# VIRTUOSO: Energy- and Latency-Aware Streamlining of Streaming Videos on SOCs

JAYOUNG LEE[ψ] PENGCHENG WANG[ψ] RAN XU[ψ] SARTHAK JAIN[ψ] VENKAT DASARI[λ]
NOAH WESTON[λ] YIN LI[δ] SAURABH BAGCHI[ψ] SOMALI CHATERJI[ψ]
ψ PURDUE UNIVERSITY, λ ARMY RESEARCH LAB, δ U OF WISCONSIN AT MADISON

Efficient and adaptive computer vision systems have been proposed to make computer vision tasks, such as image classification and object detection, optimized for embedded or mobile devices. These solutions, quite recent in their origin, focus on optimizing the model (a deep neural network, DNN) or the system by designing an adaptive system with approximation knobs. Despite several recent efforts, we show that existing solutions suffer from two major drawbacks. *First*, while mobile devices or systems-on-chips (SOCs) usually come with limited resources including battery power, most systems do not consider the energy consumption of the models during inference. *Second*, they do not consider the interplay between the three metrics of interest in their configurations, namely, latency, accuracy, and energy. In this work, we propose an efficient and adaptive video object detection system — VIRTUOSO, which is jointly optimized for accuracy, energy efficiency, and latency. Underlying VIRTUOSO is a multi-branch execution kernel that is capable of running at different operating points in the accuracy-energy-latency axes, and a lightweight runtime scheduler to select the best fit execution branch to satisfy the user requirement. We position this work as a first step in understanding the suitability of various object detection kernels on embedded boards in the accuracy-latency-energy axes, opening the door for further development in solutions customized to embedded systems and for benchmarking such solutions. VIRTUOSO is able to achieve up to 286 FPS on the NVIDIA Jetson AGX Xavier board, which is up to 45 times faster than the baseline EfficientDet D3 and 15 times faster than the baseline EfficientDet D0. In addition, we also observe up to 97.2% energy reduction using VIRTUOSO compared to the baseline YOLO (v3) — a widely used object detector designed for mobiles. To fairly compare with VIRTUOSO, we benchmark 15 state-of-the-art or widely used protocols, including Faster R-CNN (FRCNN) [NeurIPS'15], YOLO v3 [CVPR'16], SSD [ECCV'16], EfficientDet [CVPR'20], SELSA [ICCV'19], MEGA [CVPR'20], REPP [IROS'20], FastAdapt [EMDL'21], and our in-house adaptive variants of FRCNN+, YOLO+, SSD+, and EfficientDet+ (our variants have enhanced efficiency for mobiles). With this comprehensive benchmark, VIRTUOSO has shown superiority to all the above protocols, leading the accuracy frontier at every efficiency level on NVIDIA Jetson mobile GPUs. Specifically, VIRTUOSO has achieved an accuracy of 63.9%, which is more than 10% higher than some of the popular object detection models, FRCNN at 51.1%, and YOLO at 49.5%.

CCS Concepts: • **Computing methodologies** → **Machine learning**; • **Computer systems organization** → **Embedded & cyber-physical systems**.

Additional Key Words and Phrases: Adaptive inference, video object detection, energy consumption, efficiency-aware analytics, mobile GPUs, embedded computing, configuration tuning.

Author's address: Jayoung Lee[ψ] Pengcheng Wang[ψ] Ran Xu[ψ] Sarthak Jain[ψ] Venkat Dasari[λ] Noah Weston[λ] Yin Li[δ] Saurabh Bagchi[ψ] Somali Chaterji[ψ]
ψ Purdue University, λ Army Research Lab, δ U of Wisconsin at Madison.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2022 Association for Computing Machinery.
XXXX-XXXX/2022/9-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

**ACM Reference Format:**
Jayoung Lee[ψ] Pengcheng Wang[ψ] Ran Xu[ψ] Sarthak Jain[ψ] Venkat Dasari[λ] Noah Weston[λ] Yin Li[δ] Saurabh Bagchi[ψ] Somali Chaterji[ψ]
ψ Purdue University, λ Army Research Lab, δ U of Wisconsin at Madison. 2022. VIRTUOSO: Energy- and Latency-Aware Streamlining of Streaming Videos on SOCs. 1, 1 (September 2022), 33 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 INTRODUCTION

Video analytic systems have seen widespread success in various domains, ranging from computationally heavy tasks such as recognizing faces for surveillance to mobile applications such as detecting objects for mobile-based augmented reality (AR) [1, 35, 39], and to real-time systems such as localizing pedestrians and cars for autonomous driving [2, 14, 22]. A key function shared by these applications is the ability to detect objects in videos. There is a growing number of use cases for performing such video object detection on *mobile devices*. These devices are increasingly equipped with mobile GPUs, albeit they are much weaker computationally than those on server-class machines.

Nevertheless, there is an impetus to perform the video processing in (near) real-time on the streaming (video) content, on the device itself[1]. This is needed, say, to improve the user's immersive experience (e.g., in AR/VR games) or to give high-confidence outputs from streaming videos (e.g., for pedestrian recognition in autonomous driving). A typical operating point in processing each frame is 33 ms, which corresponds to the video stream rate of 30 frames per second (FPS). We find empirically that state-of-the-art (SOTA) protocols (designed for servers) when executed on mobile platforms significantly overshoot this latency margin. For example, MEGA [7], considered a SOTA solution takes 253.4 msec per frame on NVIDIA AGX Xavier while occupying all computing resources. AGX Xavier is a GPU hardware commonly fused for embedded GPU applications and one that we also utilize for VIRTUOSO. In response, a slew of efficient models and systems have been proposed to improve their efficiency or performance on mobile devices [6, 18, 24, 37, 42, 45, 51, 52].

Another approach to make streaming analytics feasible on mobile devices is to utilize both object detection and tracking, which is denoted by the technique "*tracking-by-detection*" [3, 28]. By interspersing multiple frames of tracking (lightweight compared to detection) with each frame where object detection is executed, the overall performance is sped up. The tracking-by-detection technique exposes several adaptation strategies. This can include selecting among a set of detectors and trackers and how to handle the relative frequency between the detector and tracker. Empirically, we find that the execution time of an object tracker is 10X lower than the time for an object detector. A judicious choice of such adaptation strategies is needed to satisfy the real-time requirements for energy, latency, and accuracy, on mobile devices.

The stringent requirements on mobile devices at runtime expose another challenge for streaming video analytics: *how to adapt (at runtime) to the dynamic user requirements and available resources on the device?* To address this problem, several recent works considered multi-branch solutions [12, 21, 48, 49]. Such solutions include multiple execution kernels in a system and choosing the optimal one *during runtime* to satisfy the user requirement. Yet, omitting features that control energy consumption is a major drawback of all these approaches. Further, compared to recent studies on efficient convolutional neural network (CNN) architectures, there have been limited studies and applications on efficient object detection solutions on embedded devices, along with the issue of using outdated feature extractors.

In real applications, the change of video content, available resources on mobile devices, and manual control of the requirements make dynamic adaptation even harder. The wide range of

[1]For convenience we will often use the shorthand "device" to refer to a "mobile device".

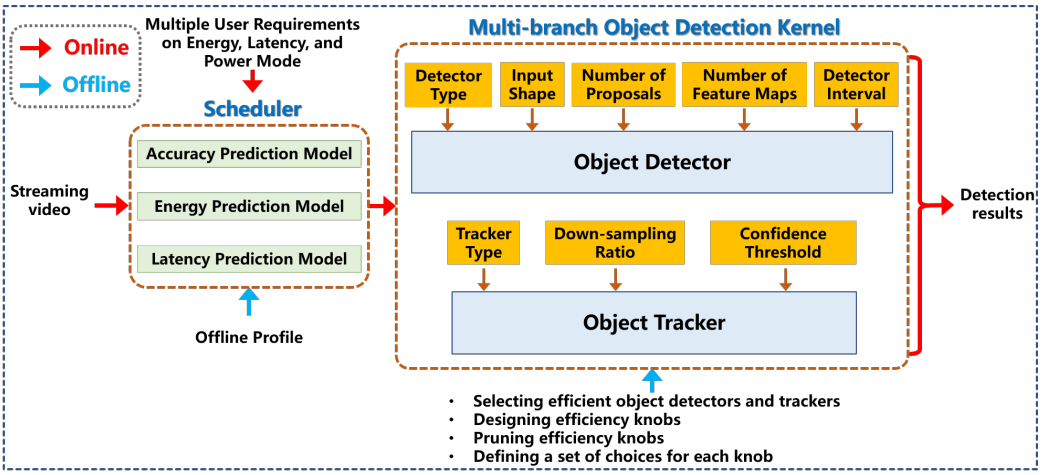



Fig. 1. The illustration of VIRTUOSO, which includes two important components, a multi-branch object detection kernel, and a scheduler. The multi-branch object detection kernel was achieved by exposing multiple tuning knobs for both the object detector and the object tracker. The scheduler can configure the kernel at runtime to satisfy user requirements of latency, energy, and accuracy.

energy, latency, and accuracy requirements means the system needs to have adaptability to a variety of scenarios. Thus, the design must satisfy the following prerequisites: (1) designing an execution kernel that is both energy and latency efficient for the mobile device, (2) analysis of the energy consumption and latency performance of all execution branches on mobile devices, and (3) an adaptive scheduler to make decisions at runtime to satisfy multiple requirements simultaneously. However, to our knowledge, no prior work has included all of these design innovations.

In this work, we present VIRTUOSO[2], which is customized for mobile devices under varying resource constraints, with additional efficiency knobs[3] for energy-aware adaptation. VIRTUOSO selects the most efficient baseline object detectors EfficientDet [45] and SSD [24], enhances them by integrating object trackers, and provides 155 execution branches by exposing 8 efficiency knobs within one system. Moreover, VIRTUOSO has a scheduler that can select the optimal branch during runtime to satisfy the energy and latency requirements at the same time, while maximizing the accuracy.

We evaluate VIRTUOSO and the baselines using the ILSVRC 2015 VID datasets on 3 NVIDIA mobile GPUs of increasing compute capacity — Jetson TX2, Xavier NX, and AGX Xavier, under 2 different power modes of AGX Xavier. On each board and under each power mode, VIRTUOSO is the best solution that can automatically choose the optimal branch to satisfy the energy and latency requirements, and maximize the accuracy performance. With the multi-branch execution kernel, VIRTUOSO can adapt its energy efficiency within a wide range (span of 89X) and also latency in a wide range (span of 128X). Such wide ranges of energy efficiency and latency can satisfy various scenarios at runtime to optimize the functionality of our video object detection system. Given 1 J per frame energy requirement, VIRTUOSO can achieve 52.0% accuracy, and given a 33.3 msec latency budget, VIRTUOSO can achieve 60.1% accuracy on an AGX Xavier board. Additionally, when energy

[2]Just like a VIRTUOSO is a person who has exceptional skill, expertise, or talent at some endeavor, we believe our system demonstrates such skill in configuring streaming video object detection on embedded or mobile devices.
[3]We use the term "efficiency knobs" rather than the more common term "tuning knobs" as we are focusing on the performance metric of accuracy, normalized by latency or energy cost to achieve that accuracy, in other words, efficient accuracy.

and latency do not have stringent limits, VIRTUOSO can achieve 63.9% accuracy, which is more than 10% higher than that of baseline models. To further explore the performance of VIRTUOSO under a more realistic scenario, we also evaluated a total of 15 baseline models on 3 different embedded boards under different *resource contention levels* to better understand the energy, latency, and accuracy performance of these latest models for video object detection.

In this paper, our contributions are as follows.

(1) We present VIRTUOSO, the best energy efficient and adaptive video object detection system for mobile devices. VIRTUOSO can dynamically adapt its runtime configurations based on the given user requirements. Our runtime scheduler solves the optimization problem for accuracy at any energy and latency level and thus can guarantee the Pareto optimal performance on resource-constrained devices. We are the only adaptive model doing this 3D (energy-latency-accuracy) tuning in real-time during inference.
(2) VIRTUOSO designs an efficient multi-branch execution kernel with a total of 8 different efficiency knobs, which are optimized for both energy efficiency and latency on mobiles. This gives VIRTUOSO more flexibility to a wider range of available resources, span of 89X energy adapting space when baseline models cannot, and 4.5X times larger latency adapting space than the baseline FastAdapt [21]. VIRTUOSO can achieve 97% lower energy consumption and up to 50 times faster execution, compared to using the object detection kernel only.
(3) We evaluate our proposed systems and 15 baselines (including 3 latest video object detection solutions and our enhanced FRCNN [38], YOLO [37], SSD [24], and EfficientDet [45] for both energy and latency on the embedded devices) on 3 NVIDIA Jetson embedded devices and under different GPU resource availability, energy efficiency constraint, and latency constraints. In addition, we also investigate and evaluate the impact of different device power modes, which have not been evaluated before. Different power modes allow several configurations with different CPU frequencies and number of CPU cores online. We show accuracy superiority to baselines given any efficiency constraint.

The rest of the paper is organized as follows. In Section. 3, we present our overall design for VIRTUOSO and its key components, such as the dynamic scheduler and efficiency knobs for handling various user requirements for latency or energy consumption. In Section. 4, we present the multi-branch kernel design and implementation of VIRTUOSO, along with different runtime environments for evaluation. Section. 5 consists of two parts: First, we evaluate the overall performance of VIRTUOSO. Next, we evaluate VIRTUOSO and its multi-branch kernels against other baselines.

## 2 RELATED WORK

**Video object detection** seeks to locate object instances in video frames using bounding boxes and simultaneously classify the instance into target categories with their class probabilities. The most widely used detection models adopt CNNs, broken down into two parts: a backbone network that extracts features from images (e.g., ResNet), and a detection network or head, which classifies object regions and refines the localization of the objects based on the extracted features (e.g., Region Proposal Network or Weighted Bi-directional Feature Pyramid Network). The detection network can be further categorized into two-stage detectors [10, 38, 45], or single-stage detectors [24, 37, 51]. One representative work of two-stage detectors is Faster R-CNN (FRCNN) [38], where plausible regions are proposed in the first stage, followed by decision refinement in the second. Specifically, CNNs extract image feature maps and feed them 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 proposal feature maps and provide these to the classifier network. On the other hand, YOLO and

SSD are the representative works for single-stage detectors. These single-stage end-to-end detection solutions do not include the step of region proposal generation, but rather, directly classify a dense set of pre-defined regions from the feature maps. One-stage detection models are usually easier to train and are more computationally efficient but often suffer from lower accuracy, especially for mAP with high IoU thresholds.

A general trend in object detection is to design deeper and more complex object detection networks to achieve higher accuracy such as in recent video object detection algorithms [7, 7, 11, 41, 47, 50, 53]. There is ongoing research on pushing the accuracy further for video object detection tasks, for example, frame aggregation [7, 50], a technique that utilizes features from other frames during inference to enhance the detection results. SELSA [47] widens the window for selecting the frames for aggregation by not only selecting neighboring frames but considering their semantic neighborhood. MEGA [7] takes the work from SELSA one step further and adds global frame aggregation where frames from other videos, sharing semantic similarity, are also taken into account. While these techniques are performed during the runtime of the inference task, REPP [41] reuses the detection output from a baseline model to further post-process the detection output to enhance the detection results after the analysis of a video. Other works make use of optical flow [53], or techniques such as knowledge distillation [11]. However, these advancements in accuracy do not necessarily target making these algorithms more efficient in terms of the network size, energy consumption, and latency of the detection task.

Several studies have been conducted to optimize accuracy and latency for video object detection tasks. Feichtenhofer *et al.* [13] combines an object detector and an object tracker to create a joint design that is trained and deployed in an end-to-end fashion so that a lightweight tracker could speed up the process of a detector-only design. Jiang *et al.* [18] use an LSTM module to propagate the high-level features across frames to reduce the computation cost resulting from the optical flow technique that captures the temporal information in the video. A "key frame" concept is used in [52] to effectively group adjacent frames with similar features, thus saving redundant computation costs. Chen *et al.* [6] also utilize the concept of "key frame", to adaptively schedule the computation path to sparsely spread out operations with high computation costs. However, most studies that tackle the optimization challenge between accuracy and latency still focus on server-class GPUs, which are much more powerful than mobile or embedded devices [4]. In many real-world object detection tasks, the task has to be carried out in a real-time fashion on a computationally constrained platform, such as a mobile device. In such cases, video object detection becomes challenging because of the resource constraints and the stringent energy budget (limited battery) and latency budget (30–50 msec/frame) for acceptable video quality.

Some recent approaches take the real-world computation constraint into account and design efficient backbones that are specifically designed to reduce the computation cost. MobileNetV2 [42] uses an inverted residual block to reduce the number of computations, and thus improve computational efficiency. AdaScale [8] makes use of the content information of videos to dynamically re-scale the images to lower resolution, and at the same time, achieve better accuracy. GhostNet [16] uses a "ghost module" to reuse some of the features from the feature map to reduce the computational cost. These are examples that utilize a human-crafted component to optimize the model. On the other hand, model architectures can be automatically optimized using a neural architecture search (NAS) technique. NAS-based models [43, 44, 46] pre-define the blocks or layers that will be used to construct the network, and search through combinations and connections of the pre-defined components. EfficientDet [45] introduces reinforcement learning (RL) based-NAS for a light-weight network. Instead of modifying or creating a new network design, some solutions add adaptive

[4]We use the terms "mobile device" and "embedded device" synonymously.

components to the pipeline, such as using a dynamic pipeline adapting to content at runtime [12, 49]. However, all of the aforementioned studies focus on the network design, which performs the object detection task in limited scenarios. Even though these works have improved the computational efficiency, they still require further development and improvement to be deployed for a real-time dynamic environment on mobile devices with changing energy and latency requirements. For example, they may require significant feature engineering to fit the specialized capabilities of the mobile GPUs as done in our prior work [15].

As an overall summary, we also include Fig. 2 to visualize the comparison between key features of VIRTUOSO. Overall, VIRTUOSO is a video object detection framework that comes with a multi-kernel design for adaptive inference and is able to consider both energy and latency performance on embedded devices. Other baselines do not consider the latency or energy performance on embedded devices or do not have adaptive features during inference.

| | Multi-kernel Design | Adaptive Inference | Real-time latency on Embedded Devices | Energy Consideration | Video Object Detection |
|---|---|---|---|---|---|
| Virtuoso | ✓ | ✓ | ✓ | ✓ | ✓ |
| MEGA [CVPR 21'] | ✗ | ✗ | ✗ | ✗ | ✓ |
| SELSA [ICCV 19'] | ✗ | ✗ | ✗ | ✗ | ✓ |
| AdaScale [MLSys 19'] | ✓ | ✓ | ✗ | ✗ | ✓ |
| ApproxDet [SenSys 20'] | ✓ | ✓ | ✓ | ✗ | ✓ |
| EfficientDet [CVPR 20'] | ✗ | ✗ | ✗ | ✗ | ✓ |
| GhostNet [CVPR 20'] | ✗ | ✗ | ✓ | ✓ | ✗ |

Fig. 2. Comparison of related works against VIRTUOSO.

**Benchmarking video object detection works on embedded devices:** With the rise of video object detection, coupled with the popularity of edge computing in recent years, video object detection tasks have been pushed to the edge/embedded devices where the data is generated. MEVBench [9] has provided a benchmark suite for a range of mobile vision applications such as face detection, object tracking, and feature extraction. However, none of SOTA works or the latest devices have been used, and its evaluation is not on the GPU, which is the *de facto* hardware for DNN-based computer vision works. AIoT bench [26] also provides an AI tasks' benchmark suite based on Android and Raspberry-Pi and covers different frameworks like TensorFlow and Caffe2. Nevertheless, there are no SOTA models in it and no evaluation is presented. Qasaimeh *et al.* [34] has conducted benchmarks of accuracy, latency, and energy on a wide range of vision kernels and neural networks on multiple embedded devices, i.e., ARM57 CPU, Nvidia Jetson TX2, and Xilinx ZCU102 FPGA. However, it only includes one GPU-enabled device and thus is not comprehensive since embedded devices with GPUs are very common nowadays. Also, it has not included the SOTA models and does not focus on video object detection, which is the VIRTUOSO's focus. Also, lots of video object detection solutions are designed by including multiple tuning knobs, Qasaimeh *et al.* [34] has not shown the accuracy-efficiency tradeoff on embedded devices. Buckler *et al.* [4] take a more detailed look at the kernels of the image signal processing (ISP) pipeline, e.g., it does a detailed investigation of how many stages of an ISP pipeline should be used, what algorithm the image sensor should use, and the quantization of the ADC. Consequently, it is less complete in

terms of its coverage of the detection kernels. It covers only one model for object detection, Faster R-CNN, which we also cover. They do not measure power consumption but use analytical formulae and simulations. Euphrates [54] optimizes the interaction between the ISP and the CNN in the CV pipeline. This also does the optimization in an SoC architecture specific manner. However, it does not focus on the video object detection task. A comprehensive benchmark is important to understand the advantages and disadvantages of different efficient and adaptive models, i.e., how accurate they are given an efficiency requirement and how much these models can adapt in terms of efficiency.

## 3 TECHNIQUES

We now present the techniques used in the design and implementation of VIRTUOSO. To achieve high energy efficiency and low latency on embedded devices, we first propose a collection of efficiency knobs (Sec. 3.1) and an efficient multi-branch object detection kernel (Sec. 3.2). Our system combines the efficiency knobs and is capable of running at different operating points in the accuracy-energy-latency axes through its multiple execution branches. We then propose our runtime scheduler to solve the constrained optimization problem at any energy efficiency or latency requirements (Sec. 3.3).

### 3.1 Efficiency knobs for Object Detection Models

To effectively make object detection backbones both energy and time efficient, the *tracking-by-detection* technique is one of the most common methods for video object detection. Particularly, considering a video as a sequence of consecutive frames, we define Group-of-Frames (GoF) as a collection of consecutive frames in which we apply the computationally expensive object detector to the first frame, and apply the light-weight object tracker, to the remaining frames. An object tracker is highly efficient since it is much cheaper in terms of computation, with at least 20 times better latency performance than an object detector (from our results). However, it relies on the relationship between the current frame and the past frame, and the detection results of the past frame. Thus, an object tracker, despite being more efficient, cannot run without an object detector. The latter provides a calibrated detection result on every first frame of a GoF.

#### *3.1.1 Efficiency Knobs for the Object Detector.*

**Object Detection Backbone**: We select a total of four different object detectors to perform the detection — EfficientDet, SSD, FRCNN, and YOLO, and call them "Object Detection Backbones". Switching among these kernels can be used as the primary adaptation strategy based on the users' requirements. Particularly, EfficientDet is a family of object detectors that are scaled up with different scaling factors, starting from the base model D0. Among the 8 variants of EfficientDet (D0-D7), we experimentally find that model variants with larger scale than D3 fail to run on our embedded devices. Thus, we select EfficientDet D0 and D3 as the most light-weight, and heavy-weight ones, among all executable variants. These variants within the same object detector family enable the tradeoff with respect to the accuracy, energy efficiency, and latency. For simplicity, an object detector backbone refers to an object detector, or a particular variant, e.g., EfficientDet D0 or D3.

**Input Image Resolution for the Object Detector Backbone**: Given the different object detector backbones, we further consider additional efficiency knobs. First is the resolution of the input image fed into the detector. Each detector backbone comes with a pre-defined image resolution that can be processed through the neural network, and all input images are resized to the pre-defined resolution as the first step. We modify the input layer of the object detector backbone to accept

images with different resolutions. This is possible as the backbones are fully convolutional. Feeding smaller-scaled image results in both less energy consumption and less computational overhead, translating to a more efficient model.

**Number of Proposals in the Object Detector Backbone**: FRCNN is a two-stage object detector. The first stage is a Region Proposal Network (RPN) to process the feature map output from the feature extractor and return a pre-defined number of object candidates in the feature map. We implement number of proposals as the efficiency knob in the RPN to modify the number of output object candidates. The number of object candidates is directly related to the computation in the second stage of the detector. Thus, we are able to leverage the accuracy vs. efficiency tradeoff by modifying the number of proposals. This knob is only available for FRCNN.

**Number of Feature Maps in the Object Detector Backbone**: To further engineer our object detector backbone, we explore the MnasFPN [5] feature pyramid in the SSD detector that is used to concatenate feature maps. MnasFPN concatenates a total of four feature maps, responsible for detecting objects on different scales. The first feature map has the largest feature map size and is responsible for detecting objects at a finer granularity. Following the path down the feature pyramid from the first feature map, the following feature maps are compressed gradually and are used to detect larger objects. We explore the tradeoff of accuracy versus efficiency by using different combinations of the four feature maps, such as [1, 2, 3], [2, 4], and so on, where 1 to 4 stand for the four feature maps.

#### 3.1.2 *Efficiency Knobs for the Object Tracker.*

**Object Tracker**: Another major component of our efficient design is the object tracker. Similar to different object detector backbones, we also utilize multiple object trackers. A total of four object trackers, MedianFlow [19], KCF [17], CSRT [25], and OpticalFlow [20], are utilized and explored.

**Resizing factor - Input Image Resolution for Object Tracker**: The resizing factor for the input image of the object tracker is changed here such that a larger image requires the tracker to process through a larger number of pixels, with a reduction in efficiency.

**Confidence Threshold of the Object Detector Backbone**: The confidence threshold of the object detector backbone is closely related to the performance of the object tracker. A typical efficient detector backbone, such as EfficientDet or SSD, has a pre-defined number of detected objects or outputs (e.g., detections with the top 100 confidence scores for both EfficientDet and SSD) to increase the accuracy performance of the model. A spike in energy and latency overhead is encountered if the tracker tracks all the detected objects. For example, the latency for tracking a single object of a 1280 x 720 image takes about 6.5 msec on the Xavier AGX board. For tracking 100 images, this value becomes 344.0 msec, which is more than 50 times degraded (higher) latency. Although the instantaneous power measurements are similar for both cases, since the energy consumption is accumulated power over time, the impact on latency affects energy consumption as well. Thus, we set a tunable threshold to control the number of objects to track.

**Detector Interval**: VIRTUOSO leverages the usage frequency for the object detector backbone and the object tracker in the GoF. Every first frame in the GoF is passed through the object detector backbone to provide calibrated detection results for the object tracker, and the rest of the frames are passed through the object tracker. We define the number of frames in the GoF as the detector interval, indicating how often the detector should be run. For example, if the detector interval is 1, we run the object detector on all frames. In contrast, if the detector interval is 8, we run the object detector every 8 frames, and the rest with the object tracker.

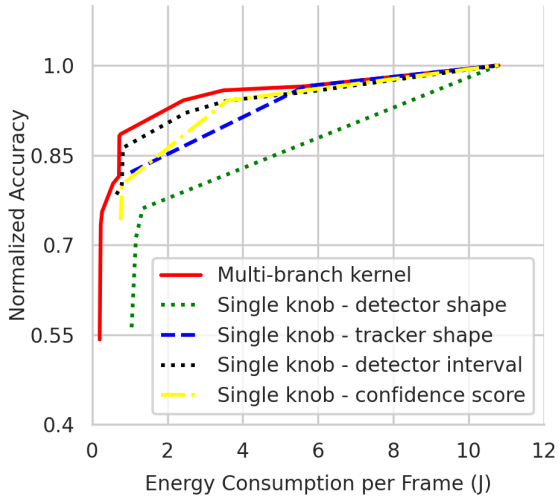



Fig. 3. Accuracy of VIRTUOSO given multiple efficiency knobs versus a single efficiency knob. The overall accuracy versus energy (same trend with accuracy vs. latency) tradeoff is always better using multiple knobs combined, compared to only using a single knob, validating the effectiveness of multiple knobs stacked together.

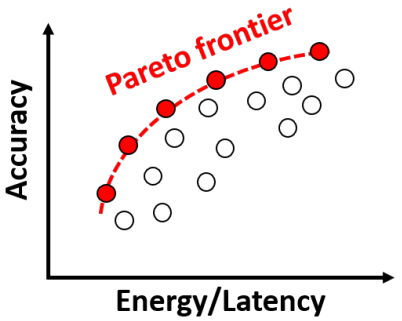



Fig. 4. A Pareto frontier for the accuracy-energy/latency tradeoff for a particular model, by varying its efficiency knobs. The Pareto frontier has to be considered separately for each embedded device.

### 3.2 Efficient Multi-Branch Object Detection Kernel

We propose our multi-branch object detection kernel as the combination of all possible combinations of the efficient methods or knobs listed in Sec. 3.1. Particularly, an execution branch corresponds to a collection of choices on each efficiency knob and each branch can independently finish the task. Rigorously, an execution branch $b$ is denoted in a tuple form,

$$b = (d, rd, np, nm, t, rf, ct, i), \tag{1}$$

where $d$ is the object detector backbone, $rd$ is the input resolution of the detector, $np$ is the number of proposals in the detector, $nm$ is the number of feature maps in the detector, $t$ is the object tracker, $rf$ is the resizing factor of the input image for the object tracker, $ct$ is the confidence threshold to track, and $i$ is the detector interval. Thus, each execution branch is an instantiation in the high-dimensional configuration space, with a certain accuracy $a(b)$, energy consumption $e(b)$, and execution time (latency) $l(b)$. However, these execution branch choices are not fully independent. For example, if we choose EfficientDet as the object detector, we cannot use the "Number of Proposals", which does not apply to EfficientDet. We further discuss the implementation details in Sec. 4.1. This notion of finding optimal configurations in large configuration space with dependencies among different parameters has been tackled in other contexts, such as for distributed database tuning [27].

The advantage of multiple efficiency knobs over one knob is that their combination achieves a better Pareto optimal accuracy frontier for any efficiency requirement. In Fig. 3, we study the accuracy of VIRTUOSO along with each efficiency knob given a certain energy constraint compared to that of VIRTUOSO using multiple knobs. The results have shown that using multiple efficiency

knobs gives a wider spectrum of accuracy vs. energy tradeoff and much higher accuracy at any energy level.

### 3.3 Scheduler

We design the scheduler in VIRTUOSO to select the most accurate branch $b_{opt}$ to satisfy users' requirements both for the execution time (a budget of $l_0$ per video frame) and the energy consumption (a budget of $e_0$ per frame), at runtime. During this optimization process, the user may specify either energy or latency as the major SLA, and the other as the minor SLA for prioritization. Fig. 4 conceptually shows the selection of the scheduler, where among all possible execution branches, the scheduler picks the Pareto frontier performance branches that achieves the highest performance in both latency or energy vs. accuracy. Such a design with multiple execution branches on the Pareto frontier is crucial to adapt and achieve high accuracy performance while conforming to user requirements that may change at runtime.

Rigorously, the scheduler solves the following optimization problem:

$$\underset{b \in \mathcal{B}}{\operatorname{argmax}}\, a(b)\ s.t.\ e(b) \leq e_0,\ l(b) \leq l_0, \tag{2}$$

where $\mathcal{B}$ is the set of all possible branches in the multi-branch object detection kernel. $a(b)$, $e(b)$, and $l(b)$ are the accuracy, energy efficiency, and the latency of branch $b$, respectively. Energy consumption has not been explored in existing studies on embedded object detectors.

Algorithm 1 presents a high-level overview of our scheduler design. Note that the $knob_config$ variable consists of all the efficiency knobs settings that define an execution branch. The scheduling process begins by taking the user requirement as input. First, with either energy or latency as the major requirement, all the branches that meet the user-specified value will be filtered. Next, if there is a user-specified minor requirement as well, the branches filtered by the major requirement will again be pruned using the minor requirement. Finally, given the remaining branches that meet the user requirements, the accuracy predictor predicts and selects the highest accuracy branch among the filtered branches. The scheduler is triggered at the beginning of the inference with $scheduler.predict()$, and then is triggered ($scheduler.reschedule$ when it detects a change in the user requirement.

As efficiency is one of the main drivers for the design of VIRTUOSO, the scheduler should also be lightweight, making immediate decisions as video frames arrive in the streaming style. To solve this problem, we model the energy consumption, latency, and accuracy of each execution branch in a data-driven manner. Particularly, we collect energy, latency, and the accuracy profile of each branch offline. Then, we train the energy, latency, and accuracy prediction models. We then use these models during the online phase so as to finish the task of the scheduler. When taking the choices of embedded devices, their power modes, and resource contention into consideration, these models are more complex than the simplified form in Eq. 2, and our detailed design follows next.

**Energy Prediction Model**: The energy consumption $e(b)$ of an execution branch is measured by calculating the average energy consumption of processing a single frame for each branch. We first profile the energy consumption on sample videos instead of the entire dataset and measure the overall energy consumption of each execution branch. This is because the overall energy consumption of each execution branch is consistent across video frames and does not require such large amount of profiling data. Since the exact energy consumption of a specific process on the embedded devices could not be measured, we use the overall energy consumption of the board as our metric. We use the following Eq. 3, where $N$ represents the number of frames within the video, $p(t)$ represents the instantaneous power measured at every 1 second interval, and $T$ represents the overall time of inference.

**Algorithm 1:** Pseudocode for the scheduler.

```
Input: Major_requirement, Minor_requirement
knob_config = scheduler.predict(Major_requirement, Minor_requirement);
for video_frame do
    detector.set_knob(knob_config);
    if detector_frame then
        average_latency /= detector_interval;
        average_energy /= detector_interval;
        average_latency = average_energy = 0;
        if new_user_requirement! = old_user_requirement then
            knob_config = scheduler.reschedule(new_user_requirement);
        end
        detect(video_frame);
        average_latency += detector_latency;
        average_energy += detector_energy;
    else
        track(video_frame);
        average_latency += tracker_latency;
        average_energy += tracker_energy;
    end
end
```

$$e(b) = \frac{\sum_{t=0}^{T} p(t)}{N * T}. \tag{3}$$

**Latency Prediction Model**: The latency $l(b)$ of an execution branch is affected by many factors. For example, due to the different computation capabilities of embedded boards, the latency on each board is different. Also, the power mode of the device and the resource contention also affect the runtime latency of an execution branch. To minimize the profiling cost, we use the following two techniques. *First*, we profile the latency on sample videos instead of on the entire dataset. This is because the latency of each execution branch is consistent across video frames and does not require such large amount of profiling data. *Second*, we decouple the profiling on the object detector and the object tracker. This allows us to profile all object detector branches and all object tracker branches, separately, and we use the following Eq. 4 to calculate the overall latency due to the "tracking-by-detection" design,

$$l(b) = \frac{l_{detector}(b) + (i-1) * l_{tracker}(b)}{i}, \tag{4}$$

where $l_{detector}(b)$ denotes the detector latency of branch $b$, $l_{tracker}(b)$ denotes the tracker latency of branch $b$, and i is the number of group of frames that matches the detector interval.

**Accuracy Prediction Model**: The accuracy $a(b)$ of an execution branch is profiled in the offline training dataset and looked up in the online phase. The intuition is that the accuracy of each branch stays the same in the online phase since both the offline training dataset and the online test dataset follow an independent and identical distribution. Considering the accuracy is meaningful given a large enough dataset and the number of execution branches is large, the cost of offline profiling is significant. Thus, we use the three following techniques to speed up the profiling. *First*, we prune

out the inferior branches in terms of accuracy and efficiency and only use efficient yet effective models for the final design. For example, only SSD and EfficientDet are considered the choices of object detectors. *Second*, we use the high-end servers to profile the accuracy of each branch since our multi-branch execution kernel produces deterministic and consistent results between servers and embedded devices. *Finally*, our profiling leverages the fact that the branches with the same configurations except for detector internal $i$ can reuse the object detection results on the frames where the object detector runs. We first profile the accuracy of all execution branches with $i = 1$ (object detector only), save the detection results, and then profile the accuracy of other execution branches and reuse the saved detection results.

To match stringent users' efficiency requirements — energy or latency — of inference at real-time (e.g., 30 or 50 FPS) on embedded devices, the low overhead of the branch prediction models must be prioritized. Our implementation of lightweight prediction models comes with the benefit of low overhead. We empirically find that the overall latency overhead of our scheduler is less than 1 msec on all of our set of Jetson boards (0.16 ms on AGX Xavier, 0.26 ms on Xavier NX, and 0.19 ms on TX2), which is marginal compared to the typical real-time frame rate of 30 FPS. Note that this overhead includes all of the branch selection time and the branch switching time. Overall, with lightweight prediction models and low overhead of the scheduler, VIRTUOSO is able to dynamically adapt at runtime based on changes in user-specified latency or energy requirement.

## 4 IMPLEMENTATION

In this section, we mainly describe the implementation details of our multi-branch object detection kernel (Sec. 4.1), training details of object detectors (Sec. 4.2), and the embedded devices we used for evaluation (Sec. 4.3).

### 4.1 Efficient Multi-Branch Object Detection Kernel

We implement our efficient multi-branch object detection kernel with the following software stacks: CUDA 10.2.89 and CuDNN 8.0.0.180 as the base libraries for GPU-accelerated DNN primitives, and TensorFlow 2.4.0 with Python 3.6, for developing the overall framework. Also, given the multiple efficiency knobs we have implemented, there exists a vast number of execution branch combinations. Thus, we propose the following implementation techniques to reduce both offline and online costs to a practically feasible level.

**Pruning the Efficiency Knobs**: First, we only consider EfficientDet and SSD as the object detector backbone in VIRTUOSO, as they show superior performance in energy, latency, and accuracy, compared to FRCNN and YOLO. Also, these two object detectors are optimized for performance on mobile devices. While VIRTUOSO only incorporates EfficientDet and SSD due to their efficient design, we still include FRCNN and YOLO with other efficiency knobs, and use them as baselines for evaluation. Next, we narrow down the efficiency knobs to be applied to each object detector backbone. The "number of proposals" knob is exclusive to FRCNN, so it is not included in VIRTUOSO. For EfficientDet D0 and D3, the models come with the built-in input image resolution that is hard-coded - 512 for EfficientDet D0 and 896 for EfficientDet D3 - matching their scaling factor and the design of the feature pyramid architecture (what they refer to as the Bi-FPN). Therefore, the input image resolution knob is implemented only for SSD. In addition, we have tested the number of feature map knobs that are exclusive to SSD and checked the accuracy and latency of all possible combinations on the ILSVRC 2015 VID dataset. Our intuition of using a subset of the feature pyramid layers in MnasFPN is that reducing the number of layers used to compute the features will result in a benefit in latency with a moderate tradeoff of accuracy. However, due to the explicit design of the MnasFPN generated by the NAS technique, each feature level had a significant drop in accuracy, while removing feature levels afforded limited latency reduction benefits.

**Defining a Set of Choices for Each Knob**: Most efficiency knobs, e.g., input resolution, and detector interval can support any integer choices. However, due to the cost of offline profiling and online scheduling, we define a discrete set of choices for each knob as follows. The SSD and EfficientDet D0 and D3 are selected as the object detector backbones. Input image resolution for the detector is implemented for the SSD, and we use the shape of [192, 256, 320] as our pre-defined set of choices. The acceptable image resolution depends on the network architecture. MedianFlow tracker is selected for its lightweight design compared to the other trackers we have explored while maintaining comparable accuracy. The input resolution for the tracker is [100%, 50%, 25%] of the original resolution on height and width dimensions. For detector interval, we have [1, 2, 4, 8, 20, 100], and finally, we use confidence thresholds of [0.15, 0.30] to post-process the detection outputs from the object detector. The confidence threshold removes objects with lower confidence scores and controls the number of objects that need to be tracked by the object tracker.

Considering all the pre-defined efficiency knobs above, we have a total of 155 execution branches for VIRTUOSO. Having hundreds of branches in a multi-branch object detection kernel does not mean we have to store and load that many copies of branches in the disk or the memory. "Input resolution of the object detector backbone", "number of proposals in the detector backbone", "number of feature maps in the detector backbone", "resizing factor of the object tracker", "confidence threshold to track", and "detector interval" can all be implemented as a control parameter with just one copy of the model. As for the choice of the object detector backbone and object tracker, we lower the switching cost among the execution branches by merging the static computation graphs of all object detector backbones. The benefit of merging the static graph is that each object detection backbone is only loaded into the GPU memory when it is executed, which saves resources for detector backbones not being used. In addition, the executed object detector backbones are preserved in a cached state, and VIRTUOSO is able to switch among loaded backbones with minimal overhead without requiring the initialization of the detector backbone.

### 4.2 Training Efficient Object Detectors

EfficientDet D0, D3, and SSD object detectors do not come with publicly available pre-trained weights for the ILSVRC 2015 VID datasets [40]. Following the widely adopted training protocols in the latest video object detection solutions that we have selected as baselines for evaluation [7, 41, 47], we train each object detector on a combined dataset of ILSVRC VID training and DET training datasets [40]. The ILSVRC 2015 VID training dataset consists of 3,862 videos with 30 object classes. From each video, 15 video frames, whose timestamps are evenly spaced, are selected. As an addition to the ILSVRC 2015 VID training dataset, images containing the 30 overlapping classes with the VID dataset are selected from the DET training dataset. Finally, a total of 111,473 video frames or images, 57,834 from the VID dataset, and 53,639 from the DET dataset are selected for the training process.

We train EfficientDet D0 and D3 from the COCO pre-trained weights that come from the official repository and follow the default training settings while freezing the backbone part (EfficientNet) during the fine-tuning. To speed up, we use 1% of ILSVRC 2015 VID training dataset (11,768 video frames) for the first 100 epochs of D0 and for the first 50 epochs of D3, and then use the aforementioned VID and DET dataset for the remaining 10 epochs. In addition, the SSD model combined with MobileNetV2 and MnasFPN also comes with the COCO pre-trained weights from the official repository (not our evaluation dataset ILSVRC 2015 VID). We follow most of the default training settings from the repository, except for using a batch size of 48 and a learning rate of 0.004. The model is trained with the aforementioned VID and DET dataset and is trained up to 180 epochs.

### 4.3 Embedded Devices

We evaluate VIRTUOSO and baseline models on NVIDIA Jetson AGX Xavier [29], Jetson Xavier NX [32], and Jetson TX2 [31]. Each device has different CPU, GPU, and memory capacities, and the relationship between their computational capacities is Jetson AGX Xavier > Jetson Xavier NX > Jetson TX2. Table 1 gives the hardware specifications of these devices. Each board has different numbers of power mode levels shown in Table 2 for Jetson Xavier AGX, and Table 3 for Jetson NX Xavier. We picked some power modes in our experiments to better understand the power modes of Jetson devices — mode 0 and mode 2 on AGX Xavier, mode 0, 2, and 4 on Xavier NX, and mode 0 on TX2 [5]. These levels can impact the performance of an object detection system by altering the maximum power budget, maximum frequency for CPU, GPU, and deep learning accelerator, and the number of online CPU cores.

Furthermore, these devices have a native Dynamic Voltage and Frequency Scaling (DVFS) functionality on the CPU and GPU, enabled by default. DVFS provides a way to reduce the static and dynamic power consumption of the embedded boards on the fly by scaling up or down the voltage and frequency based on the targeted performance of the application [30]. We find that the default DVFS functionality can cause inconsistency in our evaluation results because of the changing CPU, GPU, and memory frequencies. Therefore, we disable DVFS by fixing the frequency of the modules at their max frequencies under the corresponding power mode. We empirically determine that this step is crucial to the reproducibility of results.

| Models | Jetson AGX Xavier | Jetson Xavier NX | Jetson TX2 |
|---|---|---|---|
| CPU | 8-core NVIDIA Carmel Armv8.2 64-bit CPU 8MB L2 + 4MB L3 with max frequency at 2265MHz | 6-core NVIDIA Carmel ARMv8.2 64-bit CPU 6MB L2 + 4MB L3 with max frequency at 1900MHz | Dual-Core NVIDIA Denver 2 64-Bit CPU and Quad-Core ARM Cortex-A57 MPCore processor with max frequency at 2000MHz |
| GPU | 512-core NVIDIA Volta GPU with 64 Tensor Cores with max frequency at 1377MHz | 384-core NVIDIA Volta GPU with 48 Tensor Cores with max frequency at 1100MHz | 256-core NVIDIA Pascal GPU with max frequency at 1300MHz |
| Memory | 32 GB 256-bit LPDDR4x 136.5GB/s | 8 GB 128-bit LPDDR4x 51.2GB/s | 8 GB 128-bit LPDDR4 59.7GB/s |
| Storage | 32 GB eMMC 5.1 | 16 GB eMMC 5.1 | 32 GB eMMC 5.1 |
| Power | 10W/15W/30W | 10W/15W | 7.5W/15W |
| DL Accelerator | 2x NVDLA Engines | 2x NVDLA Engines | - |
| AI Performance | 16 TFLOPS | 10.5 TFLOPS | 1.33 TFLOPS |
| Price | $699 | $399 | $399 |

Table 1. Specifications for NVIDIA Jetson devices: AGX Xavier, Xavier NX, and TX2.

## 5 EVALUATION

To evaluate VIRTUOSO, we first introduce all multi-branch object detection kernels of VIRTUOSO and baselines in Sec. 5.1, and evaluation dataset and metrics in Sec. 5.2. We then present our evaluation results in the following sections:

[5]The default modes are: mode 7 on AGX Xavier, mode 3 on Xavier NX, and mode 3 on TX2.

| Key Parameters | Power Mode 0 | Power Mode 1 | Power Mode 2 | Power Mode 3 | Power Mode 4 | Power Mode 5 | Power Mode 6 | Power Mode 7 |
|---|---|---|---|---|---|---|---|---|
| Power Budget | N/A | 10W | 15W | 30W | 30W | 30W | 30W | 15W |
| Online CPU Cores | 8 | 2 | 4 | 8 | 6 | 4 | 2 | 4 |
| Maximal CPU Frequency (MHz) | 2265.6 | 1200 | 1200 | 1200 | 1450 | 1780 | 2100 | 2188 |
| Maximal GPU Frequency (MHz) | 1377 | 520 | 670 | 900 | 900 | 900 | 900 | 670 |
| Maximal DL Accelerator Frequency (MHz) | 1395.2 | 550 | 750 | 1050 | 1050 | 1050 | 1050 | 115.2 |

Table 2. Different power modes on the Jetson AGX Xavier board. Each power mode comes with a different maximum power budget, number of online CPU cores, CPU, GPU, and DL accelerator frequency, which gives flexibility in selecting energy efficiency.

| Key Parameters | Power Mode 0 | Power Mode 1 | Power Mode 2 | Power Mode 3 | Power Mode 4 |
|---|---|---|---|---|---|
| Power Budget | 15W | 15W | 15W | 10W | 10W |
| Online CPU Cores | 2 | 4 | 6 | 2 | 4 |
| Maximal CPU Frequency (MHz) | 1900 | 1400 | 1400 | 1500 | 1200 |
| Maximal GPU Frequency (MHz) | 1100 | 1100 | 1100 | 800 | 800 |
| Maximal DL Accelerator Frequency (MHz) | 1100 | 1100 | 1100 | 900 | 900 |

Table 3. Different Power Modes for Jetson Xavier NX.

(1) First, we show the comparison of VIRTUOSO to different baselines. In addition, we evaluate VIRTUOSO under both energy consumption and latency requirements and present the results in Sec. 5.3.
(2) Second, we rigorously investigate the impact of power mode on energy vs. latency tradeoff during the evaluation of VIRTUOSO with different user requirements and show that power modes on embedded devices can be utilized to achieve further optimization on energy or latency performance.
(3) Third, in Sec. 5.6, we present a more in-depth energy consumption analysis on our multi-branch object detection kernels and baselines by utilizing finer-grained power modes.
(4) Finally, we evaluate VIRTUOSO and all baselines comprehensively on different embedded devices under various resource contention scenarios for accuracy, latency, and energy. In Sec. 5.5, we present the impact of runtime environment on object detectors, and further show the effectiveness of VIRTUOSO's adaptive features.

### 5.1 VIRTUOSO Variants and Baselines

We consider 9 baseline video object detection solutions and their variants, with a total of 15 protocols, for our evaluation. These models are selected using the following criteria:

(1) The code and model could be deployed on a Jetson TX2 board, and
(2) The model is open-sourced and can be replicated on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2015 VID dataset [40].

The detailed explanation of each protocol is as follows.

#### 5.1.1 *Adaptive Video Object Detection Models.*

We implement and evaluate a total of 6 adaptive video object detection models.
**EfficientDet D0, D0+ and D3, D3+**: EfficientDet D0+ and D3+ are multi-branch object detection kernels, which is our improvement over EfficientDet D0 and D3 with efficiency knobs. We have four efficiency knobs for EfficientDet D0+ and D3+, as follows: 1) object tracker, 2) input image resolution for the object tracker, 3) confidence threshold to track, and 4) detector interval.
**SSD and SSD+**: SSD+ is another multi-branch object detection kernel, that is SSD as the object detector backbone combined with our efficiency knobs. There are a total of five efficiency knobs for SSD+: 1) input image resolution for the object detector backbone, 2) object tracker, 3) input image resolution for the object tracker, 4) confidence threshold to track, and 5) detector interval.
**FastAdapt**: FastAdapt [21] is an adaptive framework that is able to adapt to different latency requirements. We use FastAdapt as one of the adaptive baselines for performance comparison. FastAdapt uses a single object detector backbone, FRCNN, and also incorporates an object tracker to speed up the average inference latency.
**FRCNN and FRCNN+**: FRCNN+ is our improvement over FRCNN [38] and we added two efficiency knobs for FRCNN+: 1) image shape and 2) number of proposals. With different combinations of the image shape and number of proposals, there are 28 execution branches with the input image resolution in the following set [224, 320, 448, 576] and the number of proposals in the following set [1, 3, 5, 10, 20, 50, 100]. We use the MedianFlow tracker while keeping a detector interval of 8 frames, which is a middle-of-the-range value.
**YOLO and YOLO+**: YOLO+ is our improvement over YOLO [36, 37], combined with some of the efficiency knobs for limited adaptivity. We included one efficiency knob for YOLO, which is the shape of the input image and also used it in tandem with the MedianFlow object tracker [19] for acceleration. There are a total of 12 execution branches determined by the input image resolution in the following set [224, 256, 288, 320, 352, 384, 416, 448, 480, 512, 544, 576]. Similar to FRCNN+, we also limit the detector interval to 8 frames for YOLO+.

#### 5.1.2 *Accuracy-Optimized Video Object Detection Models.*

Several latest video detection models are considered to provide baseline experiments of *non-adaptive* models. These models are optimized for accuracy.
**REPP**: REPP [41] comes with a total of three model variants — YOLOv3, SELSA, and FGFA. However, only their implementation over a YOLOv3 baseline is able to run on a TX2 board. We address this baseline as "REPP with YOLOv3" to avoid confusion with our own YOLO implementations (YOLO and YOLO+).
**SELSA**: For SELSA [47], we utilize ResNet-50 and ResNet-101, with the corresponding variants referred to as SELSA 50 and SELSA 101.
**MEGA**: MEGA [7] is provided with two different object detector backbones with different levels of feature usage. MEGA utilizes a feature aggregation technique to further improve accuracy by capturing the content similarity for both neighboring frames and overall global frames within the video. However, this feature considers and processes multiple frames at runtime, and thus cannot run on embedded devices due to lack of memory. Here, we only use the object detector

backbone provided by the authors with ResNet-50 as the feature extractor and limit the use of feature aggregation.

### 5.2 Evaluation Dataset and Metrics

**Dataset**: We use the ILSVRC 2015 VID validation dataset [40] as the evaluation dataset for the video object detection task—to classify and localize the objects in 30 classes, over 555 videos (176,126 frames in total). For adaptive video object detection baselines, we consider a streaming setting for inference with video frames fed one by one and report the mean latency per video frame. For other methods, we use batch processing for inference with a batch size of 1 to feed the frames one by one.
**Accuracy** of all baselines is measured in terms of mean average precision (mAP), following the widely adapted evaluation protocol [23] on the dataset. mAP is defined as the mean of APs for all classes. AP is the area under the precision-recall curve that measures both localization and classification accuracy by comparing detection bounding boxes against ground-truth boxes using a fixed Intersection-over-Union (IoU) threshold of 0.5.
**Energy consumption** is measured by the native API provided by the Jetson board *tegrastats* [33] utility from NVIDIA. *Tegrastats* provides the information on instantaneous power usage of the CPU and GPU modules, and we convert the power consumption to the overall energy consumption, and then divide it by the number of frames to calculate the average energy consumption per frame. Users can also specify the interval of the execution of *tegrastats*. We use a 1-second interval in our experiments. This choice collects accurate measurements at a fine time granularity while bounding the overhead of the measurement and corresponding impact on the performance during evaluation.
**Latency** of all baselines are measured as per-frame latency over the Group of Frame (GoF) during inference[6], and further aggregated over GoFs of all videos on the dataset.

### 5.3 Satisfying Various Efficiency Requirements

We first present the overall evaluation results of VIRTUOSO and other efficient and adaptive video object detection baselines in Fig. 5. *First*, VIRTUOSO achieves higher accuracy at any latency range between 15 msec to 200 msec. Particularly, we are 10.7% and 5.9% more accurate than FastAdapt at 70 msec and 15 msec latency requirement, 11.0% more accurate than FRCNN+ at 40 msec latency, 13.3% and 12.1% more accurate than YOLO+ at 80 msec and 25 msec latency, 12.8% more accurate and 11.7 msec faster than FRCNN with maximum performance, and 14.4% more accurate and 320.7 msec faster than YOLO, again, with maximum performance. *Second*, VIRTUOSO achieves much wider latency adaptation range from 3.5 to 245.3 msec in which VIRTUOSO leads the accuracy frontier. The adaptation range is 4.5 times larger than FastAdapt, 10 times larger than FRCNN+, and 4.7 times larger than YOLO+. To conclude, VIRTUOSO is able to achieve both superior accuracy and latency while covering a wider range of adaptation, outperforming all other efficient and adaptive baselines.

We further examine the performance of VIRTUOSO given combined latency and efficiency requirements and under different power modes. We show in Fig. 6 a more detailed performance of VIRTUOSO given several energy requirements. We can see that given the most stringent energy requirement—0.5 J per frame, VIRTUOSO is able to achieve between 3.5 and 7.0 msec latency, with 32.5% to 48.3% accuracy accordingly (purple and solid curve). Then, we gradually relax the energy requirement to 1 J (green curve), 10 J (blue curve), and unlimited (red curve) and find that VIRTUOSO achieves higher accuracy (51.2% to 63.9%), at the expense of higher latency (10.6 msec to 245.3 msec). At a more real-world applicable frame rate of 30 FPS (33.3 msec per frame latency requirement),

[6]Due to our tracking-by-detection technique in Sec. 3.1, the latency of the first frame and remaining frames are uneven. Thus, we take the average over a GoF as the temporal latency.

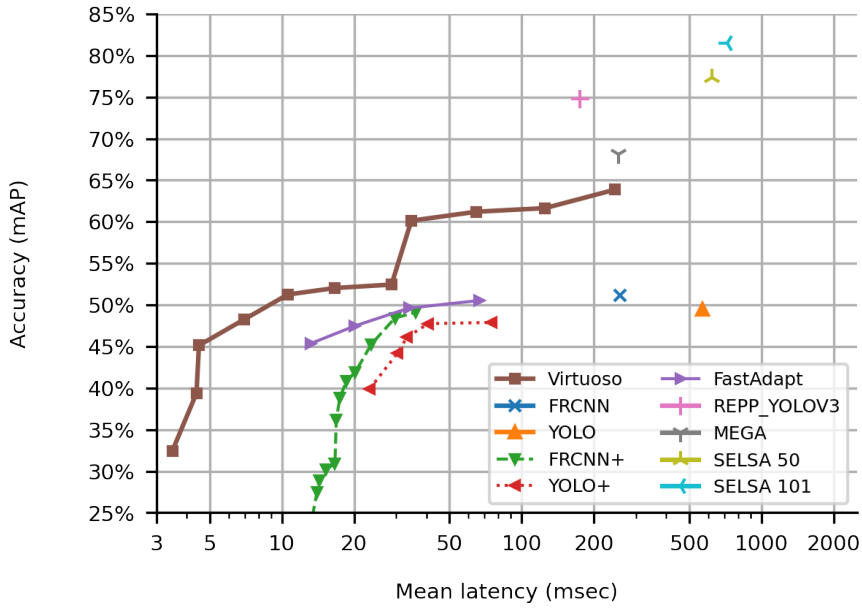



Fig. 5. Comparison of VIRTUOSO with other baseline protocols including our in-house enhancements. VIRTUOSO is able to achieve a much wider range of adaptation with superior accuracy or latency tradeoff compared to all other protocols. Note that the x-axis is in the log scale.

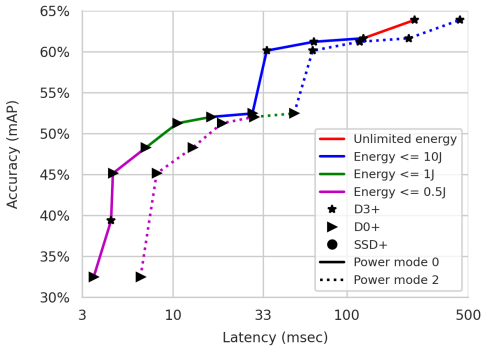



Fig. 6. Performance of VIRTUOSO's ability to adapt to different energy requirements under 2 different power modes, on NVIDIA AGX Xavier. Note that the D3+, D0+, and SSD+ in the legend indicate the kernels that VIRTUOSO has selected for that particular branch.

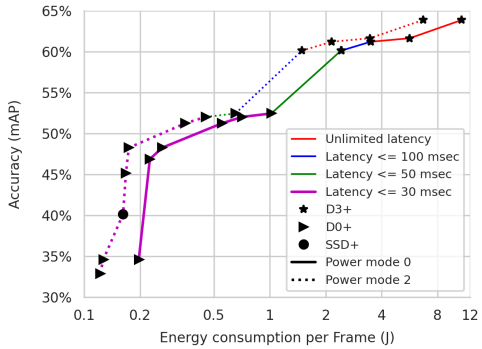



Fig. 7. Performance of VIRTUOSO's ability to adapt to different latency requirements under 2 different power modes, on NVIDIA AGX Xavier. Note that the D3+, D0+, and SSD+ in the legend indicate the kernels that VIRTUOSO has selected for that particular branch.

VIRTUOSO is able to run near 30 FPS (at 36.6 msec per frame) with an accuracy of 60.1%. Therefore, we explicitly show the improvement in accuracy performance at a higher energy requirement with different colors.

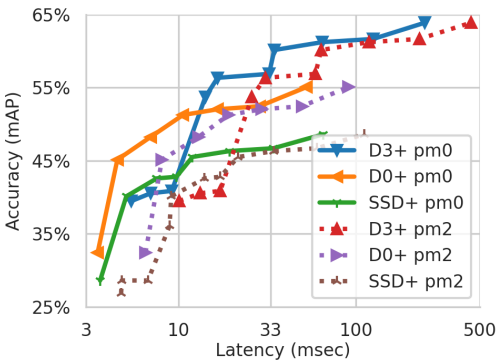



Fig. 8. Accuracy of VIRTUOSO's each object detector backbone given latency requirements, on NVIDIA AGX Xavier.

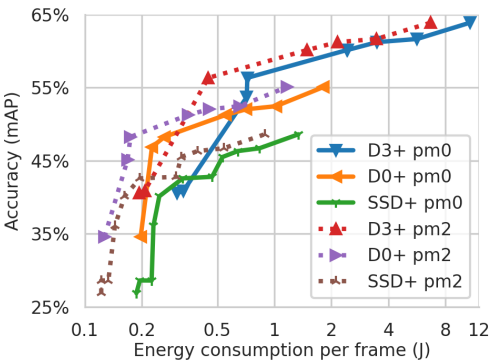



Fig. 9. Accuracy of VIRTUOSO's each object detector backbone given energy requirements, on NVIDIA AGX Xavier.

Furthermore, as we switch the power mode to 2 (from power mode 0) (dashed line), VIRTUOSO is more power efficient and achieves up to 51.2%, 52.0%, and 63.9% accuracy with more relaxed energy requirements of 0.5 J, 1J, and 10 J. The accuracy is higher than that in power mode 0 given the same energy requirement, at the expense of 3 to 4.2 times higher latency. One thing to note that is SSD+ is always inferior to EfficientDet+ in this experiment and thus has no data point in the figure (recall we are plotting the Pareto optimal curve). This is somewhat expected as EfficientDet is more recent work with further optimized accuracy performance. Further, the lower latency region is dominated by EfficientDet D0+ and the higher accuracy region is dominated by EfficientDet D3+. To conclude, the multi-requirement design of VIRTUOSO gives the flexibility to the user for picking different energy or latency requirements to match the use case and VIRTUOSO can always maximize its accuracy subject to such efficiency requirements.

We then evaluate VIRTUOSO with the latency requirement as a prioritized one and check the accuracy vs. energy tradeoff. Fig. 7 shows the adaptation performance of our overall framework for the accuracy vs. energy tradeoff under different latency requirements. While the results for accuracy vs. energy tradeoff show a similar trend to results of accuracy vs. latency, one note is that SSD+ shows up in the Pareto performance curve in power mode 2. This indicates that SSD+ is the preferred choice for some low-energy budget cases due to its high energy efficiency and no object detector can dominate all users' requirements. For all models, as the latency requirement is made stricter, the overall energy consumption of selected branches also decreases, showing a generally applicable relationship between latency and energy consumption.

To better understand the energy and latency performance of VIRTUOSO under different power modes, we show in Fig. 10 a more in-depth result where we select 31 execution branches from the multi-branch execution kernel of VIRTUOSO and show their energy consumption and latency. While energy vs. latency tends to have a linear relationship, it is also observable that the power mode is an important factor that impacts both energy and latency. Roughly, power mode 2 is 40% lower in energy consumption, with a drawback of 1.8 times higher latency compared to power mode 0.

We dig deeper by examining the performance of each object detector backbone. Fig. 8 and 9 show the performance of D3+, D0+ and SSD+ with accuracy vs. latency and accuracy vs. energy respectively. All models show a similar trend of decreasing accuracy with better energy and latency performance, due to the object tracker-related efficiency knobs being more dominant to enhance

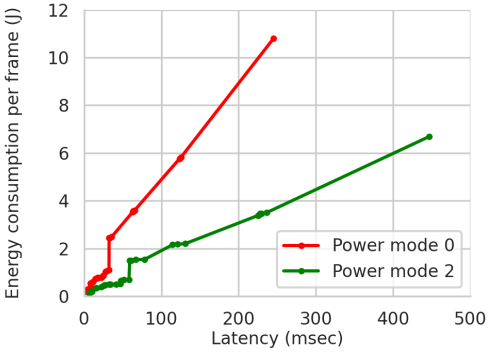



Fig. 10. Energy vs. Latency performance for different power modes, on NVIDIA AGX Xavier. The execution branches that are selected as datapoints in each power mode line are identical, and the difference in the power mode results in a proportional change between latency and energy consumption.

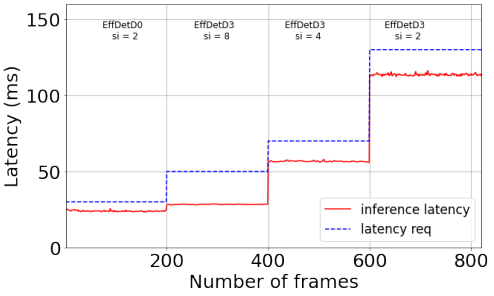



Fig. 11. Performance of VIRTUOSO with dynamically changing user requirements on latency.

the efficiency. For EfficientDet D3, the original model performs at 63.9% accuracy with 245.3 msec latency and 10.8 J energy consumption per frame in power mode 0. This is reduced down to 5.4 msec and 0.3 J per frame which is 45 times faster and 97.2% more energy efficient at the cost of lower accuracy of 39.5%. Similarly, with power mode 0, EfficientDet D0 is 8 times faster and 80.6% more energy efficient with the accuracy dropping from 52.5% to 32.5%. For SSD, it is 9.5 times faster and 77.6% more energy efficient.

The three most utilized knobs in VIRTUOSO are detector interval, resizing factor for the tracker, and confidence threshold for the tracker. The detector interval has the highest impact on both the energy consumption and latency, since the object tracker is lightweight in both energy and computation cost, and also does not utilize the GPU. It impacts both the latency and energy on large scale and is able to reduce the original latency up to 93%. Next is the resizing factor, the input image resolution for the object tracker. It is more effective in the low latency region and can reduce the latency up to 40%. Last is the confidence threshold of the tracker, and on average, it only impacts the latency by a very small value of 8%. However, the confidence threshold becomes important in specific scenarios where there are many objects in the scene to track.

To conclude, our efficiency knobs not only provide benefits in energy efficiency but also improve latency performance. We also show that the power modes on embedded devices can further optimize the latency and energy performance. In addition, VIRTUOSO is able to outperform other baselines due to the following factors. *First*, VIRTUOSO equips a single model with the tuning knobs, enabling the model to achieve different accuracy-latency tradeoffs by selecting different branches at runtime. *Second*, VIRTUOSO further incorporates multiple models, includes an orchestrated scheduler to switch between models and between branches within each model, and thus is able to explore fine-grained accuracy-latency tradeoffs and handle a vast spectrum of user requirements at runtime. In doing so, VIRTUOSO achieves higher accuracy and lower latency in comparison to those baselines.

### 5.4 Evaluation with Dynamic User Requirements

Fig. 11 presents an example of how Fig. 6 and Fig. 7 can be applied to a real-time inference scenario with dynamically changing latency user requirements. The latency user requirement (blue dashed line) is gradually increased as the inference progresses, and with the increased latency budget

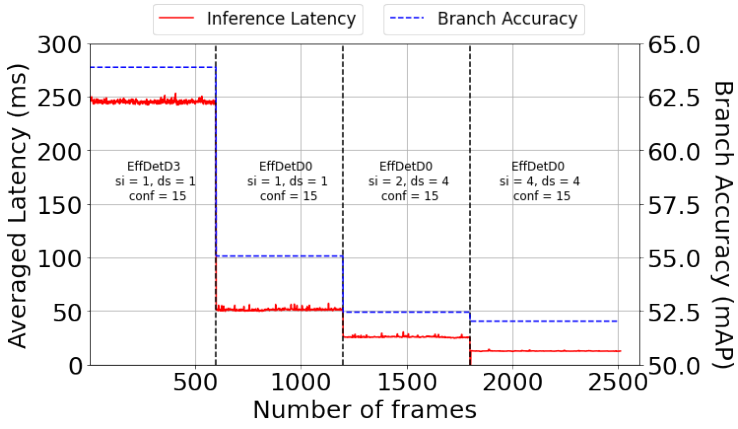



Fig. 12. Latency & Accuracy performance on AGX Xavier. The black dotted vertical line indicates the section divided per 600 frames.

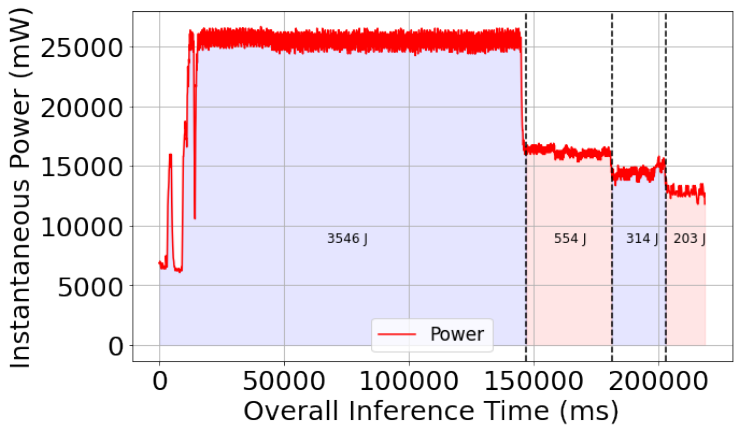
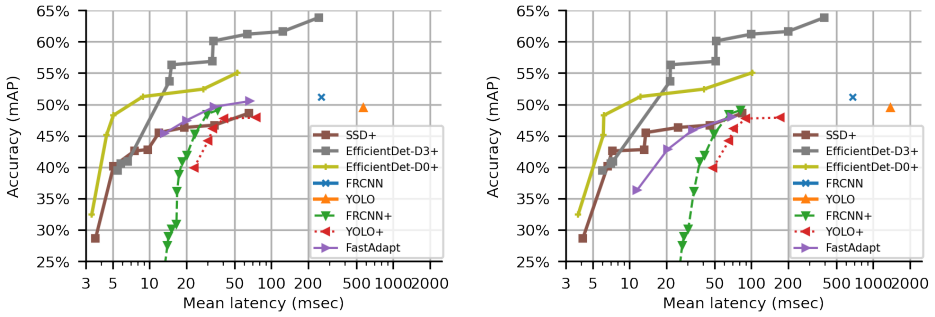



Fig. 13. Instantaneous power on AGX Xavier. The total energy consumption per section (area under the curve) is annotated for each filled region.

available, the scheduler is able to pick a more accurate, but slower branch, which is able to run under the given user requirement. In the beginning, when the latency user requirement is at 30 ms, the scheduler picks a branch with EfficientDet-D0 as the kernel, with a detector interval of 2, having an overall accuracy of 52.45 %. As the inference progresses, the latency user requirement is increased, and the scheduler reacts instantly to the change in the user requirement. At a latency user requirement of 130 ms, the scheduler picks a branch with EfficientDet-D3 as the kernel, with a detector interval of 2, having an overall accuracy of 61.64 %. Note that the branch switching happens without much overhead between different kernels, or between different detector intervals (noted as *si*).

Fig. 12 and Fig. 13 present an example where both the energy and the latency user requirement are dynamically changing. For a sample video of 2513 frames, every 600 frames, a different user requirement is input to the scheduler. Note that except for the last section, all sections consist




(a) No contention (b) 50% GPU contention

Fig. 14. Evaluating the object detector baselines on the NVIDIA Jetson AGX Xavier.

of 600 frames, with the last section being 714 frames. The first 600 frames of the inference are performed without any user requirements, which indicates VIRTUOSO running at max performance - EfficientDet D3 backbone without coupling with the tracker. Next, an energy requirement of 3 $J$ per frame is given to VIRTUOSO, and VIRTUOSO switches to the EfficientDet D0 backbone, matching the given user requirement. For the third section, an additional latency requirement of 30 ms per frame is given, having a 3 $J$ energy requirement as the major user requirement, and a 30 ms latency requirement as the minor requirement. The scheduler selects the execution branch consisting of EfficientDet D0 with the MedianFlow tracker having a tuning knob of confidence threshold of 0.15 and downsampling ratio of 4 for the input resolution of the tracker. Last, to further cut down the energy consumption, the energy user requirement is lowered to 1 $J$ per frame. As shown in Fig. 12, the inference latency is lowered with different branches at the cost of accuracy. For the energy performance, in Fig. 13, the inference time is plotted against the instantaneous power measured on the AGX Xavier board. The black vertical lines indicate the same sections from Fig. 12. As the branch switches down to a more efficient branch, both the overall power level and inference time is decreased, resulting in a big drop of overall energy consumption.

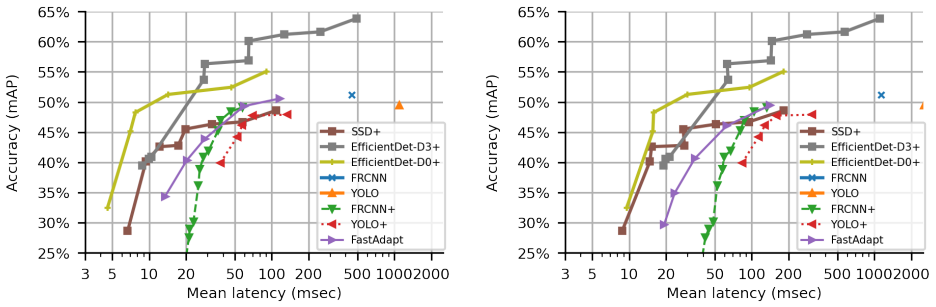
The results suggest that VIRTUOSO is able to adapt to different user requirements during inference, being capable to cope with various real-world problems such as a low battery or real-time latency requirements. Both the energy and latency performance is impacted by different branches selected by the scheduler, and note that the change of branches or kernels comes at almost no computational overhead.

## 5.5 Evaluation on the Accuracy and Latency across Devices

We further evaluate the performance of VIRTUOSO's each object detector backbone on more embedded devices — NVIDIA AGX Xavier, Xavier NX, and TX2.

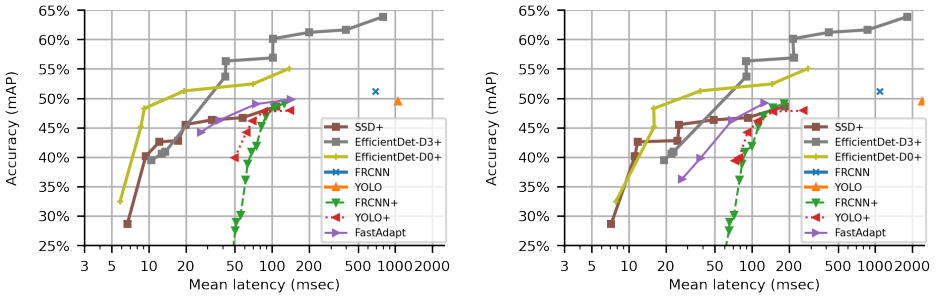
### *5.5.1 Adaptive Video Object Detection Models.*

Fig. 14 reports the accuracy and latency of all adaptive video object detection models with varying contention levels on the NVIDIA Xavier AGX board. Here, we also provide a more in-depth result of VIRTUOSO, by separately evaluating the multi-branch object detection kernels of VIRTUOSO-EfficientDet D0+, D3+ and SSD+.




(a) No contention (b) 50% GPU contention

Fig. 15. Evaluating the object detector baselines on the NVIDIA Jetson Xavier NX.




(a) No contention (b) 50% GPU contention

Fig. 16. Evaluating the object detector baselines on the NVIDIA Jetson TX2.

*First*, in Fig. 14 (a), we observe that EfficientDet D0+, D3+, and SSD+, which are VIRTUOSO variants, have the lead in the accuracy-latency frontier over FastAdapt, FRCNN+, and YOLO+, over a wide range of latency performance, varying from 5.8 msec to 245.3 msec.

Specifically, EfficientDet D0+ has an accuracy up to 55.1% mAP, running at 52.8 msec per frame (roughly 18.9 FPS) and has a latency down to 3.4 msec per frame (roughly 294.1 FPS), running at an accuracy of 32.5% mAP. The maximum performance of EfficientDet D0+ is 5.3% higher in accuracy compared to FastAdapt with the maximum performance while having lower latency of 13.5 msec at the same time. EfficientDet D3+ covers the higher accuracy range with higher latency compared to D0+. It can achieve up to an accuracy of 63.9% mAP, running at 245.3 msec per frame (roughly 4.0 FPS), and the latency can be reduced down to 5.5 msec (roughly 181.8 FPS) with an accuracy of 39.5%, combined with VIRTUOSO's efficiency knobs. For SSD+, the accuracy is at a maximum of 48.6% with a latency of 65.5 msec per frame. The lowest achieved latency is 3.6 msec with an accuracy of 28.6%, which is 0.2 msec slower and 3.9% mAP lower compared to D0+. We also show the latency/accuracy performance of FRCNN+ and YOLO+ versus FRCNN and YOLO, the latter without our optimizations. For FRCNN and YOLO, the latency is 257 and 566 msec per frame while the accuracy is 51.1% and 49.5%, respectively. Coupled with our efficiency knobs, FRCNN+ is able to achieve a real-time processing time of 29.7 msec (33.6 FPS) at an accuracy of 48.4%, which is more than 8 times faster than FRCNN, while only being 2.7% lower in accuracy. Also, FRCNN+ is able to achieve up to 49.1% accuracy with 36.2 msec latency at maximum performance. YOLO+ is able

to achieve a minimum latency of 23.3 msec at 39.9% accuracy, which is more than 24 times faster than YOLO. YOLO+ is also able to achieve 47.9% accuracy at 75.0 msec with maximum accuracy performance.

Fig. 15 (a) and Fig. 16 (a) shows the latency vs. accuracy performance on the Xavier NX and TX2 board. Given a latency requirement of 50 msec (20 FPS), EfficientDet D0+ achieves 52.5% accuracy with 46.9 msec latency per frame. This is 1.2% higher in accuracy compared to the performance on TX2 where EfficientDet D0+ achieves 51.3% accuracy with 38.0 msec latency per frame, under the same latency requirement of 50 msec. Similarly, EfficientDet D3+ achieves 56.3% accuracy with 28.4 msec latency on Xavier NX, where it achieves the same accuracy of 56.3% with 42.2 msec latency on TX2 under the 50 msec latency requirement. Dialing up the latency requirement (more stringent and realistic for video) to 33.3 msec, the accuracy performance on TX2 drops to 40.9% with 13.5 msec latency per frame. This accuracy increase is also observed with other baselines applied with our efficiency knob. Both FRCNN+ and YOLO+ cannot run under 40 msec latency requirement on TX2, but on Xavier NX, FRCNN+ can run with 47.0% accuracy at a latency of 38.3 msec, and YOLO+ can run with 39.9% accuracy at a latency of 38.1 msec.

Comparing with results from the three different devices, we observe that given a stronger computation power, our multi-branch object detection kernels of VIRTUOSO can achieve higher accuracy performance under the same latency requirement. In addition, we were able to observe that a device with higher computation power gives larger benefits to execution branches with heavier object detector backbones, where EfficientDet D3+ is able to run at 53.7% accuracy on AGX Xavier with 15.1 msec latency per frame, whereas it runs with 40.9% accuracy at 13.6 msec latency on TX2 under a real-time latency requirement of 30 msec per frame.

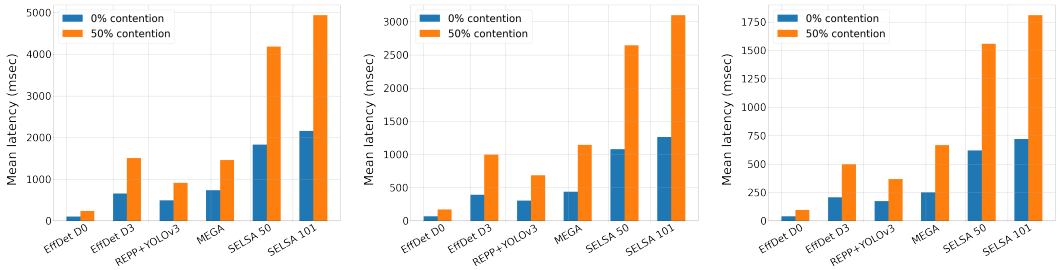
To summarize, first, multi-branch object detection kernels of VIRTUOSO- EfficientDet D0+, D3+, and SSD+ - lead the accuracy-latency performance in all scenarios, especially under stringent latency requirements. Compared to other adaptive video object detection baselines - FastAdapt, FRCNN+, and YOLO+ - EfficientDet D3+ can achieve up to 15.9% better accuracy. Second, utilizing different embedded devices shows that, given a stronger computation power, EfficientDet D0+, D3+, and SSD+ can leverage higher accuracy under a specific latency requirement, due to the latency performance boost. In contrast, non-adaptive baselines only have a single datapoint of accuracy vs. latency and cannot leverage the accuracy with the changing latency performance.

Our results show that VIRTUOSO, which leverages all three multi-branch kernels of EfficientDet D0+, D3+, and SSD+, has the ability to meet the latency requirement more flexibly under various scenarios while maintaining higher accuracy than other baselines.

#### 5.5.2 *Accuracy-Optimized Video Object Detection Models.*

Baselines without adaptive features are evaluated for accuracy and latency on different embedded devices, and we use EfficientDet D0 and D3 as the base for comparison. As we can see from the caption of Fig. 17, without latency requirements, these baselines achieve higher mAP than most adaptive baselines, ranging from 55.1% to 81.5%. For latency results, EfficientDet D0 has the lowest latency and accuracy among all models and boards. SELSA 101 achieves the highest accuracy but with the highest latency on all boards. In contrast, REPP with YOLOv3 has a reasonable accuracy, at 74.81%, while still maintaining relatively low latency. However, when considering a user latency requirement, most baselines have latency over 200 msec, or 5 FPS, even on the AGX Xavier board, which is one of the most powerful embedded boards (subfigure (c)). Such latency values do not meet the need for real-time processing.

It is also noteworthy that, while REPP with YOLOv3 has a reasonable latency alongside with a high accuracy performance, it is a post-processing technique that requires information of all




(a) Jetson TX2 (b) Jetson Xavier NX (c) Jetson AGX Xavier

Fig. 17. Benchmarking SOTA video object detection models on NVIDIA Jetson Devices under different contention levels. The accuracy measurements are the same among all boards and are as follows. EfficientDet D0: 55.07% mAP, EfficientDet D3: 63.87% mAP, REPP+YOLOv3: 74.81% mAP, MEGA base: 68.11% mAP, SELSA 50: 77.31% mAP, SELSA 101: 81.5% mAP. Unlike the adaptive baselines, non-adaptive baselines provide only a single datapoint of accuracy and latency under the given environment.

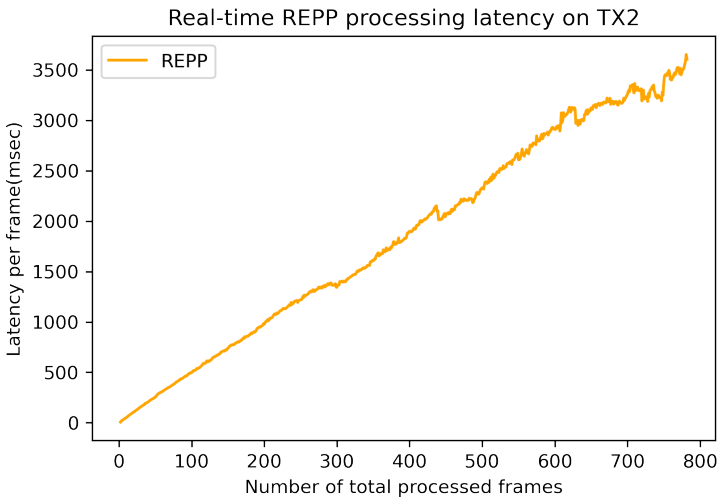



Fig. 18. Latency performance and overhead of REPP postprocessing applied to real-time streaming. As the number of frames increases, the overhead of processing each frame keeps increasing in a linear fashion.

the detection results to refine the bounding boxes for higher accuracy. However, during a video streaming condition in our case, such information of detection results is limited to only the video frame till now, and it would have to be applied frame by frame. We add a simple experiment to showcase this limitation, by running REPP in a real-time fashion, and in Fig. 18 it is shown that detection boxes accumulated over 800 video frames would result in a processing time of up to 3 seconds on a TX2 device, which is unsuitable for real-time processing.

The accuracy values obtained in our evaluation for non-adaptive baselines differ from those of the original authors, most significantly for REPP. This, we believe, is due to two reasons, as follows: First, we use an IoU threshold of 0.6 during the non-max suppression phase, consistently for all protocols which are mostly used as the default for our adaptive baselines (original authors use different thresholds for different protocols, e.g., 0.5 for REPP). Second, we use a streaming setting

of input frames, where the information to future frames that are required for frame aggregation of these non-adaptive baselines, is limited.

To conclude, we show in Fig. 17, that the 50% contention on the embedded device impacts the latency performance of each object detection kernel, and increases the latency by roughly two times. Also, a similar trend of better latency performance of using a device with a stronger computation power is observed as in Fig. 14 to Fig. 16. However, unlike our adaptive kernels of EfficientDet D0+, D3+ and SSD+, non-adaptive baselines do not have any ability to meet the latency requirement, and show a very high latency of over 300 msec, under 50% GPU contention even on the AGX Xavier board.

### 5.6 Evaluation of Energy Consumption

Embedded devices need to be energy efficient as in most cases they may need to run on limited power. For more in-depth profiling and comparison of energy consumption of different baselines with different power modes, we explicitly select the Xavier NX board as it is our middle-of-the-computational-power device and also comes with a moderate selection of power modes.

| **Power Modes** | *D0+* | *D0* | *D3+* | *D3* | *SSD+* | *SSD* | *FRCNN+* | *FRCNN* | *YOLO+* | *YOLO* | *Fast Adapt* | *REPP w YOLOv3* | *MEGA* | *SELSA 50* | *SELSA 101* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 267 | 1050 | 882 | 4786 | 295 | 692 | 2205 | 6567 | 2196 | 14270 | 2244 | 3427 | 5269 | 12233 | 14565 |
| 2 | 305 | 1183 | 937 | 5050 | 346 | 757 | 2296 | 6745 | 2246 | 14407 | 2293 | 3490 | 5368 | 12519 | 14967 |
| 4 | 297 | 1025 | 863 | 4323 | 347 | 716 | 2064 | 5906 | 1914 | 11836 | 2069 | 3122 | 4528 | 10493 | 12451 |
| Avg | 290 | 1086 | 894 | 4720 | 329 | 722 | 2189 | 6406 | 2119 | 13504 | 2202 | 3346 | 5055 | 11748 | 13995 |

Table 4. Energy consumption (J) on Jetson Xavier NX. Overall, our in-house baselines with our efficiency knobs show a huge efficiency boost in energy consumption ranging from 65% to 81% reduction, compared to the baselines without our efficiency knobs.

**Experiment setup**: With the selected power modes of Xavier NX from Section. 4.3, we first measure the power in the idle status. The results are 3.25, 3.38, and 3.08$W$ in modes 0, 2, and 4 respectively. Then, we run each video object detection baseline on a randomly selected video, which has 828 frames. The results are shown in Fig. 19 and Table 4.

**Results**: Efficient object detector backbones (SSD, SSD+, EfficientDet D0, D0+, D3, D3+) consume lower instantaneous power measured in real-time. As shown in Figure 19, the maximum peak power and the overall power level during inference is much lower up to 33%, compared to other baselines. Among all baselines, EfficientDet D0+, D3+, and SSD+ have a superior performance energy-wise, especially with SSD+ and EfficientDet D0+ both having less than 350 $J$. Compared to its original detector backbone, EfficientDet D0+ on average consumes 290 $J$, whereas EfficientDet D0 consumes 1086 $J$, which is almost 4 times larger. The difference is bigger for EfficientDet D3+ and D3, where D3 consumes more than 5 times the energy compared to D3+. SSD being a very lightweight detector backbone consumes only 722 $J$ on average, but still, SSD+ is able to cut the energy consumption down to 329 $J$ on average. In addition, FRCNN+, and YOLO+ have low average total energy consumption at around 2,200 $J$, compared to FRCNN and YOLO. REPP with YOLOv3 consumed 3,346 $J$ on average while SELSA has the highest average energy consumption of 13,995 $J$, which is more than 10 times larger than adaptive baselines. Our overall energy consumption evaluations validate our insight that adaptive baselines EfficientDet D0+, D3+, SSD+, FRCNN+, and YOLO+ demonstrate superior energy efficiency relative to their rigid variants. Since the major part of power consumption comes from the GPU module, and EfficientDet D0+, D3+ SSD+, FRCNN+, and YOLO+ which is our implementation leveraging an object tracker, we notice significant oscillations in their energy plots, compared to their original implementations (Fig. 19). This is because the object tracker mainly uses the CPU and is more energy efficient. Switching between the object detector

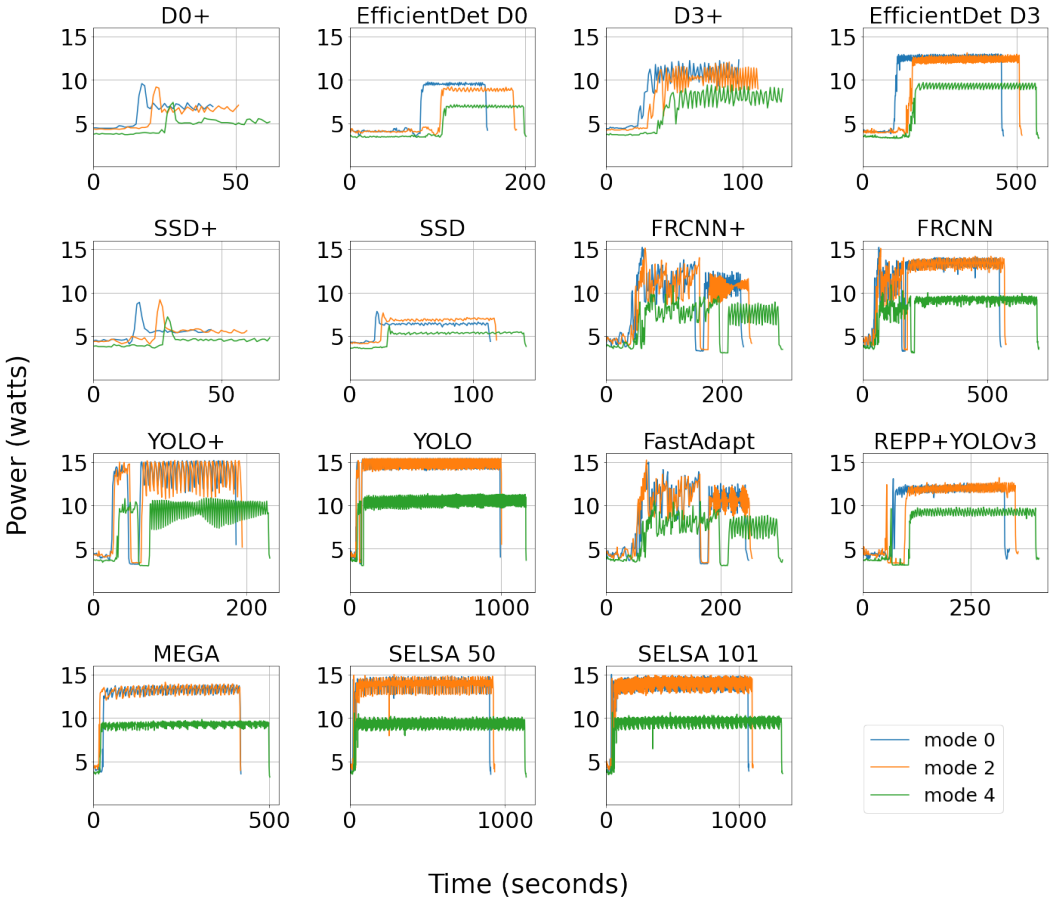
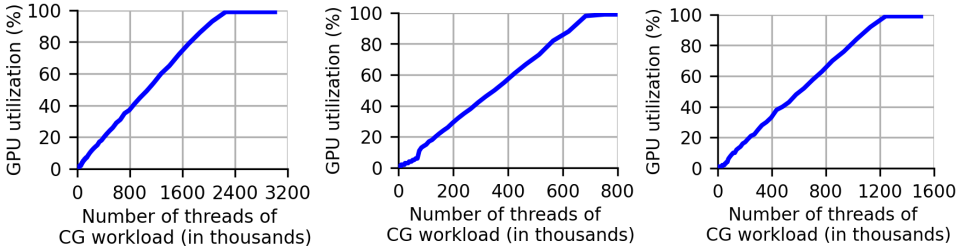



Fig. 19. Instantaneous power on Xavier NX. Overall, our in-house baselines with our efficiency knobs show a lower average power level compared to its original backbone without any efficiency knobs. Heavier and accuracy-oriented models show higher instantaneous power and longer inference times.

backbone (mainly executed on GPU) and object tracker (mainly executed on CPU) corresponds to the oscillations in the curve.

We further investigate the impact of different power modes on the energy consumption of the 15 protocols. We can see from Fig. 19 and Table 4 that all models in power mode 0 achieve lower latency than those in power modes 2 and 4, with the overall highest energy consumption. All models in power mode 4 have the slowest inference latency with the lowest instantaneous power level compared with those in power modes 0 and 2. Mainly, compared with mode 0, power mode 4 helps reduce the instantaneous power on an average over models, by around 30% and total energy consumption by around 10% despite its longer average inference time of 18%.

Overall, our efficiency knobs are able to cut down the energy consumption of object detection backbones significantly, showing at least 60% decreased energy consumption for all adaptive baselines - EfficientDet D0+, D3+, SSD+, FRCNN+, and YOLO+ compared to its counterparts. In addition, our evaluation of the energy consumption difference between different power modes suggests that the power mode can be utilized as another efficiency knob according to the user's




(a) AGX Xavier (b) Xavier NX (c) TX2

Fig. 20. Calibration of the contention generator on each embedded device.

requirements. Specifically, for video object detection tasks, the users can make their choice as to whether to focus on latency performance by choosing a higher performance power mode (e.g., mode 0) or energy saving by switching to a lower performance power mode (e.g., mode 4) of the embedded devices.

### 5.7 Additional Evaluation with Contention

To further evaluate the performance of VIRTUOSO and other baseline models in a more realistic scenario, we design and implement a synthetic contention generator (CG) to inject tunable levels of GPU resource contention for the object detection models. The CG is a stand-in for the general background and concurrent workloads executed on the device, which concomitantly consume GPU resources. With the CG, we are able to profile the performance of all models under different resource contention scenarios. The CG occupies designated levels of resources on the GPU module of the embedded device by a percentage of the maximum capacity. To achieve this goal, the CG on the GPU performs *add operation* with a CUDA kernel function. By changing the number of threads of the CG workload, we control the number of GPU cores that are kept busy per second. Thus, we are able to occupy different amounts of GPU resources and we call the amount of GPU resource that the CG occupies the *GPU contention level*. We choose 3 GPU contention levels [0%, 20%, 50%] to simulate the resource competition in the background. Since each embedded device comes with a different GPU architecture, the number of cores, and computation capabilities, a CG workload with a certain number of threads results in different GPU contention levels on different boards. We calibrate the CG on each embedded device (namely, NVIDIA Jetson AGX Xavier, Xavier NX, and TX2 boards), to produce a consistent level of contention. We show our calibration experiment results of the CG on AGX Xavier, Xavier NX, and TX2 in Fig. 20. As can be seen, with the increasing number of threads of the CG workload, the GPU contention level (GPU utilization) keeps increasing and finally saturates at 99% as the number of threads is large enough. One key observation is that the relationship between the number of threads of CG workload and the GPU contention level is linear, which makes our CG a small linear control system.

#### 5.7.1 *Evaluation of under Contention.*

In this section, we evaluate all of our baseline protocols and accuracy-oriented baselines with contention, to see the impact of contention that exists in real-world scenarios.

In Fig. 14 (b), we measure the performance of the protocols under 50% GPU contention. Under 50% GPU contention, most of the latency performance of each execution branch is increased by roughly 2 times, compared to no contention. The D0+ branch that runs with 19.3 msec latency per frame under no contention, runs at 38 msec per frame with the same accuracy of 51.3%. The D3+

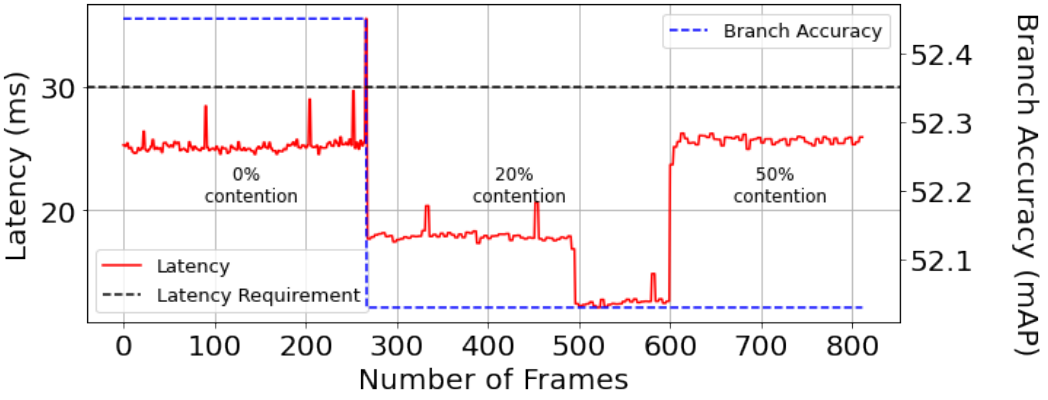



Fig. 21. Accuracy and latency of VIRTUOSO when adapting to changing contention from 0% –> 20% –> 50%.

branch that runs with 13.5 msec latency per frame under no contention runs at 23.0 msec per frame with 41% accuracy. A similar trend of increased latency is also observed for SSD+, FRCNN+, and YOLO+. The increased latency under contention reduces the accuracy performance under real-time processing latency of 30 FPS. SSD+ runs at 24.9 msec latency per frame with an accuracy of 46.3%, and FRCNN+ runs at 30.4 msec latency per frame with 30.2% accuracy. YOLO+ was not able to meet the 30 FPS latency requirement, but however, was still 28 times faster than YOLO with the fastest configuration, which is 48.7 msec, compared to YOLO's 1385.6 msec. Fig. 15 (b) shows the evaluation results on Xavier NX with contention. We observe increased latency of all baselines as the contention is given and due to the less computation power on Xavier NX, it is naturally harder to maintain a real-time latency performance. However, EfficientDet D0+, D3+, and SSD+ can still keep the real-time 30 FPS (33.3 msec per frame) with the accuracy of 51.3%, 40.9%, and 45.5% on Xavier NX when there is 50% GPU contention. Note that EfficientDet D0+ has a higher accuracy of 51.3% compared to EfficientDet D3+'s 40.9% under a stringent latency requirement. Such accuracy vs. latency tradeoff benefits VIRTUOSO with all multi-branch kernels combined, being able to switch between kernels to achieve an overall higher Pareto curve.

The impact of contention is more severe with accuracy-oriented models, as shown in Fig. 17. We find from our evaluation that all baselines, without any adaptive features, suffer from poor latency, further exacerbated under GPU contention, showing up to 2X increased latency with contention. This is because of their larger network sizes, sophisticated design of using frame aggregation as a post-processing step, and inability to adapt to the runtime environment.

To further explore VIRTUOSO's ability to adapt, we implement a simple heuristic that checks consecutive latency violations by a constant ratio during inference. We define two consecutive latency violations with a similar ratio as an effect caused by contention and divide the user requirement by this ratio for calibration. Fig. 21 shows that with this modified scheduler, VIRTUOSO is able to conform to the real-time user requirement of 30 FPS, even with increasing contention at a small accuracy penalty. These results demonstrate that VIRTUOSO can be applied to real-world applications, with contending background processes being a common occurrence.

Overall, we find that the multi-branch object detection kernels of VIRTUOSO can maintain real-time latency performance with reasonable accuracy even under resource contention. While GPU contention impacts the overall latency performance for all baselines regardless of the device being used, unlike non-adaptive baselines where the latency increased up to 2X, EfficientDet D0+, D3+,

and SSD+ were able to maintain a real-time latency of 19.3 msec, 13.5 msec, and 23.0 msec on the AGX device with 51.3%, 41%, 46.3% accuracy, respectively.

## 6 CONCLUSION

In this paper, we have proposed VIRTUOSO, an adaptive video object detection framework that consists of an object detector, object tracker, and a dynamic scheduler. A total of 8 different efficiency knobs are coupled with EfficientDet D0, D3, and SSD as the object detector backbones to create multiple execution branches. Further, our dynamic scheduler is able to predict the best performing branch at runtime. We evaluate VIRTUOSO from multiple perspectives, considering energy consumption, latency, and accuracy performance, alongside 15 different baselines. Among all evaluated baselines, VIRTUOSO is able to achieve the best Pareto optimal performance curve, covering a wide spectrum of performance tradeoffs, with an inference time down to as low as 3.5 msec on the Xavier AGX board, and accuracy up to 63.87%. Moreover, VIRTUOSO is able to show dynamic switching of branches with the flexibility to meet various user requirements. It is also observed that different power modes are able to provide further tradeoff of energy versus latency. For example, among our experimental settings of using AGX Xavier with power modes 0 and 2, we were able to achieve a 40% reduction in energy.

We take one step further and perform a more in-depth evaluation of our multi-branch object detection kernels — EfficientDet D0+, D3+, and SSD+ with the baselines on different runtime environment scenarios. Specifically, we evaluate using different embedded devices, different contention levels, and different power modes. While all scenarios had an impact on latency performance, our in-house adaptive baselines, coupled with our efficiency knobs, were able to meet certain user requirements at an acceptable accuracy. In contrast, all non-adaptive baselines suffer from a severe drop in latency performance, and more so, under GPU contention.

In the aspect of energy consumption, we show that using our efficiency knobs, our adaptive baselines — EfficientDet D0+, D3+, SSD+, FRCNN+, and YOLO+ were at least 60% more energy efficient compared to their non-adaptive counterparts.

While VIRTUOSO is mainly evaluated on Jetson NVIDIA boards, we would like to mention that evaluating on different mobile platforms such as TPUs, and FPGAs could be a potential future work to expand the application of adaptive computer vision.

We hope that this work points to further work in understanding the suitability of various object detection kernels on embedded boards. This understanding must encompass varying levels of resource availability on these devices as well as varying power modes of operation available on these devices.

## 7 ACKNOWLEDGMENTS

This material is based in part upon work supported by the National Science Foundation under Grant Numbers CNS-2038986/2038566, CNS-2146449 (NSF CAREER award), an Amazon Research Award, and funding from the Army Research Lab (Contract number W911NF-2020-221). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors. The authors also thank the reviewers for their enthusiastic comments and insightful feedback that improved the clarity and presentation of our work.

## REFERENCES

[1] Kittipat Apicharttrisorn, Xukan Ran, Jiasi Chen, Srikanth V Krishnamurthy, and Amit K Roy-Chowdhury. 2019. Frugal following: Power thrifty object detection and tracking for mobile augmented reality. In *Proceedings of the 17th Conference on Embedded Networked Sensor Systems*. 96–109.

[2] Eduardo Arnold, Omar Y Al-Jarrah, Mehrdad Dianati, Saber Fallah, David Oxtoby, and Alex Mouzakitis. 2019. A survey on 3d object detection methods for autonomous driving applications. *IEEE Transactions on Intelligent Transportation Systems* 20, 10 (2019), 3782–3795.

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

[4] Mark Buckler, Suren Jayasuriya, and Adrian Sampson. 2017. Reconfiguring the imaging pipeline for computer vision. In *Proceedings of the IEEE International Conference on Computer Vision*. 975–984.

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

[6] Kai Chen, Jiaqi Wang, Shuo Yang, Xingcheng Zhang, Yuanjun Xiong, Chen Change Loy, and Dahua Lin. 2018. Optimizing video object detection via a scale-time lattice. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 7814–7823.

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

[8] Ting-Wu Chin, Ruizhou Ding, and Diana Marculescu. 2019. Adascale: Towards real-time video object detection using adaptive scaling. *Proceedings of Machine Learning and Systems* 1 (2019), 431–441.

[9] Jason Clemons, Haishan Zhu, Silvio Savarese, and Todd Austin. 2011. MEVBench: A mobile computer vision benchmarking suite. In *2011 IEEE international symposium on workload characterization (IISWC)*. IEEE, 91–102.

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

[11] Jiajun Deng, Yingwei Pan, Ting Yao, Wengang Zhou, Houqiang Li, and Tao Mei. 2019. Relation distillation networks for video object detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 7023–7032.

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

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

[14] Di Feng, Christian Haase-Schütz, Lars Rosenbaum, Heinz Hertlein, Claudius Glaeser, Fabian Timm, Werner Wiesbeck, and Klaus Dietmayer. 2020. Deep multi-modal object detection and semantic segmentation for autonomous driving: Datasets, methods, and challenges. *IEEE Transactions on Intelligent Transportation Systems* 22, 3 (2020), 1341–1360.

[15] Asish Ghoshal, Ananth Grama, Saurabh Bagchi, and Somali Chaterji. 2015. 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*. 403–412.

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

[17] João F Henriques, Rui Caseiro, Pedro Martins, and Jorge Batista. 2014. High-speed tracking with kernelized correlation filters. *IEEE transactions on pattern analysis and machine intelligence* 37, 3 (2014), 583–596.

[18] Zhengkai Jiang, Yu Liu, Ceyuan Yang, Jihao Liu, Peng Gao, Qian Zhang, Shiming Xiang, and Chunhong Pan. 2020. Learning where to focus for efficient video object detection. In *European Conference on Computer Vision*. Springer, 18–34.

[19] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. 2010. Forward-backward error: Automatic detection of tracking failures. In *2010 20th international conference on pattern recognition*. IEEE, 2756–2759.

[20] Kiran Kale, Sushant Pawar, and Pravin Dhulekar. 2015. Moving object tracking using optical flow and motion vector estimation. In *2015 4th international conference on reliability, infocom technologies and optimization (ICRITO)(trends and future directions)*. IEEE, 1–6.

[21] Jayoung Lee, Pengcheng Wang, Ran Xu, Venkat Dasari, Noah Weston, Yin Li, Saurabh Bagchi, and Somali Chaterji. 2021. Benchmarking Video Object Detection Systems on Embedded Devices under Resource Contention. In *Proceedings of the 5th International Workshop on Embedded and Mobile Deep Learning*. 19–24.

[22] Buyu Li, Wanli Ouyang, Lu Sheng, Xingyu Zeng, and Xiaogang Wang. 2019. Gs3d: An efficient 3d object detection framework for autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 1019–1028.

[23] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. 2014. Microsoft coco: Common objects in context. In *European conference on computer vision*. Springer, 740–755.

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

[25] Alan Lukezic, Tomas Vojir, Luka ˇCehovin Zajc, Jiri Matas, and Matej Kristan. 2017. Discriminative correlation filter with channel and spatial reliability. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 6309–6318.

[26] Chunjie Luo, Fan Zhang, Cheng Huang, Xingwang Xiong, Jianan Chen, Lei Wang, Wanling Gao, Hainan Ye, Tong Wu, Runsong Zhou, et al. 2018. AIoT bench: towards comprehensive benchmarking mobile and embedded device intelligence. In *International Symposium on Benchmarking, Measuring and Optimization*. Springer, 31–35.

[27] Ashraf Mahgoub, Alexander Michaelson Medoff, Rakesh Kumar, Subrata Mitra, Ana Klimovic, Somali Chaterji, and Saurabh Bagchi. 2020. {OPTIMUSCLOUD}: Heterogeneous Configuration Optimization for Distributed Databases in the Cloud. In *2020 {USENIX} Annual Technical Conference ({USENIX}{ATC} 20)*. 189–203.

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

[29] NVIDIA Corporation. 2020. NVIDIA Jetson AGX Xavier Board. https://developer.nvidia.com/embedded/jetson-agx-xavier-developer-kit.

[30] NVIDIA Corporation. 2020. NVIDIA Jetson Linux Developer Guide. https://docs.nvidia.com/jetson/l4t/index.html#page/Tegra%20Linux%20Driver%20Package%20Development%20Guide/power_management_jetson_xavier.html#wwpID0E0VO0HA.

[31] NVIDIA Corporation. 2020. NVIDIA Jetson TX2 Board. https://developer.nvidia.com/embedded/jetson-tx2.

[32] NVIDIA Corporation. 2020. NVIDIA Jetson Xavier NX Board. https://developer.nvidia.com/embedded/jetson-xavier-nx-devkit.

[33] NVIDIA Corporation. 2020. Tegrastats Utility. https://docs.nvidia.com/jetson/archives/l4t-archived/l4t-3231/index.html#page/Tegra%20Linux%20Driver%20Package%20Development%20Guide/AppendixTegraStats.html.

[34] Murad Qasaimeh, Kristof Denolf, Alireza Khodamoradi, Michaela Blott, Jack Lo, Lisa Halder, Kees Vissers, Joseph Zambreno, and Phillip H Jones. 2021. Benchmarking vision kernels and neural network inference accelerators on embedded platforms. *Journal of Systems Architecture* 113 (2021), 101896.

[35] Jinmeng Rao, Yanjun Qiao, Fu Ren, Junxing Wang, and Qingyun Du. 2017. A mobile outdoor augmented reality method combining deep learning object detection and spatial relationships for geovisualization. *Sensors* 17, 9 (2017), 1951.

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

[37] Joseph Redmon and Ali Farhadi. 2018. YOLOv3: An Incremental Improvement. https://doi.org/10.48550/ARXIV.1804.02767

[38] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*. 91–99.

[39] Ali Rohan, Mohammed Rabah, and Sung-Ho Kim. 2019. Convolutional neural network-based real-time object detection and tracking for parrot AR drone 2. *IEEE access* 7 (2019), 69575–69584.

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

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

[42] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4510–4520.

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

[44] Mingxing Tan and Quoc Le. 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*. PMLR, 6105–6114.

[45] Mingxing Tan, Ruoming Pang, and Quoc V Le. 2020. EfficientDet: Scalable and efficient object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 10781–10790.

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

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

[48] Ran Xu, Rakesh Kumar, Pengcheng Wang, Peter Bai, Ganga Meghanath, Somali Chaterji, Subrata Mitra, and Saurabh Bagchi. 2021. ApproxNet: Content and Contention-Aware Video Object Classification System for Embedded Clients. *ACM Trans. Sen. Netw.* 18, 1, Article 11 (oct 2021), 27 pages. https://doi.org/10.1145/3463530

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

[50] Chun-Han Yao, Chen Fang, Xiaohui Shen, Yangyue Wan, and Ming-Hsuan Yang. 2020. Video object detection via object-level temporal aggregation. In *European conference on computer vision*. Springer, 160–177.

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

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

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

[54] Yuhao Zhu, Anand Samajdar, Matthew Mattina, and Paul Whatmough. 2018. Euphrates: Algorithm-SoC Co-Design for Low-Power Mobile Continuous Vision. In *Proceedings of the 45th Annual International Symposium on Computer Architecture* (Los Angeles, California) *(ISCA '18)*. IEEE Press, 547–560. https://doi.org/10.1109/ISCA.2018.00052