Elf: Accelerate High-resolution Mobile Deep Vision with Content-aware Parallel Offloading

Wuyang Zhang
wuyang@winlab.rutgers.edu
Rutgers University
Piscataway, NJ, USA

Zhezhi He
zhezhi.he@sjtu.edu.cn
Shanghai Jiao Tong University
Shanghai, China

Luyang Liu
luyangliu@google.com
Google Research
Mountain View, CA, USA

Zhenhua Jia
zjia@nvidia.com
NVIDIA Corporation
Holmdel, NJ, USA

Yunxin Liu
yunxin.liu@microsoft.com
Microsoft Research
Beijing, China

Marco Gruteser
gruteser@google.com
Google Research
Mountain View, CA, USA

Dipankar Raychaudhuri
ray@winlab.rutgers.edu
Rutgers University
Piscataway, NJ, USA

Yanyong Zhang
yanyongz@ustc.edu.cn
Corresponding author, University of Science and Technology of China
Hefei, China

ABSTRACT

As mobile devices continuously generate streams of images and videos, a new class of mobile deep vision applications are rapidly emerging, which usually involve running deep neural networks on these multimedia data in real-time. To support such applications, having mobile devices offload the computation, especially the neural network inference, to edge clouds has proved effective. Existing solutions often assume there exists a dedicated and powerful server, to which the entire inference can be offloaded. In reality, however, we may not be able to find such a server but need to make do with less powerful ones. To address these more practical situations, we propose to partition the video frame and offload the partial inference tasks to multiple servers for parallel processing. This paper presents the design of Elf, a framework to accelerate the mobile deep vision applications with any server provisioning through the parallel offloading. Elf employs a recurrent region proposal prediction algorithm, a region proposal centric frame partitioning, and a resource-aware multi-offloading scheme. We implement and evaluate Elf upon Linux and Android platforms using four commercial mobile devices and three deep vision applications with ten state-of-the-art models. The comprehensive experiments show that Elf can speed up the applications by 4.85× with saving bandwidth usage by 52.6%, while with <1% application accuracy sacrifice.

ACM MobiCom '21, October 25–29, 2021, New Orleans, LA, USA

ACM ISBN 978-1-4503-8342-4/21/10…$15.00
https://doi.org/10.1145/3447993.3448628

CCS CONCEPTS

• **Computer systems organization → Real-time system architecture; Distributed architectures**.

ACM Reference Format:
Wuyang Zhang, Zhezhi He, Luyang Liu, Zhenhua Jia, Yunxin Liu, Marco Gruteser, Dipankar Raychaudhuri, and Yanyong Zhang. 2021. Elf: Accelerate High-resolution Mobile Deep Vision with Content-aware Parallel Offloading. In *The 27th Annual International Conference on Mobile Computing and Networking (ACM MobiCom '21), October 25–29, 2021, New Orleans, LA, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3447993.3448628

1 INTRODUCTION

In the past few years, we have witnessed the rapid development of Deep Neural Networks (DNNs), due to the fast-growing computation power and data availability [1]. Thanks to these advancements, mobile applications, particularly mobile vision applications, enjoy a performance boost in various vision-related tasks such as photo beautification, object detection and recognition, and reality augmentation [2]. However, to achieve state-of-the-art performance, DNN models (e.g., [3, 4]) usually have complicated structures with numerous parameters, hence a high demand in computation and storage. As a result, it is challenging to run full-size DNN models on mobile devices, even running into heat dissipation issues. Meanwhile, mobile deep vision applications are often interactive and require fast or even real-time [1] responses. Examples include adversarial point cloud generation [5] that reconstructs 3D scenes for intuitive surrounding interpretation and video object semantic segmentation [6] that facilitates personal activity recognition. In these cases, it is hard, if not impossible, to satisfy the applications' latency requirements due to the limited processing capacity on mobile devices.

To this end, researchers have spent a great deal of effort to improve the performance of mobile deep vision applications. On

[1] Frame rate required for real-time processing is application dependent.

the one hand, various techniques have been developed to make DNN models smaller to reduce the computation load, e.g., weight and branch pruning and sharing [7, 8], tensor quantization [9, 10], knowledge distillation [11], and network architecture search [12]. However, these techniques often lead to compromised model accuracy due to the fundamental trade-off between model size and model accuracy [13]. On the other hand, people have proposed to increase the computing resources by using massive accelerators, such as GPU, FPGA [14] and ASIC [15]. Nevertheless, due to the fundamental limits of size and power, mobile devices still fall short to meet the requirements of target applications.

To solve these challenges, several offloading approaches have been proposed [16–22]. By offloading the intensive model inference to a powerful edge server, for example, AWS Wavelength [23], the inference latency can be significantly reduced. With the high bandwidth and low latency provided by the emerging 5G networks [24], offloading is promising to provide a good user experience for mobile deep vision applications. However, existing offloading methods are insufficient in two aspects. Firstly, most existing solutions use low-resolution images through the entire pipeline, which makes the inference task lightweight, but lose the opportunity to leverage the rich content of high resolution (e.g., 2K or 4K) images/frames. Taking advantage of such rich information is important for applications such as video surveillance for crowded scenes [25], real-time Autopilot system [26], and online high-resolution image segmentation [27]. Secondly, most existing methods only consider offloading tasks between a single pair of server and client, assuming that no competing clients or extra edge resources available. In practice, a single edge server is equipped with costly hardware, for example, Intel Xeon Scalable Processors with Intel Deep Learning Boost [28] or NVIDIA EGX A100 [29], which are typically shared by multiple clients (i.e., multi-tenant environment). Moreover, the heterogeneous resource demands of applications running on edge servers [30] and highly dynamic workloads by mobile users [31] lead to resource fragmentation. If the fragmentation cannot be efficiently utilized, it may produce significant resource waste across edge servers.

To this point, in order to meet the latency requirements of deep mobile vision applications with heterogeneous edge computing resources, *it is advantageous to offload smaller inference tasks in parallel to multiple edge servers*. This mechanism can benefit many real-world deep vision tasks, including multi-people keypoint detection for AR applications and multi-object tracking for autonomous driving tasks [32], where objects can be distributed to different servers for parallel task processing. Meanwhile, offloading to multiple servers imposes several challenges. Firstly, it requires the client to effectively partition the inference job into multiple pieces while maintaining the inference accuracy. In the case of keypoint detection or instance segmentation, simply partitioning a frame into several slices may split a single instance into multiple slices, therefore, dramatically decreasing the model accuracy. Secondly, the system needs to be aware of available computation resources on each server and dynamically develops the frame partitioning solution, so that it can ensure no server in the parallel offloading procedure to become the bottleneck. Finally, such a system should have a general framework design that is independent of its host deep vision applications.

To address the aforementioned challenges, we propose and design **Elf**[2], a framework to accelerate high-resolution mobile deep vision offloading in heterogeneous client and edge server environment, by distributing the computation to available edge servers adaptively. Elf adopts three novel techniques to enable both low latency and high quality of service. To eliminate the accuracy degradation caused by the frame partitioning, we first propose a content-aware frame partitioning method. It is promoted by a fast recurrent region proposal prediction algorithm with an attention-based LSTM network that predicts the content distribution of a video frame. Additionally, we design a region proposal indexing algorithm to keep track of the motion across frames and a low resolution compensation solution to handle new objects when first appear. Both work jointly to help understand frame contents more accurately. Finally, Elf adopts lightweight approaches to estimate the resource capacity of each server and dynamically creates frame partitions based on the resource demands to achieve load balance. Overall, Elf is designed as a plug-and-play extension to the existing deep vision networks and requires minimal modifications at the application level. We have implemented Elf on commercial off-the-shelf servers and four mobile platforms in Linux and Android OS, supporting Python, C++, and Java deep vision applications. We make our code open source and available in https://github.com/wuyangzhang/elf.

The main contributions of this paper are as follows.

- To the best of our knowledge, we are the first to propose a high-resolution mobile deep vision task acceleration system that offloads the computation to multiple servers to minimize the end-to-end latency.
- To perform the computation offloading from mobile to server while simultaneously considering image content, computation cost, and server resource availability, we propose a set of techniques including recurrent region proposal prediction, and region proposal centric video frame partitioning and offloading, and region proposal computation cost estimation.
- We have built a prototype system with comprehensive experiments to demonstrate that our Elf system can be integrated with 10 state-of-the-art deep vision models and speeds up the applications by parallel offloading, up to 4.85×, with using 52.6% less bandwidth on 4 edge servers while keeping the accuracy sacrifice within 1%.
- We have learned valuable lessons of the relations between inference latency and the model design. Such lessons will help the vision community to better design models to benefit more from parallel offloading.

2 MOTIVATION AND CHALLENGES

Target Applications. In this paper, we target those applications that employ state-of-the-art convolutional neural network (CNN) models to conduct a variety of challenging computer-vision tasks from images or videos. Examples include image segmentation, multi-object classification, multi-person pose estimation, and many others. In general, those applications take an input image or video frame which is often of high resolution, e.g., 1920×1080, containing multiple objects, and perform a two-step processing task. First, they

[2]Elf is a small creature in stories usually described as smart, agile, and has magic power

use CNN networks to extract feature maps from the input and generate region proposals (RPs) for every object. Each RP is a candidate region where an object of interest – for example, a cat or child – may appear. Second, they use a CNN network to evaluate each RP and output the fine-grained result such as the classified object type or the key body points of a person. These state-of-the-art CNN models are usually highly computation intensive and run at a low frame rate, e.g., from 0.5 to 10 frames per second (fps) even on a high-end GPU (e.g. NVIDIA TITIAN 1080Ti) [3, 33, 34].

Limitations of Existing Task-Offloading Approaches. Offloading the inference tasks of CNNs onto an edge server is a promising approach to realizing the target applications on mobile devices [16, 35]. However, these existing task-offloading approaches are limited in two critical aspects. First, they only support task offloading to just one server, assuming that the server has sufficient resources to finish the offloaded task in time. However, a costly offloading server, for example, Intel Xeon Scalable Processors with Intel Deep Learning Boost [28] or NVIDIA EGX A100 [29], is usually shared by multiple clients and thus may not have sufficient resources to run a task. To demonstrate it, we profiled the computing latency of ResNet50 [36]. Each client runs on NVIDIA Jetson Nano [37] with 802.11.ax and the server runs the model inference on an NVIDIA TITIAN V GPU. The computing latency goes up in a linear pattern from 25.9 ms to 162.2 ms when changing the number of concurrent clients from 1 to 4. To handle the latency burst, Amazon SageMaker [38] adopts Kubeflow Pipelines to orchestrate and dynamically configure the traffic running on each server. However, this solution cannot handle resource fragmentation and may waste the computing cycles.

Another limitation of existing solutions is that they often use low-resolution (e.g., 384×288 [39]) images or videos to make the inference task lightweight. However, cameras on today's mobile devices typically capture with a much higher resolution such as 2K and 4K. Such a big gap causes two problems. On one hand, those existing low-resolution solutions fail to leverage the rich information of high-resolution images and videos to enable advanced applications such as various video analytics, for example, smart intersection [40]. Existing studies have already shown running object recognition related tasks on high-resolution images can largely increase the detection accuracy [41] . On the other hand, supporting high resolutions requires more computations and further undermines the assumption that one server can provide sufficient resources for the entire application. Our measurement results show that the inference latency of MaskRCNN [33] running on Jetson TX2 [42] boosts by 25%, 50% and 300% with increasing the image resolution from 224×224 to 1K, 2K and 4K, respectively, making the offloading harder.

To address the limitations of the existing work and the high resource demands of the target applications, in this paper, we design Elf, a lightweight system to accelerate high-resolution mobile deep vision applications through parallel task offloading to multiple servers.

Design Challenges. There are several key challenges in designing the Elf system. The first challenge lies in how to partition the computation. Broadly speaking, there are two approaches, model-parallel and data-parallel. Model parallelism, i.e., splitting a large model into multiple subsets of layers and running them on multiple servers, generates the large intermediate outputs from convolution layers which would lead to high communication overhead among servers [43]. For example, cracking open the DNN black-box [18] demonstrates that ResNet152 [36] produces the outputs with 19-4500× larger than the compressed input video. In this work, we explore data-parallelism by partitioning an input frame and offloading each frame partition to a different server. However, as shown in Figure 1(a), the simple equal partitioning may not work because 1) offloading a partition containing parts of an object may significantly reduce the model accuracy and 2) offloading a partition containing no objects may lead to excessive waste. Instead, we need to develop a smart video frame partitioning scheme to generate the ideal image partitioning shown in Figure 1(b).

(a) Equal partitioning (b) Ideal partitioning

Figure 1: Examples of video frame partitioning. The simple partitioning method in (a) split pixels of the same object into multiple parts and yield poor inference results. We can achieve much better partitioning using Elf, close to the ideal partitioning shown in (b)

The second challenge is how to distribute the tasks to multiple servers to minimize the total model inference latency. Ideally, all the servers should finish their tasks at the same time. However, that is hard to achieve because multiple dynamic factors must be considered together: the number of objects in the input images, the resource demand of processing each object, the number of servers and their available resources. Furthermore, another challenge in Elf is to minimize the workload of the resource-limited mobile device. In particular, the video frame partitioning is the step before offloading, running on the mobile device, and thus must be efficient and lightweight.

3 OVERVIEW AND DESIGN GUIDELINES

Elf intends to address the challenges with the steps below:

(1) Recurrent region proposal prediction. On the mobile end, whenever a new video frame arrives, Elf predicts its region proposals based on the ones detected in historical frames. The prediction reports each region proposal's coordinates. Here, a *region proposal* (RP) refers to a group of pixels containing at least one object of interest, e.g., a vehicle.
(2) Frame partitioning and offloading. Given the list of predicted RPs, Elf partitions the frame into "RP boxes". All the RP boxes collectively cover all the RP pixels while discarding background pixels that are unlikely to contain RPs. Elf then offloads these partitions to proper edge servers for processing. Both partitioning and offloading consult the partition's resource demands and server resource availability.

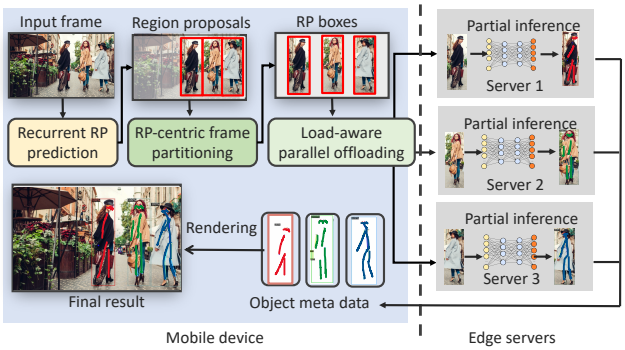

Figure 2: Elf system architecture. We explain the architecture using a multi-person pose estimation example with three edge servers

(3) Partial inference and result integration. Taking the offloaded partitions as input, the edge servers run the application-specific CNN models to yield partial inference results. These partial results are finally integrated at the mobile side to render the final result.

The above workflow is illustrated in Figure 2. While the third step is natural and easy to do, the first two steps call for careful designs to achieve the goals of Elf. In the rest of this section, we discuss the design guidelines of these two key components of Elf, focusing on how they are designed to address the challenge described in Section 2.

3.1 Recurrent Region Proposal Prediction

We adopt the following guidelines to devise the recurrent RP prediction algorithm: 1) the algorithm is lightweight; 2) the algorithm can effectively learn the motion model of the objects/RPs from history frames; and 3) the algorithm pays more attention to more recent frames. Here, a well-designed algorithm can accurately predict the RP distribution and help minimize the impact of the frame partitioning upon the deep vision applications' model accuracy. Following the guidelines above, we devise an attention-based Long Short-Term Memory (LSTM) network for recurrent RP prediction.

Note that the main-stream RP prediction/tracking algorithms require CNN models [44] that adds tens of millisecond to the system. Instead, our approach efficiently utilizes the historical RP inference results and converts the computing-intensive image regression problem to a light-weight time series prediction problem.

As part of the prediction algorithm, we also develop an RP indexing algorithm that keeps track of the motion across frames. Finally, we also propose a *Low Resolution Compensation* scheme to handle new objects when they first appear.

3.2 RP-Centric Video Frame Partitioning and Offloading

Partitioning a video frame allows Elf to offload each partition to a different edge server for parallel processing. Ideally, a well-designed frame partitioning scheme should show a negligible overhead and have heterogeneous edge servers to finish parallel inference tasks at the same time. Keeping these goals in mind, we design an RP-centric approach with the following guidelines.

Content awareness. The partitioning algorithm should be aware of the number of and locations of RPs in a frame and be inclusive. Also, Elf discards background pixels that are unlikely contain any RPs. Based on the study in AutoFocus [45], the area percentage of background in the COCO validation set takes up to 57%. Removing them can significantly reduce the computing and network transmission costs.

Computation cost awareness. Depending upon the objects contained in each partition, partitions have different computation costs. For example, it usually involves different numbers of pixels and becomes more challenging to identify multiple overlapping vehicles with similar colors than identifying a single vehicle with early-exit CNN models [46]. The algorithm should thus take into consideration this cost heterogeneity to achieve load balancing among the servers.

Resource awareness. After partitioning, the algorithm next matches these partitions to a set of edge servers. Unlike central clouds, edge cloud servers exhibit heterogeneous computing/storage/networking resources due to the distributed nature and high user mobility [17]. This makes the matching problem even more challenging. A poor match may result in job stragglers that complete much slower than their peers and thus significantly increases the overall latency.

4 FAST RECURRENT RP PREDICTION

When a new frame arrives, Elf predicts the coordinates of all the RPs in the frame, based on the RPs in the previous frames. In this section, we present three components that are key to achieve fast and effective RP prediction: an attention-based Long Short-Term Memory (LSTM) prediction network, a region proposal indexing algorithm, and a low-resolution frame compensation scheme. We choose to use attention-based LSTM for its powerful capabilities of learning rich spatial and temporal features from a series of frames. Also, it incurs a low system overhead of 3-5 ms running on mobile devices as illustrated in Section 7.5.

4.1 Problem Definition and Objective

As the objective is to train the attention-based LSTM network for acquiring the RP predictions accurately, the optimization process could be mathematically expressed as:

$$\begin{aligned} \min_{\boldsymbol{\theta}} \mathcal{L}(\hat{\boldsymbol{R}}_i^t, \boldsymbol{R}_i^t) = \min \textstyle\sum_i \big[& (\hat{x}_{i,\mathrm{tl}}^t - x_{i,\mathrm{tl}}^t)^2 + (\hat{y}_{i,\mathrm{tl}}^t - y_{i,\mathrm{tl}}^t)^2 \\ & (\hat{x}_{i,\mathrm{br}}^t - x_{i,\mathrm{br}}^t)^2 + (\hat{y}_{i,\mathrm{br}}^t - y_{i,\mathrm{br}}^t)^2 + (\hat{a}_i^t - a_i^t)^2 \big] \\ \text{s.t.} \quad & \hat{\boldsymbol{R}}_i^t = f(\{\boldsymbol{R}_i^{t-N}, \boldsymbol{R}_i^{t-N+1}, ..., \boldsymbol{R}_i^{t-1}\}, \boldsymbol{\theta}) \end{aligned} \tag{1}$$

where the vector $\boldsymbol{R}_i^t$ denotes the i-th ground-truth RP at frame t, and $\hat{\boldsymbol{R}}_i^t$ is the predictive RP counterpart. Both $\boldsymbol{R}_i^t$ and $\hat{\boldsymbol{R}}_i^t$ consist of $[x_{\mathrm{tl}}, y_{\mathrm{tl}}, x_{\mathrm{br}}, y_{\mathrm{br}}, a_i]$ as the x, y coordinates of RP's top-left and bottom-right corners, and the area, respectively. $\boldsymbol{\theta}$ is the model parameters of LSTM. Also, a_i^t is the RP's area calculated based on x_{tl}, y_{tl}, x_{br}, and y_{br},. Further, N is the number of previous frames used in the prediction network $f(\cdot)$. Next, we explain our algorithmic effort in minimizing the prediction error as calculated in Eq. (1).

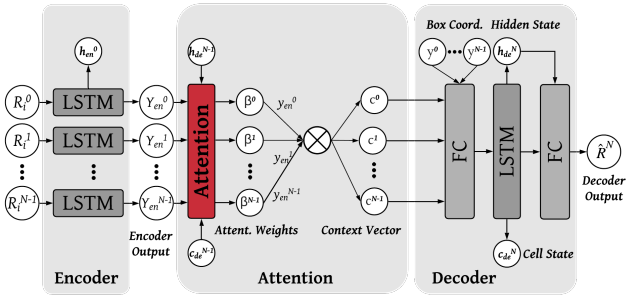

Figure 3: Our attention-based LSTM network

4.2 Attention-Based LSTM Network

Below we present the details of our attention-based LSTM RP prediction network.

4.2.1 ***Network structure.*** Recently, attention-based RNN models [47, 48] have shown their effectiveness in predicting time series data. In this work, we adapt a dual-stage attention-based RNN model [49], and develop a compact attention-based LSTM network for RP predictions. Note that adopting an LSTM based model rather than RNN can help detect periodically repeated patterns appeared in historical frames. Our model consists of three modules – an encoder, an attention module, and a decoder, as shown in Figure 3.

Encoder: To predict the i-th RP in the current frame, the encoder takes the spatial and temporal information (i.e., the RP's locations in history frames) of the i-th RP from N past frames $\boldsymbol{R}_i^t \in \mathbb{R}^{5\times 1}$ as input, and encodes them into the feature map $\{\boldsymbol{Y}_{\text{en}}^t\}$, $t \in \{0, \ldots, N-1\}$. This encoding is conducted by a two-layer LSTM [50], which can be modeled as:

$$\boldsymbol{Y}_{\text{en}}^t = f_{\text{en}}(\boldsymbol{Y}_{\text{en}}^{t-1}, \boldsymbol{R}^t), \tag{2}$$

where $f_{\text{en}}(\cdot,\cdot)$ denotes the LSTM computation.

Attention: Subsequently, we adopt an attention module which is a fully-connected layer to select the most relevant encoded feature. The first step is to generate the attention weight $\boldsymbol{\beta}$:

$$l^t = \boldsymbol{W}_2 \tanh(\boldsymbol{W}_1[\boldsymbol{Y}_{\text{en}}; c_{\text{de}}^{N-1}; h_{\text{de}}^{N-1}]) \tag{3}$$

$$\beta^t = \frac{\exp(l_i^t)}{\sum_{j=T-N-1}^{T} \exp(l_j^t)} \tag{4}$$

where $[\boldsymbol{Y}_{\text{en}}; c_{\text{de}}^{N-1}; h_{\text{de}}^{N-1}]$ is a concatenation of the encoder output $\boldsymbol{Y}_{\text{en}}$, decoder cell state vector c_{de}^{N-1} and decoder hidden state vector h_{de}^{N-1}. $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ are the weights to be optimized. The intermediate attention weight l^t is applied with softmax function to obtain the normalized attention weight $\boldsymbol{\beta}$. Thereafter, the context vector can be computed as:

$$\boldsymbol{c}^t = \sum_{j=0}^{N-1} \beta_j^t \boldsymbol{Y}_{\text{en}} \tag{5}$$

which captures the contributions of encoder outputs.

Decoder: The decoder module processes the context vector through a fully connected layer, an LSTM model, and a fully-connected regressor.

Algorithm 1 Region Proposal Indexing

Require: RP $\boldsymbol{R}_i^{t-1}=[x_{\text{tl},i}^{t-1}, y_{\text{tl},i}^{t-1}, x_{\text{br},i}^{t-1}, y_{\text{br},i}^{t-1}]$ for object i in frame $t-1$, where $i \in [0, 1, \ldots, m^{t-1}]$ and m^{t-1} is number of objects in frame $t-1$. Label set is $\boldsymbol{L}$.

Ensure: For frame at t, assign an index to each region proposal $\boldsymbol{R}^t$.

{Step-1. Initialization:}

1: **if** $t<N$ **then** ▷ label with a consistent index

2: $\quad \boldsymbol{R}_i^t[5] \leftarrow l_i^t,\ l_i^t \in \boldsymbol{L};\quad \forall i \in [0, 1, \ldots, m^{t-1}]$

3: **end if**

{Step-2. Measure distance and area:}

4: **for** $i := 1$ to m^{t-1} **do**

5: $\quad$ **for** $k := 1$ to m^t **do**

6: $\quad\quad D_{i,k}^{\text{x}} \leftarrow |(x_{\text{tl},i}^{t-1} + x_{\text{br},i}^{t-1}) - (x_{\text{tl},k}^{t} + x_{\text{br},k}^{t})|/2$ ▷ x-axis.

7: $\quad\quad D_{i,k}^{\text{y}} \leftarrow |(y_{\text{tl},i}^{t-1} + y_{\text{br},i}^{t-1}) - (y_{\text{tl},k}^{t} + y_{\text{br},k}^{t})|/2$ ▷ y-axis.

8: $\quad\quad A_{i,k} \leftarrow |1 - \dfrac{(x_{\text{br},i}^{t-1} - x_{\text{tl},i}^{t-1})(y_{\text{br},i}^{t-1} - y_{\text{tl},i}^{t-1})}{(x_{\text{br},k}^{t} - x_{\text{tl},k}^{t})(y_{\text{br},k}^{t} - y_{\text{tl},k}^{t})}|$ ▷ Area

9: $\quad$ **end for**

10: **end for**

{Step-3. Match and label:}

11: **for** $k := 1$ to m^t **do**

12: $\quad \hat{i} \leftarrow \arg\min_i \{A_{i,k}\}_{i=0}^{m^{t-1}}$; s.t. $D_{i,k}^{\text{x}} < 0.02, D_{i,k}^{\text{y}} < 0.02, A_{i,k} < 0.2$

13: $\quad$ **if** $\hat{i}$ is not *None* **then**

14: $\quad\quad \boldsymbol{R}_k^t[5] = \boldsymbol{R}_{\hat{i}}^{t-1}[5]$ ▷ label with matched RP

15: $\quad$ **else**

16: $\quad\quad \boldsymbol{R}_k^t[5] = l_k^t,\ l_k^t \in \boldsymbol{L}$ ▷ new label for unmatched

17: $\quad$ **end if**

18: **end for**

4.3 Region Proposal Indexing

To precisely predict a region proposal, we need to collect historical data, which provides necessary information such as motion models and trajectories. However, many vision applications, such as those discussed in Section 2, commonly output object labels in random order. Thus, it is hard to match and track region proposals across frames. For example, let us look at the example illustrated in Figure 4, where the same RPs in consecutive frames have different labels.

To address this issue, we devise a light consistent RP indexing algorithm. Vision-based matching algorithms are not considered because they introduce significant overheads in hundreds of milliseconds [51]. From the very first video frame, Elf assigns a unique index to each region proposal. In each upcoming frame, Elf matches each RP with the corresponding index assigned earlier. If an RP includes a new object that was not seen before, a new index will be automatically assigned.

Here, we match the RPs across frames with a combination of *RP position shift* and *RP area shift*. The *RP position shift* measures the change of the center point along the x-/y-axis between the current frame and the previous frame, as specified by Lines 6 and 7 in Algorithm 1. A larger value indicates a bigger spatial shift and thus a lower matching probability. The *RP area shift* measures the amount of area change between the RPs in two adjacent frames, as

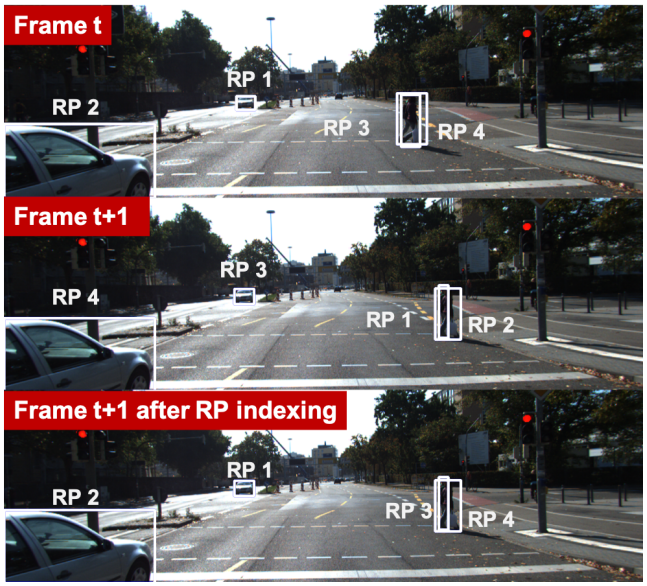

Figure 4: An example result for RP indexing

specified by Line 8 in Algorithm 1. A lower value indicates a higher matching probability. In our work, when the x and y RP position shift are both under 0.02 and the area shift ratio is under 0.2, we declare a match. The thresholds have been selected because they generate the lowest prediction loss in the evaluation. The sum of the RP position shift and RP area shift will be taken as an additional metric when there exist multiple RPs simultaneously satisfying the above threshold requirement.

RP Expansion. Another challenge in RP prediction lies in the possibility that the predicted RP bounding box may not cover all the pixels of an object due to motion. For example, as shown in Figure 5, the predicted bounding box excludes the person's hands, which will affect the object detection performed on the edge server. To address this challenge, we carefully expand the bounding box by p%. The downside of this scheme is the increased data transmission and computation. We conduct a trade-off study in Section 7.6. Here, ELF adopts different strategies to dynamically configure the value of p for different RPs. In particular, it consults the corresponding RP position shift and the prediction confidence level as the indicators to assign different weights on p.

4.4 Handling Objects When First Appear

The above attention LSTM-based prediction can deal with only the objects that already occurred in the previous frame, but not new ones never seen before. In this subsection, we discuss how to handle the new objects when they appear for the first time in a frame.

Low Resolution Compensation (LRC). To handle new objects, we propose a *low resolution compensation* (LRC) scheme with a balanced trade-off between computation overhead and new-object detection accuracy. Importantly, while inference with the down-sampled frame cannot produce fine-grained outputs that are required by the applications, such as object masks or key body points, we find that inference with down-sampled frames can still detect the presence of objects. Figure 16 and Figure 17 validate this observation. To reduce the computation overhead, LRC down-samples a high-resolution video frame by a max-pooling operation. Then ELF offloads the resized video frame, along with the partitions from regular sized partitions, to edge servers to run application-specific models, which usually consist of an object detection component. Based on the inference results, ELF can roughly locate the new objects in the frame.

Please note that here we use the same application-specified deep learning neural networks in the LRC module even though it may lead to a higher computation overhead than some lightweight networks. In this way, we do not compromise the new object detection accuracy. Meanwhile, ELF runs LRC once per n frames to reduce such an overhead. n is a hyperparameter, indicating the trade-off between computation cost and at most n-frame delay to realize new objects.

5 RP-CENTRIC VIDEO FRAME PARTITIONING AND OFFLOADING

Based on the RP predictions, ELF partitions a frame into multiple pieces, focusing on regions of interest while removing unnecessary dummy background pixels. Video frame partitioning plays a dominant role in minimizing the offloading traffic and balancing workloads across edge servers.

5.1 Problem Statement

ELF takes the following items as input: (i) video frame F_t at time t, (ii) the list of RP predictions in which $\boldsymbol{R}_i^t$ denotes the i-th RP in frame F_t, with $i\in[1,...,M]$ and M as the total number of RPs, and (iii) the available resource capacity, with p_j^t denoting the available resource capacity of the j-th server ($j\in[1,...,N]$) at time t. Based on the input, ELF packs the M RP processing tasks and one LRC task into N' offloading tasks ($N'\leq N$), and offloads each task onto an edge server.

The overall objective of the partitioning and the offloading process is to minimize the completion time of the offloading tasks that are distributed across N' edge servers. In other words, minimizing the completion time of the task which has the longest execution time among all the tasks. We assume that the mobile device only has access to a *limited* number of servers and that we try to make full use of these servers to minimize the application's completion time.

Accordingly, the optimization objective can be written as:

$$\begin{gathered}\min\ \max(\{T_k^t\}) \quad k\in[1,...,N'],\\ s.t.\ T_k^t = T_{\text{rps},k}^t + T_{\text{lrc},k}^t \cdot \mathbf{1}_{(t \bmod n=0)} \cdot \mathbf{1}_{(\arg\max\{p^t\}=k)},\\ T_{\text{rps},k}^t \approx \frac{C_{\text{rps},k}^t}{p_k^t},\ T_{\text{lrc},k}^t \approx \frac{C_{\text{lrc},k}^t}{p_k^t}\end{gathered} \tag{6}$$

where T_k^t denotes the completion time on the k-th server[3] at time-t. T_k^t consists of two completion-time terms, $T_{\text{rps},k}^t$ and $T_{\text{rps},k}^t$, for RPs and LRC respectively. $\mathbf{1}_{\text{condition}}$ returns 1 if and only if the condition meets, otherwise returns 0. Further, $C_{\text{rps},k}^t$ and $C_{\text{lrc},k}^t$ are

[3]We re-index the server with $k\in[1,...,N']$ instead of $j\in[1,...,N]$, owing to the aforementioned task packing.

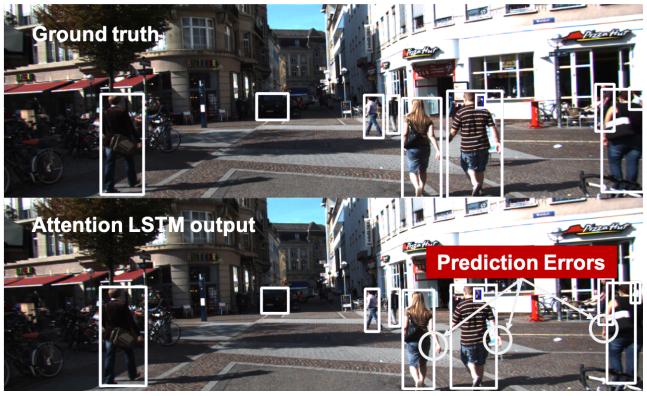

Figure 5: An example prediction error. Part of the objects are outside of the predicted RP bounding box

the computing cost of RP box and LRC offloading to server k, which will be described in Eq. (8).

5.2 Why Not Directly Schedule Each Individual RP Task?

After predicting the list of RPs, a straightforward scheduling approach is to cut out all the RPs and individually schedule each RP processing task onto edge servers based on the resource availability in a greedy fashion. While this sounds intuitive in many domains, it may not work the best for deep vision tasks due to the potential fragmentation problem.

First, the execution time of r (r is a small number such as 2 and 3) small RP (e.g., <5% size of the original image) tasks is not much less than r times of the execution time of running a single r-fold RP task. For example, Figure 9 shows that a frame with resizing into 1%, 4%, 9% takes 35.19ms, 37.9ms, and 44.24ms, respectively, to run the MaskRCNN inference. This is because in most deep vision models, except for the part extracting feature maps, the rest of the network is usually the same regardless of the size of the input.

Second, it is hard to determine a good cropping strategy. On one hand, the precise cut-out individual RPs will lead to poor detection inference accuracy due to the lack of necessary background pixels; on the other hand, if we leave large padding around the RPs, then the total offloaded data will be too large to be efficient. Third, too many cropping operations generate high memory copy overheads which may likely become problematic on mobile devices.

5.3 Partitioning & Offloading of RP-Box

RP Box Initialization. Given the above observations, ELF proposes an RP scheduling method that is more content- and resource-aware than the above naive counterpart. The key data structure here is what we call RP boxes. Compared to a single RP, an RP-box is larger and consists of one or more nearby RPs, as illustrated in Figure 6 (f) with 4 RPs and 3 RP boxes. The number of offloading tasks is determined by the number of available edge servers. Each offloading task consists of either an LRC task, or an RP-box processing task, or both. By scheduling an RP box instead of individual RPs, we can avoid the fragmentation problems mentioned above.

Before partitioning a frame, ELF first crops the area with all the RPs and horizontally partitions it into N segments (N is #available servers), where each segment corresponds to an initial RP box. The size of each RP box is initialized to be proportional to the available resource of the corresponding server, as depicted in Figure 6 (b), which is the first effort to achieve load balancing. Here, we explain how the LRC task scheduling interferes with the RP box scheduling. Note that, the LRC task is only available every n frames ($\mathbf{1}_{(t \bmod n=0)}$ in Eq. (6)). At the LRC round, we partition the cropped image into $(N-1)$ segments and have $(N-1)$ RP boxes accordingly. We reserve one server for the LRC task[4]. Regardless of the number of RP boxes, the scheduling algorithm works the same – during the LRC round, we treat the LRC task the same as another RP box processing task. Below, without the loss of generality, we assume there are N RP boxes.

RP Association. Thereafter, we associate each RP with an RP box. For each RP, ELF evaluates its spatial relationship with all the RP boxes. Given a pair of RP r and box b, their spatial relationship falls into one of the three cases:

- *Inclusion.* In this case, r is completely included in b and we conveniently associate them.
- *No-overlap.* In this case, r is not associated with b.
- *Partial-overlap.* In this case, r intersects with b. Meanwhile, it partially overlaps with at least one other box as well. Here, we choose to associate with the RP box that has the most overlap with the RP. If there is a tie, we choose the RP box with a larger gap between the server resource capacity and the computation costs of the RPs that are already associated. This association solution is the second effort to achieve load balancing.

ELF applies the association steps to all the RPs as shown in Figures 6 (c) and (d).

RP Box Adjustment. After all the RPs have got associated with a box, ELF resizes each RP box such that it can fully cover all the RPs that are associated with it. Please see Figures 6 (e) and (f). After this adjustment, the computation cost of some RP boxes may drastically increase compared to the initialization stage and thus break the intended load balancing.

To avoid this, we examine those RP boxes whose cost increase exceeds a pre-defined threshold (we discuss how to estimate an RP box's computation cost in Section 5.4). For these boxes, the associated RPs are sorted ascendingly w.r.t the computation cost. We try to re-associate the first RP on the list (the one with the lowest cost) to the neighboring box who has enough computation capacity to hold this RP.

After each re-association, the two boxes need to adjust their sizes accordingly and estimate the new computation cost. We repeat this re-association process as far as the load distribution is becoming more even. We stop this process if the re-association results in an even higher load imbalance. Here, we formally evaluate the load-balanced situation by:

$$\Theta = \mathrm{Var}(\{T_k^t\}) \tag{7}$$

[4]Special care needs to be taken with the configuration of a total of 1 or 2 edge servers, and we discuss how we handle these two special cases in Section 7.2

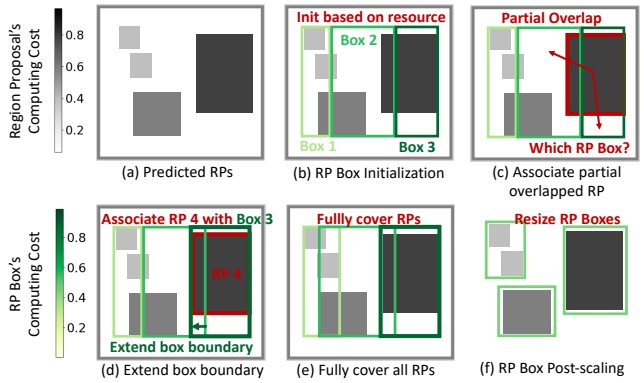

Figure 6: RP-centric frame partitioning pipeline

where Θ denotes the variance of the estimated execution time of all the tasks. A smaller Θ denotes a more balanced partitioning and offloading. We can calculate T_k^t by the following Eq. (6) where $C^t_{\text{rps},k}$ and C^t_{lrc} can be found as:

$$C^t_{\text{rps},k}=\sum_{v}\{C^t_{\text{rp},v}\},\ C^t_{\text{lrc}}=\alpha\cdot(\sum_{k=1}^{M}C^t_{\text{rps},k}) \tag{8}$$

where α is the LRC down-sample ratio.

Multi-offloading. Finally, Elf simultaneously offloads each RP box and the LRC task (if available in that round) to the corresponding edge server and executes the application-specific models in a data-parallelism fashion.

5.4 Estimating Server Capacity and RP Computation Cost

We now describe how Elf estimates the server resource capacity including both computational power and dynamic network conditions as well as each RP's computation cost.

Elf considers two ways of estimating a server's resource capacity. The first approach is through passive profiling. It calculates server m's average end-to-end latency T_m over the last n (default value of 7) offloading requests that are served by m. Then the resource capacity is defined as $1/T_m$. This passive profiling can help evaluate the trade-off between computing and network resources. The second approach is through proactive profiling: Elf periodically queries the server for its GPU utilization.

Elf also considers two ways of estimating an RP's computation cost. The first approach is based on the RP's area, assuming the cost is linearly proportional to the RP area. The second approach is through Spatially Adaptive Computation Time (SACT) [46]. Here, we briefly explain how to borrow its concept to estimate the computing cost of RPs. SACT is an optimization that early stops partial convolutional operations by evaluating the confidence upon the outputs of intermediate layers. Overall, SACT indicates how much computation has been applied with each pixel of a raw frame input. Elf can accordingly estimate the cost of an RP at the pixel level. To adopt this approach, we need to slightly modify the backbone network as instructed in [46].

We adopt the passive resource profiling and RP area-based estimation in the implementation as they are more friendly to Elf's users and require less system maintenance efforts. We will deliver other options in future work.

6 SYSTEM IMPLEMENTATION

We implement a prototype of Elf in both C++ and Python for easy integration with deep learning applications. Also, we wrap the C++ library with Java Native Interface (JNI) to support Android applications. In total, our implementation consists of 4,710 lines of codes. Our implementation is developed on Ubuntu16.04 and Android10. We integrate ZeroMQ4.3.2 [52], an asynchronous messaging library that is widely adopted in distributed and concurrent systems, for high-performance multi-server offloading. We use NVIDIA docker [53] to run offloading tasks on edge servers. We also wrap nvJPEG [54] with Pybind11 for efficient hardware-based image/video encoding on mobile devices.

Elf is designed and implemented as a general acceleration framework for diverse mobile deep vision applications. We aim to support existing applications with minimal modifications of applications. The required modifications only focus on the DNN inference functions. Here, we assume that an application can separate the DNN inference from the rest of it. Thus, the other parts and the internal logic of applications remain the same. Specifically, the host deep learning models need to implement two abstract functions:

```
1. def cat(inst_list: List[Instance]) -> Instance,
2. def extract(instance : Instance) -> List[RP].
```

Elf employs the first API to aggregate the partial inference results, and the second API to extract RPs from the data structure of partial inference results to be used in the RP prediction. With these two APIs, Elf can hide its internal details and provides a high-level API for applications:

```
3. def run(img: numpy.array) -> Instance,
```

This API can make the inference function as same as the one running locally, while Elf can run multi-way offloading and feed the merged results to applications. By following the above approach, we successfully integrate Elf with ten state-of-the-art deep learning models reported in Section 7.2. We believe that Elf requires a reasonably small effort from developers for the benefit of significantly reduced latency.

Furthermore, we discuss the placement of Elf functions. We argue that the functions should run on mobile devices but not edge servers for two reasons: 1) running the functions locally, especially frame partitioning, enables to offload less than 50% data as redundant background pixels will be cut off; 2) running all the functions only take 5-7ms on mobile devices", and thus the offloading benefit will be trivial considering half data to ship.

7 PERFORMANCE EVALUATION

We successfully integrate Elf with ten state-of-the-art deep learning networks and thoroughly evaluate Elf in the following three typical applications: instance segmentation, multi-object classification and multi-person pose estimation. Our results show that Elf can accelerate the inference up to 4.85× and 3.80× on average with using 52.6% less bandwidth on 4 edge servers while keeping the inference accuracy sacrifice within 1%.

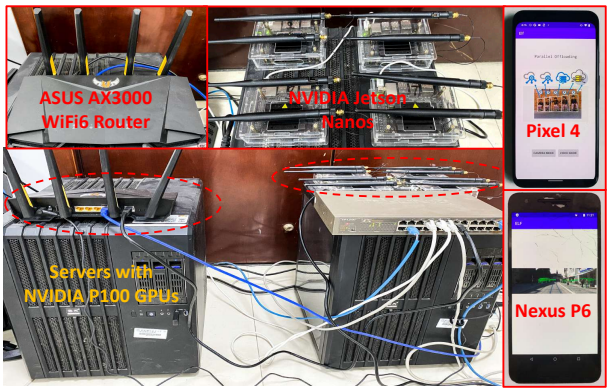

Figure 7: Our experimental evaluation hardware platform

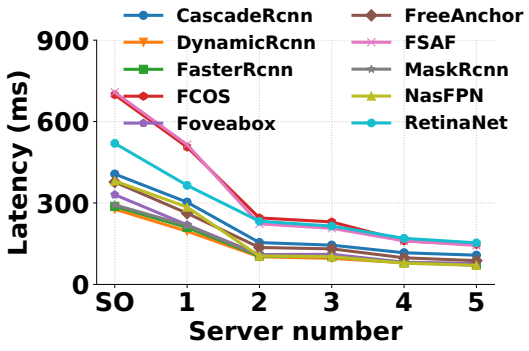

Figure 8: End-to-end latency vs server numbers

7.1 Experiment Setup

Mobile Platforms: We use four mobile platforms: Google Pixel4 (Qualcomm Snapdragon 855 chip consisting of eight Kryo 485 cores, an Adreno 640 GPU and a Hexagon 690 DSP), (2) Nexus 6P (Snapdragon 810 chip with four ARM Cortex-A57 cores and four ARM Cortex-A53 cores, an Adreno 430 GPU), (3) Jetson Nano [37] (Quad-core ARM Cortex-A57 MPCore CPU, NVIDIA Maxwell GPU with 128 CUDA cores), and (4) Jetson TX2 [42] (Dual-Core NVIDIA Denver 2 64-Bit + Quad-Core ARM Cortex-A57 MPCore CPU, NVIDIA Pascal GPU with 256 CUDA cores). The evaluation results with Jetson TX2 have been reported if not explicitly stated otherwise study the performance difference of mobile devices.

Edge Servers: We use up to 5 edge servers. Each server runs Ubuntu 16.04 and has one NVIDIA Tesla P100 GPU (3,584 CUDA Cores), Intel Xeon CPU (E5-2640 v4, 2.40GHz).

Networks: We use WiFi6 (802.11.ax, ASUS-AX3000, 690Mbps) to connect the mobile platforms and edge servers. Based on the WiFi network, we also use the Linux traffic shaping to emulate a Verizon LTE (120Mbps) link with using the parameters given by a recent Verizon network study [55]. Moreover, we randomly set the available bandwidth of each server in 70% to 100% to introduce the network heterogeneity. The emulated LTE network has been used if not explicitly stated otherwise study the network impacts. Figure 7 shows our experimental platform.

CNN Models and Datasets: We consider ten state-of-the-art models: CascadeRCNN [56], DynamicRCNN [57], FasterRCNN [3], FCOS [58], FoveaBox [59], FreeAnchor [60], FSAF [61], MaskRCNN [33], NasFPN [62], and RetinaNet [63]. Also, we use MOTS dataset [64] for instance segmentation, KITTI dataset [65] for multi-object classification, PoseTrack [66] dataset for pose estimation. MaskRCNN has been adopted if not explicitly stated otherwise study the model difference.

Comparison with Existing Offloading Work: We adopt FilterForward (FF) [19], a state-of-the-art offloading solution, as a baseline to compare with ELF. It introduces a group of "microclassifiers" that filter incoming video frames whether they contain objects of interest and only forwards the filtered results for the end-to-end model inference.

7.2 Evaluation of RP-Centric Partitioning and Offloading

Next, we evaluate the frame partitioning and offloading module. We first describe the end-to-end latency when different numbers of servers are available for ELF with ten state-of-the-art deep learning networks. Here, we assume each server has only a single GPU available for the mobile application. A special case to consider, if there is only a GPU, ELF will adopt a single RP box that covers all the RPs but removes the surrounding background pixels and stack with the LRC task. KITTI dataset has been resized to the resolution 2560×1980 to study the high-resolution scenario in the section.

Figure 8 shows the end-to-end latency with the server number from 1 to 5 as well as the Single server Offloading (*SO*). With only one server available, ELF-1 shows the applications speed up by 1.39× on average, up to 1.50× compared to *SO* as ELF can efficiently remove the redundant background pixels. When there are two servers available, ELF offloads the LRC task and one of the RP boxes to one server and the other RP box to the second server. Compared to *SO*, ELF-2 speeds up the applications by 2.80× on average, up to 3.63× . When ELF has three or more servers, it uses one server for LRC, and one RP box each on the other servers. We measure the speed up by 2.94× on average, up to 3.71× with ELF-3, 3.80× on average, up to 4.85× with ELF-4, and 4.18× on average, up to 5.43× with ELF-5 respectively.

Moreover, we profile FilterForward that speeds up the applications by 1.56× on average compared to *SO*, but its runtime could hardly be further minimized with more offloading servers. Overall, ELF shows the close speedup compared to FilterForward in the single server scenario, but delivers much more competitive performance by adding more servers.

Key Observations: We observe that the latency with different server numbers highly depends on the size of the RP boxes shipped to each edge server. With the frame partitioning algorithm, the *maximal* size of RP box compared to the raw frame as 51.7%, 23.7%, 23.7%, 15.7%, and 11.6%, the computing bottleneck in that offloading round, with the server number from 1 to 5, respectively. Please note that ELF-2 and ELF-3 have the same RP box size because both adopt 2 RP boxes but the later assigns the LRC task on the third server. Accordingly, ELF reduces bandwidth usage by 48.3%, 52.6%, 52.6%, 52.9%, and 53.6%.

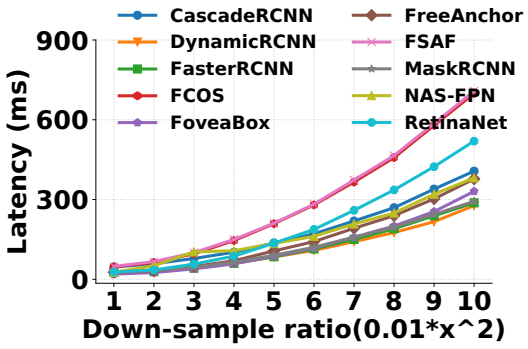

Figure 9: Processing latency vs down-sample ratio

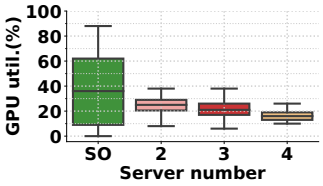

Figure 10: Average GPU utilization vs GPU numbers

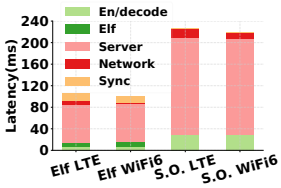

Figure 11: End-to-end latency vs network conditions

Importantly, another observation is that the model inference time strongly relates to the input size. Figure 9 shows the inference latency at the server running ten state-of-the-art models with down-sample ratio $0.01 \cdot x^2$ where x is from 1 to 10. Here, the down-sample ratios of 49%, 25%, 16%, and 9% share a rough correspondence to the RP box size with the server numbers of 1, 2 (3), 4, 5. This observation is the underlying reason why Elf can significantly reduce the inference latency by having each server inferring part of the frame.

Lessons Learned: Moreover, we identify the inference time shows distinct sensitivity among different deep vision models. First, the models, for example, FCOS [58], with more, even fully, convolutional operations present a stronger correlation between frame resolution and inference latency. Second, two-stage models, for example, RCNN series [3, 33, 56, 57], usually generate the same number of Regions of Interest (ROI) independent of the input resolution and then ship each of them down the pipeline. The second stage thus costs the same time.

Overall, the lessons we have learned regarding how to design models to benefit more from parallel offloading are: 1) one-stage models with more convolutional operations are preferred, 2) two-stage models can dynamically adjust the number of ROI based on the frame resolution as a higher resolution input potentially involves more objects.

Finally, we show the average GPU utilization under different configurations in Figure 10. In the case of *SO*, the average GPU utilization is 37% and the 95_{th} percentile is 82%. In the case of Elf-3, the average GPU utilization is 21% and the 95_{th} percentile is 31%. We note that using 3 GPUs in the case of Elf-3 shows lower per GPU utilization than *SO*. On average, Elf-3 only consumes 1.7× GPU utilization in total running with 3 GPUs, than *SO* to finish a single request. Moreover, a lower per GPU utilization allows Elf to have more chance to efficiently utilize those resource fragmentation and thus improve the total GPU utilization of edge servers.

7.3 Accuracy of Deep Vision Applications

After discussing how Elf can contribute to minimizing the latency with parallel offloading, it is critical to show that it has limited impacts upon the accuracy of deep vision applications. Three popular applications have been evaluated: instance segmentation with MaskRCNN [33], object classification with RetinaNet [63], and multi-person pose estimation with DensePose [67]. We report the inference accuracy in the following 4 settings: (1) *TX2*, a baseline running the application on Jetson TX2, (2) *Nano*, a baseline running the application on Jetson Nano, (3) *SO*, a baseline of existing offloading strategy that offloads the CNN inference to a single edge server, (4) FilterForward (FF) [19], a state-of-the-art offloading solution, (5) Elf, our approach of partitioning the frame and offloading the partial inferences to edge servers (using 3 servers as an example).

Table 1 reports the accuracy in all the settings. Compared to using the entire frame for inference as in *SO* or running locally on Jetson Nano or TX2, Elf achieves almost the same accuracy: 0.799 vs 0.803 (0.49%) for instance segmentation, 0.671 vs 0.672 (0.14%) for object classification and 0.654 vs 0.661 (1.05%) for pose estimation. FilterForward (FF) shows the accuracy 0.605 for object classification with introducing "microclassifiers" that cannot well detect small objects. The minor accuracy drop from Elf is because 1) running LRC once every 3 frames may miss/delay new or tiny objects, although it rarely happens, and 2) Elf removes the background pixels not covered by the RP boxes. However, we believe this small accuracy drop is acceptable, especially considering the significant latency reduction.

7.4 Dealing with Dynamic Network Condition and GPU Utilization

First, we compare Elf-3 and *SO* with Verizon LTE (120Mbps) and WiFi6 (690Mbps) networks. Figure 11 shows that when switching from WiFi6 to LTE, the network latency of Elf increases from 1.4ms to 6.5ms and that of *SO* increases from 10.1ms to 17.1ms. Elf is less sensitive to the network bandwidth because it offloads much less data than *SO* as shown in Figure 15. Next, we increase the GPU utilization of one server to 70% and compare our resource-aware

Table 1: Comparisons of inference accuracy (AP) in three deep vision applications: instance segmentation [33], object classification [63], and pose estimation [67]

Deep Vision Applications	Accuracy (AP)				
	TX2	Nano	SO	FF [19]	Elf
Instance Segmentation	0.803	0.803	0.803	/	0.799
Object Classification	0.672	0.672	0.672	0.605	0.671
Pose Estimation	0.661	0.661	0.661	/	0.654

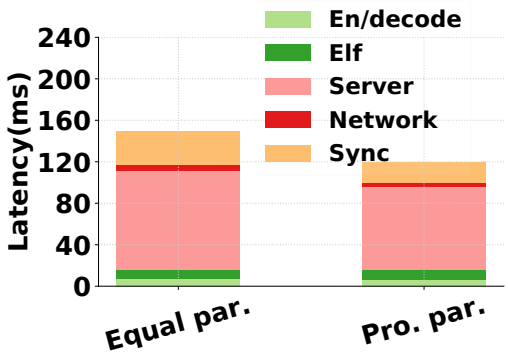

Figure 12: latency vs RP box partitioning schemes

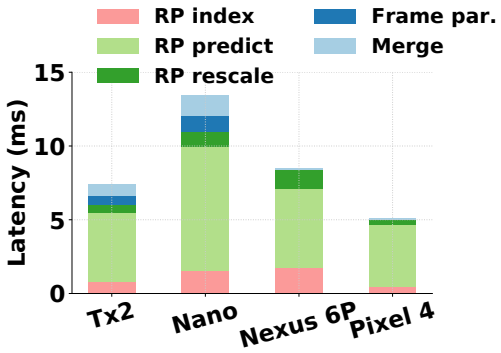

Figure 13: System overheads of Elf functions

RP box allocation method with a resource-agnostic equal RP box allocation method. Figure 12 shows that our method results in a latency of 119ms while the other method has a latency of 149ms.

7.5 Elf System Overhead on Mobile Side

Elf incurs a small amount of overhead on the mobile side. Figure 13 shows the latency of five Elf functions. Jetson TX2 and Nano are evaluated with the Python implementation and take 7ms, and 13.6ms in total. Nexus 6P and Pixel 4 are evaluated with C++ by Java native interface and cost 7.8ms and 4.8ms in total. The incurred system overhead is sufficiently low to deliver its parallel offloading functions for the significant latency reduction.

Moreover, RP prediction costs 70%+ of the total time as the attention LSTM model is implemented in Python and exported to C++ with TorchScript. We will rewrite the prediction model with TensorRT [68], a C++ library that facilitates high-performance inference, in the future work to minimize the RP prediction latency.

7.6 Evaluation of RP Prediction

Model Training: Two online available video datasets, KITTI [69] and CityScapes [70] that contain object labels for each frame, were used in the training. Using 60% of the dataset as training data, we applied the RP indexing algorithm to maintain a consistent order of region proposals. Finally, we train the network using Adam optimizer [71] with a learning rate of 1e-3 to minimize the loss function in Eq. (1).

The Effectiveness of Attention LSTM: We show the training loss curve in the first 60 epochs in Figure 18 and the loss on the test dataset in Figure 19. The trend shows that our attention LSTM outperforms vanilla LSTM that demonstrates remarkable accuracy in those prediction work [72, 73], by reducing the loss from 0.51 to 0.25, as it can pay more attention to more recent frames.

Further, we report the impact of different prediction algorithms on the inference accuracy while keeping other modules the same. Here, we also consider FastTracker [74], which predicts the current RPs as the RPs in the previous video frame, but with a scale-up of 200%, as the baseline. Figure 20 and Figure 21 show the inference accuracy and the offload ratio (defined as the ratio of the total offloading traffic, with respect to the offloading traffic in *SO*). The vanilla LSTM predictor has the lowest offloading traffic, with an 11% offload ratio, but at a considerable inference accuracy downgrade compared to our attention LSTM, 0.748 vs. 0.799. Meanwhile, FastTracker shows a slightly higher inference accuracy compared to us, 0.802 vs 0.799, but the offloading traffic almost doubles.

All things considered, our attention LSTM could achieve a good inference accuracy with reasonably low offloading traffic.

The Impact of RP Indexing: Next, we demonstrate the importance of maintaining a consistent RP index across video frames. Figure 19 shows the test loss of two prediction algorithms with and without RP indexing. With RP indexing, vanilla LSTM reduces the loss from 0.71 to 0.51 and attention LSTM reduces the loss from 0.7 to 0.25.

The Impact of RP Expansion Ratio: Moreover, the RP expansion ratio trades off the application accuracy with the average offloading traffic volume, i.e., a larger ratio leads to a higher application accuracy at the cost of more offloading data. Figures 14 and 15 show the trade-off with different expansion ratios. After increasing the RP expansion ratio to 4% or higher, the accuracy stays at 0.799, the highest Elf can achieve.

However, when Elf has the ratio under 4%, we observe an accuracy downgrade. For example, the accuracy is 0.70 and 0.75 when the expansion ratio is 1% and 2%, respectively. Also, we identify the same pattern with Elf-3 and Elf-4. On the other hand, with an RP extension ratio of 4%, the offload ratio is 7% for Elf-3 and 13% for Elf-4.

The Impact of LRC Parameters: We then show LRC can efficiently detect new objects when first appear. Figure 16 presents the accuracy using the down-sampled frames for inference with a down-sampling scale x increasing from 1 to 10, with the resulting frame size from $0.01 \times x^2$. At the scale of 8, the accuracy degrades to 0.75, and at the scale of 4, the accuracy goes down to 0.62. The result indicates that using low-resolution frames alone hurts application accuracy. Figure 17 reports the inference accuracy by offloading frame partitions and LRC with different down-sample ratios. When the LRC ratio has been increased to 16%, the accuracy keeps at 0.799 (the SO solution achieves 0.803).

8 RELATED WORK

Video Analytics Optimizations. AWS Wavelength [23] moves Amazon Web Services to Verizon's 5G edge computing platforms

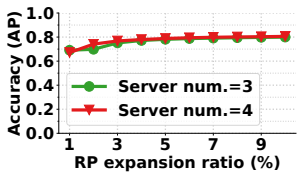

Figure 14: Inference accuracy vs RP expansion ratio

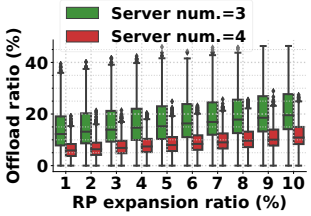

Figure 15: Offload ratio vs RP expansion ratio

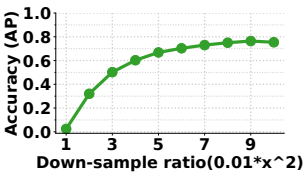

Figure 16: Inference accuracy vs downsample ratio

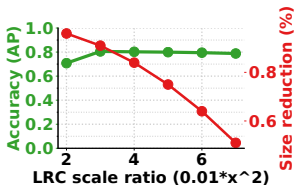

Figure 17: Inference accuracy vs LRC ratio

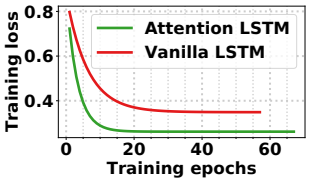

Figure 18: Training loss in the first 60 epochs

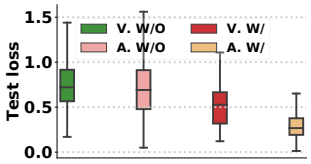

Figure 19: Test loss w/- and w/o RP indexing

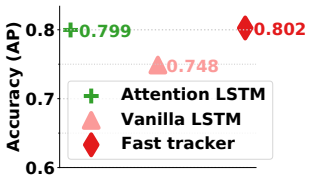

Figure 20: Inference accuracy

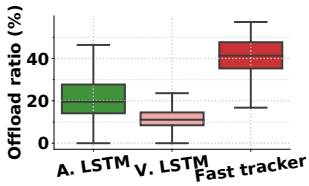

Figure 21: Offload ratios

to deliver low latency video services. Intel and NVIDIA contribute Intel Xeon Scalable Processors with Intel Deep Learning Boost [28] and NVIDIA EGX A100 [29] to enable real-time AI processing at the edge. Pano [75] proposes a quality-adaptation scheme that balances user-perceived video quality and video encoding efficiency. Chameleon [76] dynamically selects the best configurations for existing NN-based video analytics to save computing resources by up to 10×. VideoStorm [77] and AWStream [78] adapt the configuration to balance accuracy and processing delay.

Offloading Deep Neural Networks. Cracking open the DNN [18], Neurosurgeon [79], DeepThings [80] partition then distribute the layers of deep learning networks over edge servers and mobile/IoT devices (model parallelism) to reduce the inference latency. Filter-Forward [19], Reducto [20], Glimpse [21] and Vigil [22] perform selective data offloading based on the feature type, filtering threshold, query accuracy, and video content to minimize the running latency. EdgeAssisted [16] uses a motion vector based object tracking to adaptively offload those regions of interests. Frugal Following [81] dynamically tracks objects and only runs a DNN with significantly changes. DDS [82] continuously sends a low-quality video stream to the server that runs the DNN to determine where to re-send with higher quality to increase the inference accuracy. These works only support single-server offloading. Instead, Elf is designed to accelerate high-resolution vision tasks through distributed offloading of multiple servers.

Accelerating Model Inferences. Branch pruning and sharing [7, 83–85] remove redundant or less critical parameters [8] to trade-off the model complexity with inference latency. Tensor quantization [9] uses fewer bits to represent parameters for model compression. DeepCache [86] caches and reuses the result of convolutional operations to reduce the repeated computation. Simultaneously handle tasks with a single model through multi-task learning [87] for less computation. Furthermore, massive accelerators, e.g., GPU [37, 42], FPGA [14], and ASIC [15, 88], are designed to perform model inference in a high-throughput and low-latency fashion. All these works are complementary to ours.

9 CONCLUSION AND FUTURE WORK

We designed and implemented Elf, an acceleration framework for mobile deep learning networks. Elf can partition a video frame into multiple pieces and offload them simultaneously to edge servers for parallel computing. The main contribution stems from its recurrent region proposal prediction and content-aware video frame partitioning algorithms. Our study shows that Elf is promising to minimize the end-to-end latency of emerging mobile deep vision applications. Moving forward, we will continue to investigate how to further drive down the latency, e.g., integrating the data-parallelism approach of Elf with those model-parallelism solutions. We will also investigate the impact of access network bandwidth on Elf task assignment and mapping. Another future work topic is the efficient model design for deep vision applications to better benefit from parallel offloading. Finally, we will study how to efficiently orchestrate heterogeneous edge resources to minimize the AI processing latency.

10 ACKNOWLEDGMENTS

We sincerely thank the reviewers and anonymous shepherd for their valuable comments that help us improve this work and prepare the camera ready version. We thank Ivan Seskar for his help to setup the experimental platform. We also thank Sugang Li for the discussion to brainstorm the initial research idea. This work is partially supported by the Platforms for Advanced Wireless Research (PAWR) project with the National Science Foundation grant No.182792, the 2030 National Key AI Program of China 2018AAA 0100500 and Key Research Program of Frontier Sciences, CAS, Grant No.ZDBS-LY-JSC001.

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[2] M. Xu, J. Liu, Y. Liu, F. X. Lin, Y. Liu, and X. Liu, "A first look at deep learning apps on smartphones," in *The World Wide Web Conference*, pp. 2125–2136, 2019.

[3] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, pp. 91–99, 2015.

[4] M. Teichmann, M. Weber, M. Zoellner, R. Cipolla, and R. Urtasun, "Multinet: Real-time joint semantic reasoning for autonomous driving," in *2018 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1013–1020, IEEE, 2018.

[5] C. Xiang, C. R. Qi, and B. Li, "Generating 3d adversarial point clouds," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 9136–9144, 2019.

[6] S. Xu, D. Liu, L. Bao, W. Liu, and P. Zhou, "Mhp-vos: Multiple hypotheses propagation for video object segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 314–323, 2019.

[7] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1389–1397, 2017.

[8] B. Fang, X. Zeng, and M. Zhang, "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*, pp. 115–127, ACM, 2018.

[9] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized convolutional neural networks for mobile devices," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4820–4828, 2016.

[10] Z. He and D. Fan, "Simultaneously optimizing weight and quantizer of ternary neural network using truncated gaussian approximation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 11438–11446, 2019.

[11] J. Yim, D. Joo, J. Bae, and J. Kim, "A gift from knowledge distillation: Fast optimization, network minimization and transfer learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4133–4141, 2017.

[12] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8697–8710, 2018.

[13] T. Lee, Z. Lin, S. Pushp, C. Li, Y. Liu, Y. Lee, C. Xu, F. Xu, L. Zhang, and J. Song, "Occlumency: Privacy-preserving remote deep-learning inference using sgx," in *Proceedings of the 25th Annual International Conference on Mobile Computing and Networking, MobiCom 2019, October 21-25, 2019, Los Cabos, Mexico*, ACM, 2019.

[14] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 161–170, ACM, 2015.

[15] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pp. 1–12, IEEE, 2017.

[16] L. Liu, H. Li, and M. Gruteser, "Edge assisted real-time object detection for mobile augmented reality," in *The 25th Annual International Conference on Mobile Computing and Networking*, pp. 1–16, 2019.

[17] W. Zhang, S. Li, L. Liu, Z. Jia, Y. Zhang, and D. Raychaudhuri, "Hetero-edge: Orchestration of real-time vision applications on heterogeneous edge clouds," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, IEEE, 2019.

[18] J. Emmons, S. Fouladi, G. Ananthanarayanan, S. Venkataraman, S. Savarese, and K. Winstein, "Cracking open the dnn black-box: Video analytics with dnns across the camera-cloud boundary," in *Proceedings of the 2019 Workshop on Hot Topics in Video Analytics and Intelligent Edges*, pp. 27–32, 2019.

[19] C. Canel, T. Kim, G. Zhou, C. Li, H. Lim, D. G. Andersen, M. Kaminsky, and S. R. Dulloor, "Scaling video analytics on constrained edge nodes," *arXiv preprint arXiv:1905.13536*, 2019.

[20] Y. Li, A. Padmanabhan, P. Zhao, Y. Wang, G. H. Xu, and R. Netravali, "Reducto: On-camera filtering for resource-efficient real-time video analytics," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pp. 359–376, 2020.

[21] S. Naderiparizi, P. Zhang, M. Philipose, B. Priyantha, J. Liu, and D. Ganesan, "Glimpse: A programmable early-discard camera architecture for continuous mobile vision," in *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, pp. 292–305, 2017.

[22] T. Zhang, A. Chowdhery, P. Bahl, K. Jamieson, and S. Banerjee, "The design and implementation of a wireless video surveillance system," *MobiCom, ACM*, 2015.

[23] "Aws wavelength: Bring aws services to the edge of the verizon 5g network.." https://enterprise.verizon.com/business/learn/edge-computing/.

[24] A. Narayanan, E. Ramadan, J. Carpenter, Q. Liu, Y. Liu, F. Qian, and Z.-L. Zhang, "A first look at commercial 5g performance on smartphones," in *Proceedings of The Web Conference 2020*, pp. 894–905, 2020.

[25] S. Zhou, W. Shen, D. Zeng, M. Fang, Y. Wei, and Z. Zhang, "Spatial–temporal convolutional neural networks for anomaly detection and localization in crowded scenes," *Signal Processing: Image Communication*, vol. 47, pp. 358–368, 2016.

[26] N. Tijtgat, W. Van Ranst, T. Goedeme, B. Volckaert, and F. De Turck, "Embedded real-time object detection for a uav warning system," in *The IEEE International Conference on Computer Vision (ICCV) Workshops*, Oct 2017.

[27] H. Zhao, X. Qi, X. Shen, J. Shi, and J. Jia, "Icnet for real-time semantic segmentation on high-resolution images," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 405–420, 2018.

[28] "Intel xeon scalable processors." https://www.intel.com/content/www/us/en/products/processors/xeon/scalable.html.

[29] "Nvidia egx a100: delivering real-time ai processing and enhanced security at the edge." https://www.nvidia.com/en-us/data-center/products/egx-a100/.

[30] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella, "Multi-resource packing for cluster schedulers," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 455–466, 2014.

[31] L. Peterson, T. Anderson, S. Katti, N. McKeown, G. Parulkar, J. Rexford, M. Satyanarayanan, O. Sunay, and A. Vahdat, "Democratizing the network edge," *ACM SIGCOMM Computer Communication Review*, vol. 49, no. 2, pp. 31–36, 2019.

[32] S. Yang, E. Bailey, Z. Yang, J. Ostrometzky, G. Zussman, I. Seskar, and Z. Kostic, "Cosmos smart intersection: Edge compute and communications for bird's eye object tracking," in *Proc. 4th International Workshop on Smart Edge Computing and Networking (SmartEdge'20)*, 2020.

[33] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *Proceedings of the IEEE international conference on computer vision*, pp. 2961–2969, 2017.

[34] H.-S. Fang, S. Xie, Y.-W. Tai, and C. Lu, "Rmpe: Regional multi-person pose estimation," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2334–2343, 2017.

[35] X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen, "Deepdecision: A mobile deep learning framework for edge video analytics," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pp. 1421–1429, IEEE, 2018.

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

[37] "Nvidia jetson nano, the ai platform for autonomous everything." https://www.nvidia.com/jetson-nano.

[38] "Amazon sagemaker: Machine learning for every developer and data scientist." https://aws.amazon.com/sagemaker/.

[39] K. Sun, B. Xiao, D. Liu, and J. Wang, "Deep high-resolution representation learning for human pose estimation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

[40] D. Raychaudhuri, I. Seskar, G. Zussman, T. Korakis, D. Kilper, T. Chen, J. Kolodziejski, M. Sherman, Z. Kostic, X. Gu, *et al.*, "Challenge: Cosmos: A city-scale programmable testbed for experimentation with advanced wireless," in *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, pp. 1–13, 2020.

[41] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, *et al.*, "Speed/accuracy trade-offs for modern convolutional object detectors," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7310–7311, 2017.

[42] "Nvidia jetson tx2, the fastest, most power-efficient embedded ai computing device." https://developer.nvidia.com/embedded/jetson-tx2.

[43] M. Wang, C.-c. Huang, and J. Li, "Supporting very large models using automatic dataflow graph partitioning," in *Proceedings of the Fourteenth EuroSys Conference 2019*, p. 26, ACM, 2019.

[44] P. Voigtlaender, M. Krause, A. Osep, J. Luiten, B. B. G. Sekar, A. Geiger, and B. Leibe, "Mots: Multi-object tracking and segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7942–7951, 2019.

[45] M. Najibi, B. Singh, and L. S. Davis, "Autofocus: Efficient multi-scale inference," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 9745–9755, 2019.

[46] M. Figurnov, M. D. Collins, Y. Zhu, L. Zhang, J. Huang, D. Vetrov, and R. Salakhutdinov, "Spatially adaptive computation time for residual networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1039–1048, 2017.

[47] D. Bahdanau, J. Chorowski, D. Serdyuk, P. Brakel, and Y. Bengio, "End-to-end attention-based large vocabulary speech recognition," in *2016 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pp. 4945–4949, IEEE, 2016.

[48] Y. Wang, M. Huang, X. Zhu, and L. Zhao, "Attention-based lstm for aspect-level sentiment classification," in *Proceedings of the 2016 conference on empirical methods in natural language processing*, pp. 606–615, 2016.

[49] Y. Qin, D. Song, H. Chen, W. Cheng, G. Jiang, and G. Cottrell, "A dual-stage attention-based recurrent neural network for time series prediction," *arXiv preprint arXiv:1704.02971*, 2017.

[50] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," 1999.

[51] Q. Wang, L. Zhang, L. Bertinetto, W. Hu, and P. H. Torr, "Fast online object tracking and segmentation: A unifying approach," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1328–1338, 2019.

[52] P. Hintjens, *ZeroMQ: messaging for many applications.* " O'Reilly Media, Inc.", 2013.

[53] "Build and run docker containers leveraging nvidia gpus." https://github.com/NVIDIA/nvidia-docker.

[54] "Nvidia gpu-accelerated jpeg encoder and decoder." https://developer.nvidia.com/nvjpeg.

[55] A. Narayanan, J. Carpenter, E. Ramadan, Q. Liu, Y. Liu, F. Qian, and Z.-L. Zhang, "A first measurement study of commercial mmwave 5g performance on smartphones," *arXiv preprint arXiv:1909.07532*, 2019.

[56] Z. Cai and N. Vasconcelos, "Cascade r-cnn: Delving into high quality object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 6154–6162, 2018.

[57] H. Zhang, H. Chang, B. Ma, N. Wang, and X. Chen, "Dynamic r-cnn: Towards high quality object detection via dynamic training," *arXiv preprint arXiv:2004.06002*, 2020.

[58] Z. Tian, C. Shen, H. Chen, and T. He, "Fcos: Fully convolutional one-stage object detection," in *Proceedings of the IEEE international conference on computer vision*, pp. 9627–9636, 2019.

[59] T. Kong, F. Sun, H. Liu, Y. Jiang, L. Li, and J. Shi, "Foveabox: Beyound anchor-based object detection," *IEEE Transactions on Image Processing*, vol. 29, pp. 7389–7398, 2020.

[60] X. Zhang, F. Wan, C. Liu, R. Ji, and Q. Ye, "Freeanchor: Learning to match anchors for visual object detection," in *Advances in Neural Information Processing Systems*, pp. 147–155, 2019.

[61] C. Zhu, Y. He, and M. Savvides, "Feature selective anchor-free module for single-shot object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 840–849, 2019.

[62] G. Ghiasi, T.-Y. Lin, and Q. V. Le, "Nas-fpn: Learning scalable feature pyramid architecture for object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7036–7045, 2019.

[63] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, pp. 2980–2988, 2017.

[64] P. Voigtlaender, M. Krause, A. Osep, J. Luiten, B. B. G. Sekar, A. Geiger, and B. Leibe, "Mots: Multi-object tracking and segmentation," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[65] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[66] M. Andriluka, U. Iqbal, E. Ensafutdinov, L. Pishchulin, A. Milan, J. Gall, and S. B., "PoseTrack: A benchmark for human pose estimation and tracking," in *CVPR*, 2018.

[67] R. Alp Güler, N. Neverova, and I. Kokkinos, "Densepose: Dense human pose estimation in the wild," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7297–7306, 2018.

[68] "Nvidia tensorrt programmable inference accelerator." https://developer.nvidia.com/tensorrt.

[69] M. Menze and A. Geiger, "Object scene flow for autonomous vehicles," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[70] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[71] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *ICLR*, 2015.

[72] Y. Guan and T. Plötz, "Ensembles of deep lstm learners for activity recognition using wearables," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 1, no. 2, pp. 1–28, 2017.

[73] P. Zhang, W. Ouyang, P. Zhang, J. Xue, and N. Zheng, "Sr-lstm: State refinement for lstm towards pedestrian trajectory prediction," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 12085–12094, 2019.

[74] D. Held, S. Thrun, and S. Savarese, "Learning to track at 100 fps with deep regression networks," in *European Conference on Computer Vision*, pp. 749–765, Springer, 2016.

[75] Y. Guan, C. Zheng, X. Zhang, Z. Guo, and J. Jiang, "Pano: Optimizing 360 video streaming with a better understanding of quality perception," in *Proceedings of the ACM Special Interest Group on Data Communication*, pp. 394–407, 2019.

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

[77] H. Zhang, G. Ananthanarayanan, P. Bodik, M. Philipose, P. Bahl, and M. J. Freedman, "Live video analytics at scale with approximation and delay-tolerance," in *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, pp. 377–392, 2017.

[78] B. Zhang, X. Jin, S. Ratnasamy, J. Wawrzynek, and E. A. Lee, "Awstream: Adaptive wide-area streaming analytics," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pp. 236–252, 2018.

[79] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 1, pp. 615–629, 2017.

[80] Z. Zhao, K. M. Barijough, and A. Gerstlauer, "Deepthings: Distributed adaptive deep learning inference on resource-constrained iot edge clusters," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2348–2359, 2018.

[81] K. Apicharttrisorn, X. Ran, J. Chen, S. V. Krishnamurthy, and A. K. Roy-Chowdhury, "Frugal following: Power thrifty object detection and tracking for mobile augmented reality," in *Proceedings of the 17th Conference on Embedded Networked Sensor Systems*, pp. 96–109, 2019.

[82] K. Du, A. Pervaiz, X. Yuan, A. Chowdhery, Q. Zhang, H. Hoffmann, and J. Jiang, "Server-driven video streaming for deep learning inference," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pp. 557–570, 2020.

[83] A. Veit and S. Belongie, "Convolutional networks with adaptive inference graphs," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 3–18, 2018.

[84] S. Liu, Y. Lin, Z. Zhou, K. Nan, H. Liu, and J. Du, "On-demand deep model compression for mobile devices: A usage-driven model selection framework," in *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, pp. 389–400, 2018.

[85] S. Jiang, Z. Ma, X. Zeng, C. Xu, M. Zhang, C. Zhang, and Y. Liu, "Scylla: Qoe-aware continuous mobile vision with fpga-based dynamic deep neural network reconfiguration," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pp. 1369–1378, IEEE, 2020.

[86] M. Xu, M. Zhu, Y. Liu, F. X. Lin, and X. Liu, "Deepcache: principled cache for mobile deep vision," in *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, pp. 129–144, ACM, 2018.

[87] M. Long, H. Zhu, J. Wang, and M. I. Jordan, "Unsupervised domain adaptation with residual transfer networks," in *Advances in Neural Information Processing Systems*, pp. 136–144, 2016.

[88] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: efficient inference engine on compressed deep neural network," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pp. 243–254, IEEE, 2016.