1.14.0.

2. **AdaScale**: AdaScale [158] is an adaptive object detection framework that can adaptively re-scale the input image to one of a number of preset resolutions. This feature gives AdaScale the ability to perform inference at different latencies, while maintaining optimal accuracy. Even with the adaptive features, the primary focus of AdaScale is not efficiency, and thus its base latency on embedded boards are too high to be compared with resource contention. Therefore, we execute AdaScale on TX2 without contention and only compare their mAP and latency values.

3. **YOLO+**: YOLOv3 [138] is a popular one-stage object detector with a faster speed due to its single-stage design. However, the implementation is still far from achieving real-time processing on our embedded devices. Thus, we enhance the efficiency of YOLOv3, call it YOLO+, by exposing four tuning knobs similar to the ApproxDet work – *shape* of video frame fed into YOLOv3, size of GoF (*si*), type of *tracker*, and downsampling (*ds*) ratio of the frame fed into the tracker. The YOLO implementation is YOLOv3 in PyTorch v1.4.0 from Ultralytics [210].

4. **SSD+**: SSD [55] is also a popular one-stage object detector that can be combined with multiple backbones as the feature extractor. In our work, we use the MobileNetV2 as

the backbone, combined with MnasFPN [211] to further enhance the object detector. We further engineer SSD for efficiency and adaptability and name it SSD+. This is done by exposing the same tuning knobs as for YOLO. An additional tuning knob is the confidence threshold of the detector that controls the number of objects to be tracked by the tracker. The SSD implementation is with Tensorflow v1.14, following the official implementation from Tensorflow Object Detection API [212].

5. **EfficientDet**: EfficientDet [139] is one of the recent SOTA object detectors, engineered for both accuracy and efficiency, with a total of 8 model variants in its family (D0-D7), each having a different scaling factor. From our observation, heavier model variants above D3 cannot run on the Jetson TX2 board due to insufficient memory, thus we have selected D0 and D3, which are the lightest and heaviest models within the executable candidates.

6. **SELSA** [191], **MEGA** [189], **REPP** [192]: We do a more limited benchmarking of these three recent solutions with the best benchmarking results for video object detection (on server-class machines, not embedded). We use pre-trained models, trained on the same dataset as ours. This comparison is more limited since these models are not optimized for efficiency, and therefore, have unacceptable latency on embedded devices. Additionally, they cannot execute with resource contention on our embedded boards — their base latency is already too high and they crash or hang with resource contention. Therefore, we execute them on TX2, without contention, and compare mAP and latency.

5.5 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.

5.5.1 Evaluation Setup

Dataset and Metrics: We evaluate LiteReconfig on the ILSVRC 2015 VID dataset [93], which contains 3,862 videos in the training set, and 555 videos, in the validation set. Both datasets are fully annotated with classes of the objects and their localizations. We report the mAP on the VID validation dataset as the accuracy metric, following widely adopted protocols [142], [144], [165], [189], [198]. We use 90% of the ILSVRC training dataset to train the vision algorithms (detection backbones and heads), including all baselines, and the remaining 10% of the training dataset, to build the following: the latency prediction model $L(b)$, the accuracy prediction model $A(b, f)$, the switching overhead model $C_{b_0}(b)$, and the benefit of heavy features $Ben(F)$. We report violation rate of the per-frame 95th percentile (P95) latency over SLA, as the latency metric, where the scheduler of each protocol aims to guarantee a violation rate $< 5\%$. This is the standard metric used for the evaluation of latency-sensitive ML systems [116], [213]–[215]. Our reported latency also includes the execution times from all parts of LiteReconfig, such as, the feature extractor, the model execution, and the scheduler. The latency of object detection on a frame is much higher than that of object tracking on a frame (7X to 330X in our system). Thus, we follow the widely used "*tracking-by-detection*" technique [216]–[218], where the object detector is invoked once in a GoF, and the tracker is used for the rest of the frames. If the averaged latency violates the latency SLA, the entire GoF is considered to violate the SLA.

5.5.2 End-to-end Evaluation

We first evaluate LiteReconfig for the accuracy of the entire system by setting several latency requirements from 100 msec (loose) to 33.3 (tight) msec per frame (10 to 30 fps) on the NVIDIA Jetson TX2 board, where no resource contention is injected. Table 5.2 summarizes the comparisons of LiteReconfig's four variants with the baselines. We should read the mAP and P95 per-frame latency, separated by slash, and corresponding to three latency requirements from tight to loose; a note of "F" means that solution failed to meet that latency objective. First, we observe that LiteReconfig and our in-house enhanced baselines (SSD+ and YOLO+) have improved efficiency over the SOTA ApproxDet from

Table 5.2. Performance comparison on the ImageNet VID validation set. "F" in the mAP column indicates that the protocol fails to meet the latency SLA and thus the accuracy results are not comparable (one exception for LiteReconfig in one cell to show the complete accuracy data). The bold text of mAP shows the highest accuracy in each scenario and requirement, while the italicized text of latency highlights that the 95% latency SLA is violated. An "F" in a latency cell means that that protocol did not execute at all.

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

33.3 msec to 100 msec. Second, LiteReconfig achieves 3.5% mAP improvement over ApproxDet given the 100 msec latency requirement — a significant improvement for an object detection task. Third, LiteReconfig achieves consistent accuracy improvement over the content-agnostic variant, *i.e.* LiteReconfig-MinCost, by 1.6%, 0.1%, and 1.3% mAP, respectively, for each latency requirement. Fourth, among all of LiteReconfig's four variants and our enhanced baselines, LiteReconfig and SSD+ achieve the highest accuracy under 33.3 msec requirement, LiteReconfig-MaxContent-ResNet is the most

Table 5.3. Performance comparison between LiteReconfig and the video object detection solutions optimized for accuracy.

Models, latency SLA	mAP (%)	Mean latency per-frame (msec)	Memory (GB)
SELSA-ResNet-101 [191], no SLA	81.5	2334	6.91
SELSA-ResNet-50, no SLA	77.31	2112	6.70
MEGA-ResNet-101 [189], no SLA	OOM	OOM	9.38
MEGA-ResNet-50, no SLA	OOM	OOM	6.42
MEGA-ResNet-50 (base), no SLA	68.11	861	3.16
REPP [192], over FGFA[142], no SLA	OOM	OOM	10.02
REPP, over SELSA	OOM	OOM	8.13
REPP, over YOLOv3	74.8	565	2.43
EfficienetDet D3	63.9	796	5.68
EfficienetDet D0	55.1	138	2.22
AdaScale-MS, no SLA	56.3	976.4	3.26
AdaScale-SS-600, no SLA	55.7	1049.4	3.20
AdaScale-SS-480, no SLA	59.0	710.5	3.18
AdaScale-SS-360, no SLA	59.4	434.0	3.18
AdaScale-SS-240, no SLA	56.5	227.9	3.18
LiteReconfig, 100 msec	50.3	72.0	3.67
LiteReconfig, 50 msec	46.5	38.4	4.09
LiteReconfig, 33.3 msec	45.4	28.2	4.12

accurate under 50 msec requirement, and LiteReconfig is again the most accurate under 100 msec requirement. Finally, though LiteReconfig-MaxContent variants face the issue of violating the latency requirement due to their high cost of running feature extractors and models, *LiteReconfig is strictly always below the latency requirement.* To summarize, in addition to efficiency improvement over SOTA, our cost and content-aware solution scales accuracy frontiers, with latency guaranteed to meet the requirement.

Further, we extend our evaluation to a higher resource contention scenario, *i.e.* 50% GPU resource is occupied by other concurrent applications. As can be seen in Table 5.2, first, under higher resource contention on TX2, our efficiency-enhanced baselines still fail to meet the latency requirement due to lack of awareness to adapt to the resource contention. Second, we are 1.8% mAP more accurate than the best adaptive systems. Third, our full implementation is always one of best protocols among all variants and models, satisfying the latency requirement (slight exception of 34.0 msec under 33.3 msec latency requirement).

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

Comparison to Recent, Accuracy-optimized Object Detection Solutions: Table 5.3 further compares LiteReconfig to recent solutions optimized for accuracy, which however cannot achieve real-time efficiency on the embedded device. We also add our evaluation on AdaScale, one of the content-aware baselines here since its fastest variant, the one always using the smallest scale, is running at 227.9 msec on the TX2 board and is much less efficient than LiteReconfig. Its accuracy of 55.7% to 59.4% mAP, though higher than LiteReconfig, is still the worst among all accuracy-optimized models. In addition, EfficientDet D0 and D3 achieve reasonable accuracy while maintaining relatively low latency compared with other accuracy-optimized solutions such as SELSA [191], MEGA [189], and REPP [192]. EfficientDet D0 runs at 138 msec at an mAP of 55.1% coming close to 100 msec, while the accuracy is higher compared to LiteReconfig. With no adaptive features, both EfficientDet D0 and D3 fail to match the 100 msec latency requirement.

Compared to most accurate models SELSA, MEGA and REPP, LiteReconfig is 90.3X, 36.8X, and 24.1X faster (for SLA of 33.3 msec). On the flip side, our accuracy is significantly lower. Thus, one may argue that each kind of solution has its own applicability — if real-time processing is required on these embedded boards, LiteReconfig is a good solution, while if frames can be sub-sampled for detection and high accuracy is required, then these recent solutions should be chosen. It should be noted that our experimental mAP values are lower than the published numbers in the papers (3.2% lower for SELSA, 9.2% for MEGA, and 23.8% for REPP), even though we used the author pre-trained models. We can explain

Table 5.4. Effectiveness of individual content-specific features on the accuracy given different latency budgets.

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%

the accuracy reduction in Table 5.3 to three factors (1) the change of the backbone feature extractors due to the memory constraint on the TX2, *e.g.* from ResNet-101 to ResNet-50, (2) the removal of the part of the solution that refers to future frames because our problem context requires streaming and real-time processing, and (3) the usage of the same mAP calculation script across all protocols.

5.5.3 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 we achieve, with the latency requirement applied to the MBEK only, and *ignoring the overhead of that feature*. In Table 5.4, we see that all content features achieve higher utility than the content-agnostic (labeled as “None”). The maximum accuracy improvement achieved by a single content feature (over “None”) is 2.3%, 0.7%, and 1.3%, respectively, for each latency requirement. This validates our design to use cost-benefit analysis to select the best features among all (feature) options and also determine whether the benefit is enough over the content-agnostic protocol, considering its cost.

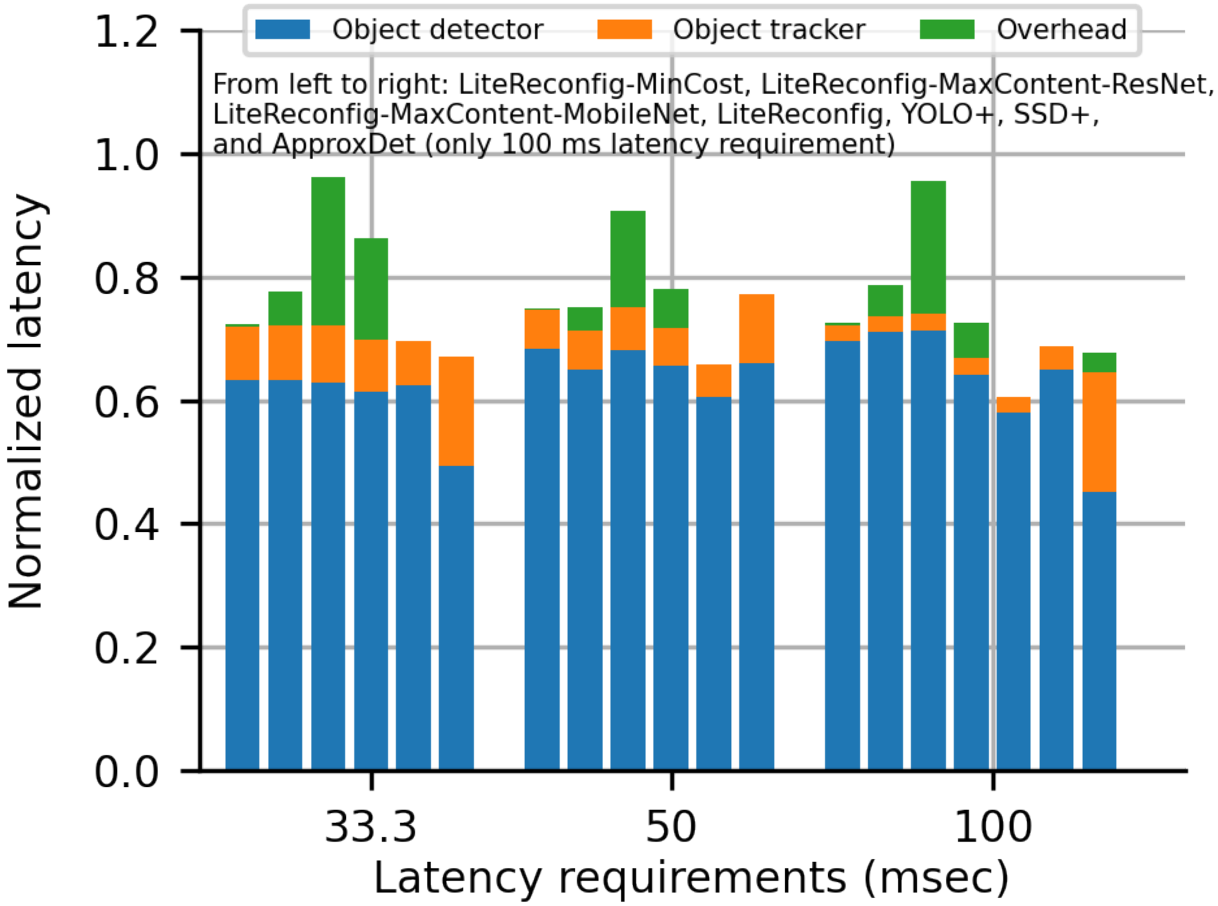


Figure 5.3. Percentage latency of each system component, normalized over the latency SLA, profiled on the TX2. FRCNN and YOLO cannot meet the 33.3 msec SLA and thus their bars are missing.

5.5.4 Understanding Latency-Accuracy Tradeoff

We examine the detailed latency breakdown of each system component in LiteReconfig to uncover the source of our benefit. Figure 5.3 shows the percentage latency (normalized by the latency SLA) of the object detector, the object tracker, and the cost, where the cost is either modeling (feature execution, regression models, solving optimization) or switching between execution branches. There is no bar for ApproxDet for 33.3 and 50 msec latencies because it cannot satisfy those SLAs. First, we can observe the cost of LiteReconfig is between the two LiteReconfig-MaxContent variants, owing to its cost benefit analysis on feature selection. Second, the overhead of LiteReconfig is always below 10%, and much less, for the higher latency requirements (50 ms and 100 ms). Finally, LiteReconfig wins over YOLO+, SSD+, and ApproxDet, due to higher latency SLA assigned to the object detector. Also, as we select the content-dependent optimal branches, even the latency of these branches may seem similar, but our selected branches finally lead to higher accuracy (Table 5.2). One may wonder why LiteReconfig does not try to use up the latency budget to get close to the 1.0 normalized latency value. This is merely an artifact of the presentation of this result — we are reporting mean latency while SLA is specified in terms of 95-th percentile latency (P95). Thus, LiteReconfig is using up its latency budget prudently, without causing too frequent SLA violations.

We further investigate the branch coverage (*i.e.* the number of distinct execution branches invoked) within the four LiteReconfig variants. Figure 5.4 shows that using heavy features (orange and green bars) tend to explore more branches tailored for the video content and thus is the driving force for higher accuracy. However, using light features can reduce more latency for the MBEK. The complete LiteReconfig (red) through its cost-benefit analysis balances these two tendencies and leads to the winning overall accuracy over other variants and baselines, and probabilistically guarantees the latency requirement. ApproxDet, covering a far higher number of execution branches (100 msec latency requirement), is still less accurate than any variant of LiteReconfig.

5.5.5 Switching Cost and Benefit

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 Equation 5.3. In Figure 5.5, we show empirically the switching overhead between any two execution branches in the object detector. The *offline* training data on Figure 5.5(a) shows that generally the switching overhead is below 10 msec but it is higher with a light source branch, *e.g.* *shape*=576 and *nprop*=1, or with a heavy destination branch, *e.g.* *shape*=576 and *nprop*=100. Figure 5.5(b) shows the *online* switching cost with 33.3 msec latency requirement at the top, and 50 msec latency requirement at the bottom. The online data confirms that the switching overhead is mostly less than 10 msec and we also observe the

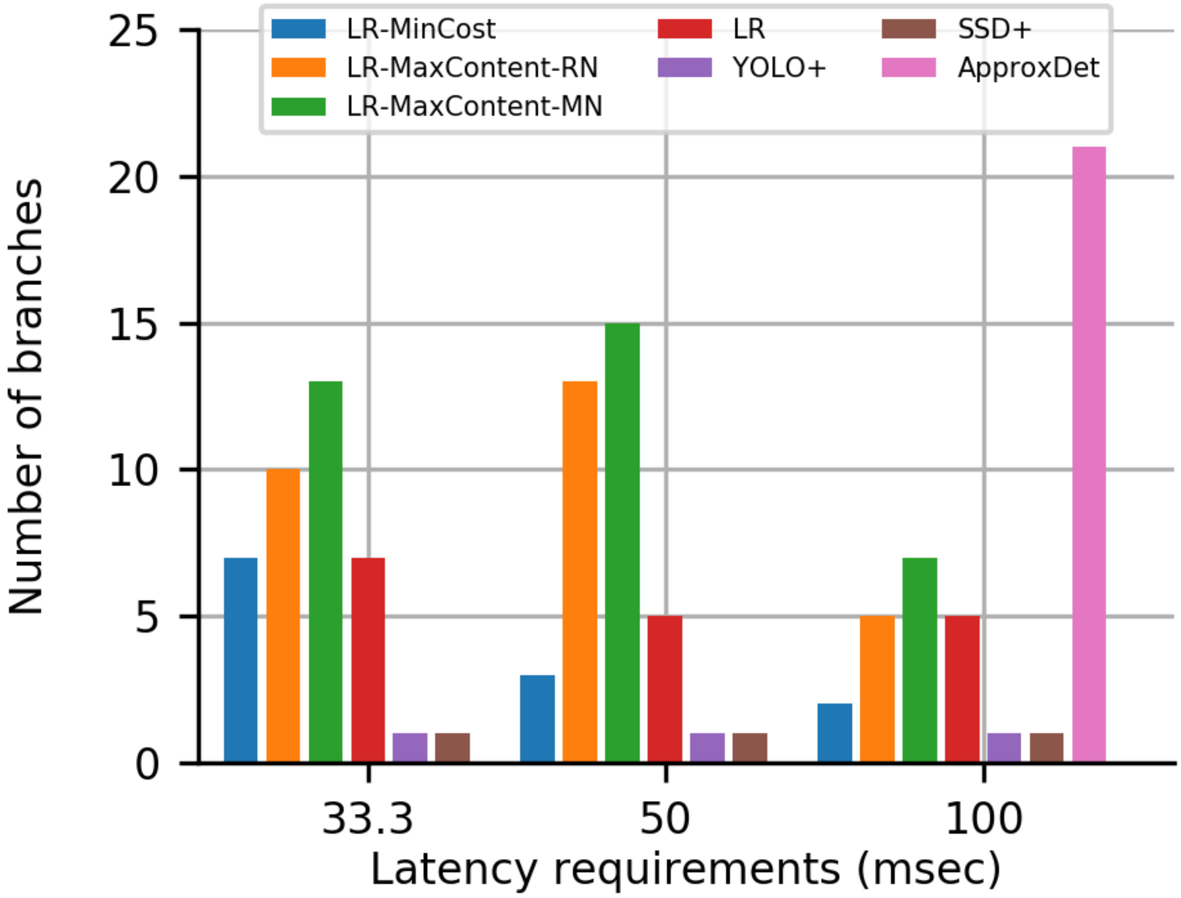
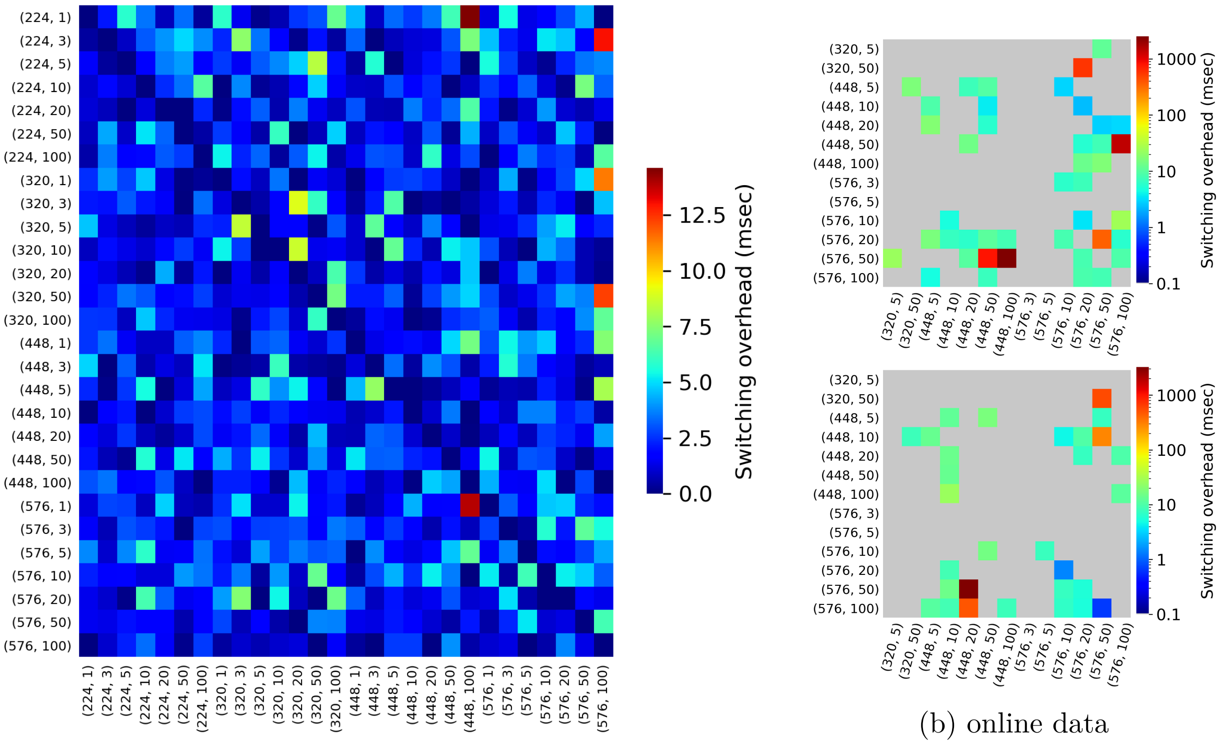


Figure 5.4. Branch coverage between the four variants of LiteReconfig and three other baselines. LiteReconfig is able to explore more number of beneficial execution branches and avoids fruitless switches between execution branches, leaving greater part of the latency budget available for the execution kernel, the object detector, and the tracker, leading to improved accuracy.

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.

5.6 Related Work

Object Detection and Tracking in Videos. Modern object detectors make use of DNNs to localize the recognized object instances within an input image. Examples include the



(b) online data

(a) offline training data

Figure 5.5. Switching overhead between execution branches in object detectors from (a) offline training data and (b) online data given 33.3 msec latency SLA on the top and 50 msec latency SLA on the bottom. The source branches are on the y-axis and the destination branches are on the x-axis, with (*shape*, *nprop*) notation.

YOLO series [56], [138], Faster-RCNN [57], EfficientDet [139], and Cascade R-CNN [173]. However, it remains challenging to apply these image-based detectors to videos, due to motion blur, occlusion, and defocus that frequently occur in videos. In parallel, visual tracking has evolved from lightweight trackers focusing on motion analysis and trajectory prediction [153], [219], [220] to DNN-based trackers that learn to match appearance patterns of target objects [143], [221], [222]. These developments have led to works on video object detection that use tracking to aggregate temporal features [142], [165], [195], [198] or to associate detected objects [2], [70], [118], [143]. Unfortunately, the majority of previous methods are meant to run on server-class systems, and only a few solutions exist for edge devices [70], [118]. Video object detection at the edge processes the videos where they are generated, and thus can improve latency and decrease network congestion.

Computer Vision on Mobiles. Resource utilization and management is one of the key factors to match the Quality of Experience (QoE) for end users in mobile devices. Many previous works focus on the design of lightweight DNNs to handle resource limits, including hand-crafted network architectures [97], [176], [187] , and network architectures built by neural architecture search [175], [177], [200]. 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 5.1 and 5.2). To sum up, based on the image content or latency budget, adaptive configuration can occur within one model [112], [118], [155], [158], within an ensemble of models within a system [77], multi-exit solutions [76], [182], or by generating a light network that is specific to a given dataset [190]. These methods, however, lack the ability to be fully adaptive to content and contention change (*e.g.* limited to a single dataset or a specific time interval) or to do a cost-benefit analysis to guide their adaptation.

Cost Benefit Analysis in Online Reconfigurable Systems. The specific context dictates many of the technical challenges in this space, like what are the cost and the benefit functions, how easily can the parameters for the functions be collected, and how should the cost-benefit analysis feed into changes made by a scheduler into the system. Some prominent examples in this line of work are for clustered database servers [110], for serverless jobs [223], for VM allocation and consolidation [224], and for VM migration [225]. In the context of mobile computing, such cost-benefit analysis has influenced decision making for mobile sensing [226], offloading from mobile to edge or cloud [227], or context-aware application scheduling on mobile devices [228]. While principles from this volume of prior work inspire our design, they do not directly solve our problem of reconfiguration of video object detection.

5.7 Conclusion

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

6. SMARTADAPT: MULTI-BRANCH OBJECT DETECTION FRAMEWORK FOR VIDEOS ON MOBILES

6.1 Introduction

Object detection is arguably one of central problems in computer vision. Much progress has been made over the past few years in deep learning based object detectors. Despite their impressive accuracy results on standard benchmarks, these models come at a price of their complexity and computational cost. This imposes a major barrier to deploy these models under resource-constrained settings with strict latency requirements, such as detecting objects in streaming videos on mobile or embedded devices. Several recent works seek to address this issue by designing light-weight models on mobiles [97], [175]–[177], [187], in particular, for video object detection [158], [188], [195], [229]. The assumption and the common belief are that those object detectors optimized for accuracy, such as Faster R-CNN with a ResNet-50 backbone, are too expensive for mobile vision tasks.

Indeed, detectors optimized for accuracy are rather complex, often trained with different input resolutions, and equipped with multiple stages (*e.g.* proposal generation). It is perhaps not surprising that these detectors can adapt to different settings at inference time. Consider the example of Faster R-CNN [57] with ResNet-50 [53], one can reduce the input resolution or the number of proposals for a lower latency while still maintaining a reasonable accuracy. Such choice combinations of tunable parameters would constitute a multi-branch object detection framework (MBODF), and the Faster R-CNN detector using a specific input resolution and a particular number of proposals from our previous example could be considered as one execution branch.

Our key observation is that if one is allowed to select at inference time from a large set of fine-grained execution branches, the detection accuracy and latency can be significantly improved (see Figure 6.1). Then, the key research questions are: *How to expose the right set of execution branches in an existing object detector and then how to schedule the optimal one at inference time?*

An ideal scheduler must not only consider the branches in the model and their properties (accuracy and latency), but also the input content. For example, if the input video only contains larger objects, using a lower input resolution for the detector suffices. However, the design of such a scheduler is challenging for streaming videos. This is because the scheduler has to "predict" potential future content change in order to select the best branch at the current time.

In this work, we focus on the challenging and practical task of streaming video object detection on mobile devices, and present a simple adaptive object detection method. Our key innovation is to leverage standard existing object detectors (Faster R-CNN, EfficientDet, SSD, YOLO) to construct an MBODF for adaptive video object detection. An MBODF combines an object detector and an object tracker and provides many execution choices (branches).

We demonstrate that our method, notwithstanding its simplicity, can adapt to a wide range of latency requirements, ranging from 10 to 50 FPS on a mobile GPU device, the NVIDIA Jetson TX2 (a widely used device for embedded/mobile vision benchmarking [230]–[232]) with only a minor accuracy loss. For example, our method when running at 20 FPS in streaming mode on TX2, achieves an mAP of 70% on a large-scale dataset (ImageNet video object detection benchmark). In contrast, the best performing detector, MEGA [189] that supports streaming videos, has an mAP of 75.4%, and runs only at 1.2 FPS. Further, SmartAdapt achieves 20.9% to 23.6% higher mAP than the state-of-the-art multi-branch algorithm [233] given the same constraint on the streaming latency (33-100 msec per frame) (Figure 6.6, FR+MB vs. FastAdapt).

Next, we uncover the importance of making a content-aware decision on the branch to run, as the optimal one is conditioned on the video content. We explore a content-aware scheduler—an Oracle one, and then a practical one, which uses various light-weight feature extractors, to adapt at runtime to the content. We show that our content-aware Oracle scheduler achieves a 6.6%–8.3% higher mAP than a content-agnostic one (Table 6.3, FR+MB+Oracle vs. FR+MB). When our realistic content-aware scheduler (CAS) is used, the gains are more modest but still present, ranging from 0.1%–2.3% (Table 6.3, FR+MB+CAS vs. FR+MB, FastAdapt+CAS vs. FastAdapt). The strength of SmartAdapt is due to the synergistic use of a carefully orchestrated set of features that demonstrate both a low computational overhead and high accuracy, and expose a fine-grained set of branches using MBODF.

Thus, our contributions can be summed up as:

1. We point out that modern detectors are intrinsically adaptive, and can be re-purposed as an MBODF, achieving varying latency and accuracy trade offs at inference time, using a set of (individually) proven adaptive attributes.

2. Object detection solutions with multiple sub-model or branch choices have been proposed in the past [118], [158], [164], [234]. However, SmartAdapt's ability to combine a set of knobs (such as the input resolution and number of proposals) and also to tune the ranges and step sizes to be fine-grained, enables MBODF on a mobile device to span a wide range of latencies with only a minor drop in accuracy (3% mAP) across this range of latencies.

3. We show that an MBODF can achieve significant performance gains when the choice of the execution branches is optimally conditioned on the input content (*e.g.* size and speed of objects). We also take an exploratory step toward practical *content-aware* adaptive object detection.

6.2 Related Work

Efficient Object Detection Models. Efficiency is paramount on embedded or mobile devices, where power is limited. Many solutions design more efficient network architectures (*e.g.* [178]) or components within an architecture (*e.g.* [235]). MicroNet [236] factorizes a convolution matrix into a lower rank one, which can cut down computational cost effectively. TinyNet [237], inspired by EfficientNet [178], uses a compound scaling method to downsize the neural architectures in a fixed ratio. Skip-Conv [238] proposes to use a gating function that allows convolutions to run on a sparse set of locations. While these methods can effectively reduce the computation cost in terms of FLOPS, they are rarely evaluated on mobile GPUs. Further, reduction in FLOPs does not always translate to reduction in latency [175], [238].

Another stream of studies for efficient object detection is to combine a costly object detection module with a relatively inexpensive object tracker module via the "tracking-by-detection" scheme [143], [196]. [239] reallocates the computation of the detection task along the pipeline, by utilizing multiple networks and propagating the detection results across these networks. However, existing studies do not consider the wide range of configurations that may impact the detection performance and rarely focus their evaluations on mobile GPUs.

Adaptive Inference for Image Recognition. These models leverage the content characteristics from the input images and make execution decisions conditioned on them. RA-Net [164] constructs multiple sub-networks with increasing depth and processes images with different complexity at different depths. Similarly, BranchyNet, MSDNet, and Approx-Net [75], [76], [155] propose multiple exits along the network pipeline for images of different complexities. AdaScale [158] resizes images at different scales targeting higher speed and accuracy. However, previous works are limited to adaptation in one dimension, and with a narrow range, and do not optimize the execution choice by ingesting a rich set of input features available to object detection pipelines.

Adaptive Inference for Video Recognition. Unlike a single image, videos have a temporal continuity among neighboring frames, and capturing the overall features effectively yet efficiently is one of the key challenges. There are two mainstream approaches that leverage this to make video object detection computationally efficient. One approach is to integrate a new module to extend the capabilities of the main neural network. This involves using an LSTM model [240], [241] or a policy network [242] to capture the feature information of the video frames, thus reducing computational cost by identifying redundancy. For example, FrameExit [243] uses a cascade of gating modules to determine at which layer to exit processing. Another approach is to focus on the efficiency of the network architecture, such as X3D [244] that expands the base 2D image classification architecture into multiple axes. Other works in this space use network quantization and spatiotemporal convolutions [245], [246]. However, only a handful of previous works considered video object detection, which is fundamentally distinct from video classification.

6.3 Efficient Object Detection Models for Streaming Videos

Our goal is to maximize the accuracy of video object detection models for streaming videos at stringent latency constraints (*e.g.* 33 msec) on mobile GPUs. We now present our solution design and describe our techniques.

6.3.1 Multi-branch Object Detection Framework

Tracking-by-detection. This is our starting point to significantly reduce the latency of the object detection models with a minor accuracy reduction. Rigorously, we define a Group of Frames (GoF) as a sequence of *d*i (detection interval) consecutive frames in a streaming video, in which we run object detectors (*e.g.* Faster R-CNN, EfficientDet, YOLO) on the first frame and run object tracker (*e.g.* MedianFlow, KCF) on the remaining frames. In the streaming scenario, as we process the video frame-by-frame, an object detector can run on any frame with no prerequisite while an object tracker depends on the detection results, either from a detector, or from the tracker on the previous video frame. Considering our implementation on a Faster R-CNN object detector (in PyTorch, with mobile GPU) and a MedianFlow object tracker (in OpenCV, with mobile CPU), the tracker runs up to 114X faster, boosting the efficiency.

Tuning Knobs at Inference Time. To further improve the efficiency and avoid a large accuracy reduction, we design tuning knobs for such tracking-by-detection scheme. Our design explores the accuracy-latency tradeoff in five independent dimensions: (1) the detector interval (*d*i), controlling how often an object detector is triggered, (2) the input resolution of the detector (rd), controlling the shape of the resized image fed into the object detector, (3) the number of proposals ($nprop$), controlling the maximum number of region proposals generated from the RPN module of the Faster R-CNN detector, (4) the input resolution of the object tracker (rt), controlling the shape of the resized image fed into the object tracker, and (5) the confidence threshold to track (ct), controlling a minimum threshold on the confidence score of the objects below which the objects are not tracked and output by the tracker. The multi-knob design leads to a combinatorial configuration space as we can tune each knob independently and in various step sizes. This allows a wide range of adaptations and is key to SmartAdapt's impressive empirical results in what follows.

MBODF. We name the multi-knob tracking-by-detection scheme, with the range and step sizes for each knob, a *Multi-branch Object Detection Framework (MBODF)*.

An execution branch in the MBODF is an instance of the values of each knob. Note that not every branch in the configuration space is valid, *e.g.* for branches that run object detector on every frame ($d\text{i} = 1$), the *rt* and *ct* knobs are not relevant.

Figure 6.1 shows the accuracy comparison between a 2-knob 54-branch, a 5-knob 368-branch, and a 5-knob 3,942-branch MBODF, where each point on the Pareto optimal curve stands for the accuracy and latency performance of a single branch on the ILSVRC VID dataset. A 5-knob MBODF is much more efficient than a 2-knob MBODF (*rd* and *nprop*). It achieves a 6.1X speedup, with only a 2.41% mAP reduction, compared to 3.0X speedup, with a 2.37% mAP reduction in the 2-knob MBODF. In contrast, the 5-knob MBODF with 10X more branches (3,942) is only slightly better than the one with a subset of branches (368) at any given value of a latency constraint. The root cause of such reduced accuracy improvement is the *lack of* smarts in choosing the execution branch conditioned on the video content. In other words, if only applying a single static branch on an entire dataset, without finer-grained content revelations (as revealed from Figure 6.3), one cannot reap the benefit of the much larger-scaled MBODF.

6.3.2 Branch Selection Problem

A scheduler is a component in the object detection system with an MBODF that decides which branch is the optimal one to run, subject to some criteria. Considering an MBODF with $m = |\mathcal{M}|$ independent execution branches $b \in \{b_1, b_2, ..., b_m\}$ that are capable of finishing the object detection task on streaming videos, we use the latency of the branch as the constraint and maximize its accuracy as the optimization goal as follows:

$$b_{opt} = \arg\max_{b} a(b, \hat{X}), s.t.\ l(b, \hat{X}) \leq l_0 \tag{6.1}$$

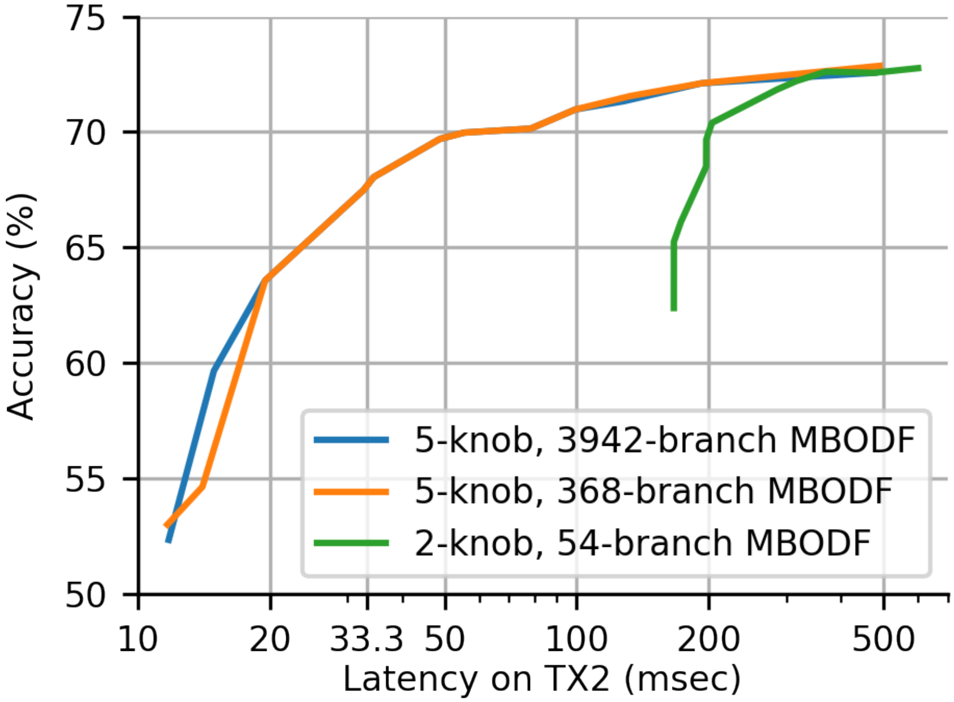


Figure 6.1. Accuracy comparison of a 5-knob MBODF and a 2-knob sub-framework (input resolution, number of proposals).

where $\hat{X}$ denotes the input video frame, l_0 denotes the latency constraints for each frame on average, and $a(b, \hat{X})$ and $l(b, \hat{X})$ represent the accuracy and latency of the branch b. Figure 6.2 shows the workflow of SmartAdapt where the scheduler takes the video frame as an input and determines the execution branch in the MBODF to run. Inside the scheduler, the workflow is as follows: (1) extracts the content features, (2) predicts the accuracy with a content-aware accuracy predictor, and then (3) uses a branch selector to choose the optimal branch. Particularly, given the tracking-by-detection scheme in the MBODF, where a GoF is a unit for scheduling, $\hat{X}$ is relaxed to a GoF. In the streaming scenario, a scheduler should be able to make a decision at any frame x_t in the streaming video where the $\hat{X}$ is the GoF starting from the frame x_t.

As the optimal branch selection is conditioned on the current frame and a few future frames, [1] a content-aware scheduler leverages the content characteristics in such a GoF to

[1] ↑The size of the GoF is typically between 1 and 100 in the MBODF.

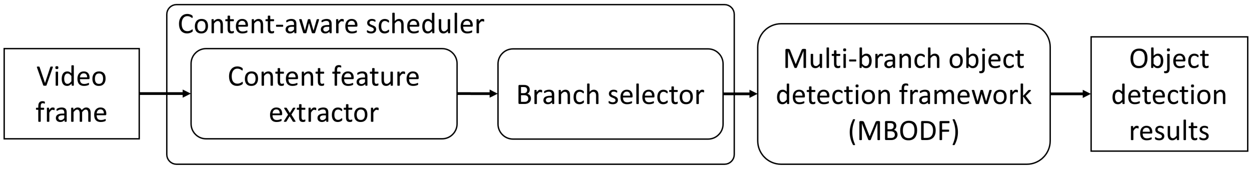


Figure 6.2. Workflow of SmartAdapt.

maximize accuracy. In contrast, a *content-agnostic* scheduler considers the average accuracy of different branches across the entire dataset, which loses the nuances of the snippet-level video characteristics.

In Figure 6.3, we show the Pareto optimal branches for three randomly selected video snippets of different content characteristics, and the one that inputs the entire dataset for $\hat{X}$. We glean that the accuracy-latency frontiers vary significantly from snippet to snippet and are different from the "average" for the entire dataset (red curve). This motivates use of a content-aware scheduler for identifying the execution branches for the end-to-end video object detection pipeline. According to our study, 83.4% branches in the MBODF are most accurate for at least one video snippet at any latency requirement. Among a dataset of 1,256 video snippets, derived from the ILSVRC VID dataset, we find 627 unique sets of accuracy-latency frontier branches. Thus, we conclude that it is important to determine the optimal branch for a given video snippet rather than use a single branch for an entire dataset. This latter approach is also observed with benefit in some prior works, addressed either by using a content-agnostic scheduler [233], or enabling multiple sub-models to choose from [164], [234].

6.3.3 Content-aware Oracle Scheduler

We name a perfect content-aware scheduler for an MBODF $\mathcal{M}$ an "Oracle" scheduler, which selects the optimal branch b_{opt} to execute. The accuracy-latency performance of an Oracle scheduler establishes the upper-bound performance of a content-aware scheduler, something that has not been established up until now.

To realize an Oracle scheduler, we grant three impractical powers to it—(1) it has access to the future frames in the GoF, (2) it has the annotation of the objects to calculate the ground truth accuracy $a(b, f(\hat{X}))$ so that no predictions are performed, (3) it exhaustively

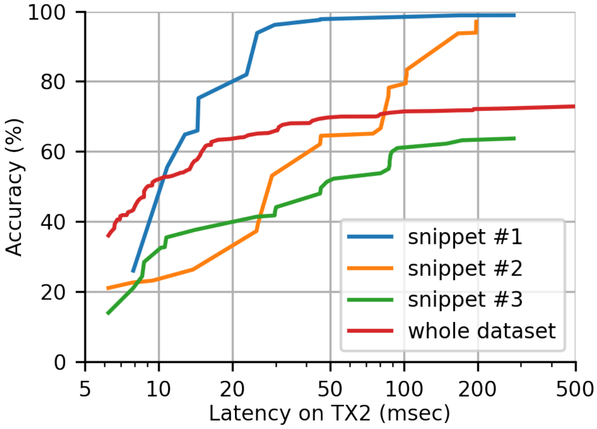


Figure 6.3. Pareto optimal branches for three selected video snippets of different content characteristics and the whole dataset.

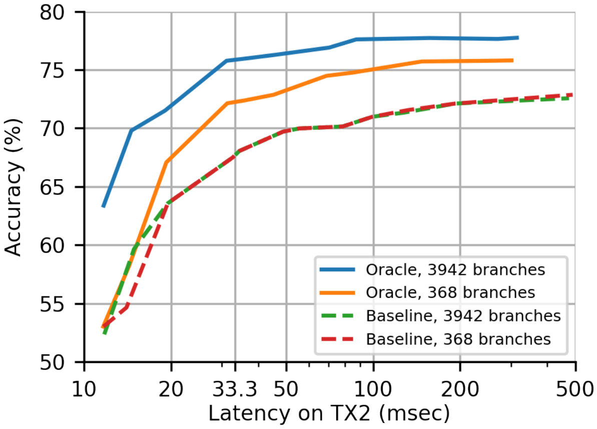


Figure 6.4. Upper bound performance of a content-aware scheduler, *i.e.* an Oracle.

tests all available branches and selects the most accurate one, subject to the latency constraint. Figure 6.4 shows the performance of the Oracle scheduler on two 5-knob MBODF instantiations, with 3,942 and 368 (a subset) branches and compares with a content-agnostic scheduler, which chooses a single static branch for the entire dataset. We observe that the Oracle scheduler has a 3.2% to 4.6% mAP improvement in the 368-branch MBODF at 10, 20, 30, and 50 FPS, four typical latency constraints on mobile devices. This is relative to the baseline with 368 branches. Interestingly, the mAP improvement of the Oracle scheduler is higher for the 3,942-branch MBODF, 6.6%–8.3%, compared to the above-mentioned 3.2%–4.6%. In contrast, such large-scaled MBODF has no benefit in the content-agnostic setting. The large gap motivates a content-aware scheduler that can adapt over a large and fine-grained range of knobs.

6.3.4 Designing a Content-aware Scheduler (CAS)

We have to design a light-weight scheduler that determines the content-specific execution branches on-the-fly, bereft of the impractical powers that we granted to the Oracle. As Equation 6.2 suggests, the branch selector in the scheduler requires a latency predictor and

Table 6.1. Feature extractors in SmartAdapt's content-aware scheduler.

Name	Dim.	Trainable	Description
light	4	No	Composed of height, width, number of objects, averaged size of the objects
HoC	768	No	Histograms of Color on red, green, blue channels
HOG	5400	No	Histograms of Oriented Gradients
ResNet50	1024	No	ResNet50 features from the object detector in the MBODF, average pooled over height and width dimensions, and only preserving the channel dimension
CPoP	31	No	"Class Predictions on the Proposal" feature (CPoP) from the object detector of the MBODF, averaged pooled over all region proposals, and only preserving the class dimension (including a background class)
MobileNet	1280	**Yes**	Efficient, effective feature extractor, average pooled from the feature map before the fully-connected layer, and only preserving the channel dimension

an accuracy predictor to solve the optimization problem. The former has been studied in an existing work [118] through a resource contention sensor and a content-aware latency predictor on each execution branch. Thus, in this work, we focus on the design of a *content-aware accuracy predictor* based on simplified content features.

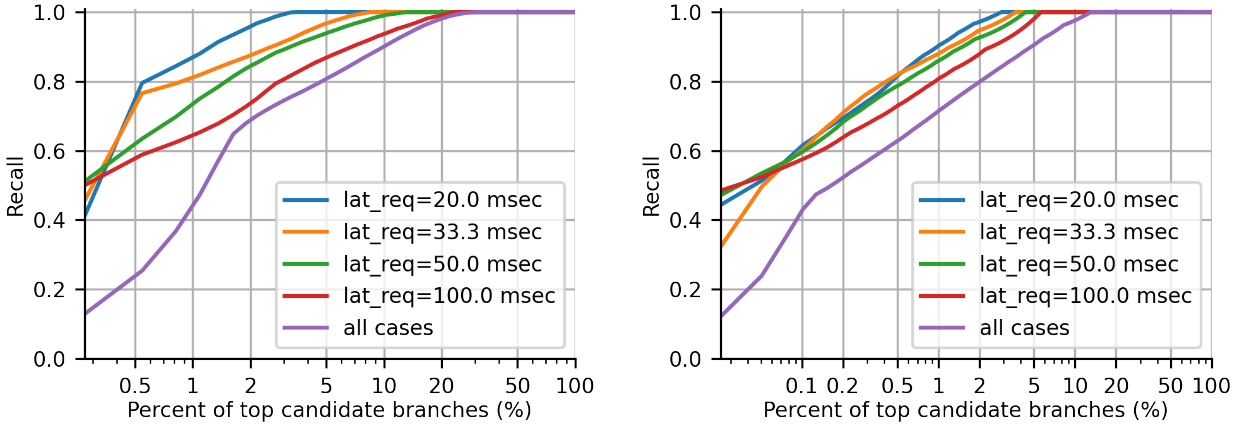
Content Feature Extractors. A content feature extractor aims to build a mapping $f(\cdot)$ from the frame representation $\hat{X}$ to its feature representation since the frame representation carries too much redundancy. The content feature extractor is expected to be discriminative so that the feature values it carries can be used to predict the content-specific accuracy of each execution branch. Then, a content-aware accuracy prediction model aims to build a mapping $a(\cdot)$ from the feature representation $f(\hat{X})$ to the accuracy of a given execution branch b. Thus, the scheduler model can be formulated as follows:

$$b_{opt} = \arg\max_{b} a(b, f(\hat{X})), s.t.\ l(b, \hat{X}) \leq l_0 \tag{6.2}$$

A well-designed content feature extractor should be rich in content characteristics, discriminative enough, and light-weight in the computational overhead. In Table 6.1, we summarize the list of content features, specs, and descriptions. We start from some light features that come with no cost to extract, *i.e.* the height and width of the video frame, the number of objects, and average size of the objects. We then choose two traditional vision features—Histograms of Color (HoC) and Histograms of Oriented Gradients (HOG) to characterize the color and gradient information. Considering that the object detection in the MBODF is composed of a deep neural network with many feature extraction layers, we choose one from the intermediate layer after the feature extractor head of Faster R-CNN backbone, *i.e.* ResNet-50, and one from the last but one layer of the object detector, *i.e.* the prediction logits on the object classes. These two features are attractive as they incur no extra computation cost. Finally, we propose to use a widely used DNN-based feature extractor, MobileNetV2. It is light-weight in terms of the computation cost and jointly trainable with the downstream content-aware accuracy predictor. Naturally, at inference time, the scheduler has to run ahead of the MBODF and thus has to rely on extracted content features from the previous GoF. Due to the well-known continuity property in video content characteristics [242], [243], this simplification works well in practice.

Content-aware Accuracy Predictor. A content-aware accuracy predictor predicts the accuracy of all branches in the MBODF given a feature vector. We use a 5-layer fully connected neural network (NN) with a residual connection [53] for each layer. We use ReLU as the activation function and have 256 neurons in each hidden layer. Particularly, as the dimensions of the light features and other features vary significantly in 1 to 3 orders of magnitude, we add a feature projection layer before the features are concatenated and fed into the 5-layer NN. The feature projection layer projects both light features and other high-dimensional features to fixed 256-dimensional vectors so that they are equally representative in the accuracy predictor. We use MSE loss for the accuracy predictor using as ground truth accuracy of the branches on each video snippet in a derived snippet-granularity dataset from the ILSVRC 2015 VID dataset (see Section 6.4 for details).

Joint Modeling of Content and Latency Requirement. We additionally explore a network that jointly models content and latency requirement for branch selection. Different from the previous design, this model does not pair with a latency predictor and thus is simpler in design. Specifically, we begin by embedding content and latency requirement into separate feature vectors using multi-layer perceptrons (MLPs). Following FiLM [247],



(a) A 368-branch MBODF

(b) A 3,942-branch MBODF

Figure 6.5. Recall of top candidate branches (by percentage of the number of branches in MBODF) from the Optimal Branch Election.

our model regresses a set of affine weights γ and biases β from the latency feature F_l using another MLP and subsequently transforms the content feature F_c as $F_c^{'} = \gamma \cdot F_c + \beta$. In doing so, our model adapts to the current latency requirement through the modulation of content features. An MLP further processes the modulated content features $F_c^{'}$ and predicts accuracy of all branches. We train our model using the same MSE loss as before, except that we set the target accuracy of a branch to zero when latency requirement is violated. We show in our experiments that this joint modeling scheme is most effective under tight latency constraint.

Candidate Branches. Predicting on hundreds and thousands of execution branches is challenging. SmartAdapt narrows down the number of candidate execution branches in the design phase to top K. The intuition is that the top K execution branches should cover the majority of optimal branches across videos of different content characteristics and different latency constraints, for properly chosen K. We use the method called *Optimal Branch Election* (OBE) to select the K candidate branches. Figure 6.5(a) shows the recall of using K branches (*i.e.* proportion where the optimal branch belongs to one of the top K) , rather than all 368 branches. We see that in the 368-branch MBODF, 10.1% branches suffice to achieve 90% recall. Also, if we consider the candidate branches *for a particular latency constraint*, even fewer can be considered. To achieve a 90% recall, the percentages of K branches are 1.4%, 2.7%, 3.3%, and 7.1%, given 20, 33.3, 50, and 100 msec latency constraints. Figure 6.5(b) shows such recall-K relation on a larger-scaled MBODF with 3,942 branches. The percentages of branches that need to be considered are lower. Thus, the content-aware accuracy predictor on K candidates can benefit with a smaller model and reduce the cost of profiling.

6.4 Implementations

All models are trained on a server with two NVIDIA P100 GPUs; evaluated on NVIDIA TX2 with a 256-core NVIDIA Pascal GPU on a 8GB unified memory.

Profiling. Once the tuning knobs are determined for each object detector, it is important to determine the ranges and step sizes of values for each knob. We profile the multi-knob tracking-by-detection scheme and evaluate the accuracy-latency relation on each knob. We then determine the ranges and step sizes according to the monotonic range of such relation and the constraints of each knob. Finally, we implement our MBODF on top of Faster R-CNN (a 368-branch and a 3,942-branch variant), EfficientDet, YOLOv3, and SSD (Table 6.2). Note that we allow two object detector variants (dv)—EfficientDet D0 and D3, for ED+MB, and allow four object trackers—MedianFlow [153], KCF [161], CSRT [162], and dense Optical Flow [166], for YL+MB.

Snippet-granularity Dataset. We derive a snippet-granularity dataset to study the content-aware accuracy of the execution branches. Given a video dataset $\{v_1, v_2, ...v_h\}$ with h videos, we clip each video into l-frame video snippets, and each video snippet is our unit for evaluating content-specific accuracy. Too small an l value makes mAP meaningless, and too large an l, reduces the content-aware granularity. We choose $l = 100$ for the ILSVRC 2015 VID dataset. To further enlarge the training dataset, we use sliding windows to extract more video snippets. Supposing a temporal stride of s frames, every l-frame snippet starting at the frame whose index is the multiple of s is selected as a video snippet (we use $s = 5$), enlarging the training dataset by a factor of l/s.

Table 6.2. Choices of the tuning knobs in the MBODF with Faster R-CNN, EfficientDet (ED), YOLOv3 (YL), and SSD object detectors (* indicates additional choices in the 3,942-branch variant, $^+$ indicates that it can only support the MedianFlow object tracker). Notations are: *d*i for the detector interval, *dv* for the variant of the detector, *tv* for the variant of the tracker, *rd* for the input resolution of the detector, *rt* for the input resolution of the tracker, *ct* for the confidence threshold of objects to be tracked.

(a) FR+MB

di	***rd***	***nprop***	***rt***	***ct***
1,2,4,8 20,50,100*	224*,352,384,288,320, 416*,448*,480*,512*	3*,5*,10*,20* 100, 1000	25%,50% 100%	0.05,0.1 0.2,0.4*

(b) ED+MB

di	***dv***	***rt***	***ct***
1,2,4,8,20,100	D0,D3	100%,50%,25%	0.15,0.3

(c) YL+MB

di	***rd***	***rt***	***tv***
1,2,4,8, 20,50, 100	224,256,288,320, 352,384,416,448 480,512,544,576	100%$^+$, 50%$^+$, 25%	MedianFlow, KCF, CSRT, Optical Flow

(d) SSD+MB

di	***rd***	***rt***	***ct***
1,2,4,8,20,100	192,256,320	100%,50%,25%	0.15,0.3

Training content-aware scheduler model. We train our content-aware accuracy predictors for 400 epochs, with a batch size of 64, a weight decay of 0.01, and an SGD optimizer of fixed learning rate of 0.01, and momentum of 0.9.

6.5 Experiments

Our experimental results are composed of three parts. *First*, we evaluate our best performing models over multiple backbone object detectors and compare with the *content-agnostic baselines*. *Second*, we perform ablation studies of our techniques over the MBODF with Faster R-CNN (FR+MB+CAS) and FastAdapt (FastAdapt+CAS) protocols and study the impact of *content-aware techniques*. *Finally*, we discuss the benefit of post-processing methods on the accuracy and latency cost of both the offline profiling and the online scheduler. We report results on the ILSVRC 2015 VID dataset and a snippet-granularity derivative of the dataset (only Table 6.4), and use different latency constraints to demonstrate the strength of our method. We achieve 70% mAP accuracy at 20 FPS and lead the accuracy frontier at a wide range of latency constraints. Before we present our results, we summarize our evaluation scenario, dataset and metrics, and naming convention for the protocols.

Streaming Inference: As we study the efficient and adaptive object detection systems on mobiles, the typical usage scenario is to process the videos at the speed of their source, *i.e.* 30 FPS, in the streaming style. This means (1) one cannot use the raw video frame or features of video frames in the future to refine the detection results on the current frame, (2) one cannot refine the detection results of past frames, and (3) the algorithm should process the video frame-by-frame in the timestamp order. We discuss the comparison with other protocols in the offline mode with post-processing techniques in Section 6.5.3.

Dataset and Metrics: We use ILSVRC 2015 VID dataset for our evaluation. Particularly, we train our feature extractors and accuracy predictors on our snippet-granularity dataset derived from the ILSVRC 2015 VID training dataset, which contains 3,862 videos. Our snippet-granularity dataset of 1,256 video snippets is derived from 10% videos in the training dataset, considering the significant amount of execution branches in our MBODF. We evaluate our models on both ILSVRC 2015 VID validation dataset and our snippet-granularity dataset. The former contains 555 videos, and we evaluate object detection performance by reporting (1) mean Average Precision (mAP) at IoU=0.5 as the accuracy metric and (2) mean execution latency per frame on the NVIDIA Jetson TX2 as the latency metric. The latter has 1,965 video snippets. Here we evaluate our accuracy prediction results, and report Mean Squared Error (MSE), Spearman Rank Correlation (SRC), and Recall of the most accurate branches between the predicted accuracy and the ground truth accuracy.

Protocols. We formulate several protocols that implement a set of techniques for efficient object detection on videos. We replicate the SOTA object detection models and create MBODF for each object detection model by designing tuning knobs and determining ranges and step sizes for each knob (Section 6.4). The variants of SmartAdapt (anything with "MB" or "CAS" in the name) and baselines are as follows:

1. **FR+MB** Our MBODF on top of the Faster R-CNN [57] object detector with ResNet-50 [53] and FPN [248]. We have a 368-branch and a 3,942-branch variant due to the different ranges and step sizes in each knob.

2. **ED+MB**: Our MBODF on EfficientDet [139].

3. **YL+MB**: Our MBODF on YOLOv3 [138].

4. **SSD+MB**: Our MBODF on SSD [55].

5. **FastAdapt** [**233**]: An adaptive object detection system with 1,036 approximation branches and a content-agnostic scheduler.

6. **ApproxDet** [**118**]: Another adaptive object detection system, but less efficient than FastAdapt.

7. **FR+MB+CAS**: Our content-aware scheduler with our MBODF on top of Faster R-CNN.

8. **FastAdapt+CAS**: Our content-aware scheduler with an off-the-shelf adaptive object detection system.

9. **AdaScale** [**158**]: an adaptive and efficient video object detection model with a scale knob. We evaluate a multi-scale (MS) variant as its main design, and include several single scales (SS) for comparison.

10. **Skip-Conv ED D0** [**238**]: We use the norm-gate variant of Skip-Conv on top of an EfficientDet D0 model. The original implementation only shows MAC and wall time reduction on *CPUs*. We evaluate Skip-Conv on the mobile GPU to compare with SmartAdapt.

11. **MEGA RN101** [**189**]: ResNet 101 version of MEGA. In our streaming inference scenario, we cannot get access to the frames or features in the future or refine the detection in the past. Thus, we report the accuracy of the still-image object detection baseline in MEGA. This applies to SELSA RN101 and REPP YOLOv3 as well.

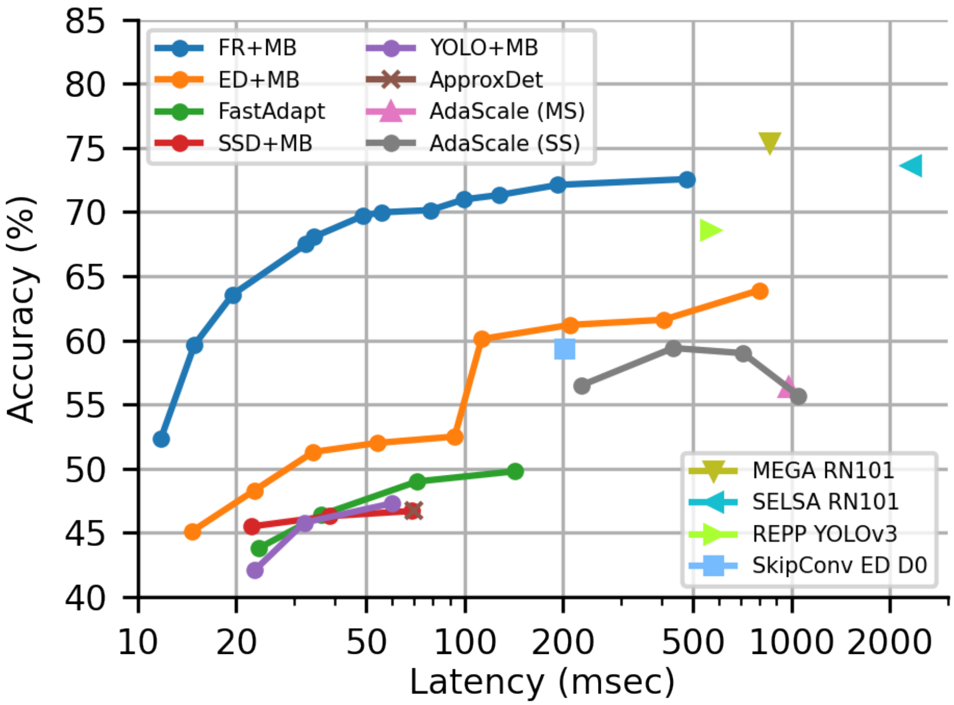


Figure 6.6. Accuracy-latency frontier. Our MBODF on top of Faster R-CNN (FR+MB) achieves higher accuracy at a wide latency range (2 - 85 FPS).

12. **SELSA RN101** [191]: ResNet-101 version of SELSA.

13. **REPP YOLOv3** [192]: YOLOv3 version of REPP.

6.5.1 Performance of MBODFs

Figure 6.6 shows the accuracy and latency performance of each protocol, in which the latency scale is logarithmic to include a large variety of protocols. We can observe that our FR+MB protocol leads the accuracy-latency frontiers compared to baselines and other MBODFs in our work. Particularly, FR+MB achieves 67.5% mAP at 30 FPS, 69.7% mAP at 20 FPS, 71.0% mAP at 10 FPS on the TX2. The adaptation range is 40.5x in latency (9.8x within 3% accuracy reduction) and the accuracy is superior to all other protocols given the same latency constraint. On the other hand, our ED+MB, YL+MB, and SSD+MB also enhance the efficiency to achieve the real-time inferencing speed (30 FPS). As for baseline protocols, MEGA and SELSA, with their deeper ResNet 101 kernel, they are 2.9% and

Table 6.3. Accuracy comparison of SmartAdapt over all efficient baselines given stringent latency constraints on the ILSVRC VID validation dataset. The object detectors FR, ED, SSD, and YOLO cannot meet the 100 msec latency constraint with a MBODF and thus not shown. N/A means that the accuracy is unusably low.

Protocols	20.0 ms	33.3 ms	50.0 ms	100.0 ms
FR+MB+Oracle (3,942 br.)	71.5%	75.8%	76.3%	77.6%
FR+MB+Oracle (368 br.)	67.1%	72.1%	72.9%	74.8%
FR+MB+CAS	**64.1%**	**68.3%**	**69.8%**	**71.1%**
FR+MB	63.6%	67.5%	69.7%	71.0%
FastAdapt+CAS	N/A	**46.1%**	**47.1%**	**50.3%**
FastAdapt	N/A	43.8%	46.4%	49.0%
ED+MB	45.1%	51.3%	52.0%	52.5%
SSD+MB	N/A	45.5%	46.3%	46.7%
YL+MB	N/A	42.1%	45.8%	47.3%
ApproxDet	N/A	N/A	N/A	46.8%

1.1% more accurate than our most accurate branch in FR+MB and much slower than us (running at 1.2 and 0.4 FPS). REPP, SkipConv, AdaScale, FastAdapt and ApproxDet are both worse than our FR+MB protocol with lower accuracy and higher latency. To conclude, our MBODFs on top of four popular object detectors can greatly enhance the efficiency to achieve real-time speed and the best of them, FR+MB, leads the accuracy-latency frontier and has comparable accuracy with the accuracy optimized models.

We then look into all adaptive and efficient protocols that are able to run within 100 msec per frame (10 FPS speed) and examine the accuracy at 50, 30, 20, and 10 FPS in Table 6.3. The results show that FR+MB+CAS achieves marginally better accuracy results than FR+MB by 0.1% to 0.8% mAP through its content-aware scheduler. Compared to the FastAdapt baseline, our content-aware scheduler achieves a higher benefit, 0.7% to 2.3% mAP improvement. Note that our CAS results are still far from our Oracle results because (1) we cannot exhaustively run every branch and select the most accurate one, (2) we cannot access annotations online to calculate the ground truth accuracy, and (3) we cannot access future frames and the scheduler's choice is based on the features of current and past frames.

Table 6.4. Evaluation of our content-aware MBODF on top of Faster R-CNN object detector with different content extractors against the content-agnostic MBODF (baseline) on the snippet-level dataset. N/A means the training cannot finish in a reasonable time.

metrics	MSE		SRC		Recall	
features	368 br.	3,942 br.	368 br.	3,942 br.	368 br.	3,942 br.
baseline	0.091	0.109	0.377	0.376	0.354	0.343
light	0.083	0.109	0.385	**0.385**	0.368	0.347
HoC	0.083	0.109	**0.387**	**0.385**	**0.369**	**0.348**
HOG	0.084	0.103	0.386	0.384	0.347	**0.348**
MobileNet	**0.082**	**0.102**	0.385	**0.385**	0.368	0.347
MobileNet Tr.	0.083	N/A	0.385	N/A	0.361	N/A

Table 6.5. Ablation study for the components in the CAS, over a FastAdapt baseline (content-agnostic) on the ILSVRC 2015 VID validation dataset, at a real-time 33.3 msec latency constraint (30 frames/sec video quality). * denotes the CAS with the jointly trainable feature extractor and the content-aware accuracy predictor. $^+$ denotes the CAS with joint modeling of content and latency requirement.

	light	HoC	HOG	Resnet50	CPoP	MobileNet	MobileNet*	MobileNet$^+$
Content-agnostic	43.8%							
All br.	44.3%	44.4%	44.3%	44.4%	44.8%	44.3%	44.0%	45.7%
200 br.	43.8%	44.1%	44.3%	44.1%	44.1%	43.8%	45.2%	**46.1%**

To summarize, in addition to the illuminating results in Figure 6.6, our exploration on the content-aware design boosts the accuracy-latency frontier further.

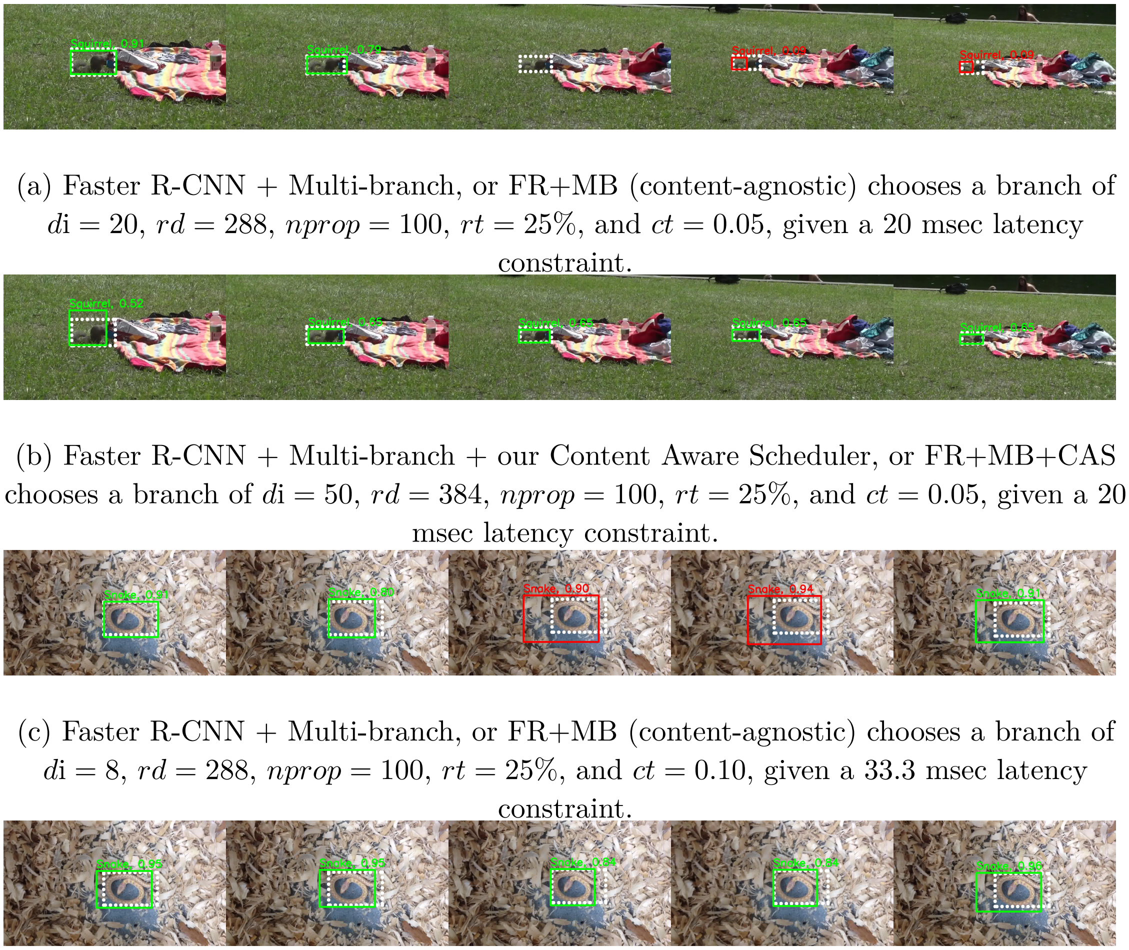
We also evaluate the CAS with different feature extractors. On the snippet-level dataset, Table 6.4 shows the MSE, SRC, and recall of our full stack of techniques with different off-the-shelf and trainable feature extractors, on top of a 368-branch and a 3,942-branch FR+MB. The results show consistent lower MSE, higher SRC, and recall in the CAS of all feature extractors compared to the content-agnostic baseline.

We further study FastAdapt as a baseline on how a content-aware scheduler (FastAdapt+CAS) improves such off-the-shelf adaptive object detection systems. Table 6.5 shows light content features, though coming with only 4 values, are better than the content-agnostic baseline by 0.5% mAP. However, directly applying off-the-shelf content features does not always yield accuracy improvements—when we train accuracy predictors on all branches, HoC, HOG, ResNet-50, MobileNet, and even trainable MobileNet's results are all neutral or negative, compared to the light features' results. Only the scheduler with the CPoP feature extractor achieves 0.5% mAP higher results than that with the light features, owing to these features extracted from the last layers of the object detector. Next, we narrow down the number of candidate branches to 200 (from 1,036 available ones), according to the scheme described in Section 6.3.4. In brief, we consider a subset of branches, top-K, which covers a large fraction of the optimal branches. With this reduced number of branches, we observe that a better performing setting is FastAdapt+CAS with the jointly trainable MobileNet feature extractor and accuracy predictor, with 1.4% higher mAP than the baseline. Reducing the number of branches can be considered as imposing a regularizer that drives the model away from noisy branches, *i.e.* those which just happen to be optimal for one (or a few) videos.

Finally, we further add the joint modeling of both content and latency requirements and reach an mAP of 46.1%, which is 2.3% mAP higher than the content-agnostic baseline. To summarize, it is hard to achieve an accuracy gain by simply plugging in an off-the-shelf feature extractor. Further, a smaller subset of branches using our technique can make the entire model easier to train and converge. This subsetting must be done carefully, ensuring (with high likelihood) that branches that appear on the Pareto optimal curve are not left out. A MobileNet feature extractor with the joint modeling of both content and latency requirements yields the best content-aware results. This aspect of joint training of the feature extractor gives another dimension of performance improvement of content-aware models on top of the content-agnostic performance (FastAdapt) (Figure 6.6).

6.5.2 Visualization

In Figure 6.7(a) and (b), we show a visualization to demonstrate the benefits of CAS over the MBODF with Faster R-CNN (FR+MB). In this example, the latency constraint is strict, only 20 msec per frame, and the scheduler has to either choose a larger di (less frequent invocation of the detector and correspondingly, more frames on the object tracker), a smaller resolution of the object detector rd, or more efficient choices on other knobs. The



(a) Faster R-CNN + Multi-branch, or FR+MB (content-agnostic) chooses a branch of $d\mathrm{i} = 20$, $rd = 288$, $nprop = 100$, $rt = 25\%$, and $ct = 0.05$, given a 20 msec latency constraint.

(b) Faster R-CNN + Multi-branch + our Content Aware Scheduler, or FR+MB+CAS chooses a branch of $d\mathrm{i} = 50$, $rd = 384$, $nprop = 100$, $rt = 25\%$, and $ct = 0.05$, given a 20 msec latency constraint.

(c) Faster R-CNN + Multi-branch, or FR+MB (content-agnostic) chooses a branch of $d\mathrm{i} = 8$, $rd = 288$, $nprop = 100$, $rt = 25\%$, and $ct = 0.10$, given a 33.3 msec latency constraint.

(d) Faster R-CNN + Multi-branch + our Content Aware Scheduler, or FR+MB+CAS chooses a branch of $d\mathrm{i} = 20$, $rd = 384$, $nprop = 100$, $rt = 25\%$, and $ct = 0.05$, given a 33.3 msec latency constraint.

Figure 6.7. Qualitative results on comparing the content-agnostic FR+MB and content-aware FR+MB+CAS on two videos, one with a small squirrel, and the second, with a still snake. The first frame out of every 10 frames is visualized. The white boxes (dashed lines) show the ground truth annotations, the red boxes denote false positive boxes, and the green boxes denote true positive boxes. Predicted class labels and confidence scores are displayed on top of the detection boxes.

content-agnostic protocol FR+MB chooses a smaller resolution of the object detector to meet the latency constraint. This branch ($d\mathrm{i} = 20$, $rd = 288$, $nprop = 100$, $rt = 25\%$, and $ct = 0.05$) results in missing detections and wrong localizations for this video due to the small size of the object (small squirrel). In contrast, FR+MB+CAS, being content-

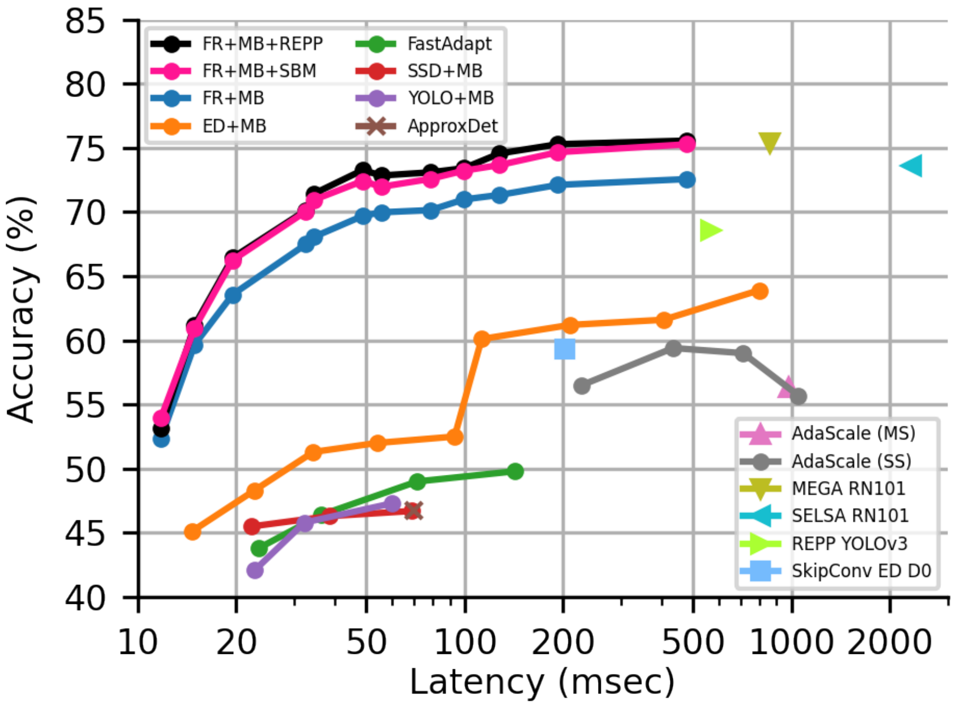


Figure 6.8. Accuracy-latency frontiers with post-processing techniques in the offline mode.

aware, smartly chooses a branch with a larger di and a larger rd. Such a branch allows precise calibrations on the frames with the object detector since the input resolution is higher ($rd = 384$) and maintains correct detections with the object tracker over a longer GoF. We show another example in Figure 6.7(c) and (d) with a video of a still snake. The snake, with its camouflage, is localized wrongly by the FR+MB baselines in the third and the fourth frames due to the small resolution of the object detector ($rd = 288$). In contrast, FR+MB+CAS, being content-aware, detects the snake correctly on all frames with a larger di and a larger rd. To summarize, given the MBODF with thousands of execution branches, CAS is able to choose a more accurate branch that is best adapted to the content and subject to the latency constraint.

Table 6.6. Accuracy comparison of FR+MB and FR+MB+CAS without post-processing techniques, with SBM, and with REPP post-processing techniques, given stringent latency constraints on the ILSVRC VID validation dataset.

Protocols	20.0 ms	33.3 ms	50 ms	100 ms
FR+MB+CAS+REPP	**66.8%**	**70.8%**	**73.4%**	**74.1%**
FR+MB+CAS+SBM	66.7%	70.7%	72.5%	73.8%
FR+MB+REPP	66.4%	70.1%	73.3%	73.4%
FR+MB+SBM	66.2%	70.0%	72.4%	73.2%
FR+MB+CAS	64.1%	68.3%	69.8%	71.1%
FR+MB	63.6%	67.5%	69.7%	71.0%

6.5.3 Further Discussions

Impact of Post-processing & Accuracy in Offline Mode. To fairly compare SmartAdapt with the accuracy optimized models, we apply REPP [192] and Seq-BBox Matching (SBM) [249] post-processing methods to our detection results in the offline mode. The averaged mAP improvement over FR+MB is 2.60% and 2.38%. While the latency cost of these processing techniques is heavily dependent on the number of objects in a given video, the overall averaged latency cost per frame is 24 msec for REPP and 9 msec for SBM.

The results of the accuracy-latency Pareto frontier branches are shown in Figure 6.8 (FR+MB+REPP and FR+MB+SBM). We observe that the post-processing techniques yield a larger accuracy improvement for the branches with higher latency. This is because those branches perform much more object detection than object tracking (di is smaller) and the detection results from the object detector can benefit more than those from the object tracker due to the nature of the re-scoring techniques in REPP and SBM.

To study how the post-processing can benefit FR+MB+CAS in the offline mode, we compare the accuracy in Table 6.6 at stringent latency constraints. The results show that both REPP and SBM can benefit FR+MB+CAS, which is our best performing protocol. Further, REPP is slightly better than SBM and makes FR+MB+CAS+REPP the best performing protocol with post-processing techniques in the offline mode.

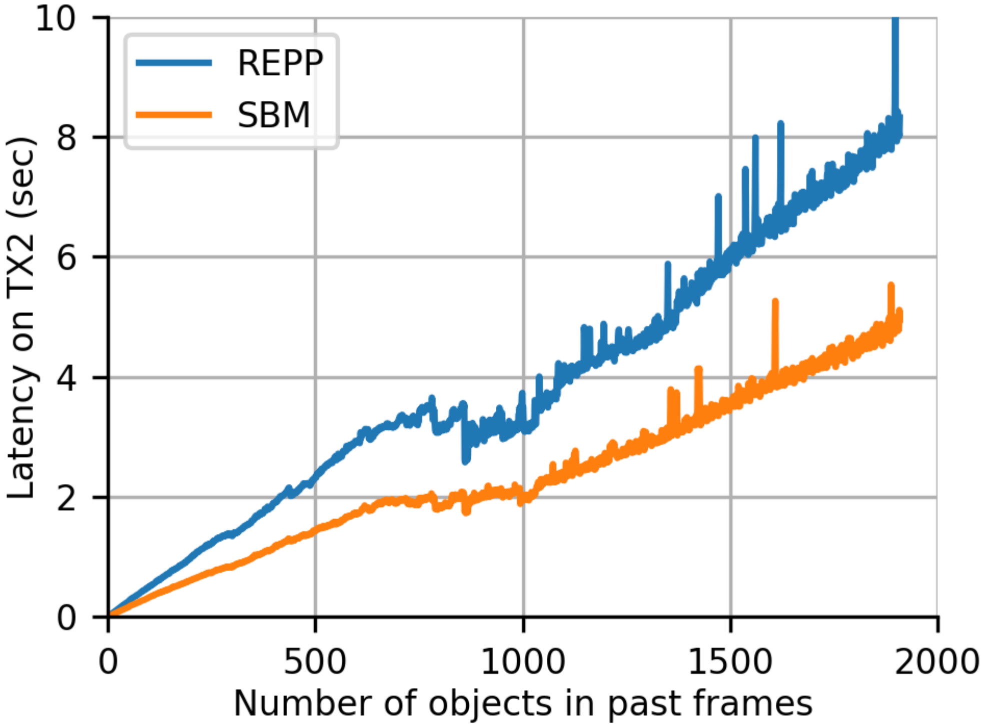


Figure 6.9. Latency of REPP and SBM in the online mode by the number of objects to post-process in the past frames, measured on the NVIDIA TX2 embedded board.

Furthermore, we have evaluated these post-processing methods in online mode. We empirically find the latency cost of REPP and SBM is significant larger in the online mode—277X and 385X larger than those in the offline mode, averaged on the entire dataset and measured on the TX2 board. Figure 6.9 uncovers the root cause of such large latency cost by showing the latency per frame vs. the number of objects in past frames. The basic version of post-processing considers all prior frames when doing post processing of the last frame. Hence, when considering later frames of a video, the total number of objects that need to be considered is cumulative and grows. We observe that as the number of objects accumulates in the later frames of a video, the latency to post-process the results increases linearly and reaches 8 seconds per frame when post-processing 1,900 objects in the past frames. Moreover, running post-processing for *every* frame in the online mode instead of doing once for the entire video (as is done in the offline mode) is another cause of such high latency cost. On the other hand, we also find that the accuracy improvement of the post-processing techniques diminishes in the online mode as the technique has to operate in streaming mode, *i.e.* it cannot go back and refine the detection results of the past frames and it cannot use the detection results from the future frames. For example, the accuracy is 0.12%-0.61% mAP *lower* than that without post-processing technique in the SBM case. Thus, we conclude that post-processing iteratively for every frame, even for every N frames, is not acceptable for our model in the online mode due to the significant latency cost and diminishing accuracy improvements.

Offline Profiling Cost. The cost of profiling MBODF to realize an Oracle scheduler and derive a snippet-granularity dataset to study content-aware accuracy is significant. Considering the 3,942-branch MBODF on top of the FRCNN object detector, in the basic case, we need to run every branch on the training and test datasets to collect its latency and accuracy. We deploy a set of engineering techniques to reduce the cost—(1) accuracy and latency profiling can be done separately, the former on the server, and the latter, on the embedded device but on a smaller set of videos since the latency does not vary significantly across videos for a given branch, (2) we profile the detector-only branches first and reuse the object detection results for other execution branches with di > 1, (3) the profiling of multiple execution branches can trivially be done in parallel and distributed in multiple servers. Combining these techniques, we are able to finish the profiling within 5 days on two servers (specs in Section 6.4).

Overhead of the Content-aware Scheduler (CAS). Though the CAS can further improve accuracy-latency frontier of the MBODF, we also evaluate the latency overhead of it because a naïve design will result in additional overhead of the scheduler on top of the latency of MBODF. Figure 6.10 shows the latency breakdown in the CAS. The cost of light feature is zero, and the cost of ResNet50 and CPoP feature extractors are minor, since ResNet50 and CPoP features come from the object detector itself. The costs of the HoC and HOG features are intermediate, between 20 to 35 msec per run, adding a minor overhead considering its triggering frequency is between once every 8 to 50 frames. The cost of a MobileNet features, whether trainable or not, is around 65 msec per run.

Figure 6.11 further evaluates FastAdapt+CAS with execution runs at 33.3 msec latency constraint. The results show the latency of the execution kernel is almost the same and

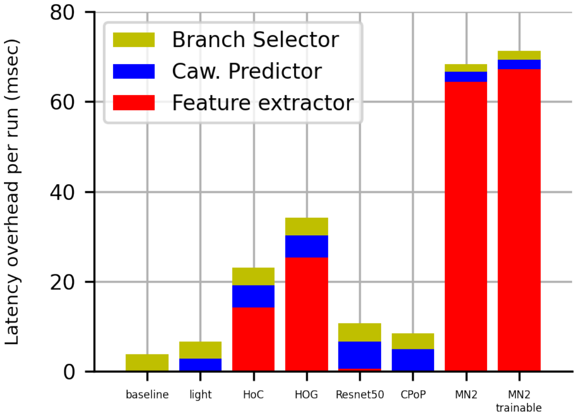


Figure 6.10. Latency breakdown in the CAS, per-run, measured on the TX2 board. Note that the CAS does not run on every frame. Thus higher cost is acceptable.

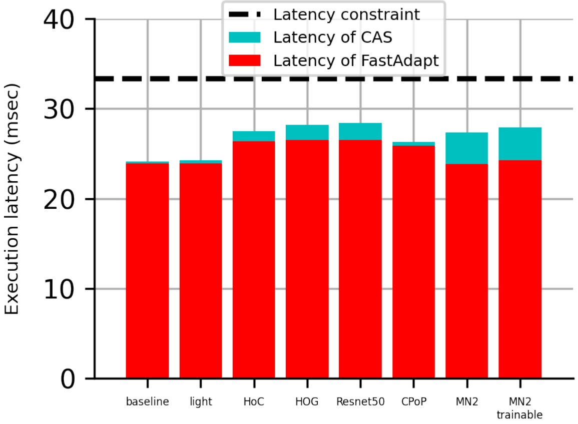


Figure 6.11. Latency of the content-aware scheduler averaged along the video, on top of the latency of the execution kernel of FastAdapt.

summed latency meets the latency budget for all feature extractors (including the relatively expensive MobileNet), owing to a conservative branch selection strategy where the branch selector uses 95th percentile latency as the criteria to select branch. Furthermore, we find that the latency cost of MobileNet can be reduced by 20% using a smaller input resolution of 64x64x3, with less than 0.1% change on the output features—one of many optimizations that we can leverage to further reduce the cost. We also find that changing the floating point precision to 16 bits does not reduce the latency—an interesting lesson.

6.6 Conclusion

We have demonstrated in a multi-branch (video) object detection framework (MBODF) how to expose the right set of execution branches and then how to schedule the optimal one at inference time. We uncovered the importance of making a *content-aware* decision on the execution branch to run. Finally, we explored a content-aware scheduler SmartAdapt, an Oracle one, and then a practical one, which uses various light-weight feature extractors, to adapt at runtime to the content. We demonstrated that our method, notwithstanding its simplicity, can adapt to a wide range of latency requirements (range of 40X), on a mobile GPU device, NVIDIA Jetson TX2. SmartAdapt, integrated with Faster R-CNN as the MBODF, leads the Pareto optimal accuracy-vs-latency frontier over 7 baselines. SmartAdapt outperforms a content-agnostic MBODF baseline, FastAdapt, by 20.9%–23.6% mAP. Our work can benefit from ongoing work in better feature extractors (to be used in our content-aware scheduler), video object detection and tracking models, and post-processing algorithms.

7. CONCLUSION

Designing approximate streaming video analytic systems is challenging. Our journey starts from the video processing pipelines. VideoChef in Chapter 2 proposes to leverage the concept of canary inputs, an accurate error mapping model, and a directed search scheme to meet real-time processing speed with the guaranteed accuracy performance. We then look into a more challenging video object classification task. ApproxNet in Chapter 3 proposes a single-model approximation-enabled DNN and a set of integrated techniques in the system design to satisfy different requirements. Furthermore, ApproxDet advances the application domain to the video object detection task. This work proposes the joint design of the multi-branch object detection kernel and the content and contention aware scheduler to address the challenging problems in this area. Finally, LiteReconfig focuses more on the cost and benefit of being content-awareness to further strengthen the system design of the approximate video object detection systems. On the other hand, SmartAdapt looks into the design of multi-branch object detection framework and content features to further improve the accuracy-latency frontiers.

In short, this dissertation focuses on an important problem–approximation for streaming video analytics on mobile devices, proposes a set of integrated techniques with systems design to solve these problems, and evaluates on three popular streaming video analytic systems with the superior performance to the state-of-the-art work.

REFERENCES

[1] S. Fouladi, R. S. Wahby, B. Shacklett, *et al.*, "Encoding, fast and slow: Low-latency video processing using thousands of tiny threads.," in *NSDI*, 2017, pp. 363–376.

[2] H. Zhang, G. Ananthanarayanan, P. Bodik, M. Philipose, P. Bahl, and M. J. Freedman, "Live video analytics at scale with approximation and delay-tolerance.," in *NSDI*, 2017, pp. 377–392.

[3] T. Zhang, A. Chowdhery, P. V. Bahl, K. Jamieson, and S. Banerjee, "The design and implementation of a wireless video surveillance system," in *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, ACM, 2015, pp. 426–438.

[4] E. Union, *General data protection regulation*, http://www.eugdpr.org/, 2017.

[5] M. Buckler, S. Jayasuriya, and A. Sampson, "Reconfiguring the imaging pipeline for computer vision," in *The IEEE International Conference on Computer Vision (ICCV)*, 2017.

[6] K. Uesaka and M. Kawamata, "Evolutionary synthesis of digital filter structures using genetic programming," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 50, no. 12, pp. 977–983, 2003.

[7] K. Sharman, A. Alcazar, and Y. Li, "Evolving signal processing algorithms by genetic programming," in *Genetic Algorithms in Engineering Systems: Innovations and Applications, 1995. GALESIA. First International Conference on (Conf. Publ. No. 414)*, IET, 1995, pp. 473–480.

[8] Z. Farbman, R. Fattal, and D. Lischinski, "Convolution pyramids.," *ACM Trans. Graph.*, vol. 30, no. 6, pp. 175–1, 2011.

[9] L. Lou, P. Nguyen, J. Lawrence, and C. Barnes, "Image perforation: Automatically accelerating image pipelines by intelligently skipping samples," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 5, p. 153, 2016.

[10] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman, "Enerj: Approximate data types for safe and general low-power computation," in *PLDI*, 2011.

[11] S. Misailovic, S. Sidiroglou, H. Hoffmann, and M. Rinard, "Quality of service profiling," in *ICSE*, 2010.

[12] S. Sidiroglou-Douskos, S. Misailovic, H. Hoffmann, and M. Rinard, "Managing performance vs. accuracy trade-offs with loop perforation," in *FSE*, 2011.

[13] S. Chaudhuri, S. Gulwani, R. Lublinerman, and S. Navidpour, "Proving programs robust," in *FSE*, 2011.

[14] M. Samadi, D. A. Jamshidi, J. Lee, and S. Mahlke, "Paraprox: Pattern-based approximation for data parallel applications," in *ASPLOS*, 2014.

[15] K. Mahadik, S. Chaterji, B. Zhou, M. Kulkarni, and S. Bagchi, "Orion: Scaling genomic sequence matching with fine-grained parallelization," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, IEEE Press, 2014, pp. 449–460.

[16] W. Baek and T. M. Chilimbi, "Green: A framework for supporting energy-conscious programming using controlled approximation," in *PLDI*, 2010.

[17] Í. Goiri, R. Bianchini, S. Nagarakatte, and T. D. Nguyen, "Approxhadoop: Bringing approximations to mapreduce frameworks," in *ASPLOS*, 2015.

[18] H. Hoffmann, S. Sidiroglou, M. Carbin, S. Misailovic, A. Agarwal, and M. Rinard, "Dynamic knobs for responsive power-aware computing," in *ASPLOS*, 2011.

[19] H. Hoffmann, "Jouleguard: Energy guarantees for approximate applications," in *SOSP*, 2015.

[20] S. Mitra, M. K. Gupta, S. Misailovic, and S. Bagchi, "Phase-aware optimization in approximate computing," in *Proceedings of the 2017 International Symposium on Code Generation and Optimization (CGO)*, IEEE Press, 2017, pp. 185–196.

[21] K. V. Palem, "Energy aware computing through probabilistic switching: A study of limits," *IEEE Transactions on Computers*, vol. 54, no. 9, pp. 1123–1137, 2005.

[22] J. Meng, S. Chakradhar, and A. Raghunathan, "Best-effort parallel execution framework for recognition and mining applications," in *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, IEEE, 2009, pp. 1–12.

[23] M. Samadi, J. Lee, D. A. Jamshidi, A. Hormati, and S. Mahlke, "Sage: Self-tuning approximation for graphics engines," in *MICRO*, 2013.

[24] V. K. Chippa, D. Mohapatra, A. Raghunathan, K. Roy, and S. T. Chakradhar, "Scalable effort hardware design: Exploiting algorithmic resilience for energy efficiency," in *DAC*, 2010.

[25] X. Sui, A. Lenharth, D. S. Fussell, and K. Pingali, "Proactive control of approximate programs," in *ASPLOS*, 2016.

[26] M. A. Laurenzano, P. Hill, M. Samadi, S. Mahlke, J. Mars, and L. Tang, “Input responsiveness: Using canary inputs to dynamically steer approximation,” in *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 2016, pp. 161–176.

[27] L. Bao, Q. Yang, and H. Jin, “Fast edge-preserving patchmatch for large displacement optical flow,” in *IEEE CVPR*, 2014, pp. 3534–3541.

[28] E. R. Dougherty and R. A. Lotufo, *Hands-on morphological image processing*. SPIE press, 2003, vol. 59.

[29] H. Jiang, A. (Helal, A. K. Elmagarmid, and A. Joshi, “Scene change detection techniques for video database systems,” *Multimedia Systems*, vol. 6, no. 3, pp. 186–195, May 1998.

[30] B. Juurlink, M. Alvarez-Mesa, C. C. Chi, A. Azevedo, C. Meenderinck, and A. Ramirez, “Understanding the application: An overview of the h. 264 standard,” *Scalable Parallel Programming Applied to H. 264/AVC Decoding*, pp. 5–15, 2012.

[31] S. G. Kim, M. Harwani, A. Grama, and S. Chaterji, “Ep-dnn: A deep neural network-based global enhancer prediction algorithm,” *Scientific reports*, vol. 6, p. 38 433, 2016.

[32] K. Mahadik, C. Wright, J. Zhang, M. Kulkarni, S. Bagchi, and S. Chaterji, “Sarvavid: A domain specific language for developing scalable computational genomics applications,” in *Proceedings of the 2016 International Conference on Supercomputing*, ACM, 2016, p. 34.

[33] J. Ko, *Libvideo: A lightweight .net library to download youtube videos*, https://github.com/jamesqo/libvideo, 2016.

[34] S. T. Welstead, *Fractal and Wavelet Image Compression Techniques*, 1st. Bellingham, WA, USA: Society of Photo-Optical Instrumentation Engineers (SPIE), 1999, ISBN: 0819435031.

[35] R. Hamzaoui and D. Saupe, *Fractal image compression - in ”Document and Image Compression”*. CRC Press, 2006.

[36] N. Thomos, N. V. Boulgouris, and M. G. Strintzis, “Optimized transmission of jpeg2000 streams over wireless channels,” *IEEE Transactions on image processing*, vol. 15, no. 1, pp. 54–67, 2006.

[37] S. Venkataramani, V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, “Quality programmable vector processors for approximate computing,” in *MICRO*, 2013.

[38] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, “Neural acceleration for general-purpose approximate programs,” in *MICRO*, IEEE Computer Society, 2012.

[39] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Architecture support for disciplined approximate programming," in *ASPLOS*, 2012.

[40] J. Ansel, Y. L. Wong, C. Chan, M. Olszewski, A. Edelman, and S. Amarasinghe, "Language and compiler support for auto-tuning variable-accuracy algorithms," in *International Symposium on Code Generation and Optimization (CGO 2011)*, IEEE, 2011, pp. 85–96.

[41] M. Carbin, S. Misailovic, and M. C. Rinard, "Verifying quantitative reliability for programs that execute on unreliable hardware," in *OOPSLA*, 2013.

[42] A. Sampson, A. Baixo, B. Ransford, *et al.*, "Accept: A programmer-guided compiler framework for practical approximate computing," *University of Washington Technical Report UW-CSE-15-01*, 2015.

[43] D. S. Khudia, B. Zamirai, M. Samadi, and S. Mahlke, "Rumba: An online quality management system for approximate computing," in *ISCA*, 2015.

[44] A. Ranjan, A. Raha, S. Venkataramani, K. Roy, and A. Raghunathan, "Aslan: Synthesis of approximate sequential circuits," in *DATE*, 2014.

[45] A. Sampson, J. Nelson, K. Strauss, and L. Ceze, "Approximate storage in solid-state memories," *ACM TOCS*, 2014.

[46] D. Jevdjic, K. Strauss, L. Ceze, and H. S. Malvar, "Approximate storage of compressed and encrypted videos," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, ACM, 2017, pp. 361–373.

[47] Y. Ding, J. Ansel, K. Veeramachaneni, X. Shen, U.-M. O'Reilly, and S. Amarasinghe, "Autotuning algorithmic choice for input sensitivity," in *PLDI*, 2015.

[48] M. Ringenburg, A. Sampson, I. Ackerman, L. Ceze, and D. Grossman, "Monitoring and debugging the quality of results in approximate programs," in *ASPLOS*, 2015.

[49] A. Raha, S. Venkataramani, V. Raghunathan, and A. Raghunathan, "Quality configurable reduce-and-rank for energy efficient approximate computing," in *DATE*, 2015.

[50] S. Achour and M. C. Rinard, "Approximate computation with outlier detection in topaz," in *OOPSLA*, 2015.

[51] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[52] C. Szegedy, W. Liu, Y. Jia, *et al.*, “Going deeper with convolutions,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[53] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[54] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 4700–4708.

[55] W. Liu, D. Anguelov, D. Erhan, *et al.*, “SSD: Single shot multibox detector,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, vol. 9907, 2016, pp. 21–37.

[56] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779–788.

[57] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards real-time object detection with region proposal networks,” in *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, 2015, pp. 91–99.

[58] K. Shankar, P. Wang, R. Xu, A. Mahgoub, and S. Chaterji, “JANUS: Benchmarking commercial and open-source cloud and edge platforms for object and anomaly detection workloads,” in *Proceedings of the 13th IEEE International Conference on Cloud Computing (CLOUD)*, 2020, pp. 590–599.

[59] O. M. Parkhi, A. Vedaldi, A. Zisserman, *et al.*, “Deep face recognition.,” in *BMVC*, vol. 1, 2015, p. 6.

[60] F. Schroff, D. Kalenichenko, and J. Philbin, “Facenet: A unified embedding for face recognition and clustering,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 815–823.

[61] Y. Wen, K. Zhang, Z. Li, and Y. Qiao, “A discriminative feature learning approach for deep face recognition,” in *European Conference on Computer Vision*, Springer, 2016, pp. 499–515.

[62] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, “Deepface: Closing the gap to human-level performance in face verification,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 1701–1708.

[63] S. Ji, W. Xu, M. Yang, and K. Yu, “3d convolutional neural networks for human action recognition,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 1, pp. 221–231, 2013.

[64] R. Poppe, “A survey on vision-based human action recognition,” *Image and vision computing*, vol. 28, no. 6, pp. 976–990, 2010.

[65] J. Liu, A. Shahroudy, D. Xu, and G. Wang, “Spatio-temporal lstm with trust gates for 3d human action recognition,” in *European Conference on Computer Vision*, Springer, 2016, pp. 816–833.

[66] K. Simonyan and A. Zisserman, “Two-stream convolutional networks for action recognition in videos,” in *Advances in neural information processing systems*, 2014, pp. 568–576.

[67] L. N. Huynh, Y. Lee, and R. K. Balan, “DeepMon: Mobile gpu-based deep learning framework for continuous vision applications,” in *Proceedings of the Annual International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2017, pp. 82–95.

[68] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, and A. Krishnamurthy, “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)*, 2016, pp. 123–136.

[69] A. H. Jiang, D. L.-K. Wong, C. Canel, *et al.*, “Mainstream: Dynamic stem-sharing for multi-tenant video processing,” in *Proceedings of USENIX Annual Technical Conference (USENIX ATC)*, 2018, pp. 29–42.

[70] L. Liu, H. Li, and M. Gruteser, “Edge assisted real-time object detection for mobile augmented reality,” pp. 1–16, 2019.

[71] O. Kayiran, N. C. Nachiappan, A. Jog, *et al.*, “Managing gpu concurrency in heterogeneous architectures,” in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, IEEE, 2014, pp. 114–126.

[72] R. Ausavarungnirun, V. Miller, J. Landgraf, *et al.*, “Mask: Redesigning the gpu memory hierarchy to support multi-application concurrency,” in *ACM SIGPLAN Notices*, ACM, vol. 53, 2018, pp. 503–518.

[73] S. Bagchi, T. F. Abdelzaher, R. Govindan, *et al.*, “New frontiers in iot: Networking, systems, reliability, and security challenges,” *IEEE Internet of Things Journal*, vol. 7, no. 12, pp. 11 330–11 346, 2020.

[74] A. G. Howard, M. Zhu, B. Chen, *et al.*, “MobileNets: Efficient convolutional neural networks for mobile vision applications,” *arXiv preprint arXiv:1704.04861*, 2017.

[75] G. Huang and D. Chen, “Multi-scale dense networks for resource efficient image classification,” *ICLR 2018*, 2018.

[76] S. Teerapittayanon, B. McDanel, and H. Kung, “BranchyNet: Fast inference via early exiting from deep neural networks,” in *Proceedings of IEEE International Conference on Pattern Recognition (ICPR)*, 2016, pp. 2464–2469.

[77] B. Fang, X. Zeng, and M. Zhang, “NestDNN: Resource-aware multi-tenant on-device deep learning for continuous mobile vision,” in *Proceedings of the Annual International Conference on Mobile Computing and Networking (MobiCom)*, 2018, pp. 115–127.

[78] K. He, X. Zhang, S. Ren, and J. Sun, “Spatial pyramid pooling in deep convolutional networks for visual recognition,” in *european conference on computer vision*, Springer, 2014, pp. 346–361.

[79] R. Xu, J. Koo, R. Kumar, *et al.*, “VideoChef: Efficient approximation for streaming video processing pipelines,” in *Proceedings of the USENIX Annual Technical Conference (USENIX ATC)*, 2018, pp. 43–56.

[80] J. Mars, L. Tang, R. Hundt, K. Skadron, and M. L. Soffa, “Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations,” in *Proceedings of the 44th annual IEEE/ACM International Symposium on Microarchitecture*, 2011, pp. 248–259.

[81] H. Yang, A. Breslow, J. Mars, and L. Tang, “Bubble-flux: Precise online qos management for increased utilization in warehouse scale computers,” in *International Symposium on Computer Architecture (ISCA)*, ACM, vol. 41, 2013, pp. 607–618.

[82] R. Xu, S. Mitra, J. Rahman, *et al.*, “Pythia: Improving datacenter utilization via precise contention prediction for multiple co-located workloads,” in *Proceedings of the International Middleware Conference (Middleware)*, 2018, pp. 146–160.

[83] Y. Zhang, M. A. Laurenzano, J. Mars, and L. Tang, “Smite: Precise qos prediction on real-system smt processors to improve utilization in warehouse scale computers,” in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, IEEE, 2014, pp. 406–418.

[84] C. Delimitrou and C. Kozyrakis, “Paragon: Qos-aware scheduling for heterogeneous datacenters,” *ACM SIGPLAN Notices*, vol. 48, no. 4, pp. 77–88, 2013.

[85] R. K. Panta, S. Bagchi, and S. P. Midkiff, “Efficient incremental code update for sensor networks,” *ACM Transactions on Sensor Networks (TOSN)*, vol. 7, no. 4, pp. 1–32, 2011.

[86] H. Yu and S. Winkler, “Image complexity and spatial information,” in *2013 Fifth International Workshop on Quality of Multimedia Experience (QoMEX)*, IEEE, 2013, pp. 12–17.

[87] I. Mario, M. Chacon, D. Alma, and S. Corral, “Image complexity measure: A human criterion free approach,” in *NAFIPS 2005-2005 Annual Meeting of the North American Fuzzy Information Processing Society*, IEEE, 2005, pp. 241–246.

[88] M. Cardaci, V. Di Gesù, M. Petrou, and M. E. Tabacchi, “A fuzzy approach to the evaluation of image complexity,” *Fuzzy Sets and Systems*, vol. 160, no. 10, pp. 1474–1484, 2009.

[89] B. Jähne, H. Haussecker, and P. Geissler, *Handbook of computer vision and applications.* Citeseer, 1999, vol. 2.

[90] D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and C. Kozyrakis, “Heracles: Improving resource efficiency at scale,” in *International Symposium on Computer Architecture (ISCA)*, ACM, vol. 43, 2015, pp. 450–462.

[91] N. Corporation. “Jetson tx2 module.” (2018), [Online]. Available: https://developer.nvidia.com/embedded/buy/jetson-tx2.

[92] M. Harris. “Unified memory for cuda beginners.” (2017), [Online]. Available: https://devblogs.nvidia.com/unified-memory-cuda-beginners/.

[93] O. Russakovsky, J. Deng, H. Su, *et al.*, “ImageNet large scale visual recognition challenge,” *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[94] K. Kang, H. Li, J. Yan, *et al.*, “T-CNN: Tubelets with convolutional neural networks for object detection from videos,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 28, no. 10, pp. 2896–2907, 2017.

[95] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, Ieee, 2009, pp. 248–255.

[96] Z. Lu, S. Rallapalli, K. Chan, and T. La Porta, “Modeling the resource requirements of convolutional neural networks on mobile devices,” in *Proceedings of the 25th ACM international conference on Multimedia*, ACM, 2017, pp. 1663–1671.

[97] X. Zhang, X. Zhou, M. Lin, and J. Sun, “Shufflenet: An extremely efficient convolutional neural network for mobile devices,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2018.

[98] R. J. Wang, X. Li, and C. X. Ling, "PELEE: A real-time object detection system on mobile devices," in *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, 2018, pp. 1963–1972.

[99] C. M. Power. "Sport cars drag race video." (2016), [Online]. Available: https://www.youtube.com/watch?v=Qj21A8HLQ0M.

[100] W. Han, P. Khorrami, T. L. Paine, *et al.*, "Seq-nms for video object detection," *arXiv preprint arXiv:1602.08465*, 2016.

[101] Q. V. Le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, and A. Y. Ng, "On optimization methods for deep learning," in *Proceedings of the 28th International Conference on International Conference on Machine Learning*, Omnipress, 2011, pp. 265–272.

[102] N. D. Lane, S. Bhattacharya, P. Georgiev, *et al.*, "Deepx: A software accelerator for low-power deep learning inference on mobile devices," in *Proceedings of the 15th International Conference on Information Processing in Sensor Networks*, IEEE Press, 2016, p. 23.

[103] B. Reagen, P. Whatmough, R. Adolf, *et al.*, "Minerva: Enabling low-power, highly-accurate deep neural network accelerators," in *ACM SIGARCH Computer Architecture News*, IEEE Press, vol. 44, 2016, pp. 267–278.

[104] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2017.

[105] E. Park, D. Kim, S. Kim, *et al.*, "Big/little deep neural network for ultra low power inference," in *Proceedings of the 10th International Conference on Hardware/Software Codesign and System Synthesis*, 2015, pp. 124–132.

[106] S. Han, X. Liu, H. Mao, *et al.*, "EIE: Efficient inference engine on compressed deep neural network," vol. 44, no. 3, pp. 243–254, 2016.

[107] A. Parashar, M. Rhu, A. Mukkara, *et al.*, "SCNN: An accelerator for compressed-sparse convolutional neural networks," vol. 45, no. 2, pp. 27–40, 2017.

[108] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, "TETRIS: Scalable and efficient neural network acceleration with 3D memory," *ACM SIGOPS Operating Systems Review*, vol. 51, no. 2, pp. 751–764, 2017.

[109] S. Zhang, Z. Du, L. Zhang, *et al.*, "Cambricon-X: An accelerator for sparse neural networks," in *Proceedings of the Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016, pp. 1–12.

[110] A. Mahgoub, P. Wood, A. Medoff, *et al.*, "Sophia: Online reconfiguration of clustered nosql databases for time-varying workloads," in *USENIX Annual Technical Conference (USENIX ATC)*, 2019, pp. 223–240.

[111] A. Mahgoub, A. M. Medoff, R. Kumar, *et al.*, "OptimusCloud: Heterogeneous configuration optimization for distributed databases in the cloud," in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, 2020, pp. 189–203.

[112] J. Jiang, G. Ananthanarayanan, P. Bodik, S. Sen, and I. Stoica, "Chameleon: Scalable adaptation of video analytics," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, 2018, pp. 253–266.

[113] K. Hsieh, G. Ananthanarayanan, P. Bodik, *et al.*, "Focus: Querying large video datasets with low latency and low cost," in *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, 2018, pp. 269–286.

[114] A. Mahgoub, K. Shankar, S. Mitra, A. Klimovic, S. Chaterji, and S. Bagchi, "SONIC: Application-aware data passing for chained serverless applications," in *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, 2021, pp. 1–15.

[115] A. W. S. Inc. "Aws lambda." (2018), [Online]. Available: https://aws.amazon.com/lambda/.

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

[117] R. Xu, H. Wang, S. Petrangeli, V. Swaminathan, and S. Bagchi, "Closing-the-loop: A data-driven framework for effective video summarization," in *Proceedings of the 22nd IEEE International Symposium on Multimedia (ISM)*, 2020, pp. 201–205.

[118] R. Xu, C.-l. Zhang, P. Wang, *et al.*, "ApproxDet: Content and contention-aware approximate object detection for mobiles," in *Proceedings of the 18th Conference on Embedded Networked Sensor Systems (SenSys)*, 2020, pp. 449–462.

[119] A. Ghoshal, A. Grama, S. Bagchi, and S. Chaterji, "An ensemble svm model for the accurate prediction of non-canonical microrna targets," in *Proceedings of the 6th ACM Conference on Bioinformatics, Computational Biology and Health Informatics*, 2015, pp. 403–412.

[120] T. E. Thomas, J. Koo, S. Chaterji, and S. Bagchi, "Minerva: A reinforcement learning-based technique for optimal scheduling and bottleneck detection in distributed factory operations," in *2018 10th International Conference on Communication Systems & Networks (COMSNETS)*, IEEE, 2018, pp. 129–136.

[121] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, “Quantized neural networks: Training neural networks with low precision weights and activations,” *Journal of Machine Learning Research*, vol. 18, pp. 187–1, 2017.

[122] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, “Deep learning with limited numerical precision,” in *Proceedings of the International Conference on Machine Learning (ICML)*, 2015, pp. 1737–1746.

[123] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, “DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients,” *arXiv preprint arXiv:1606.06160*, 2016.

[124] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, “XNOR-Net: ImageNet classification using binary convolutional neural networks,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, vol. 9908, 2016, pp. 525–542.

[125] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, “Exploiting linear structure within convolutional networks for efficient evaluation,” in *Advances in neural information processing systems*, 2014, pp. 1269–1277.

[126] S. Bhattacharya and N. D. Lane, “Sparsification and separation of deep learning layers for constrained resource inference on wearables,” in *Proceedings of the 14th ACM Conference on Embedded Network Sensor System (SenSys)*, 2016, pp. 176–189.

[127] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, “SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size,” in *Proceedings of International Conference on Learning Representations (ICLR)*, 2016, pp. 1–13.

[128] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen, “Compressing neural networks with the hashing trick,” in *Proceedings of the International Conference on Machine Learning (ICML)*, 2015, pp. 2285–2294.

[129] S. Han, J. Pool, J. Tran, and W. Dally, “Learning both weights and connections for efficient neural network,” in *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, 2015, pp. 1135–1143.

[130] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, “Learning structured sparsity in deep neural networks,” in *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, 2016, pp. 2074–2082.

[131] P. Panda, A. Sengupta, and K. Roy, “Conditional deep learning for energy-efficient and enhanced pattern recognition,” in *Proceedings of IEEE Conference on Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2016, pp. 475–480.

[132] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[133] A. Krizhevsky, G. Hinton, *et al.*, “Learning multiple layers of features from tiny images,” 2009.

[134] Z. Wu, T. Nagarajan, A. Kumar, *et al.*, “BlockDrop: Dynamic inference paths in residual networks,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 8817–8826.

[135] T. P. Company, *Pokémon GO*, 2020. [Online]. Available: https://pokemongolive.com/en/.

[136] T. Y.-H. Chen, L. Ravindranath, S. Deng, P. Bahl, and H. Balakrishnan, “Glimpse: Continuous, real-time object recognition on mobile devices,” in *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2015, pp. 155–168.

[137] X. Shen, A. Hertzmann, J. Jia, *et al.*, “Automatic portrait segmentation for image stylization,” in *Computer Graphics Forum*, Wiley Online Library, vol. 35, 2016, pp. 93–102.

[138] J. Redmon and A. Farhadi, “YOLOv3: An incremental improvement,” *arXiv preprint arXiv:1804.02767*, 2018.

[139] M. Tan, R. Pang, and Q. V. Le, “EfficientDet: Scalable and efficient object detection,” pp. 10 781–10 790, 2020.

[140] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding,” pp. 1–13, 2016.

[141] Z.-Q. Zhao, P. Zheng, S.-t. Xu, and X. Wu, “Object detection with deep learning: A review,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 11, pp. 3212–3232, 2019.

[142] X. Zhu, Y. Wang, J. Dai, L. Yuan, and Y. Wei, “Flow-guided feature aggregation for video object detection,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 408–417.

[143] C. Feichtenhofer, A. Pinz, and A. Zisserman, “Detect to track and track to detect,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 3038–3046.

[144] X. Zhu, J. Dai, L. Yuan, and Y. Wei, “Towards high performance video object detection,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 7210–7218.

[145] S. Bagchi, V. Aggarwal, S. Chaterji, *et al.*, "Vision paper: Grand challenges in resilience: Autonomous system resilience through design and runtime measures," *IEEE Open Journal of the Computer Society*, vol. 1, pp. 155–172, 2020.

[146] S. Chaterji, P. Naghizadeh, M. A. Alam, *et al.*, "Resilient cyberphysical systems and their application drivers: A technology roadmap," *arXiv preprint arXiv:2001.00090*, 2019.

[147] S. Chaterji, N. DeLay, J. Evans, *et al.*, "Artificial intelligence for digital agriculture at scale: Techniques, policies, and challenges," *arXiv preprint arXiv:2001.09786*, 2020.

[148] A. Developer, *Services provided by an app that require it to run in the background*, 2019. [Online]. Available: https://developer.apple.com/documentation/bundleresources/information_property_list/uibackgroundmodes.

[149] R. Bulat, *React Native: Background Task Management in iOS*, 2020. [Online]. Available: https://medium.com/@rossbulat/react-native-background-task-management-in-ios-d0f05ae53cc5.

[150] A. Developer, *Guide to background processing: Android*, 2019. [Online]. Available: https://developer.android.com/guide/background.

[151] C. Delimitrou and C. Kozyrakis, "Ibench: Quantifying interference for datacenter applications," in *2013 IEEE International Symposium on Workload Characterization (IISWC)*, IEEE, 2013, pp. 23–33.

[152] A. K. Maji, S. Mitra, B. Zhou, S. Bagchi, and A. Verma, "Mitigating interference in cloud services by middleware reconfiguration," in *Proceedings of the 15th International Middleware Conference*, ACM, 2014, pp. 277–288.

[153] Z. Kalal, K. Mikolajczyk, and J. Matas, "Forward-backward error: Automatic detection of tracking failures," in *Proceedings of the 20th International Conference on Pattern Recognition (ICPR)*, IEEE, 2010, pp. 2756–2759.

[154] A. Mahgoub, P. Wood, S. Ganesh, *et al.*, "Rafiki: A middleware for parameter tuning of nosql datastores for dynamic metagenomics workloads," in *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference*, 2017, pp. 28–40.

[155] R. Xu, R. Kumar, P. Wang, *et al.*, "ApproxNet: Content and contention-aware video analytics system for embedded clients," *ACM Transactions on Sensor Networks (TOSN)*, vol. 18, no. 1, pp. 1–27, Feb. 2022.

[156] B. Glasbergen, M. Abebe, K. Daudjee, and A. Levi, "Sentinel: Universal analysis and insight for data systems," *Proceedings of the VLDB Endowment*, vol. 13, no. 12, pp. 2720–2733, 2020.

[157] C. Li, S. Wang, H. Hoffmann, and S. Lu, “Statically inferring performance properties of software configurations,” in *Proceedings of the Fifteenth European Conference on Computer Systems*, 2020, pp. 1–16.

[158] T.-W. Chin, R. Ding, and D. Marculescu, “Adascale: Towards real-time video object detection using adaptive scaling,” *arXiv preprint arXiv:1902.02910*, 2019.

[159] J. Dai, Y. Li, K. He, and J. Sun, “R-FCN: Object detection via region-based fully convolutional networks,” in *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, 2016, pp. 379–387.

[160] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, “Focal loss for dense object detection,” in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 2980–2988.

[161] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista, “High-speed tracking with kernelized correlation filters,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 3, pp. 583–596, 2014.

[162] A. Lukežič, T. Voj’iř, L. Čehovin Zajc, J. Matas, and M. Kristan, “Discriminative correlation filter tracker with channel and spatial reliability,” *International Journal of Computer Vision*, vol. 126, pp. 671–688, 2018.

[163] C. Min, A. Montanari, A. Mathur, and F. Kawsar, “A closer look at quality-aware runtime assessment of sensing models in multi-device environments,” in *Proceedings of the 17th Conference on Embedded Networked Sensor Systems*, 2019, pp. 271–284.

[164] L. Yang, Y. Han, X. Chen, S. Song, J. Dai, and G. Huang, “Resolution adaptive networks for efficient inference,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 2369–2378.

[165] X. Zhu, Y. Xiong, J. Dai, L. Yuan, and Y. Wei, “Deep feature flow for video recognition,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 2349–2358.

[166] G. Farnebäck, “Two-frame motion estimation based on polynomial expansion,” in *Proceedings of Scandinavian Conference on Image Analysis*, 2003, pp. 363–370.

[167] T.-Y. Lin, M. Maire, S. Belongie, *et al.*, “Microsoft COCO: Common objects in context,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, Springer, 2014, pp. 740–755.

[168] J. McCalpin, “Stream: Sustainable memory bandwidth in high performance computers,” *http://www.cs.virginia.edu/stream/*, 2006.

[169] J. D. McCalpin, “Memory bandwidth and machine balance in current high performance computers,” *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter*, pp. 19–25, Dec. 1995.

[170] S. Guha, N. Mishra, G. Roy, and O. Schrijvers, “Robust random cut forest based anomaly detection on streams,” in *Proceedings of the International Conference on Machine Learning (ICML)*, 2016, pp. 2712–2721.

[171] M. Tancreti, M. S. Hossain, S. Bagchi, and V. Raghunathan, “Aveksha: A hardware-software approach for non-intrusive tracing and profiling of wireless embedded systems,” in *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, 2011, pp. 288–301.

[172] S. Che, M. Boyer, J. Meng, *et al.*, “Rodinia: A benchmark suite for heterogeneous computing,” in *2009 IEEE International Symposium on Workload Characterization (IISWC)*, Ieee, 2009, pp. 44–54.

[173] H. Shuai, Q. Liu, K. Zhang, J. Yang, and J. Deng, “Cascaded regional spatio-temporal feature-routing networks for video object detection,” *IEEE Access*, vol. 6, pp. 3096–3106, 2018.

[174] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “MobileNetv2: Inverted residuals and linear bottlenecks,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 4510–4520.

[175] M. Tan, B. Chen, R. Pang, *et al.*, “Mnasnet: Platform-aware neural architecture search for mobile,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2820–2828.

[176] A. Howard, M. Sandler, G. Chu, *et al.*, “Searching for MobileNetV3,” in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2019, pp. 1314–1324.

[177] B. Wu, X. Dai, P. Zhang, *et al.*, “FBNet: Hardware-aware efficient convnet design via differentiable neural architecture search,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 10 734–10 742.

[178] M. Tan and Q. Le, “Efficientnet: Rethinking model scaling for convolutional neural networks,” in *International Conference on Machine Learning*, PMLR, 2019, pp. 6105–6114.

[179] T. Elsken, J. H. Metzen, and F. Hutter, “Neural architecture search: A survey,” *Journal of Machine Learning Research*, vol. 20, no. 55, pp. 1–21, 2019.

[180] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, “Pruning filters for efficient ConvNets,” in *Proceedings of International Conference on Learning Representations (ICLR)*, 2017, pp. 1–13.

[181] J.-H. Luo, H. Zhang, H.-Y. Zhou, C.-W. Xie, J. Wu, and W. Lin, “ThiNet: Pruning CNN filters for a thinner net,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, no. 10, pp. 2525–2538, 2018.

[182] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang, “Network trimming: A data-driven neuron pruning approach towards efficient deep architectures,” *arXiv preprint arXiv:1607.03250*, 2016.

[183] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, “Binarized neural networks,” in *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, 2016, pp. 4107–4115.

[184] S. Zilberstein, “Using anytime algorithms in intelligent systems,” *AI magazine*, vol. 17, no. 3, pp. 73–73, 1996.

[185] A. Veit and S. Belongie, “Convolutional networks with adaptive inference graphs,” *International Journal of Computer Vision*, vol. 128, pp. 730–741, 2019.

[186] K. Apicharttrisorn, X. Ran, J. Chen, S. V. Krishnamurthy, and A. K. Roy-Chowdhury, “Frugal following: Power thrifty object detection and tracking for mobile augmented reality,” in *Proceedings of the Conference on Embedded Networked Sensor Systems (SenSys)*, 2019, pp. 96–109.

[187] K. Han, Y. Wang, Q. Tian, J. Guo, C. Xu, and C. Xu, “Ghostnet: More features from cheap operations,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1580–1589.

[188] M. Liu, M. Zhu, M. White, Y. Li, and D. Kalenichenko, “Looking fast and slow: Memory-guided mobile video object detection,” *arXiv preprint arXiv:1903.10172*, 2019.

[189] Y. Chen, Y. Cao, H. Hu, and L. Wang, “Memory enhanced global-local aggregation for video object detection,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 10 337–10 346.

[190] B. Fang, X. Zeng, F. Zhang, H. Xu, and M. Zhang, “Flexdnn: Input-adaptive on-device deep learning for efficient mobile vision,” in *Proceedings of the 5th ACM/IEEE Symposium on Edge Computing (SEC)*, 2020.

[191] H. Wu, Y. Chen, N. Wang, and Z. Zhang, "Sequence level semantics aggregation for video object detection," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 9217–9225.

[192] A. Sabater, L. Montesano, and A. C. Murillo, "Robust and efficient post-processing for video object detection," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2020, pp. 10 536–10 542.

[193] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 2961–2969.

[194] S. Zhang, L. Wen, X. Bian, Z. Lei, and S. Z. Li, "Single-shot refinement neural network for object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 4203–4212.

[195] M. Liu and M. Zhu, "Mobile video object detection with temporally-aware feature maps," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 5686–5695.

[196] C.-H. Yao, C. Fang, X. Shen, Y. Wan, and M.-H. Yang, "Video object detection via object-level temporal aggregation," in *European conference on computer vision*, Springer, 2020, pp. 160–177.

[197] K. Kang, W. Ouyang, H. Li, and X. Wang, "Object detection from video tubelets with convolutional neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 817–825.

[198] S. Wang, Y. Zhou, J. Yan, and Z. Deng, "Fully motion-aware network for video object detection," in *Proceedings of the European Conference on Computer Vision (ECCV)*, vol. 11217, 2018, pp. 542–557.

[199] K. A. Joshi and D. G. Thakore, "A survey on moving object detection and tracking in video surveillance system," *International Journal of Soft Computing and Engineering*, vol. 2, no. 3, pp. 44–48, 2012.

[200] J. Lin, W.-M. Chen, Y. Lin, J. Cohn, C. Gan, and S. Han, "Mcunet: Tiny deep learning on iot devices," in *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, 2020, pp. 1–15.

[201] X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen, "Deepdecision: A mobile deep learning framework for edge video analytics," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, IEEE, 2018, pp. 1421–1429.

[202] C. L. Novak, S. A. Shafer, *et al.*, "Anatomy of a color histogram.," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 1992, 1992, pp. 599–605.

[203] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, vol. 1, 2005, pp. 886–893.

[204] X. Huang, C. Shen, X. Boix, and Q. Zhao, "Salicon: Reducing the semantic gap in saliency prediction by adapting deep neural networks," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 262–270.

[205] H. Kasture and D. Sanchez, "Ubik: Efficient cache sharing with strict qos for latency-critical workloads," *ACM SIGPLAN Notices (ASPLOS)*, vol. 49, no. 4, pp. 729–742, 2014.

[206] N. Roy, A. Dubey, and A. Gokhale, "Efficient autoscaling in the cloud using predictive models for workload forecasting," in *2011 IEEE 4th International Conference on Cloud Computing*, IEEE, 2011, pp. 500–507.

[207] N. Corporation, *NVIDIA Jetson TX2 Board*, https://developer.nvidia.com/embedded/jetson-tx2.

[208] N. Corporation, *NVIDIA Jetson AGX Xavier Board*, https://developer.nvidia.com/embedded/jetson-agx-xavier-developer-kit.

[209] R. Xu, *Approxdet: Content and contention-aware approximate object detection for mobiles*, https://github.com/StarsThu2016/ApproxDet, 2020.

[210] Ultralytics, *Yolov3 in pytorch*, https://github.com/ultralytics/yolov3, 2021.

[211] B. Chen, G. Ghiasi, H. Liu, *et al.*, "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)*, 2020, pp. 13 607–13 616.

[212] Tensorflow, *Official implementation of ssd-mobilenetv2-mnasfpn*, https://github.com/tensorflow/models/tree/master/research/object_detection, 2021.

[213] H. Kasture and D. Sanchez, "Tailbench: A benchmark suite and evaluation methodology for latency-critical applications," in *2016 IEEE International Symposium on Workload Characterization (IISWC)*, IEEE, 2016, pp. 1–10.

[214] Y. Lee, A. Scolari, B.-G. Chun, M. D. Santambrogio, M. Weimer, and M. Interlandi, "PRETZEL: Opening the black box of machine learning prediction serving systems," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2018, pp. 611–626.

[215] W. Reda, M. Canini, L. Suresh, D. Kostić, and S. Braithwaite, "Rein: Taming tail latency in key-value stores via multiget scheduling," in *Proceedings of the Twelfth European Conference on Computer Systems*, 2017, pp. 95–110.

[216] S.-H. Bae and K.-J. Yoon, "Robust online multi-object tracking based on tracklet confidence and online discriminative appearance learning," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition (CVPR)*, 2014, pp. 1218–1225.

[217] J. Ju, D. Kim, B. Ku, D. K. Han, and H. Ko, "Online multi-object tracking based on hierarchical association framework," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2016, pp. 34–42.

[218] M. Muller, A. Bibi, S. Giancola, S. Alsubaihi, and B. Ghanem, "Trackingnet: A large-scale dataset and benchmark for object tracking in the wild," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 300–317.

[219] B. Babenko, M.-H. Yang, and S. Belongie, "Visual tracking with online multiple instance learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, 2009, pp. 983–990.

[220] H. Grabner, M. Grabner, and H. Bischof, "Real-time tracking via on-line boosting.," in *Proceedings of the British Machine Vision Conference (BMVC)*, vol. 1, 2006, pp. 47–56.

[221] X. Li, C. Ma, B. Wu, Z. He, and M.-H. Yang, "Target-aware deep tracking," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2019.

[222] N. Wojke, A. Bewley, and D. Paulus, "Simple online and realtime tracking with a deep association metric," in *2017 IEEE international conference on image processing (ICIP)*, IEEE, 2017, pp. 3645–3649.

[223] A. Klimovic, Y. Wang, P. Stuedi, A. Trivedi, J. Pfefferle, and C. Kozyrakis, "Pocket: Elastic ephemeral storage for serverless analytics," in *13th Usenix Symposium on Operating Systems Design and Implementation (OSDI)*, 2018, pp. 427–444.

[224] N. T. Hieu, M. Di Francesco, and A. Ylä-Jääski, "Virtual machine consolidation with usage prediction for energy-efficient cloud data centers," in *2015 IEEE 8th International Conference on Cloud Computing*, IEEE, 2015, pp. 750–757.

[225] R. Singh, D. Irwin, P. Shenoy, and K. K. Ramakrishnan, “Yank: Enabling green data centers to pull the plug,” in *Presented as part of the 10th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 13)*, 2013, pp. 143–155.

[226] A. Anjomshoaa, F. Duarte, D. Rennings, T. J. Matarazzo, P. deSouza, and C. Ratti, “City scanner: Building and scheduling a mobile sensing platform for smart city services,” *IEEE Internet of Things Journal*, vol. 5, no. 6, pp. 4567–4579, 2018.

[227] S. Yu, R. Langar, X. Fu, L. Wang, and Z. Han, “Computation offloading with data caching enhancement for mobile edge computing,” *IEEE Transactions on Vehicular Technology*, vol. 67, no. 11, pp. 11 098–11 112, 2018.

[228] J. Lee, K. Lee, E. Jeong, J. Jo, and N. B. Shroff, “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*, 2016, pp. 1235–1246.

[229] H. Luo, W. Xie, X. Wang, and W. Zeng, “Detect or track: Towards cost-effective video object detection/tracking,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 8803–8810.

[230] D. Wofk, F. Ma, T.-J. Yang, S. Karaman, and V. Sze, “Fastdepth: Fast monocular depth estimation on embedded systems,” in *2019 International Conference on Robotics and Automation (ICRA)*, IEEE, 2019, pp. 6101–6108.

[231] J. Lin, C. Gan, and S. Han, “Tsm: Temporal shift module for efficient video understanding,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 7083–7093.

[232] C.-Y. Wang, H.-Y. M. Liao, Y.-H. Wu, P.-Y. Chen, J.-W. Hsieh, and I.-H. Yeh, “Cspnet: A new backbone that can enhance learning capability of cnn,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, 2020, pp. 390–391.

[233] J. Lee, P. Wang, R. Xu, *et al.*, “Benchmarking video object detection systems on embedded devices under resource contention,” in *Proceedings of the 5th International Workshop on Embedded and Mobile Deep Learning*, 2021, pp. 19–24.

[234] L. Yang, H. Jiang, R. Cai, *et al.*, “Condensenet v2: Sparse feature reactivation for deep networks,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 3569–3578.

[235] S.-H. Gao, Y.-Q. Tan, M.-M. Cheng, C. Lu, Y. Chen, and S. Yan, “Highly efficient salient object detection with 100k parameters,” in *European Conference on Computer Vision*, Springer, 2020, pp. 702–721.

[236] Y. Li, Y. Chen, X. Dai, *et al.*, “Micronet: Improving image recognition with extremely low flops,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 468–477.

[237] K. Han, Y. Wang, Q. Zhang, W. Zhang, C. Xu, and T. Zhang, “Model rubik’s cube: Twisting resolution, depth and width for tinynets,” *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[238] A. Habibian, D. Abati, T. Cohen, and B. E. Bejnordi, “Skip-convolutions for efficient video processing,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2021.

[239] K. Chen, J. Wang, S. Yang, *et al.*, “Optimizing video object detection via a scale-time lattice,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7814–7823.

[240] Z. Wu, C. Xiong, Y.-G. Jiang, and L. S. Davis, “Liteeval: A coarse-to-fine framework for resource efficient video recognition,” *Advances in Neural Information Processing Systems*, vol. 32, pp. 7780–7789, 2019.

[241] Z. Wu, C. Xiong, C.-Y. Ma, R. Socher, and L. S. Davis, “Adaframe: Adaptive frame selection for fast video recognition,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 1278–1287.

[242] Y. Meng, C.-C. Lin, R. Panda, *et al.*, “AR-Net: Adaptive frame resolution for efficient action recognition,” in *European Conference on Computer Vision*, Springer, 2020, pp. 86–104.

[243] A. Ghodrati, B. E. Bejnordi, and A. Habibian, “Frameexit: Conditional early exiting for efficient video recognition,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 15 608–15 618.

[244] C. Feichtenhofer, “X3d: Expanding architectures for efficient video recognition,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 203–213.

[245] D. Tran, H. Wang, L. Torresani, J. Ray, Y. LeCun, and M. Paluri, “A closer look at spatiotemporal convolutions for action recognition,” in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2018, pp. 6450–6459.

[246] X. Sun, R. Panda, C.-F. R. Chen, A. Oliva, R. Feris, and K. Saenko, "Dynamic network quantization for efficient video inference," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 7375–7385.

[247] E. Perez, F. Strub, H. De Vries, V. Dumoulin, and A. Courville, "Film: Visual reasoning with a general conditioning layer," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018.

[248] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2117–2125.

[249] H. Belhassen, H. Zhang, V. Fresse, and E.-B. Bourennane, "Improving video object detection by seq-bbox matching.," in *VISIGRAPP (5: VISAPP)*, 2019, pp. 226–233.

VITA

Ran Xu was a Ph.D. candidate in the Elmore Family School of Electrical and Computer Engineering at Purdue University, where he was advised by Professor Saurabh Bagchi. His research focuses on the joint discipline of computer vision and systems with an emphasis on streaming video analytic systems on mobiles. He has published his work in many premier systems conferences and jounals, including USENIX ATC, SenSys, TOSN, Middleware and so on. Before studying at Purdue, he received his Bachelor of Engineering degree in Electronic Engineering from Tsinghua University, China in 2016.

PUBLICATIONS

Rigorously Reviewed Conference Papers:

[C1] J. Lee, P. Wang, R. Xu, et al., "Benchmarking video object detection systems on embedded devices under resource contention," in Proceedings of the 5th International Workshop on Embedded and Mobile Deep Learning, 2021, pp. 19-24.

[C2] R. Xu, C.-l. Zhang, P. Wang, et al., "ApproxDet: Content and contention-aware approximate object detection for mobiles," in Proceedings of the 18th Conference on Embedded Networked Sensor Systems (SenSys), 2020, pp. 449-462.

[C3] R. Xu, H. Wang, S. Petrangeli, V. Swaminathan, and S. Bagchi, "Closing-the-loop: A data-driven framework for effective video summarization," in Proceedings of the 22nd IEEE International Symposium on Multimedia (ISM), 2020, pp. 201-205.

[C4] K. Shankar, P. Wang, R. Xu, A. Mahgoub, and S. Chaterji, "Janus: Benchmarking commercial and open-source cloud and edge platforms for object and anomaly detection workloads," in Proceedings of the IEEE International Conference on Cloud Computing (CLOUD), 2020, pp. 590-599.

[C5] R. Xu, S. Mitra, J. Rahman, et al., "Pythia: Improving datacenter utilization via precise contention prediction for multiple co-located workloads," in Proceedings of the International Middleware Conference (Middleware), 2018, pp. 146-160.

[C6] R. Xu, J. Koo, R. Kumar, et al., "VideoChef: Efficient approximation for streaming video processing pipelines," in Proceedings of the USENIX Annual Technical Conference (USENIX ATC), 2018, pp. 43-56.

Serial Journal Articles:

[J1] R. Xu, R. Kumar, P. Wang, et al., "ApproxNet: Content and contention-aware video analytics system for embedded clients," ACM Transactions on Sensor Networks (TOSN), vol. 18, no. 1, pp. 1-27, Feb. 2022.

[J2] S. Bagchi, T. Abdelzaher, R. Govindan, et al., "New frontiers in IoT: networking, systems, reliability, and security challenges," IEEE Internet of Things Journal (IoT-J), vol. 7, no. 12, pp. 11330-11346, Dec. 2020.