SCYLLA: QoE-aware Continuous Mobile Vision with FPGA-based Dynamic Deep Neural Network Reconfiguration

Shuang Jiang†, Zhiyao Ma†, Xiao Zeng★, Chenren Xu†, Mi Zhang★, Chen Zhang♮, Yunxin Liu♮

†Peking University ★Michigan State University ♮Microsoft Research

***Abstract*—Continuous mobile vision is becoming increasingly important as it finds compelling applications which substantially improve our everyday life. However, meeting the requirements of quality of experience (QoE) diversity, energy efficiency and multi-tenancy simultaneously represents a significant challenge. In this paper, we present SCYLLA, an FPGA-based framework that enables QoE-aware continuous mobile vision with dynamic reconfiguration to effectively address this challenge. SCYLLA pre-generates a pool of FPGA design and DNN models, and dynamically applies the optimal software-hardware configuration to achieve the maximum overall performance on QoE for concurrent tasks. We implement SCYLLA on state-of-the-art FPGA platform and evaluate SCYLLA using drone-based traffic surveillance application on three datasets. Our evaluation shows that SCYLLA provides much better design flexibility and achieves superior QoE trade-offs than status-quo CPU-based solution that existing continuous mobile vision applications are built upon.**

I. INTRODUCTION

Continuous mobile vision is becoming an increasingly important service on today's mobile devices, ranging from smartphones, wearable gadgets to drones. State-of-the-art vision-based mobile applications such as augmented reality [1], traffic surveillance [2] and cognitive assistance [3], [4] can substantially revolutionize the way we interact with ambient environments for improving our everyday life.

The unique characteristics of continuous mobile vision, *i.e.,* quality of experience (QoE) (*e.g.,* inference accuracy and latency) difference across tasks, energy efficiency, and heterogeneous multi-tenancy need to be well satisfied and balanced over the journey – it is always desirable to accurately and timely recognize all the in-view objects in fine-grain while extend the battery life as long as possible. Most recently, Deep Neural Network (DNN) has become the *de facto* technology for computer vision tasks because of its superiority in accuracy [5], [6], [7]. However, the model complexities pay for high computational resources [8], [9], leading to high end-to-end latency in CPU-based implementation. GPU exploits operator-level parallelism to achieve low latency, but at a much greater energy cost, which is often not acceptable for mobile devices.

As another feature, in continuous mobile vision, the application context continuously changes. The in-view objects of interest can come and go, and thus they natively carry the multi-tenancy property. For instance, for a drone-based traffic surveillance system, a traffic estimation (*e.g.,* for flow control) task requires a generic object detection and recognition engine, while for each detected vehicle, the type classification and/or license plate recognition task can come as a follow-up query. This feature requires highly efficient system support for multi-tenancy, especially from concurrency and heterogeneity perspective. GPU's SIMT (Single Instruction, Multiple Thread) architecture makes it not able to efficiently execute concurrent and heterogeneous tasks – it suffers from poor performance in concurrent tasks [10] and several seconds of model switching overhead. Most recently, AI chips emerge as an ASIC solution for application-specific performance optimization. Unfortunately, they are not designed for concurrency and heterogeneity. In a nutshell, the nature of *dynamic application QoE requirement, limited energy budget, and heterogeneous multi-tenancy* in continuous mobile vision makes today's mainstream computing engines highly inefficient and inflexible.

Motivated by the problems aforementioned, we present SCYLLA, an FPGA-based framework that enables QoE-aware continuous mobile vision with dynamic reconfiguration to meet the unique characteristics of continuous mobile vision applications. The design of SCYLLA sits on top of FPGA, not only utilizing its intrinsic properties of low latency and energy efficiency, but also exploiting its hardware support for parallelism and its unique reconfiguration capability. Our key insight is that DNN models typically do not fully occupy the on-board resources. Consequently, we can have different FPGA designs to either allow concurrent execution of multiple small (heterogeneous) tasks, or optimize the latency or accuracy which typically requires more on-board resources, with the objective to jointly optimize the compute resource allocation and task scheduling for heterogeneous multi-tenancy in an energy-efficient manner.

Challenges and Solutions. The design of SCYLLA addresses two key challenges. First, it is difficult to provide multiple QoE profiles and fast switch among them to support multi-tenancy and QoE with limited on-board resources. To address this challenge, SCYLLA pre-generates a pool of FPGA design and DNN model profiles with different trade-offs among latency, energy and accuracy. It dynamically re-configures FPGA and selects DNN models for different QoE requirements. With fast reconfiguration[1], SCYLLA is able to efficiently switch among different FPGA design and DNN model profiles to provide flexible supports for QoE and multi-tenancy. Second, it is non-trivial to optimize the overall performance across multiple concurrently running tasks with different QoE requirements [12], [13]. To address this challenge, SCYLLA employs utility functions that encode different QoE metrics together, and uses a QoE-aware task scheduler to select FPGA design and DNN model profiles to maximize the total utility value of the tasks. In doing so, SCYLLA is able to select the "optimal"

[1]We have evaluated that the time for reconfiguration of the logic is about 80$\sim$90 ms on the Xilinx ZCU102 FPGA board [11].

Authorized licensed use limited to: Yale University. Downloaded on January 25,2022 at 19:01:55 UTC from IEEE Xplore. Restrictions apply.

software-hardware configuration and achieve the maximum overall performance on QoE for all the concurrent tasks.

Performance Evaluation. We implemented SCYLLA on state-of-the-art FPGA platform and evaluated SCYLLA using drone-based traffic surveillance application on two real-world datasets and one synthetic dataset. Our evaluation results show that compared to status-quo CPU-based solution, SCYLLA is able to provide flexible and superior trade-offs in the design space among accuracy, processing latency, energy consumption. Specifically, SCYLLA is able to reduce the processing latency by 11.9x and the energy consumption by 71.5x. Moreover, SCYLLA is able to reduce the frame drop percentage by about 61.3% and achieve 60% improvement on the percentage of tasks that meet the latency bounds of latency-critical tasks.

Key Contributions.

- To our best knowledge, SCYLLA is the first FPGA-based framework that enables QoE-aware continuous mobile vision with dynamic DNN reconfiguration to meet the unique characteristics of continuous mobile vision.
- We design a QoE profiling scheme that generates multiple FPGA design and DNN model profiles to support dynamic QoE requirements. We also design a QoE-aware multi-tenant task scheduling scheme to achieve the maximum overall performance on QoE for concurrent tasks.
- Our evaluation shows that SCYLLA provides much better design flexibility and achieves superior QoE trade-offs than status-quo CPU-based solution that existing real-world continuous mobile vision applications are built upon. We believe SCYLLA sheds light on leveraging new hardware and hardware-software co-design techniques to enhance the performance for continuous mobile vision systems.

II. MOTIVATION AND BACKGROUND

A. Motivating Scenario

Our work is motivated by killer applications of continuous mobile vision such as drone-based traffic surveillance – it not only presents a flexible solution not constrained by fixed monitoring point, but also poses research challenges in handling dynamic vision tasks in an energy-efficient manner. For instance, a drone runs the traffic estimation (*i.e.,* vehicle counting) task as a background service for real-time flow control purpose. This process essentially relies on object detection algorithm, which locates every vehicle in the frame and then triggers finer-grained tasks such as vehicle type classification and license plate recognition, to be carried in immediate or future queries. Such tasks come with different QoE requirements from time to time and require completely different computational (*e.g.,* DNN) engines. Such highly dynamic context change poses new challenges to meet the QoE requirements for all the queries.

Why FPGA, not GPU? Given its highly parallel architecture, GPU becomes one of the most popular platforms for DNNs. However, we argue that, compared to GPU, FPGA fits much better for continuous mobile vision for the following two reasons. First, the SIMT design of GPU is optimized for executing a single DNN model. Recent work has shown GPU being highly inefficient in executing multiple DNN models concurrently [10]. However, the multi-tenant characteristic of continuous mobile vision requires systems to concurrently run multiple applications at the same time. The MIMD architecture of FPGA naturally accommodates such multi-tenancy. Second, the energy consumption is crucial to battery-powered mobile systems. The gap in energy consumption makes FPGA more attractive for power-hungry continuous mobile vision. Previous work has already shown that FPGA is energy-efficient and fit for DNN-based vision applications [14], [15], [16].

B. FPGA Primer

In general, FPGA is a large set of programmable logic blocks. The logic blocks can be regarded as many parallel "cores", and (re)configured with arbitrary parallelism design to implement customized functions for performance (*e.g.,* latency and throughput) optimization. FPGA usually contains Look-Up Tables (LUTs) that can be programmed to execute any combinational logic, Digital Signal Processing (DSP) units for arithmetic operations, and Block-RAMs (BRAMs) to store data on the chip. The FPGA is usually connected to an on-board co-processor or a host PC for co-processing. Typically, a piece of off-the-shelf FPGA contains tens of thousand LUTs, thousands of DSPs and hundreds of Block RAMs.

FPGA's special advantages are appealing for achieving the goals of SCYLLA and inspire us to design DNN accelerators and different QoE profiles on it. FPGA is good at latency-sensitive job because it has special capability of circuit-level customization on its massively parallel computing units and on-chip storage banks, which saves a large portion of overheads in general-purpose processors, like instruction fetch, cache miss, task scheduling *etc.* Existing FPGA accelerators achieved 3-30x speedup compared with CPU on different DNN models [17], [15]. What is more, FPGA is an MIMD architecture, which guarantees high concurrency. With proper circuit partition, FPGA is able to support multiple DNN models running at the same time without any need to do time-multiplexing. Besides, FPGA is natively energy-efficient, making it friendly for the mobile scenarios.

III. SCYLLA DESIGN

A. Design Goals

SCYLLA is aimed to support multi-tenant DNN-based computer vision tasks with different QoE requirements. To achieve its full promise, SCYLLA needs to fulfill the following goals:

- **High Concurrency.** As most single computer vision task does not occupy the whole on-board resource, the system should allow as much concurrency as possible to fully utilize the available resources to parallelize the tasks. The parallelization together with FPGA-based acceleration should reduce the processing latency of the tasks.
- **QoE-aware Scheduling.** User requests can come with different QoE requirements. The system design should be able to flexibly adapt to such dynamics while minimizing the overhead in switching between different task executions.

Authorized licensed use limited to: Yale University. Downloaded on January 25,2022 at 19:01:55 UTC from IEEE Xplore. Restrictions apply.

- **Energy-aware Operation.** Energy is often constrained in mobile devices, but can be traded for better performance. The system should have a power-saving mechanism that allows favoring longer operation time by sacrificing QoE.

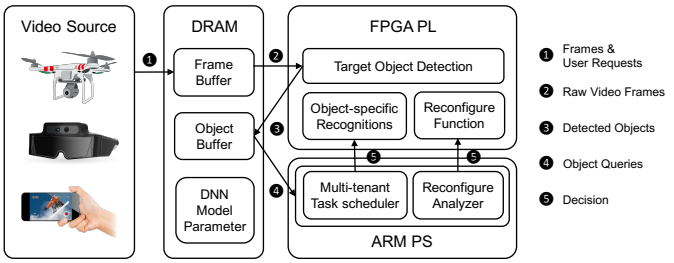


Figure 1: SCYLLA's System Design and Operational Flow.

B. System Overview

As an overview, the SCYLLA design is based on multi-processor system-on-chip with both software (*i.e.,* ARM) and hardware (*i.e.,* FPGA) programmability. We leverage this feature to adopt a *software-hardware co-design scheme*: we implement the query management and task scheduling logic, namely Frame/Object Buffer and Multi-tenant Task Scheduler in the processing system (PS) part, and leave all the compute-intensive workloads (*i.e.,* DNN-based computer vision algorithm) to be accelerated in the re-configurable FPGA programmable logic (PL) part.

The operational flow of SCYLLA is illustrated in Fig. 1. Firstly, Frame Buffer takes all the external input ❶, including both the frames from continuous video stream along with the user request (*i.e.,* objects of interest and requirements on latency, accuracy and energy) arriving randomly. Specifically, it keeps all the input frames and periodically sends these frames to the *Target Object Detection* solver ❷ for recognizing all the in-frame objects. The recognized objects will be ❸ returned to the Object Buffer waiting for further fine-grained processing. Later the queries will be ❹ sent to the Multi-tenant Task Scheduler, which monitors the run-time resource and runs the scheduling algorithm to select an "optimal" FPGA design and DNN models from the profiles for each query to maximize the overall performance of QoE (§V-C). Then the queries and scheduling decisions are sent to the PL part ❺, which will then (optionally) invoke the Reconfigure Function to reconfigure the FPGA design, and execute the corresponding tasks. Upon finishing processing these queries, the system continues to process the frames in the Frame Buffer and repeats the previous operations.

IV. QOE PROFILE GENERATION

One key feature of SCYLLA is to satisfy different QoE requirements for concurrently running computer vision tasks. This requires SCYLLA to provide multiple QoE profiles (*e.g.,* trade-off among latency, accuracy and energy) for each task. Inspired by the customizable and reconfigurable feature of FPGA, we propose to generate multiple FPGA design profiles with different trade-offs between latency and resource usage (achieving different parallelism and energy consumption), and

```
Loop_1: for(o = 0; o < OUT; ++o){
#pragma LOOP_UNROLL factor = Q1
  Loop_2: for(r = 0; r < ROW; ++r){
    Loop_3: for(c = 0; c < COL; ++c){
#pragma LOOP FLATTEN
#pragma LOOP_UNROLL factor = Q2
      Loop_4: for(i = 0; i < IN; ++i){
        Loop_5: for(f1 = 0; f1 < F1; ++f1){
          Loop_6: for(f2 = 0; f2 < F2; ++f2){
            output[o][r][c] +=
              filter[o][i][f1][f2] * input[i][r+f1][c+f2];
}}}}}}
```

Figure 2: Pseudo code of a convolution layer in DNN.

dynamically reconfigure the FPGA at runtime. In the mean-time, model compression is a typical approach to generate multiple DNN models for different QoE settings [18], [19]. We thus leverage *model quantization*, one type of model compression approaches that is suitable for FPGA acceleration to generate multiple DNN model profiles with different trade-offs between latency and accuracy. Lastly, we combine the FPGA design profiles and the DNN model profiles together to provide multiple configurations that vary in terms of QoE. Such $\langle FPGA\ design, DNN\ model \rangle$ combination enriches the QoE profiles, providing more opportunities to optimize the overall performance of the concurrent vision tasks.

A. FPGA Design Profiles

As mentioned earlier (§II), FPGA can be configured with dedicated parallelism design by utilizing different amounts of resources. In other words, we can implement a single "large" accelerator utilizing all the resources to execute one single task; or we can implement multiple "small" instances on FPGA to execute several tasks in parallel. Inspired by this, we design multiple FPGA accelerators that are different in terms of performance and resource utilization for the given DNN-based vision tasks. Specifically, we modify the parameters of optimization methods, including changing the factor of unrolled loops, the pipeline initiation interval (the clock cycles needed to process a new input), to justify the parallelism of accelerators and their resource utilization on FPGA. It should be noted that the obtained FPGA design profiles do not change the precision of the DNN model parameters, meaning that running the same DNN model under different FPGA profiles will result in different latency, but the inference accuracy of DNN models will remain unchanged. Besides, with the occupied resource decreasing, the energy consumption on FPGA will also be reduced, meaning that we can save more energy by running a DNN model on a "small" instance with a relatively long latency as trade-off.

We take the convolution layer which is usually the most computation-intensive component [9], [20] in DNN models as an example. A convolution layer plays the role of "feature extractor" by convolving the input images or *feature maps* from previous layers with a set of filters, which has the dominant computation workload in deep learning algorithms. Fig. 2 illustrates a standard 6-loop convolution algorithm (the stride is set as 1 here). It generates OUT feature maps whose size are $ROW \times COL$ by convolving IN input feature maps

Authorized licensed use limited to: Yale University. Downloaded on January 25,2022 at 19:01:55 UTC from IEEE Xplore. Restrictions apply.

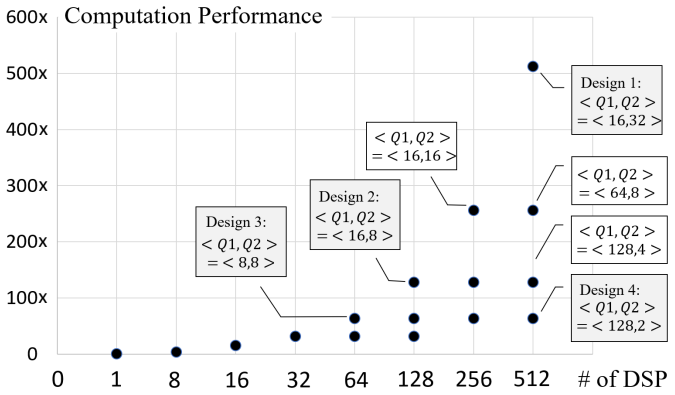


Figure 3: Computation performance under different configurations for the 3rd conv layer in YOLO-tiny.

with OUT filters sized in $IN \times F1 \times F2$. To parallelize the code for acceleration on FPGA, *Loop Unrolling* is an effective method. For instance, if we unroll *Loop_1*, *Loop_2* and *Loop_3* with a unrolling factor pair $\langle Q1, Q2\rangle$[2][3], $Q1 \times Q2$ processing units (*e.g.,* the DSP units) will be used to run in parallel, and the latency is expected to be reduced by about $Q1 \times Q2$ times ideally. Increasing $\langle Q1, Q2\rangle$ will increase parallelism and improve performance. Therefore, the computation performance of FPGA accelerator for CNN can be formulated as the following equation:

$$Comp\ perf\ = \frac{OUT \cdot ROW \cdot COL \cdot IN \cdot F1 \cdot F2}{\lceil \frac{OUT}{Q1} \rceil \cdot \lceil \frac{ROW \cdot COL}{Q2} \rceil \cdot IN \cdot F1 \cdot F2} \quad (1)$$

Given the constraint that $Q1 \times Q2$, the total used DSP number, should be smaller than the resource upper bound provided by FPGA. $Q1,\ Q2$ are non-negative variables.

$$0 < Q1 \times Q2 < \#\ of\ DSPs \quad (2)$$

We use the convolution from layer 9 (the 3rd convolution layer) in YOLO-tiny model as an example to illustrate our scheme. Tab. I shows the layer configuration of this convolution. With above formulation, we can define a design space for this convolution configuration with variance to different $\langle Q1, Q2\rangle$. Fig. 3 shows this design space for the example. From this figure, we have two observations. First, even with the same number of DSPs, different $\langle Q1, Q2\rangle$ design performance varies. For $\#\ of\ DSPs = 512$, optimal design configuration "design 1" is 8x faster than the worst "design 4". Second, with increase of the number of DSPs in use, the performance of optimal design increases accordingly. Thus, we leverage this optimization scheme to help us explore the trade-offs between DSP resource usage and latency. In the following sections, we will evaluate three design instances.

[2]The unroll factor *Q1* means that *Loop_1* will only execute *OUT/Q1* times, and each time *Q1* duplications of the inner logic will execute in parallel.

[3]Since *Loop_2* and *Loop_3* are row and column dimensions for output image. They can be flattened to a single loop and use one unrolling factor

Input feature map (IN)	64
Output feature map (OUT)	32
Feature map size ($ROW \times COL$)	112×112
Kernel size ($F1 \times F2$)	3×3

Table I: Configuration of 3rd convolution in YOLO-tiny.

B. DNN Model Profiles

To satisfy the different requirements on accuracy and latency, we seek to generate DNN models that vary on accuracy and latency with *model compression* techniques. *Model quantization* is one type of compression techniques. The key idea is to quantize the DNN model parameters from high bit-width floating point representations (*e.g.,* 32-bit floating point) to lower bit-width fixed point [21], [22] or even binarized ones [23]. In doing so, the models can be stored and computed under low bit-width; and the latency for executing the quantized models can be effectively reduced. At the same time, as the quantization reduces the precision of parameters, it also leads to a loss on inference accuracy. Usually, less bit width results in lower accuracy but faster processing speed [22], [24]. The existence of accuracy-latency trade-off enables us to generate multiple DNN model profiles with different trade-offs to meet diverse QoE requirements via quantization.

Due to the customizable feature, FPGA can be designed to work at any bit width (from 1 to a hardware-related upper bound). Thus, quantization is naturally suitable for FPGA-based acceleration. For each vision task and its DNN models, we generate a set of quantized models with different bit width (*e.g.,* 8-bit and 6-bit) as the model profiles. These model profiles with different accuracy-latency trade-offs are used together with the FPGA design profiles. Each $\langle FPGA\ design, DNN\ model\rangle$ pair can decide a $\langle latency, accuracy, resource\ utilization\rangle$ vector for a specific task. This combination enriches the space of QoE profiles, providing more choices for the task scheduler to optimize the performance on QoE of given vision tasks.

C. Performance Study

We implement three FPGA configuration plans on the Xilinx ZCU102 board [11] and choose three DNN models to study the performance of our profiling method. As mentioned

	DSP	LUT	FF	BRAM
Design 1	51.9	57.6	21.7	79.0
Design 2	38.1	70.2	26.98	81.15
Design 3	33.24	81.16	31.4	78.02

Table II: Resource utilization (%) of FPGA.

	YOLO-tiny (mAP)	MobileNetSSD	GoogLeNet
6-bit	52.2	84.8	87.99
8-bit	56.5	86.6	89.91
32-bit (original)	57.1	87.7	91.2

Table III: Accuracy (%) of model profiles.

Authorized licensed use limited to: Yale University. Downloaded on January 25,2022 at 19:01:55 UTC from IEEE Xplore. Restrictions apply.

before, we implement generic convolution kernels with different parallelism. Tab. II illustrates the total resource utilization on FPGA. Design 1 uses the most computation resources with the best performance. Design 2 and 3 use less resource and thus take a longer time.

	YOLO-tiny		MobileNetSSD		GoogLeNet	
	6-bit	8-bit	6-bit	8-bit	6-bit	8-bit
Design 1	188.3	294.1	48.1	89.3	31.8	44.4
Design 2	303.3	384.6	116.3	208.3	80.6	135.1
Design 3	400.1	526.3	263.1	370.1	228.8	322.6

Table IV: Inference latency (ms) under different profiles.

	YOLO-tiny		MobileNetSSD		GoogLeNet	
	6-bit	8-bit	6-bit	8-bit	6-bit	8-bit
Design 1	0.873	1.361	0.222	0.412	0.149	0.208
Design 2	0.796	1.01	0.191	0.342	0.214	0.367
Design 3	0.684	0.964	0.172	0.242	0.391	0.514

Table V: Energy cost (J/image) under different profiles.

For each DNN model, we generate 6-bit and 8-bit fixed-point instances via model quantization. The accuracy of them are shown in Tab. III. For all the DNNs here, the accuracy of 8-bit instance is higher than the 6-bit ones. Combining the FPGA configuration and quantized DNN models, we evaluate the processing latency of each DNN model under different profiles and Tab. IV shows the results. We can see that under the same FPGA configuration, the latency of 6-bit quantized model is shorter than that of the 8-bit model for each DNN. On the other hand, for a given DNN model, its latency also varies under different FPGA configurations. Design 3 has the longest latency as the computing power of each kernel is relatively low, while design 1 achieves the shortest latency. Tab. V lists the energy consumption and we can observe similar results that for each DNN model, the energy consumption varies with different FPGA configurations and quantization bits. Such trade-off provides us multiple selections to schedule multiple vision tasks to maximize the overall performance of QoE.

V. QoE-aware Task Scheduling

A. Principles of QoE-aware Task Scheduling

With the generated FPGA design and DNN model profiles, SCYLLA is able to optimize the overall performance for multiple vision tasks. The key is to formulate and jointly maximize the "rewards" of QoE parameters including latency, accuracy and energy. Unlike the scheduling strategies of CPU-based platforms, in which each request can acquire a part of the CPU time that makes them run "concurrently", the number of tasks that can be processed in parallel is decided by the number of computing engines for a certain FPGA design. If the number of requests exceeds the number of engines, some requests must wait until an engine is available. This makes the scheduling problem on FPGA different from previous works as we must consider the impact of waiting time of tasks.

We are to balance among three factors of tasks, namely latency, accuracy and energy. Ideally, we want each task to be completed as soon as possible, *i.e.,* with short latency, along with high recognition accuracy and low energy consumption. However, there is a complex interaction or trade-off among latency, accuracy and energy as described in §IV-C. To deal with the complexities, we leverage the principle of utility function to combine those three factors for joint optimization, which is a common practice in computer systems [13], [25], [26]. We search through the possible schedule plans, and choose the one that achieves the highest utility value. As the optimization of utility value is NP-Hard, we design a heuristic approach to search for an approximate optimal solution.

B. Problem Formulation

Let U represent the set of mobile vision tasks to be scheduled and P represent the set of FPGA design profiles. For a task $u \in U$, we let M_u represent the set of DNN models to execute u. Given an FPGA profile $p \in P$ and a model profile $m_u \in M_u$, we let $t(p, m_u, u)$, $acc(p, m_u, u)$ and $e(p, m_u, u)$ represent the processing latency, accuracy and energy of task u under these conditions, respectively. In fact, $t(p, m_u, u)$ contains the *time for execution on FPGA*, *time to wait for available engines* and *time to reconfigure the FPGA*, which can be written in the following form:

$$t(p, m_u, u) = t_{exec}(p, m_u, u) + t_{wait} + \mu \cdot t_{reconfig}(p) \quad (3)$$

where μ is 0 or 1, representing whether the reconfiguration process happens. t_{wait} of a task is decided by the number of tasks before it, their processing latency and the number of computing engines under FPGA profile p. Besides, we let $N(p)$ denote the number of engines on FPGA for given p, and $N_{cur}(t)$ denote the number of tasks running in parallel at moment t. $N(p)$ is mainly decided by the "bottleneck" resource to implement one engine (*e.g.,* the BRAM utilization shown in Tab. II). Our utility function is defined as follows:

$$\begin{aligned} U(p, m_u, u) =& \alpha_T \cdot min(0, t_{max}(u) - t(p, m_u, u)) + \\ & \alpha_A \cdot (acc(p, m_u, u) - acc_{min}(u)) + \\ & \alpha_E \cdot (e_{max}(u) - e(p, m_u, u)) \end{aligned} \quad (4)$$

where $t_{max}(u)$, $acc_{min}(u)$ and $e_{max}(u)$ are the minimum or maximum QoE goals for task u, respectively. α_T, α_A and α_E are constant parameters in the range of $[0, 1]$ to control the QoE trade-off preference.

To maximize the performance of our system, we choose to *maximize sum of utilities*, which is a typical scheduling scheme used in previous work [18], [13], [27]. The optimization objective can be formulated as follows:

$$\begin{aligned} &\max_{p \in P, m_u \in M_u} \sum_{u \in U} U(p, m_u, u) \\ &s.t. \forall p \in P, \forall t, N_{cur}(t) \leq N(p) \end{aligned} \quad (5)$$

The solution should select a DNN model for each task and a FPGA design profile to execute all the tasks, as well as the order to execute the given tasks. Solving such a non-convex problem to maximize the sum of utilities is computationally

Authorized licensed use limited to: Yale University. Downloaded on January 25,2022 at 19:01:55 UTC from IEEE Xplore. Restrictions apply.

hard. Based on this optimization problem, we consider the strategies for FPGA reconfiguration and energy consumption discussed before, and propose a heuristic scheduling scheme to support online multi-tenant scheduling.

C. QoE-aware Task Scheduling Scheme

The FPGA platform we use can be configured to have 1, 2 or 4 kernels (*i.e.,* tasks that can be processed in parallel). Tasks must be divided into 1, 2 or 4 sets accordingly. To partition tasks in real time, we design a greedy division scheme as shown in Alg. 1. It repeatedly chooses the set which has the minimum total execution time and assigns a task to it.

Algorithm 1 Task Division Scheme

Input: A sequence of tasks U
Output: A division of tasks to several task sets S, where $|S| = 1, 2$ *or* 4
for each $u \in U$ **do**
  Select $s \in S$ with smallest total execution time
  Assign u to s
end for
return S

Algorithm 2 Executing Sequence Determination and Model Selection Scheme

Input: A set of tasks s
Output: Plan: <*execution order, model profile*>
Sort all $u \in s$ in ascending order of $t_{max}(u)$
for each $u \in s$ **do**
  Set $quant(u) = 8bit$
end for
while $\exists u \in s$ s.t. $t(p, m_u, u) > t_{max}(u)$ and $\exists u \in U$ s.t. $quant(u) == 8bit$ **do**
  for each $u \in s$ s.t. $quant(u) == 8bit$ **do**
    Calculate total utility increase $\Delta util(u)$ when setting $quant(u) = 6bit$
  end for
  Select $u_{sel} = \text{argmax}_u \, \Delta util(u)$
  if $\Delta util(u_{sel}) > 0$ **then**
    Set $quant(u) = 6bit$
  else
    Break the loop
  end if
end while

Given the divided sets of tasks, we design a scheme to determine the executing sequence and the quantization of model to use for each set of the tasks. Specifically, as illustrated in Alg. 2, considering that the latency will accumulate with the tasks waiting for available engines, we firstly sort tasks by their latency requirements to let the tasks with shorter latency bounds execute earlier. We set all tasks to use 8-bit quantization at the beginning. If all latency requirements are met, the searching is done; otherwise, we repeatedly choose one task that changing its model from 8-bit to 6-bit increases the total utility value the most, until that we cannot further increase total utility value or all models are set to 6-bit.

Finally, we compare among the FPGA design profiles and choose the one with the greatest total utility value.

D. Put Everything Together

Fig.4 shows a real instance of scheduling and execution time pattern, from which we can see the alternative execution of object detection, *i.e.,* the YOLO-tiny, and fine-grained recognition, *i.e.,* the GoogLeNet and MobileNetSSD. In particular,

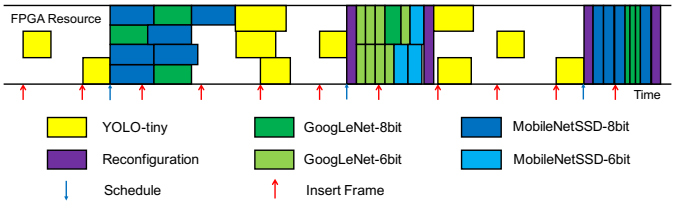


Figure 4: SCYLLA's scheduling sequence snapshot.

the scheduler dynamically selects the FPGA design and DNN model profiles based on the calculated utility values. The width of YOLO-tiny blocks varies because the complexity of frame influences the running time. When executing fine-grained tasks on 4 kernels, each job takes a relatively prolonged time, but 4 tasks can run in parallel. On the other hand, 1-kernel profile permits each task to run at full speed but in sequence. Reconfiguration happens when the current FPGA design is not the needed one. One thing to mention is that we run 3 YOLO-tiny tasks in parallel instead of 4 tasks, because the bandwidth of the on-chip bus is not large enough to support 4 YOLO-tiny tasks in parallel.

VI. IMPLEMENTATION

We use a Xilinx ZCU102 FPGA board [11], one of the newest FPGA developing board, as the hardware platform to prototype SCYLLA. The Xilinx ZCU102 board contains a 1.2-GHz ARM Cortex-A53 processor and a XCZU9EG FPGA. We implement the DNN accelerators (*i.e.,* the FPGA PL part) and generate the FPGA design profiles based on CHaiDNN [28] using Xilinx SDx 2018.2 Tool. Specifically, the DNN accelerators are firstly implemented in C/C++, and then we analyze the C/C++ code to localize the most compute-intensive "bottlenecks", and insert HLS pragmas to let SDx generate pipelines, unrolled loops or leverage other optimization techniques to improve the parallelism. Then SDx invokes HLS toolchains to synthesize hardware code (*e.g.,* Verilog) from the C/C++ implementation and generate a bitstream to configure the FPGA logic. We leverage the basic implementation of DNN engines (*e.g.,* convolution) in CHaiDNN, and modify the source codes for different parallelism and resource utilization to generate the FPGA design profiles. Besides, we also modify the software stack of CHaiDNN, moving its online parameter quantization process to be executed *offline*. The modification reduces the initialization time of running DNN inference from nearly 30s to about 2s for the DNN models we used, making CHaiDNN able to support the online mobile scenarios and fast re-initialization when changing DNN models (Tab. VI). In general, our modification extends the function and improves the performance of CHaiDNN. The *Task Scheduler* is implemented using C++ and running on the ARM co-processor.

	YOLO-tiny	MobileNetSSD	GoogLeNet
CHaiDNN	21.67	8.31	5.07
Modified	2.08	1.58	1.02

Table VI: Loading time (s) of DNN models.

Authorized licensed use limited to: Yale University. Downloaded on January 25,2022 at 19:01:55 UTC from IEEE Xplore. Restrictions apply.

VII. Evaluation

A. Experimental Setup

Tasks and DNN models. We choose *drone-based traffic surveillance*, a typical and important continuous mobile vision application for our evaluation. Due to the mobility, drone is able to track traffic conditions in large areas from dynamic views, providing services that fixed surveillance cameras cannot provide. Our application includes three types of vision tasks in traffic surveillance:

- **Object Detection.** Object detection is the "backbone" task in traffic surveillance systems as the detected objects can be used for further classification. It detects the generic category of objects (*e.g.,* cars, person, traffic signs *etc.*) and finds their locations in the frame. We select YOLO [29], one of the most commonly used DNN models, which is well-known for its fast speed and acceptable accuracy, for our object detection task. The model is trained on the Pascal VOC 2012 dataset [30] which contains 21 classes.
- **License Plate Recognition.** This task detects the position of license plates and identifies the numbers on the plates, which usually works together with traffic violation detection systems. For this task, we use a MobileNetSSD model provided in the HyperLPR project [31]. In our system, this application takes the detected cars from YOLO, detects and recognizes the license plate numbers.
- **Car Type Classification.** Our last task aims to recognize the specific type (brands and models, *e.g.,* Audi A6) of detected cars for further data analysis as described in [13]. We select the GoogLeNet model in the Caffe Model Zoo [32] for this task. The model was trained on the CompCars dataset [33] that contains nearly all common car types.

Video Datasets. We evaluate SCYLLA using two real-world datasets and one synthetic dataset. Due to lack of public drone-based surveillance datasets, we use our own video clips captured by DJI Mavic Pro [34], the most advanced commercial drones. We also capture video clips on streets using a Sony dsc-RX100 camera [35] and select five public videos captured by the surveillance cameras on streets from YouTube. These video clips captured by fixed cameras are similar with the cases when drones are hovering in the air (*e.g.,* serving at crossroads in the countryside without surveillance cameras). To cover more cases, we generated synthetic video data using Unreal CV [36]. Fig. 5 illustrates three example frames of different datasets. We select and split 22 representative video clips and the length of each clip is about 10∼20s. During evaluation, we extract raw images from the clips, store them on the FPGA board, and feed them sequentially, mimicking the real video ingestion. The tasks, DNN models and datasets for training are summarized in Tab. VII.

Task	DNN Model	Training Dataset
Object Detection	YOLO-tiny	Pascal VOC 2012
License Plate Recognition	MobileNetSSD	HyperLPR
Car Type Classification	GoogLeNet	CompCars

Table VII: Tasks, DNN models and datasets in our work.

(a) Drone Data (b) Fixed Camera Data (c) Synthetic Data

Figure 5: Example frames of the three datasets.

Baseline. The status-quo for real-world deployed continuous mobile vision applications is based on CPU. We thus set up a CPU-based design as our baseline. For fair comparison, we use an Intel Core-i7 7700HQ CPU with 4 GB memory (the same as that on ZCU102 board) as the hardware platform. We use the CPU version of Caffe [37] framework to execute the DNN models because CHaiDNN only supports Caffe's model format as input. We firstly evaluate the latency and energy consumption to execute the three DNN models on this CPU platform. We then take the scheduled execution trace of SCYLLA as input and conduct a trace-driven emulation. For the multiple-kernel execution on FPGA, we create given number of threads and make the threads sleep for a given duration of time to emulate the process of execution. As we do not have support for 8-bit/6-bit quantization on the CPU platform, we use the original DNN model parameters (32-bit) without quantization. Tab. VIII shows the performance of the baseline.

	YOLO-tiny	MobileNetSSD	GoogLeNet
Latency (ms)	579.1	629.3	563.5
Energy (J/image)	17.25	18.87	17.06

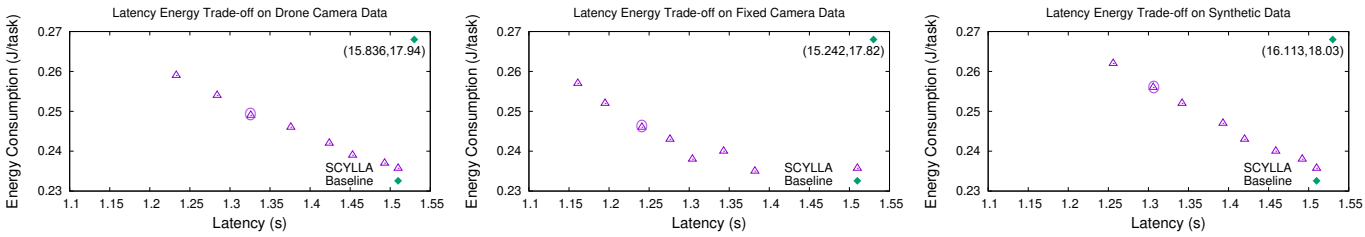
Table VIII: Performance on the CPU Platform.

B. Results

We process all the video clips in the three datasets with SCYLLA and the baseline. For SCYLLA, we change the α_T, α_A and α_E in the utility function to tune the scheduling preference and obtained different performance results.

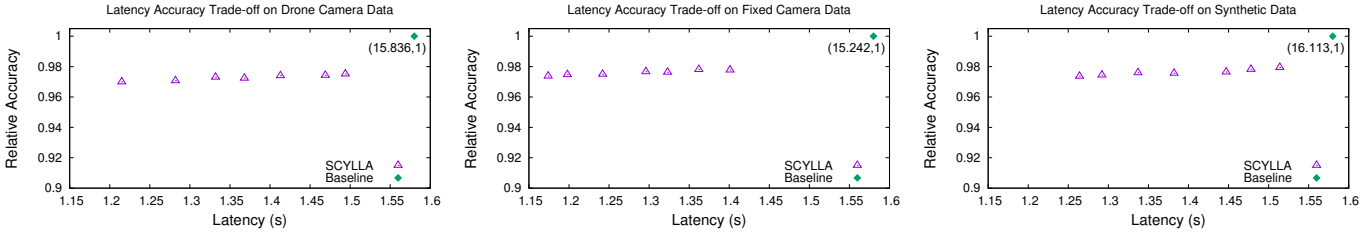
Latency-Energy Trade-off. Firstly, we keep α_A fixed and tune the ratio of α_T and α_E to evaluate the overall performance on latency and energy consumption. Fig. 6 compares the performance of SCYLLA and baseline over the three datasets. We have two key observations. First, by tuning the ratio of α_T and α_E, SCYLLA is able to achieve different trade-offs between latency and energy consumption while the baseline can only provide fixed performance, demonstrating the superiority of SCYLLA on providing flexible selections on QoE. Second, SCYLLA performs much better on both latency and energy consumption than baseline. For the drone dataset, the best configuration of SCYLLA in our evaluation (the circled purple triangle point that has the longest normalized Euclidean distance to baseline, meaning the best overall performance) reduces the latency by 11.9x. For the fixed camera data and synthetic data, the latency is reduced by 12.3x and 11.4x, respectively. One main reason is that the time to execute YOLO-tiny on CPU is long. It can only process 1.7 frames per second. The frames accumulate rapidly and cannot be consumed in time, resulting in long delay before processing,

Authorized licensed use limited to: Yale University. Downloaded on January 25,2022 at 19:01:55 UTC from IEEE Xplore. Restrictions apply.



(a) Drone Data (b) Fixed Camera Data (c) Synthetic Data

Figure 6: Latency-Energy Performance of SCYLLA and Baseline.



(a) Drone Data (b) Fixed Camera Data (c) Synthetic Data

Figure 7: Latency-Accuracy Performance of SCYLLA and Baseline.

and thus causing long latency. As for energy consumption, the best configuration of SCYLLA can reduce the average energy consumption by nearly 71.5x on average over the three datasets, meaning that SCYLLA is much more energy-efficient.

Latency-Accuracy Trade-off. Next, we keep α_E fixed and tune the ratio of the other two parameters to see the overall performance on processing latency and inference accuracy. The results are shown in Fig. 7. As we use the original floating-point model parameters for our CPU-based baseline, its accuracy is marginally higher than that in SCYLLA due to the precision differences. We normalized the highest accuracy (89.57% on average over the three datasets) as 1.0 to see the relative accuracy loss. We have similar observation that SCYLLA has a trade-off between latency and accuracy by adjusting α_T and α_A. Compared with the baseline, SCYLLA can reduce the latency by 12.1x with only 2.3% loss in terms of relative accuracy on average. When tuning α_T and α_A, the changes on latency is relatively small. This is because we can only provide two selections of accuracy (*i.e.,* 6-bit and 8-bit) for each DNN model given the constraint of CHaiDNN. With CHaiDNN planning to support more selections of model quantization, SCYLLA will be able to exhibit larger advantage in terms of latency-accuracy trade-off.

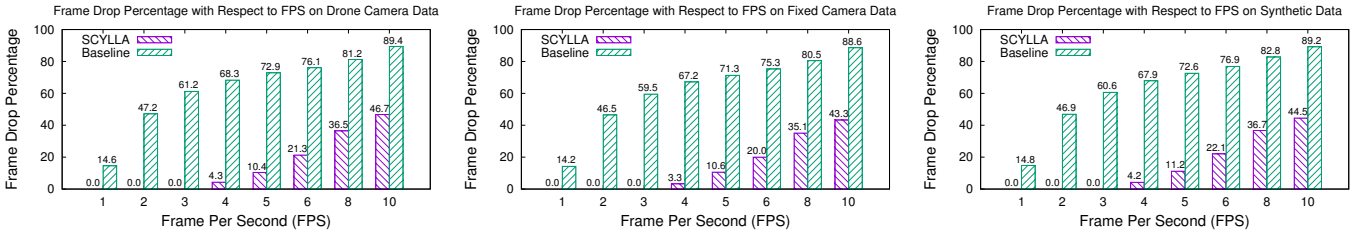
Frame Drop Percentage. When the input frames cannot be processed by YOLO-tiny in time, they will accumulate in the *Frame Buffer*. However, the size of *Frame Buffer* is always limited in real systems. Frames will be dropped if the *Frame Buffer* is full. In other words, smaller *Frame Drop Percentage* means better performance of the system. We evaluate the *Frame Drop Percentage* under different input frame rates in unit of frames per second (FPS). We set the size of *Frame Buffer* to 3x of the FPS (*e.g.,* 15 for 5 FPS) to provide a 3-second latency tolerance. For SCYLLA, we choose the "best" configurations in Fig. 6 (*i.e.,* the circled purple triangle points) for the three datasets, respectively. We can see Fig. 8 that for all the three datasets, with FPS increasing, the frame drop percentage increases for both SCYLLA and the baseline, but SCYLLA achieves much lower value than CPU. SCYLLA begins to drop a few frames (4.3% on average) when FPS increases to 4, while CPU drops about 14.5% of the frames even if FPS is only 1. When FPS achieves 10, SCYLLA drops nearly half of the frames (45% on average), while CPU drops nearly 90% of the frames. We can safely say that SCYLLA can work at 5 FPS well with only 10% frames dropped which is acceptable in many cases, reducing the frame drop percentage by 61.3% compared with CPU. This reduction significantly improves the user experience.

Latency Bound Meet Percentage. For many continuous mobile vision applications, latency is the most important QoE metric. The number of processed tasks that meet the latency "deadline" or *Maximum Latency Bound (MLB)* defined by users can reflect the performance of system. Thus, we evaluate the percentage of tasks whose latency is within the MLB. We set several different values of MLB, and choose the best configurations of SCYLLA to compare it with the baseline. As shown in Fig. 9, SCYLLA performs much better than CPU on all the three datasets. When using SCYLLA, about 31.4% of the tasks finished within the 1-second MLB, which is near real-time in terms of human perception [4], while no task can be finished within 1 second when using the CPU baseline. With MLB increasing, the percentage of tasks that meet the MLB increases rapidly on SCYLLA. When MLB is 3, nearly all the tasks (98.4% on average) running on SCYLLA can meet this latency bound, while this value is only 24.3% for CPU. Even if the MLB increases to 5, only 39.5% of the tasks can meet the latency bound on CPU. We can safely conclude that SCYLLA can achieve more than 60% improvement on finishing the tasks within a MLB that is less than 5 seconds.

VIII. DISCUSSION

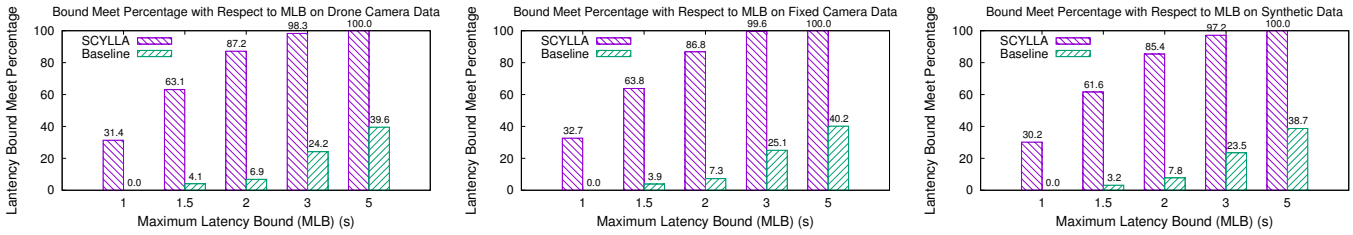
Scalability. While in this work, we prototyped SCYLLA on a single FPGA board, the design of SCYLLA can be easily scaled

Authorized licensed use limited to: Yale University. Downloaded on January 25,2022 at 19:01:55 UTC from IEEE Xplore. Restrictions apply.



(a) Drone Data (b) Fixed Camera Data (c) Synthetic Data

Figure 8: Frame Drop Percentage of SCYLLA and Baseline.



(a) Drone Data (b) Fixed Camera Data (c) Synthetic Data

Figure 9: Latency Bound Meet Percentage of SCYLLA and Baseline.

to cluster or even LAN scale with centralized management, and can potentially play critical roles in mobile edge computing to serve more clients with (predictive) dynamic requests.

Fine-grained Reconfiguration. In this work, we design SCYLLA under the constraint of 3 reconfigurable choices with different numbers of same kernels on FPGA. The number of choices is limited because CHaiDNN only allows such configurations. In fact, FPGA's MIMD architecture and reconfiguration capability enables combination of different types of computation engines. With best-fit hardware design for specific DNN structures, SCYLLA can provide more reconfiguration choices and schedule massive concurrent small and heterogeneous tasks, and we would expect higher performance gain.

IX. Related Work

SCYLLA is inspired by the increasing importance of continuous mobile vision and multi-tenant task scheduling, but differs from existing work on hardware platform and techniques that support QoE-aware multi-tenant task scheduling.

Continuous Mobile Vision. Recent years have witnessed the great efforts to optimize the performance of continuous mobile vision based on CPU [12], [20], [18], GPU [38], [39] and FPGA [40], [41]. As for on-device solutions, DeepCache [20] caches and reuses the results of previous frames via effective image matching to reduce latency and energy cost. NestDNN [18] builds multi-capacity DNN models to provide resource-accuracy trade-offs and leverages a resource-aware scheduler to jointly optimize the performance of latency and accuracy. Our work is inspired by the aforementioned work but differs with the main innovation that leverages the unique reconfiguration property of FPGA, provides flexible performance trade-off among different QoE metrics, and optimizes the overall performance on QoE accordingly.

QoE-aware Multi-Tenant Task Scheduling. Many existing systems such as Morpheus [42] and Jockey [43] dynamically allocate resources to optimize the latency of streaming tasks for data centers. Some real-time systems leverage utility or cost functions to satisfy the soft deadlines of multi-tenant tasks [44], [13], [18]. In particular, VideoStorm [13] firstly considers "processing quality"(*i.e.,* QoE) for live video analytic and jointly optimizes the performance across multiple queries via task scheduling. Compared with the existing systems, SCYLLA focuses on the resource-constrained continuous mobile vision scenario instead of large-scale clusters, and it introduces new challenges for QoE-aware task scheduling specific to FPGA.

X. Conclusion

This paper presents SCYLLA, an FPGA-based framework that enables multi-tenant dynamic QoE support for continuous mobile vision applications. SCYLLA proposes a novel reconfiguration-based profile generation approach which pre-generates a pool of FPGA design and DNN model profiles with different QoE performance. At runtime, SCYLLA dynamically selects the optimal software-hardware configuration via QoE-aware task scheduling to jointly optimize the performance on latency, energy and accuracy for the concurrent vision tasks. Our evaluation shows that SCYLLA is able to reduce the processing latency by 11.9x and saving 71.5x of the energy consumption compared to status-quo CPU-based solution. Besides its superior QoE performance, SCYLLA is able to reduce the frame drop percentage by 61.3% and achieve 60% higher latency bound meet percentage.

Acknowledgments

This work is supported in part by National Key Research and Development Plan, China (Grant No. 2016YFB1001200), National Natural Science Foundation of China (Grant No. 61802007), Science and Technology Innovation Project of Foshan City, China (Grant No. 2015IT100095), and NSF Awards CNS-1617627 and PFI:BIC-1632051. Chenren Xu is the corresponding author: chenren@pku.edu.cn

Authorized licensed use limited to: Yale University. Downloaded on January 25,2022 at 19:01:55 UTC from IEEE Xplore. Restrictions apply.

REFERENCES

[1] H. Qiu, F. Ahmad, F. Bai, M. Gruteser, and R. Govindan, "Avr: Augmented vehicular reality," in *ACM MobiSys*, 2018.

[2] K. Kanistras, G. Martins, M. J. Rutherford, and K. P. Valavanis, "Survey of unmanned aerial vehicles (uavs) for traffic monitoring," *Handbook of unmanned aerial vehicles*, 2015.

[3] K. Ha, Z. Chen, W. Hu, W. Richter, P. Pillai, and M. Satyanarayanan, "Towards Wearable Cognitive Assistance," in *ACM MobiSys*, 2014.

[4] Z. Chen, W. Hu, J. Wang, S. Zhao, B. Amos, G. Wu, K. Ha, K. Elgazzar, P. Pillai, R. Klatzky, D. Siewiorek, and M. Satyanarayanan, "An Empirical Study of Latency in an Emerging Class of Edge Computing Applications for Wearable Cognitive Assistance," in *ACM SEC*, 2017.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012.

[6] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva, "Learning Deep Features For Scene Recognition Using Places Database," in *NIPS*, 2014.

[7] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deepface: Closing the Gap to Human-level Performance in Face Verification," in *IEEE CVPR*, 2014.

[8] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, 2017.

[9] C. Zhang, G. Sun, Z. Fang, P. Zhou, P. Pan, and J. Cong, "Caffeine: Towards Uniformed Representation and Acceleration For Deep Convolutional Neural Networks," *IEEE TCAD*, 2018.

[10] P. Jain, X. Mo, A. Jain, H. Subbaraj, R. S. Durrani, A. Tumanov, J. Gonzalez, and I. Stoica, "Dynamic space-time scheduling for gpu inference," in *NeurIPS*, 2018.

[11] "Xilinx Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit," https://www.xilinx.com/products/boards-and-kits/ek-u1-zcu102-g.html.

[12] R. LiKamWa and L. Zhong, "Starfish: Efficient Concurrency Support for Computer Vision Applications," in *ACM MobiSys*, 2015.

[13] 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 *USENIX NSDI*, 2017.

[14] S. Jiang, D. He, C. Yang, C. Xu, G. Luo, Y. Chen, Y. Liu, and J. Jiang, "Accelerating Mobile Applications at the Network Edge with Software-Programmable FPGAs," in *IEEE INFOCOM*, 2018.

[15] Q. Xiao, Y. Liang, L. Lu, S. Yan, and Y.-W. Tai, "Exploring Heterogeneous Algorithms for Accelerating Deep Convolutional Neural Networks on FPGAs," in *ACM DAC*, 2017.

[16] R. Zhao, W. Song, W. Zhang, T. Xing, J.-H. Lin, M. Srivastava, R. Gupta, and Z. Zhang, "Accelerating Binarized Convolutional Neural Networks with Software-Programmable FPGAs," in *ACM FPGA*, 2017.

[17] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks," in *ACM FPGA*, 2015.

[18] B. Fang, X. Zeng, and M. Zhang, "NestDNN: Resource-Aware Multi-Tenant On-Device Deep Learning for Continuous Mobile Vision," in *ACM MobiCom*, 2018.

[19] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, and A. Krishnamurthy, "MCDNN: An Approximation-Based Execution Framework for Deep Stream Processing Under Resource Constraints," in *ACM MobiSys*, 2016.

[20] M. Xu, M. Zhu, Y. Liu, F. X. Lin, and X. Liu, "DeepCache: Principled Cache for Mobile Deep Vision," in *ACM MobiCom*, 2018.

[21] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep Learning With Limited Numerical Precision," in *ICML*, 2015.

[22] S. Anwar, K. Hwang, and W. Sung, "Fixed Point Optimization of Deep Convolutional Neural Networks for Object Recognition," in *IEEE ICASSP*, 2015.

[23] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training neural networks with weights and activations constrained to +1 or -1," *arXiv e-print*, Feburary 2016.

[24] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized Convolutional Neural Networks for Mobile Devices," in *IEEE CVPR*, 2016.

[25] L. Amini, N. Jain, A. Sehgal, J. Silber, and O. Verscheure, "Adaptive Control of Extreme-scale Stream Processing Systems," in *IEEE ICDCS*, 2006.

[26] R. Johari and J. N. Tsitsiklis, "Efficiency Loss in a Network Resource Allocation Game," *Mathematics of Operations Research*, vol. 29, no. 3, 2004.

[27] P. Marbach, "Priority Service and Max-Min Fairness," in *IEEE INFOCOM*, 2002.

[28] "CHaiDNN: An HLS based Deep Neural Network Accelerator Library for Xilinx Ultrascale+ MPSoCs," https://github.com/Xilinx/CHaiDNN.

[29] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-time Object Detection," in *IEEE CVPR*, 2016.

[30] "The Pascal VOC 2012 Dataset," http://host.robots.ox.ac.uk/pascal/VOC/voc2012/.

[31] "HyperLPR: A High Performance License Plate Recognition Framework," https://github.com/zeusees/HyperLPR.

[32] "Caffe Model Zoo," http://caffe.berkeleyvision.org/model_zoo.html.

[33] "The Comprehensive Cars (CompCars) dataset," http://mmlab.ie.cuhk.edu.hk/datasets/comp_cars/index.html.

[34] "DJI Mavic Pro." 2018, https://www.dji.com/mavic.

[35] "SONY dsc-RX100 V," https://www.sony.com/electronics/cyber-shot-compact-cameras/dsc-rx100m5a.

[36] "UnrealCV," https://unrealcv.org/.

[37] "Caffe Deep Learning Framework," http://caffe.berkeleyvision.org/.

[38] L. N. Huynh, Y. Lee, and R. K. Balan, "DeepMon: Mobile GPU-based Deep Learning Framework for Continuous Vision Applications," in *ACM MobiSys*, 2017.

[39] 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.

[40] S. Jiang, D. He, C. Yang, C. Xu, G. Luo, Y. Chen, Y. Liu, and J. Jiang, "Accelerating mobile applications at the network edge with software-programmable fpgas," in *IEEE INFOCOM*, 2018.

[41] S. Wang, C. Zhang, Y. Shu, and Y. Liu, "Live video analytics with fpga-based smart cameras," in *ACM HotEdgeVideo*, 2019.

[42] S. A. Jyothi, C. Curino, I. Menache, S. M. Narayanamurthy, A. Tumanov, J. Yaniv, R. Mavlyutov, Í. Goiri, S. Krishnan, J. Kulkarni *et al.*, "Morpheus: Towards Automated SLOs for Enterprise Clusters," in *USENIX OSDI*, 2016.

[43] A. D. Ferguson, P. Bodik, S. Kandula, E. Boutin, and R. Fonseca, "Jockey: Guaranteed Job Latency in Data Parallel Clusters," in *ACM EuroSys*, 2012.

[44] B. Ravindran, E. D. Jensen, and P. Li, "On Recent Advances in Time/Utility Function Real-Time Scheduling and Resource Management," in *IEEE ISORC*, 2005.

Authorized licensed use limited to: Yale University. Downloaded on January 25,2022 at 19:01:55 UTC from IEEE Xplore. Restrictions apply.