Real-Time Object Detection System with Multi-Path Neural Networks

Seonyeong Heo
POSTECH
Pohang, Republic of Korea
heosy@postech.ac.kr

Sungjun Cho
POSTECH
Pohang, Republic of Korea
allencho1222@postech.ac.kr

Youngsok Kim
Yonsei University
Seoul, Republic of Korea
youngsok@yonsei.ac.kr

Hanjun Kim
Yonsei University
Seoul, Republic of Korea
hanjun@yonsei.ac.kr

***Abstract*—Thanks to the recent advances in Deep Neural Networks (DNNs), DNN-based object detection systems become highly accurate and widely used in real-time environments such as autonomous vehicles, drones and security robots. Although the systems should detect objects within a certain time limit that can vary depending on their execution environments such as vehicle speeds, existing systems blindly execute the entire long-latency DNNs without reflecting the time-varying time limits, and thus they cannot guarantee real-time constraints. This work proposes a novel real-time object detection system that employs multi-path neural networks based on a new worst-case execution time (WCET) model for DNNs on a GPU. This work designs the WCET model for a single DNN layer analyzing processor and memory contention on GPUs, and extends the WCET model to the end-to-end networks. This work also designs the multi-path networks with three new operators such as skip, switch, and dynamic generate proposals that dynamically change their execution paths and the number of target objects. Finally, this work proposes a path decision model that chooses the optimal execution path at run-time reflecting dynamically changing environments and time constraints. Our detailed evaluation using widely-used driving datasets shows that the proposed real-time object detection system performs as good as a baseline object detection system without violating the time-varying time limits. Moreover, the WCET model predicts the worst-case execution latency of convolutional and group normalization layers with only 27% and 81% errors on average, respectively.**

I. Introduction

Identifying where an object exists and what the object is, object detection plays a crucial role in autonomous vehicles, drones and security robots. Thanks to the recent advances in Deep Neural Networks (DNNs), the object detection systems begin to heavily utilize DNNs [1]–[4] which provide superior accuracy over traditional algorithms [5], [6]. The DNN-based object detection systems employ a series of layers that extract the representative features of objects, increasing accuracy of the region identification and the object classification.

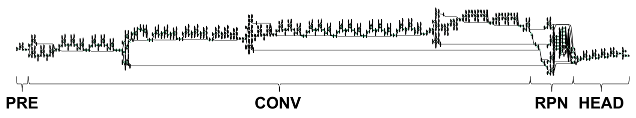
The DNN-based object detection systems should respect the real-time constraints that can vary depending on their execution environments. Since the systems are widely adopted in real-time environments where missing time constraints may cause a catastrophe such as collisions, the systems should detect objects within a certain time limit. Here, the time limit can vary depending on the execution environments. For example, when a car drives fast, the system should quickly detect objects in the front to avoid collisions. On the other hand, when the car drives slow, the system has a longer time to detect the objects because the time for the car to reach the objects increases. Therefore, the DNN-based object detection systems should be aware of the time-varying time limits.

One important aspect of the real-time object detection systems is that their execution latency and accuracy conflict with each other. First, deeper DNNs that have more layers generally outperform shallower ones; however, deeper DNNs tend to incur higher execution latency than the shallower ones. The DNN-based object detection systems [1]–[4] make predictions by extracting and evaluating features from a given image through a series of spatial operations such as convolutions and pooling (e.g., ResNet [7]). Additional operations on additional layers can increase the accuracy, but also increase execution latency. Second, each of the identified regions requires to execute DNNs. The object detection proposes a set of image regions that may contain an object. Considering a larger number of the region proposals improve the accuracy; however, increasing the number of the region proposals also leads to higher execution latency. Therefore, the real-time object detection systems must be aware of the trade-offs between the latency and accuracy along with the varying time constraints.

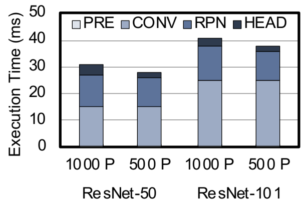
Unfortunately, the existing DNN-based object detection systems [1]–[4] cannot satisfy the varying time constraints. Although Faster R-CNN [4] and Mask R-CNN [3] obtain much lower execution latency compared with R-CNN [1] and Fast R-CNN [2], the networks are not aware of real-time constraints and frequently exceed the time limits. Dynamic model compression and DVFS [8]–[15] can dynamically adjust execution latency. However, since the model compression is limited to pruning specific filters of convolutional layers, and the DVFS suffers from huge overheads on changing voltage and frequency, they cannot efficiently satisfy the varying time constraints of object detection. Similarly, anytime algorithms [16]–[19] can meet the dynamic time constraints by changing their execution paths according to a given time limit, but their applicability is limited to a single algorithm, so they cannot be used in object detection.

This work proposes a novel real-time object detection system that employs *multi-path* neural networks based on a new worst-case execution time (WCET) model for DNNs on a GPU. The multi-path neural networks provide several execution paths having different latency-accuracy trade-offs,

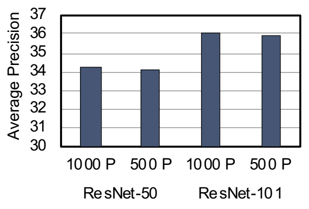
2642-7346/20/$31.00 ©2020 IEEE
DOI 10.1109/RTAS48715.2020.000-8



(a) Network structure



(b) Execution time



(c) Average precision

Fig. 1: Overall structure of Faster R-CNN and its execution time and average precision with different configurations

and the WCET model shows expected latencies for each path for the worst-case, so the system can dynamically select and execute an appropriate execution path satisfying the time-varying time constraints. To design and implement the system, this work first designs the WCET model for a single-layer of DNNs analyzing processor and memory contention on GPUs, and extends the WCET model to the end-to-end networks. Second, this work designs the multi-path networks with three new operators such as *skip*, *switch*, and *dynamic generate proposals* that dynamically change their execution paths. The skip operator allows the system to bypass a few consecutive DNN layers, the switch operator lets the system select one of the series of DNN layers, and the dynamic generate proposals adjusts the number of target object regions. The skip and switch operators are inserted into multiple locations of a target DNN, so the system can dynamically adjust the remaining execution paths at run-time. Finally, this work proposes a path decision model that chooses the optimal execution path at run-time reflecting dynamically changing environments and time constraints.

To demonstrate the real-time awareness of the proposed real-time object detection system, this work trains and evaluates the system with well-known driving datasets: Cityscapes [20] and BDD100K [21]. The evaluation results demonstrate that the system adaptively changes its execution paths in object detection respecting the time-varying time constraints without any deadline miss while achieving similar accuracy to a baseline single-path object detection system. Moreover, the evaluation shows that the WCET model predicts the worst-case execution latency of convolutional and group normalization layers with only 27% and 81% errors on average, respectively.

In summary, the contributions of this paper are:

- Worst-case execution latency modeling for DNN-based object detection on GPUs
- Multi-path neural networks which adaptively change their execution paths according to a given time constraint
- Path decision model that dynamically determines the execution path of the multi-path neural networks reflecting the varying time constraints

II. Background & Motivation

A. Deep Neural Networks for Object Detection

Object detection systems can greatly benefit from highly accurate DNNs. Object detection networks can be largely categorized as one-stage networks such as Single Shot Detector (SSD) [22] and YOLO [23]–[25] and two-stage networks such as R-CNN [1], Faster R-CNN [3] and Mask R-CNN [4]. The major difference between one-stage networks and two-stage networks is whether region proposal and classification steps are separated or not. One-stage networks integrate region proposal and classification steps to reduce the inference time, whereas two-stage networks separate region proposal and classification steps to improve object localization. Among the two kinds of networks, this work focuses on two-stage networks because the two-stage networks have more configurable features to solve dynamic time constraint problems.

First employed by R-CNN [1], two-stage object detection networks first identify region proposals (i.e., bounding boxes), which may contain a valid object, and classify the valid objects in the region proposals. We provide a brief background on Faster R-CNN [3], one of the most widely-used two-stage object detection networks due to its low amount of computation over its predecessors (e.g., R-CNN [1] and Fast R-CNN [2]), as an example.

Faster R-CNN consists of the following four major phases (Fig. 1(a)): PRE, CONV, RPN and HEAD.

1) **PRE** is a pre-processing phase which applies basic routines such as scaling and cropping to a given image, so that the size of the image matches what the rest of the DNN expects.
2) **CONV** contains a deep convolutional network (mainly, ResNet [7]) to extract high-level features of potential objects useful for RPN and HEAD
3) **RPN** contains a region proposal network which proposes region proposals which indicate specific regions in the given image to look into.
4) **HEAD** is a post-processing phase which finally classifies the object in each region from RPN by examining the high-level features with a convolutional network.

Faster R-CNN has various configurable parameters that this work can exploit to achieve target accuracy-latency trade-offs. Fig. 1(b) and Fig. 1(c) show the execution time breakdown and average precision of simple Faster R-CNN networks with different configurations. Here, average precision (AP) indicates a metric for evaluating the accuracy of object detection systems [26]. AP is defined as the average proportion of the image area where the inferred bounding boxes correctly overlap the ground truth bounding boxes.

The graphs clearly demonstrate that the accuracy-latency trade-offs in the object detection networks. First, the graph shows that the deeper CONV phase (ResNet-101) generates more accurate object detection results while consuming more time than the shallower CONV phase (ResNet-50). Similarly, the larger number of region proposals (1000P) enhances the accuracy of the object detection networks than the smaller number of region proposals (500P). In summary, the depth of the CONV phase and the number of region proposals to generate can affect both the execution time and the accuracy of object detection networks. This work uses this property to make object detection networks to adapt time-varying real-time requirements.

B. Deep Neural Networks on Graphics Processing Units

Deep neural networks demand high throughput to achieve low execution latency as they incur large amounts of computation. To satisfy the high performance demands, Graphics Processing Units (GPUs), which provide orders of magnitude higher throughput than CPUs [27], have become a popular platform. GPUs achieve high throughput by having thousands of simple arithmetic cores and high-bandwidth off-chip memory. A GPU consists of Streaming Multiprocessors (SMs), which have a fixed number of GPU cores, e.g., 64 cores in NVIDIA Turing GPUs. Each SM contains warp schedulers, a register file, shared memory, and L1 caches. The warp scheduler groups the threads into a set of *warps*, which is the scheduling granularity of each SM (e.g., 32 threads/warp in NVIDIA GPUs), and executes the threads within the same warp in a lock-step manner. When the threads issue memory access requests, the L1 caches interact with the global L2 data cache and DRAM memory to serve the requests.

To fully exploit the high throughput of GPUs, programmers need to write GPU-friendly functions called *kernels*. Then, programmers can initiate the execution of a GPU kernel by specifying the total number of threads and the thread block size, the programmer-specified number of threads that share a set of hardware resources. Since the threads within the same thread block use the shared memory, the GPU assigns the threads of a thread block to a specific SM as a whole. The computations specified by the kernel depend on the executing thread ID, so the parallel execution of the kernel by the threads follows the Single Instruction, Multiple Threads (SIMT) execution model [28].

Due to the rapid growth of deep learning, GPU vendors provide a set of kernels implementing representative DNN operations and optimized for their GPUs (e.g., cuDNN [29]). By invoking the optimized kernels, users can execute the layers of their neural networks on their target GPUs to achieve high throughput. Using the DNN libraries, deep learning frameworks typically execute one layer on a GPU at a time as each neural network layer incurs a large amount of computation.

C. WCET Analysis for DNN Executions on GPUs

Real-time systems demand Worst-Case Execution Time (WCET) analyses on their target applications and platforms to ensure real-time guarantees. The WCET analyses verify the system's real-time awareness by calculating the longest possible execution latency of the systems; if a WCET analysis on a target system predicts that the execution of the system always completes within a given time limit, we can validate that the system is fully real-time aware. When predicting the WCET of a system, a naïve, conservative WCET analysis would overestimate the system's execution latency; however, it is desirable to minimize the gap between the actual and the estimated execution latencies as minimizing the gap allows us to predict the WCET of the system more accurately.

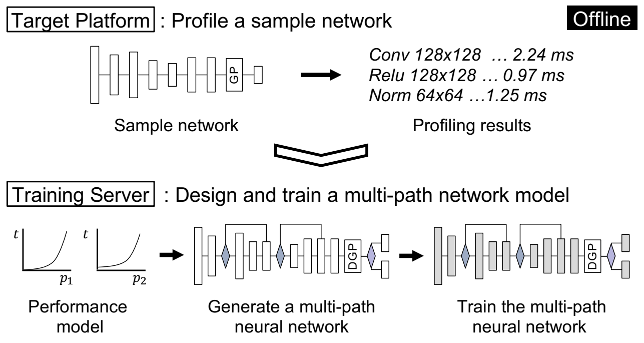


Fig. 2: Offline training processes

Some prior studies perform WCET analyses on GPUs so that GPU-based real-time systems can evaluate their real-time-awareness [30], [31]. Unfortunately, it is difficult to employ the WCET analyses for GPU-based object detection networks as the WCET analyses demand the source code of their target applications and most DNN libraries for GPUs are proprietary (e.g., NVIDIA's cuDNN).

As an alternative, prior studies estimate the execution latency of DNNs on GPUs by employing linear regression [32], [33]; however, their linear regression for DNNs is based on high-level tensor information such as input channel size without considering GPU architectures and memory contention. Since DNNs read and write a large amount of memory, there exist contentions on global and shared memory affecting the execution latency. Thus, the existing memory contention-oblivious linear regression fails to precisely estimate the WCET, and then elaborate architectural modeling that reflects memory contention is necessary.

In conclusion, real-time object detection systems demand a new WCET analysis which can avoid detailed source-level analyses and provide an architectural background.

III. System Design

This work designs a real-time object detection system which satisfies dynamic time constraints using multi-path neural networks. Fig. 2 and Fig. 3 show the overall design of our system. To allow the system to satisfy the dynamic time constraints, the multi-path neural networks introduce *skip* and *switch* operators which provide alternative execution paths to the system. The skip operators allow the system to bypass a few consecutive layers, and the switch operators let the system select the optimal execution path among multiple series of layers. At run-time, the multi-path neural networks select appropriate execution paths through the skip and switch operators and by dynamically adjusting the numbers of region

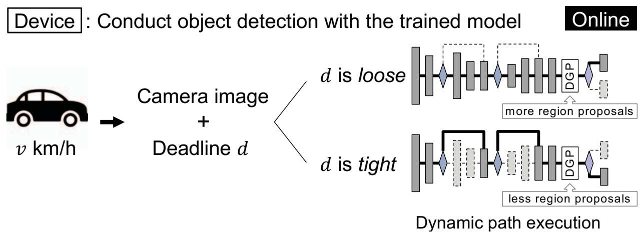


Fig. 3: Online execution of a multi-path network

proposals the multi-path neural networks produce. When selecting the appropriate execution paths, the system considers the remaining time limits and the dynamic system states such as the speed of a vehicle (Fig. 3). If the deadline is tight, then the multi-path neural network chooses a shorter execution path and generates a small number of region proposals to reduce the execution latency. On the other hand, if the deadline is loose, then the multi-path network achieves higher accuracy by selecting a longer execution path and by generating a large number of region proposals.

To design the multi-path neural networks, the system first profiles a sample neural network on its target platform (Fig. 2). Our system uses an existing object detection network [3] as the sample network. To collect necessary profiling results, the system profiles relevant performance metrics such as total execution cycles by running the sample neural network several times on the target GPU. Based on the profiling results, the system generates a multi-path neural network by inserting the skip and switch operators to the sample network. Then, the system (re-)trains the multi-path neural network to minimize the potential accuracy losses by the skip and switch operators. When the training process gets completed, the system sends the trained multi-path neural network to the target platform.

During run-time, the system satisfies the dynamic time constraints by (1) utilizing the skip and switch operators of the multi-path neural network, and (2) dynamically adjusting the number of region proposals the multi-path neural network should produce. Depending on the remaining time limits, the skip and switch operators select the appropriate execution paths which do not violate the time limits. When selecting the appropriate execution paths, the operators utilize a *WCET model* to predict the execution latency of an execution path. To construct the WCET model, the system employs an interval-driven WCET analysis for single-layer executions and extends the WCET analysis for the entire multi-path neural network. The system also adjusts the number of region proposals to ensure real-time guarantees; when the system is running out of time (i.e., the deadline is tight), the system reduces the number of region proposals to reduce the execution latency.

By employing the multi-path neural networks and the WCET model, our system ensures the real-time guarantees and maximizes the object detection accuracy. The skip and switch operators provide alternative execution paths to satisfy the dynamic time constraints. The WCET model lets the operators select the optimal execution path which does not violate the dynamic time constraints and achieve the highest accuracy among the possible execution paths.

IV. WCET Analysis for Neural Networks

In this section, we present our GPU performance model along with WCET analysis. Our real-time object detection system utilizes the WCET analysis to ensure real-time guarantees.

A. Basic Assumptions

To model the execution latency of DNNs on GPUs, we make the following key observations on our target GPU platforms.

First, we observe that **the number of active threads in a warp is constant for different DNN layers in an object detection DNN**. We measure warp execution efficiency using the NVIDIA profiling tool [34] when executing convolution and group normalization layers of a sample Fast R-CNN network. The warp execution efficiency indicates the ratio of active threads over the maximum possible active threads in a warp (i.e., warp size). Fig. 4(a) shows the cumulative distribution of the warp execution efficiency on NVIDIA GTX 1050 Ti and NVIDIA Titan Xp GPUs. For every different layer, the GPU kernels that implement the DNN layers always obtain 100% warp execution efficiency. In other words, each warp always has the maximum number of active threads supported by a processor.

Second, we find that **each SM has at least one active warp during execution if no memory contention exists**. Using the NVIDIA profiling tool, we measure the average number of eligible warps per cycle. The metric indicates the average number of warps that are ready to issue their next instructions per cycle. As Fig. 4(b) shows, about 97.7% of different layers have at least one eligible warp per cycle on average. Based on the profiling result, we can safely assume that each SM can always schedule at least one warp if no memory contention exists. Note that having no memory contention is necessary for the assumption as memory contention causes warp stalls and possibly reduces the number of eligible warps per cycle.

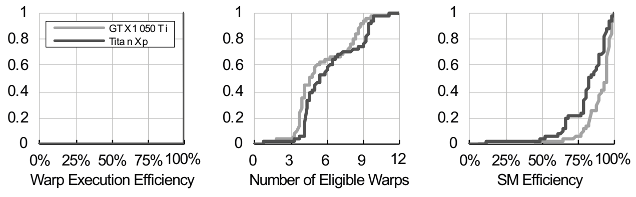
Third, we focus on the **Greedy-Then-Oldest (GTO) warp scheduling policy** which is the most common warp scheduling policy in the current GPUs. The GTO policy schedules the instructions of a warp until the warp stalls. Upon a stall, the policy selects the warp whose instructions have not been issued for the longest amount of time.

B. Single-layer Performance Modeling

As a DNN consists of a series of different layers, we first model single-layer performance as the first step toward DNN performance modeling. Our modeling utilizes an interval analysis where an interval is defined as the duration of a warp from the time when the warp is rescheduled to the time when the warp can issue new instructions after the warp is stalled. In Fig. 5(a), C_i^{exec} denotes the number of execution cycles until the warp stalls, and C_i^{stall} denotes the number of stall cycles until the warp is ready to issue an instruction. Then, C_i^{exec} and C_i^{stall} compose the interval i. Note that C_i^{exec} and C_i^{stall}

TABLE I: Notation for performance modeling

Notation	Description	Notation	Description
N_{block}	Number of thread blocks	N_i^{warp}	Number of eligible warps in interval i
N_{proc}	Number of streaming multiprocessors	D_i^{proc}	Additional delay due to processor contention in interval i
C_{total}	Total number of cycles of executing a single layer	D_i^{mem}	Additional delay due to memory bandwidth contention in interval i
C_i	Number of cycles of all warps in a thread block in interval i	D_i^{mshr}	Additional delay due to MSHR contention in interval i
C_i^{exec}	Number of execution cycles for a warp in interval i	D_i^{dram}	Additional delay due to memory contention in interval i
C_i^{stall}	Number of stall cycles for a warp in interval i	D_i^{bank}	Additional delay due to shared memory bank contention in interval i
L^M	Cache miss latency	R_i^M	Number of memory instructions in interval i
L^S	Shared memory access latency	R_i^S	Number of shared memory instructions in interval i



(a) Warp efficiency (b) # Eligible Warps (c) SM efficiency

Fig. 4: Cumulative distributions of three GPU performance metrics when executing layers in an object detection network

assume that the warp does not experience any contention with other warps or processors (i.e., SMs of NVIDIA GPUs).

We formulate the number of cycles in the i-th interval of all warps in a thread block using the notation in Table I;

$$C_i = C_i^{exec} + C_i^{stall} + D_i^{proc} + D_i^{mem}$$

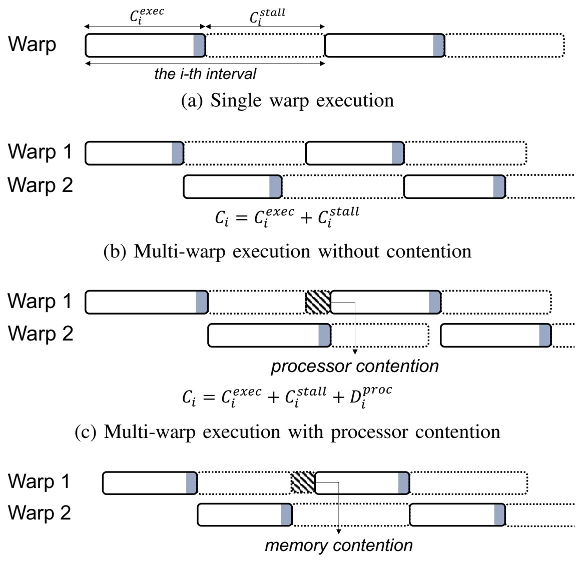
GPU schedulers assign each thread block to a specific processor to execute multiple warps in the thread block alternately. Then, the total cycles for executing a single layer with N_{block} thread blocks on N_{proc} processor become:

$$C_{total} = \frac{N_{block}}{N_{proc}} \sum_i C_i$$

Fig. 5 illustrates how the number of cycles changes from the case of single-warp execution to the case of multi-warp execution with additional delays.

1) Processor Contention Modeling: Processor contention can occur when a processor has multiple active warps to schedule. Based on the GTO scheduling policy, the processor greedily executes a warp until the warp stalls. Then, the stalled warp waits until all the other warps stall before issuing new instructions. In such a case, if the number of stall cycles of a warp is larger than the number of execution cycles of the other warps, then there would be no additional delay due to processor contention as shown in Fig. 5(b). On the other hand, if the number of stall cycles of a warp is smaller than the number of execution cycles of the other warps, the warp must bear an additional delay until the SM reschedules it as Fig. 5(c) shows.

Based on our second basic assumption, the processors can hide the stall cycles C_i^{stall} by continuously re-scheduling the ready warps. In other words, a warp should wait $(N_i^{warp} - 1)$


C_i^{exec}
C_i^{stall}
Warp
the i-th interval
Warp 1
Warp 2
$C_i = C_i^{exec} + C_i^{stall}$
processor contention
$C_i = C_i^{exec} + C_i^{stall} + D_i^{proc}$
memory contention
$C_i = C_i^{exec} + C_i^{stall} + D_i^{mem}$
(a) Single warp execution

(b) Multi-warp execution without contention

(c) Multi-warp execution with processor contention

(d) Multi-warp execution with memory contention

Fig. 5: Interval-driven single-layer execution time modeling

warps until it is rescheduled in the worst case. In summary, the worst-case additional delay due to processor contention becomes:

$$D_i^{proc} = (N_i^{warp} - 1) \cdot C_i^{exec} - C_i^{stall}$$

2) Memory Contention Modeling: Memory contention can occur when multiple processors access to DRAM or multiple threads access the same shared memory bank simultaneously. In the case, the additional delay is:

$$D_i^{mem} = \max\left\{D_i^{mshr} + D_i^{dram}, D_i^{bank}\right\}$$

(i) MSHR Contention: MSHR is a register that handles a cache miss with the information on the cache miss such as memory address. MSHR contention can cause additional delay because the number of MSHRs is limited. Based on the MSHR contention model of the prior work [35], we simplify the model as:

$$D_i^{mshr} = L^M \cdot \left(\left\lceil \frac{R_i^M}{\texttt{NUM\_MSHR}} \right\rceil - 1\right) < L^M \cdot \frac{R_i^M}{\texttt{NUM\_MSHR}}$$

considering the worst-case MSHR contention where `NUM_MSHR` is the number of MSHRs.

(ii) DRAM Bandwidth Contention: When multiple processors access the off-chip DRAM simultaneously, the DRAM bandwidth limits the processing of the memory requests. Similarly, we simplify the existing DRAM contention model [35] as:

$$D_i^{dram} = \texttt{CORE\_FREQ} \cdot \frac{\texttt{CACHE\_LINE}}{\texttt{MEM\_BAND}} \cdot \frac{R_i^M}{2}$$

considering the worst-case bandwidth contention (i.e., the DRAM utilization is very high) where `CORE_FREQ` is the core frequency, `CACHE_LINE` is the cache line size, and `MEM_BAND` is the DRAM bandwidth.

(iii) Shared Memory Bank Contention: Shared memory is an on-chip scratchpad memory space which the threads in the same thread block can share. To maximize the throughput, each processor has the shared memory with multiple shared memory banks so that all of the threads in a warp can concurrently access different banks. In such a case, the additional delay for accessing the shared memory banks of the shared memory (D_i^{bank}) is simply zero as no bank conflicts occur.

However, when the threads happen to access the same bank, a bank conflict can occur and the accesses get served in a serialized manner. In the worst case where all threads access to the same shared memory bank, a thread should wait for $\left(R_i^S - 1\right)$ before obtaining an access to the bank. Then, the worst-case additional delay due to shared memory bank contention is:

$$D_i^{bank} = L^S \cdot \left(R_i^S - 1\right)$$

Note that we subtract the shared memory access latency of a request when calculating its additional delay; the latency is already counted as the stall cycles (C_i^{stall}).

In summary, the number of total cycles for executing a single layer becomes:

$$\begin{aligned} C_{total} &= \frac{N_{block}}{N_{proc}} \sum_i C_i \\ &= \frac{N_{block}}{N_{proc}} \sum_i \left\{C_i^{exec} + C_i^{stall} + D_i^{proc} + D_i^{mem}\right\} \\ &= \frac{N_{block}}{N_{proc}} \sum_i \{N_i^{warp} \cdot C_i^{exec}\} + \frac{N_{block}}{N_{proc}} \sum_i D_i^{mem} \end{aligned}$$

Here, our performance model can accurately predict the execution latency of each layer while a GPU runs other applications. Since GPUs support spatial multitasking by distributing SMs to different applications, accurately predicting the execution latency on such multi-tenant scenarios requires accurate modeling of the per-SM behaviors. Our method achieves high per-SM modeling accuracy by using the concept of intervals, and models the DRAM bandwidth consumption shared between DNNs. Therefore, using the two highly-accurate models, our performance model can easily be extended to cover multi-tenant scenarios as well.

C. WCET Analysis

Based on the performance modeling, this work designs the WCET analysis methodology for deep neural networks on GPUs. To obtain the worst-case execution time from the performance model, we should determine the unknown parameters in the performance model. Since the full specifications of GPU architectures are not open to the public, we find the parameters through linear regression based on the profiling results of layers with different configurations. Table II lists the events and metrics of the NVIDIA profiling tool [34] that we use in the evaluation.

TABLE II: Profiling events and metrics for WCET analysis

Term	**Event**	**Description**
C_{total}	elapsed_cycles_pm	Elapsed clocks for a kernel
$\frac{N_{block}}{N_{proc}} \sum_i N_i^{warp} \cdot C_i^{exec}$	active_cycles_pm	Number of cycles where a multiprocessor has at least one active warp
Term	**Metric**	**Description**
$N_{block} \sum_i (R_i^M + R_i^S)$	ldst_executed	Number of executed local, global, shared, and texture memory load and store insts
$N_{block} \sum_i R_i^M$	global_load_requests global_store_requests	Number of global load requests Number of global store requests

Assumption 1: The total execution cycles have a linear relation with the number of computation, load, and store instructions:

$$N_{block} \sum_i \{N_i^{warp} \cdot C_i^{exec}\} = \alpha N_{comp} + \beta N_{load} + \gamma N_{store} + \delta$$

where N_{comp}, N_{load}, and N_{store} are the number of computation, load, and store instructions, respectively.

Assumption 2: The total memory requests have a linear relation with the number of load and store instructions:

$$N_{block} \sum_i R_i^M = \alpha N_{load}^M + \beta N_{store}^M + \gamma$$

where N_{load}^M and N_{store}^M are the number of global load and store instructions, respectively.

Assumption 3: The total shared memory requests have a linear relation with the number of shared load and store instructions:

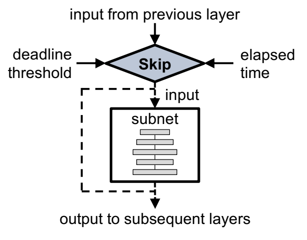
$$N_{block} \sum_i R_i^S = \alpha N_{load}^S + \beta N_{store}^S + \gamma$$

where N_{load}^S and N_{store}^S are the number of shared load and store instructions, respectively.

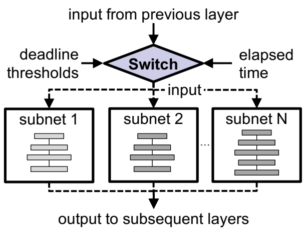
For the other unknown parameters (i.e., L^M and L^S), we estimate the worst-case value of each parameter referring to the architectural analysis in previous work [36], [37] and the profiling results of different layers from the NVIDIA profiling tool [34] such as shared memory throughput and DRAM throughput.

D. Extending to End-to-End Network

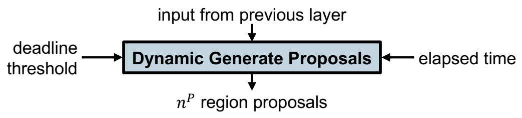
End-to-end networks include pre-processing or post-processing computations besides layer executions, but the pre-processing and post-processing time makes a small percentage of the end-to-end network latency. Therefore, we use a simple



(a) Skip operator

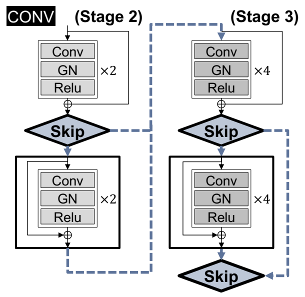


(b) Switch operator

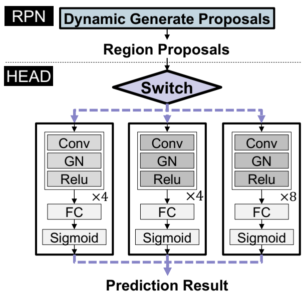


(c) Dynamic generate proposals operator

Fig. 6: Conceptual diagrams for the new operators of multi-path neural networks



(a) CONV phase



(b) RPN and HEAD phases

Fig. 7: Applying the three operators in Fig. 6 to an object detection network

regression model to stochastically estimate the worst-case pre-processing and post-processing time between layers. Finally, we can estimate the end-to-end network latency by summing the worst-case execution time of the layers in a path using our WCET analysis method in and the pre-processing and post-processing time between the layers.

V. Multi-path Object Detection Network

A. Network Design

To make neural networks aware of dynamic time constraints themselves, this work introduces three new operators: *skip*, *switch*, and *dynamic generate proposals*. Fig. 6 illustrates the three operators with their inputs and outputs. First, a skip operator (Fig. 6(a)) contains a single subnet (a set of layers) and decides whether to run the subnet or not according to the relative deadline and skip threshold. Second, a switch operator can contain multiple subnets and (Fig. 6(b)) selects one of the subnets to run according to the relative deadline and switch thresholds for the subnets. Third, a dynamic generate proposals operator (Fig. 6(c)) decides the number of region proposals to pass to the HEAD phase according to the relative deadline and proposal threshold. Section V-B will describe how the path decision model determines the threshold values of each operator.

Using the three operators, our system constructs a multi-path object detection network on top of existing object detection networks [3], [4]. The existing object detection networks provide various design spaces for configuring the networks. First, the networks can have a different number of layers in the CONV phase. Second, the networks can generate a different number of region proposals. Third, the networks can have a different number of convolutional blocks with a different number of hidden channels in the HEAD phase which affects the classification accuracy.

Exploring the three network design spaces, this work designs a multi-path object detection network that can change its path dynamically. First of all, the system inserts skip operators into the CONV phase, which is normally very deep as shown in Fig. 1(a). Therefore, skip operators are more suitable than switch operators for the CONV phase in terms of memory usage because switch operators require to maintain individual weight matrices for each subnet.

For the CONV phase, this work regards ResNet [7], one of the most common backbone networks for object detection networks. ResNet consists of multiple residual stages, and each residual stage contains identical residual blocks. Since ResNet keeps the same input dimension in the same stage, ResNet facilitates applying skip operators without extra overhead to the original network. For other convolutional neural networks such as VGG [38], adding skip operators may require extra layers to match output and input dimensions. Therefore, those networks are less suitable for multi-path object detection networks. Fig. 7(a) shows how the system adds multiple skip operators in the CONV phase.

Next, this work extends the RPN phase to dynamically change the number of region proposals on demand by replacing the original generate proposal operator with the dynamic generate proposal operator. The dynamic generate proposals operator allows to dynamically change the number of region proposals, which determines the batch size of inputs in the HEAD phase. The path decision model gives a threshold value for the dynamic generate proposals operator to find the appropriate number of region proposals.

Finally, the system generates multiple convolutional subnets for the HEAD phase. The HEAD phase generally contains a much shallower convolutional network than the CONV phase. Thus, maintaining multiple subnets is acceptable for the HEAD phase. Moreover, the HEAD phase includes fully-connected layers that determine the final prediction result of the object detection network. Then, sharing the fully-connected layers among different subnets can contaminate the output prediction result from different execution paths. Therefore, the system creates multiple subnets and configures each subnet with a different number of convolutional blocks and hidden layer dimensions. Fig. 7(b) shows how the system adds a switch operator in the HEAD phase.

The system configures multi-path networks based on the

TABLE III: Network configuration parameters

Parameter	Description
n^{RS}	The number of residual stages in the CONV phase
n_i^{RB}	The number of residual blocks in the residual stage i
n_{max}^P	The maximum number of region proposals
n_{min}^P	The minimum number of region proposals
n^{SN}	The number of subnets in the HEAD phase
n_j^{CB}	The number of convolutional blocks in the j-th subnet
n_j^{HC}	The number of hidden channels in the j-th subnet

configurable parameters in the existing object detection system [39] to satisfy the following two constraints:

1) **Deadline constraint:** The maximum execution time of a multi-path network (t_{max}^T) should be less than or equal to the maximum relative deadline (D_{max}), i.e.,

$$t_{max}^T \leq D_{max}$$

2) **Memory constraint:** The total memory necessary for a multi-path network should be less than or equal to the total memory of the target platform (M_{max}), i.e.,

$$\sum_{i=1}^{n} M_i \leq M_{max}$$

where n is the number of different layers in the network and M_i is the amount of memory that each layer requires.

To obtain a multi-path network model that satisfies both the deadline constraint and the memory constraint, the system starts from a minimal object detection network and gradually extends the network. To guarantee the minimum desirable accuracy, the system preserves several necessary network layers such as the first few layers in ResNet for the multi-path object detection networks. Then, the system gradually inserts residual or convolutional blocks to the networks until the parameters violate one of the constraints referring to the existing object detection networks.

The system automatically constructs the overall network structure following the design policies in a way to reduce accuracy loss. First, the system adds skip operators with convolutional blocks in the CONV phase imitating the original ResNet structures (i.e., ResNet-50 and ResNet-101). Then, the system inserts a switch operator with extra subnets in the HEAD phase after the CONV phase becomes as deep as ResNet-101, because too deep residual networks may not obtain a good performance if a training dataset is not large enough. If free memory space is still available, the system additionally adds smaller subnets to the switch operator.

B. Path Decision Model

With the network structure, the path decision model determines how each operator should act according to the remaining time until the deadline. Each operator checks the elapsed time (t_e) from when the inference task starts, and makes an appropriate decision to meet the deadline (D) eventually. Note that each operator tracks the real-time value of t_e, then each operator observes different values of t_e along with the execution of the object detection network.

Before introducing a path decision model, we first formulate the total execution time of a multi-path network as

$$t^T = t_{PRE}^T + t_{CONV}^T + t_{RPN}^T + t_{HEAD}^T$$

where t_{PRE}^T, t_{CONV}^T, t_{RPN}^T and t_{HEAD}^T indicate the execution time of each phase in the object detection network (Fig. 1(a)). Since t_{PRE}^T and t_{RPN}^T change little with different network configurations as shown in Fig. 1(b), we assume t_{PRE}^T and t_{RPN}^T are constant. Then, we can formulate t_{CONV}^T and t_{HEAD}^T as:

$$t_{CONV}^T \leq \sum_{i=1}^{n^{RS}} \left\{ n_i^{RB} \cdot t_i^{RB} \right\}$$
$$t_{HEAD}^T \leq \max \left\{ t_j^{SN}(n_{max}^P) \right\}_{j \in [1, n^{SN}]}$$

using the network configuration parameters in Table III, where t_i^{RB} denotes the execution time of a residual block in the residual stage i and $t_j^{SN}(n)$ denotes the execution time of the subnet j in the HEAD phase when the number of region proposals is n.

First, the path decision model determines when each skip operator in the CONV phase should skip residual blocks. If the minimum execution time of the rest of the network exceeds the remaining time until the deadline, the skip operator decides to skip its residual blocks. Then, the threshold τ of the skip operator in the residual stage k becomes:

$$\tau = n_k^{RS} \cdot t_k^{RS} + t_{RPN}^T + \min \left\{ t_j^{SN}\left(n_{min}^P\right) \right\}_{j \in [1, n^{SN}]} \leq D - t_e$$

Second, the path decision model finds an appropriate number of region proposals (n^P) for the dynamic region proposals. Since the number of region proposals is one of the important factors that affect the detection accuracy, the network should try to maximize the number of region proposals assuming that in the HEAD phase the network would run the smallest subnet. Then, the threshold τ of the dynamic generate region proposals becomes:

$$\tau = \min \left\{ t_j^{SN}\left(n_{max}^P\right) \right\}_{j \in [1, n^{SN}]}$$

With τ, the dynamic generate proposals operator determines the appropriate number of proposals as:

$$n^P = \min \left\{ \max \left\{ \left\lfloor n_{max}^P \cdot \frac{D - t_e}{\tau} \right\rfloor, n_{min}^P \right\}, n_{max}^P \right\}$$

Finally, the path decision model chooses a single subnet to run among the subnets of the switch operator in the HEAD phase. The decision model chooses the subnet with the largest execution time t_j^{SN} such that t_j^{SN} is less than or equal to the remaining time until the deadline (i.e. $t_j^{SN} \leq D - t_e$).

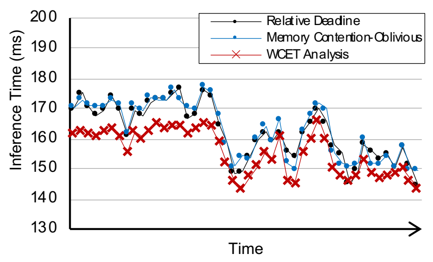
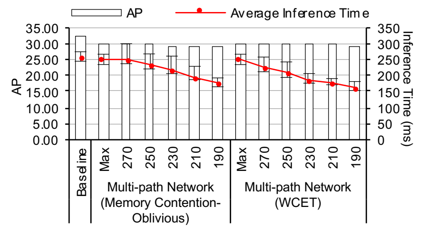
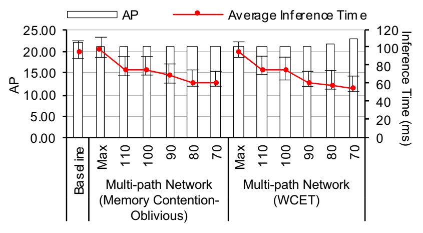


Fig. 8: Inference time for each target relative deadline



(a) Cityscapes



(b) BDD100k

Fig. 9: Average precision of the baseline network and the multi-path network for each target relative deadline

TABLE IV: Maximum AP of training strategies

Strategy	NS-NS	RS-NS	NS-RS	RS-RS
Maximum AP	19.87	23.55	29.94	29.84

C. Training Multi-path Neural Networks

This work tries to find an efficient strategy to train multi-path neural networks empirically. Unlike static neural networks where gradients always propagate to all paths, multi-path neural networks have multiple possible paths to which gradients can propagate. Depending on the network structures (i.e. the number of skip and switch operators), the number of possible paths can be huge. Therefore, finding an efficient training strategy is one of the challenges in using multi-path neural networks.

We design different training strategies for the skip and switch operators and train the same multi-path neural network with the following four training strategies to find the best:

- **No Skipping - No Switching (NS-NS)**: runs the subnet of each skip operator without skipping it and aggregates the results from all subnets for each switch operator
- **Random Skipping - No Switching (RS-NS)**: randomly runs the subnet of each skip operator and aggregates the results from all subnets of each switch operator
- **No Skipping - Random Switching (NS-RS)**: runs the subnet of each skip operator without skipping it and randomly runs a single subnet for each switch operator
- **Random Skipping - Random Switching (RS-RS)**: randomly runs the subnet of each skip operator and randomly runs a single subnet for each switch operator

Note that when selecting an execution path randomly, multi-path neural networks change their paths randomly per iteration (a pair of forward and backward passes). In other words, multi-path neural networks stick to a specific path within an iteration to prevent gradients from propagating through a wrong path.

Table IV shows the maximum AP when applying each training strategy. We train an empty multi-path network with the Cityscapes dataset [20] and measure the maximum AP by executing the longest path of the multi-path network. Among the four training strategies, NS-RS and RS-RS obtain the better accuracy than NS-NS and RS-NS. The result implies whether to switch randomly or not gives a significant impact on the accuracy. We suppose this accuracy loss might come from the vanishing gradient problem by the aggregation operator. On the other hand, whether to skip randomly or not incurs meaningful accuracy change only with the no switching strategy.

VI. Evaluation

To evaluate our real-time object detection system, this work implements a prototype system on top of Caffe2 deep learning framework and Detectron object detection system [39]. We extend Caffe2 to support the three operators of the multi-path neural networks shown in Fig. 6, and Detectron to create the multi-path neural networks and the timer operators to check the elapsed time before executing each operator. We use a Faster R-CNN network with ResNet-101 and FPN employing group normalization layers as a baseline network. On top of the baseline network, we build a multi-path network by inserting five skip operators and one switch operator to the baseline network. Among 25 potential positions for skip operators, we choose five positions considering meaningful inter-skip latency to adapt to time-varying deadlines and skipping overheads.

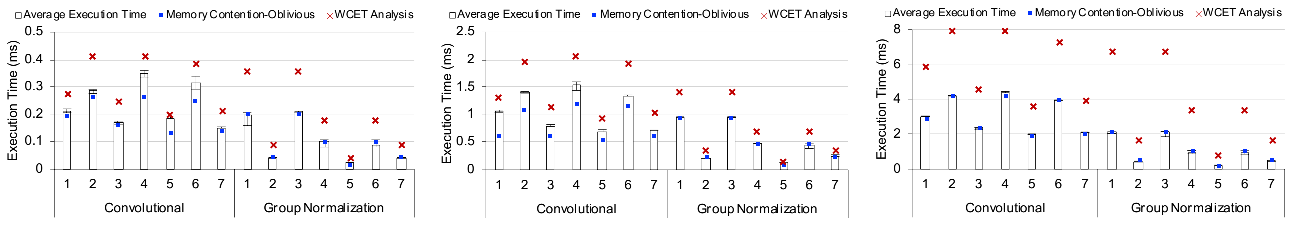
To train and test the networks, we use two widely used driving datasets: Cityscapes [20] and BDD100K [21]. For the Cityscapes dataset, we use the `fine` training set and employ the compact set of categories as done in prior work [4]. Table V summarizes the characteristics of the datasets and the training parameters we used. Note that we force the same training iterations in Table V for both the baseline network and the multi-path network. We utilize an NVIDIA Titan Xp GPU as the target platform of our object detection system.

TABLE V: Datasets and training parameters

Information	Cityscapes (fine) [20]	BDD100K [21]
Number of train images	2,975	70,000
Number of validation images	500	10,000
Learning rate	0.0025	0.005
Iteration	196,000	180,000
Number of GPUs	1	2
Batch size	1	4
Image scale	[700, 950]	[540, 720]

TABLE VI: Configuration of convolutional and group normalization layers

#	Convolutional Layer							Group Normalization Layer			
	$\text{Channel}_{\text{in}}$	$\text{Height}_{\text{in}}$	Width_{in}	$\text{Channel}_{\text{out}}$	Kernel	Stride	Padding	$\text{Channel}_{\text{in}}$	$\text{Height}_{\text{in}}$	Width_{in}	Group Size
1	64	136	240	256	1	1	0	64	272	480	32
2	256	136	240	128	1	1	0	64	136	240	32
3	128	68	120	512	1	1	0	256	136	240	32
4	256	136	240	512	1	2	0	128	136	240	32
5	512	68	120	128	1	1	0	128	68	120	32
6	512	68	120	256	1	1	0	512	68	120	32
7	256	34	60	1024	1	1	0	256	68	120	32



(a) Titan Xp (b) GTX 1050 Ti (c) Jetson TX2

Fig. 10: Single-layer execution time

A. *Real-time Object Detection Using Multi-path Networks*

In this experiment, we evaluate whether our real-time object detection system can satisfy dynamic time constraints. Specifically, the experiment examines whether the multi-path neural network can conservatively adapt to the dynamic time constraints using the WCET model. In addition, the experiment examines an execution of the multi-path neural network without memory contention modeling; the memory contention-oblivious latency estimation model only employs the linear relation of Assumption 1 in our WCET analysis when selecting its execution path. The experiment assumes a driving scenario where the speed of the vehicle dynamically changes over time. The allowed inference time (i.e., the time limit for real-time object detection systems) changes with respect to the vehicle's velocity.

The experimental result clearly indicates that our system using the WCET model fully satisfies the dynamic time constraints. On the other hand, the memory contention-oblivious model-based execution fails to ensure real-time guarantees violating the dynamic time constraints. Fig. 8 shows the relative deadlines over time, and the multi-path neural network execution latency either without or with memory contention modeling in our WCET model. Our system is fully real-time aware as the execution of the multi-path neural network using our WCET model always finishes before the deadlines. However, without memory contention modeling which helps to find an appropriate safety margin, the multi-path neural network often fails to meet the deadlines.

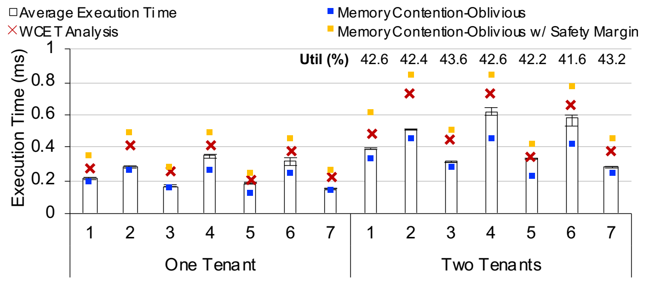
We also evaluate the changes in the inference accuracy of the multi-path neural network as the time limit changes. The evaluation results shown in Fig. 9 indicate that our multi-path neural network incur little accuracy losses even when the time limit becomes tight; our system incurs less than 3.1 and 1.1 points of accuracy losses for the validation sets of Cityscapes and BDD100K, respectively. For the multi-path network, we employ both our WCET models with and without memory contention modeling. The memory contention-oblivious latency estimation model incurs similar accuracy losses; however, our entire WCET model not only incurs little accuracy losses, but also ensures real-time guarantees.

In summary, our multi-path neural networks with the WCET model enables real-time object detection with little accuracy losses. The memory contention-oblivious model, however, fails to fully satisfy the dynamic time constraints because it estimates the inference latency too tightly.

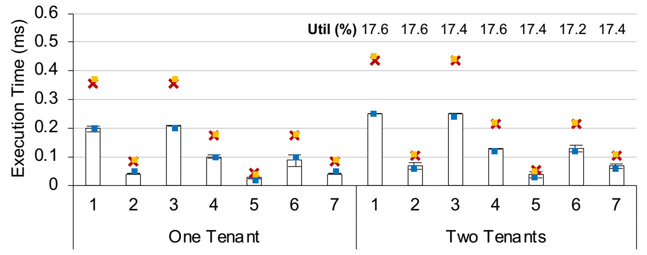
B. *Accurate Single-layer WCET Analysis*

We now evaluate the accuracy of our single-layer WCET analysis. Achieving accurate single-layer WCET predictions is essential as our system relies on the single-layer WCET analysis to ensure real-time guarantees. For this experiment, we first construct two single-layer neural networks, each with convolutional and group normalization layers on top of Caffe2. Then, we obtain the single-layer WCET models for the two types of layers by profiling the two neural networks with various configurations (e.g., input size). After that, the single-layer WCET models are validated against real hardware platforms and compared with the WCET models without memory contention modeling. To evaluate the platform portability of the WCET models, we employ an NVIDIA GTX 1050 Ti GPU and Jetson TX2 as additional target platforms.

The experimental results show that our single-layer WCET models accurately predict the single-layer execution latency; the maximum absolute errors for convolutional layers are only 0.06 ms, 0.38 ms, and 2.68 ms for Titan Xp, GTX 1050 Ti, and Jetson TX2, respectively. Fig. 10 shows the actual and estimated execution time of the two types of layers on Titan Xp, GTX 1050 Ti, and Jetson TX2, respectively.



(a) Convolutional



(b) Group Normalization

Fig. 11: Single-layer execution time with one and two tenants

The error bars indicate the minimum and the maximum execution times from five profiling runs. On average, the WCET model for convolutional layers obtains 27%, 35%, and 85% relative errors on Titan Xp, GTX 1050 Ti, and Jetson TX2 over the maximum execution time, respectively. For the group normalization layers, the worst-case execution time model obtains 81%, 44%, and 223% relative errors on Titan Xp, GTX 1050 Ti, and Jetson TX2, respectively. The memory contention-oblivious model tends to incur smaller prediction errors; however, it does not consider WCET and cannot guarantee the WCET in a number of cases. Note that the errors from our WCET analysis will not get accumulated through the execution of the multi-path neural network; each operator dynamically tracks the actual elapsed time instead of the WCET.

We also evaluate our WCET model in multi-tenant scenarios by running another GPU kernel in parallel while executing a layer. Fig. 11 shows the actual and estimated execution time on Titan Xp in one- and two-tenant scenarios. In Fig. 11, GPU utilization indicates the average percentage of multiprocessors used by the other tenant. We set a safety margin for the memory contention-oblivious model as the maximum relative error between the actual and estimated execution time in the experiment (45%), not to violate time constraints at least for the evaluated results. On average, the WCET model obtains 51% and 3% less errors compared with the memory contention-oblivious model with a safety margin for convolutional and group normalization layers, respectively. The result shows that our WCET model estimates the worst-case execution time with smaller errors in multi-tenant scenarios, whereas the memory contention-oblivious model might not be able to estimate the execution time properly even with the safety margin.

VII. Related Work

A. *Real-Time DNN Frameworks*

Some prior work [8]–[10] aims to satisfy the varying time constraints on DNNs using approximation and DVFS mechanisms. For example, ApNet [8] treats the layers of a DNN, instead of the entire DNN, as schedulable tasks and applies per-layer approximation to meet the target deadline. When selecting the approximation strategy to use for each layer, ApNet examines how the resource utilization of the target device (e.g., GPUs) changes with respect to the approximation mechanisms. As another example, PredJoule [9], a timing-predictable energy optimization framework for DNNs, exploits the fact that the energy usage patterns differ among different layers. By considering the correlation between power and performance characteristics of the layers into an account, PredJoule selects the optimal DVFS strategy for executing DNNs without violating the target time constraint. DeepRT [10] also exploits DVFS to satisfy the target time constraints, but focuses on executing DNNs on mobile devices. As mobile devices have limited amounts of resources, DeepRT utilizes both DVFS and compression; it dynamically adjusts the CPU and GPU clock frequencies, and employs dynamic model compression using singular value decomposition.

Our work shares the common goal of satisfying the varying time constraints of DNN executions with the prior work; however, our multi-path networks achieve higher applicability by requiring neither accuracy-affecting techniques (e.g., per-layer approximation, compression) nor low-level system-wide controls (e.g., DVFS). Instead, the multi-path networks simply extend the existing DNNs with multiple execution paths having different execution latencies, and utilize the skip/switch operators to dynamically change a DNN's execution path during run-time with respect to the remaining time limit.

B. *Reducing the DNN Execution Latency*

To reduce the execution latency of DNNs on resource-constrained devices, some existing work proposes dynamic resource-aware DNN systems which reduce the computational overheads of DNNs. First, some frameworks reduce the computational overheads of DNNs by eliminating near-zero weights in convolutional and fully-connected layers [40]–[42]. By removing the near-zero weights, the number of multiply-accumulate operations reduces and the frameworks can achieve lower execution latency. Second, other frameworks propose to employ filter pruning which reduces the computational overheads by eliminating some filters of convolutional layers [11], [12]. For example, NestDNN [11] allows DNNs to expand and shrink the size of the filters in convolutional layers according to the dynamic availability of hardware resources. Third, the other frameworks fine-tune a given DNN with respect to the available hardware resources [13]–[15]. As an example, NetAdapt [13] fine-tunes a DNN to a target platform by applying different levels of filter pruning to the DNN's convolutional and fully-connected layers, and by estimating the resource consumption of the reconfigured DNNs. Among the fine-tuned

DNNs which meet the resource budget, NetAdapt selects the most-accurate one as the optimal DNN for the target platform.

Reducing the execution latency of DNNs can help the frameworks meet deadlines on real-time environments; however, the frameworks cannot adapt to varying time constraints, and focus on reducing the computational overheads of a DNN by eliminating the DNN's operations (e.g., filter pruning). On the other hand, our multi-path networks not only adapt to the varying time constraints using accurate performance modeling, but also offer higher flexibility by allowing different paths to consist of different series of layers. For example, our multi-path networks can provide subnets where different levels of filter pruning have been applied, or subnets which consist of completely different series of layers as alternative execution paths. Furthermore, layers can be skipped on the multi-path networks whereas the existing frameworks often do not.

C. GPU Performance Modeling

Modeling the performance of GPUs has been an active research topic as accurate performance prediction brings several benefits (e.g., performance bottleneck identification and faster simulations). Hong and Kim [43] introduce the concept of Memory Warp Parallelism (MWP) to estimate the number of parallel memory requests to predict the execution latency. Using the estimated MWP of a GPU kernel, their model estimates the costs of memory accesses to derive the total execution latency of the kernel. Baghsorkhi et al. [44] propose to abstract a GPU kernel as a Work Flow Graph (WFG) to estimate the kernel's execution latency. The nodes of the WFG represent the operations (e.g., computation, memory, synchronization), and the weights of the WFG's edges are the average numbers of cycles required to execute their source nodes. Zhang and Owens [45] employ a microbenchmark-driven approach to construct a GPU performance model. Instead of building an analytical model first and then verifying the model using microbenchmarks, they build a throughput model by profiling the microbenchmarks on the target GPU. Huang et al. [35] present an interval analysis-based performance model for GPUs considering how hardware components operate. The model chooses a representative warp by clustering all profiling results of warps to estimate the performance of a target program. Using a functional simulator, the model consumes shorter analysis time than other models based on a cycle-accurate simulator. More recently, Wu et al. [46] utilize machine learning to predict a GPU kernel's execution latency. They first train a neural network which predicts the performance scaling behavior of a GPU kernel. Then, to estimate the latency and power of a kernel on the target GPU, they predict the kernel's performance scaling behavior using the neural network and scale the kernel's performance on a base hardware configuration.

Unlike the prior work on GPU performance modeling, this work focuses on the WCET analysis of deep neural networks to guarantee deadline. Our performance modeling is based on the work by Huang et al. [35] by employing their interval analysis technique; however, by specializing in modeling the performance of DNNs and extending the model with the worst-case resource contention modeling, our method simplifies and extends the prior work for accurate performance modeling of DNN executions on GPUs.

D. Worst-Case Execution Time Analysis for GPUs

Worst-Case Execution Time (WCET) analyses enable real-time scheduling of GPU programs by helping them meet deadlines even in the worst-case execution scenarios. In terms of the WCET analyses for GPUs, Hirvisalo [47] conducts static timing analysis of GPU kernels with abstract Cooperative Thread Array (CTA) simulation. Hirvisalo calculates the WCET by cumulating instruction execution and memory stall times of the abstract warp with respect to thread divergence, memory latencies and scheduling choices. Huangfu and Zhang [31] propose a static WCET analysis method based on a predictable scheduling policy (i.e., *Greedy Then Round-Robin* scheduling). With the timing model for the scheduling policy, they present a static analyzer that analyzes the assembly codes of GPU programs and provides the WCET estimations for the GPU programs. Betts and Donaldson [30] introduce a hybrid WCET analysis which combines dynamic profiling and static analysis, extending an existing hybrid WCET analysis for sequential programs. First, they slice program traces into a set of Warp Specific Traces (WSTs) generated by a specific warp on a particular multiprocessor. To provide warp-specific WCET estimate with the traces, they propose two ways to further analyze them: a pure dynamic technique and a hybrid technique. The pure dynamic technique dynamically infers the worst-case release jitter of the final warp. The hybrid technique builds a model of how warps arrive and computes the worst-case release jitter of the final warp. Then, based on the worst-case release jitter, they estimate the WCET of a GPU program.

The existing WCET analyses require the source code of their target applications; however, as GPU vendors do not fully disclose the details of their GPUs and optimized DNN libraries (e.g., cuDNN), it is difficult for real-time DNN-based object detection systems to employ the existing WCET analyses. Open-source frameworks such as Caffe and TensorFlow also utilize the proprietary libraries as well. Therefore, we utilized the nvprof profiling tool to obtain the information (e.g., cycle counts, memory instruction counts) necessary for estimating the disclosed/unknown parameters instead.

VIII. Conclusion

Real-time object detection systems need to satisfy environment-varying time constraints; however, the existing object detection systems which employ DNNs to achieve high accuracy fail to satisfy the real-time requirements. This work proposes a novel real-time object detection system that employs multi-path neural networks and a WCET performance model for the neural networks. The networks can dynamically select their execution paths to meet the varying time constraints. Our detailed experiments using widely used driving datasets clearly show that the system successfully satisfies any time limit while maintaining high accuracy.

Acknowledgements

We thank the anonymous reviewers and shepherd for their valuable feedback. This research was supported by Samsung Research Funding Center of Samsung Electronics under Project Number SRFC-TB1703-03. This research was also partly supported by the Yonsei university research fund of 2019. Hanjun Kim is the corresponding author of this paper.

References

[1] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

[2] R. Girshick, "Fast r-cnn," *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015.

[3] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, 2017.

[4] K. He, G. Gkioxari, P. Dollar, and R. Girshick, "Mask r-cnn," *2017 IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[5] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2001.

[6] P. Viola and M. J. Jones, "Robust real-time face detection," *International Journal of Computer Vision*, vol. 57, no. 2, pp. 137–154, May 2004.

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[8] S. Bateni and C. Liu, "Apnet: Approximation-aware real-time neural network," in *2018 IEEE Real-Time Systems Symposium (RTSS)*, 2018.

[9] S. Bateni, H. Zhou, Y. Zhu, and C. Liu, "Predjoule: A timing-predictable energy optimization framework for deep neural networks," in *2018 IEEE Real-Time Systems Symposium (RTSS)*, Dec 2018.

[10] W. Kang and J. Chung, "Deeprt: Predictable deep learning inference for cyber-physical systems," *Real-Time Systems*, vol. 55, no. 1, p. 106135, Jan. 2019.

[11] 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*, ser. MobiCom 18. ACM, 2018.

[12] W. Kang, D. Kim, and J. Park, "Dms: Dynamic model scaling for quality-aware deep learning inference in mobile and embedded devices," *IEEE Access*, vol. 7, 2019.

[13] T.-J. Yang, A. Howard, B. Chen, X. Zhang, A. Go, M. Sandler, V. Sze, and H. Adam, "Netadapt: Platform-aware neural network adaptation for mobile applications," in *Computer Vision – ECCV 2018*, V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, Eds. Springer International Publishing, 2018.

[14] Z. Xu, Z. Qin, F. Yu, C. Liu, and X. Chen, "Direct: Resource-aware dynamic model reconfiguration for convolutional neural network in mobile systems," in *Proceedings of the International Symposium on Low Power Electronics and Design*, ser. ISLPED 18. Association for Computing Machinery, 2018.

[15] Z. Xu, F. Yut, C. Liu, and X. Chen, "Reform: Static and dynamic resource-aware dnn reconfiguration framework for mobile device," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*, 2019.

[16] R. Mangharam and A. A. Saba, "Anytime algorithms for gpu architectures," in *2011 IEEE 32nd Real-Time Systems Symposium (RTSS)*, Nov 2011.

[17] A. Grubb and D. Bagnell, "Speedboost: Anytime prediction with uniform near-optimality," in *Proceedings of the 15th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2012.

[18] Y. V. Pant, H. Abbas, K. Mohta, T. X. Nghiem, J. Devietti, and R. Mangharam, "Co-design of anytime computation and robust control," in *2015 IEEE Real-Time Systems Symposium (RTSS)*, 2015.

[19] B. Lim, S. Son, H. Kim, S. Nah, and K. M. Lee, "Enhanced deep residual networks for single image super-resolution," *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, Jul 2017.

[20] 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 *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[21] F. Yu, W. Xian, Y. Chen, F. Liu, M. Liao, V. Madhavan, and T. Darrell, "Bdd100k: A diverse driving video database with scalable annotation tooling," 2018.

[22] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European Conference on Computer Vision*. Springer, 2016, pp. 21–37.

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

[24] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017, pp. 6517–6525.

[25] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," 2018.

[26] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *Computer Vision – ECCV 2014*, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds. Springer International Publishing, 2014, pp. 740–755.

[27] S. Ryoo, C. I. Rodrigues, S. S. Baghsorkhi, S. S. Stone, D. B. Kirk, and W. mei W. Hwu, "Optimization Principles and Application Performance Evaluation of a Multithreaded GPU Using CUDA," in *Proc. 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, 2008.

[28] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym, "NVIDIA Tesla: A Unified Graphics and Computing Architecture," *IEEE Micro*, vol. 28, no. 2, 2008.

[29] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cuDNN: Efficient Primitives for Deep Learning," 2014, https://arxiv.org/abs/1410.0759.

[30] A. Betts and A. Donaldson, "Estimating the wcet of gpu-accelerated applications using hybrid analysis," in *2013 25th Euromicro Conference on Real-Time Systems (ECRTS)*, 2013.

[31] Y. Huangfu and W. Zhang, "Static wcet analysis of gpus with predictable warp scheduling," in *2017 IEEE International Symposium on Real-Time Computing (ISORC)*, 2017.

[32] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '17. ACM, 2017.

[33] Y. Kim, J. Kim, D. Chae, D. Kim, and J. Kim, "μlayer: Low latency on-device inference using cooperative single-layer acceleration and processor-friendly quantization," in *Proceedings of the Fourteenth EuroSys Conference 2019*, ser. EuroSys '19. ACM, 2019.

[34] "nvprof," https://docs.nvidia.com/cuda/profiler-users-guide/index.html#nvprof-overview.

[35] J. Huang, J. Lee, H. Kim, and H. Lee, "Gpumech: Gpu performance modeling technique based on interval analysis," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, 2014.

[36] Z. Jia, M. Maggioni, B. Staiger, and D. P. Scarpazza, "Dissecting the nvidia volta gpu architecture via microbenchmarking," 2018.

[37] Z. Jia, M. Maggioni, J. Smith, and D. P. Scarpazza, "Dissecting the nvidia turing t4 gpu via microbenchmarking," 2019.

[38] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations (ICLR)*, 2015.

[39] R. Girshick, I. Radosavovic, G. Gkioxari, P. Dollár, and K. He, "Detectron," https://github.com/facebookresearch/detectron, 2018.

[40] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS15. Cambridge, MA, USA: MIT Press, 2015.

[41] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Jiao, L. Qendro, and F. Kawsar, "Deepx: A software accelerator for low-power deep learning inference on mobile devices," in *Proceedings of the 15th International Conference on Information Processing in Sensor Networks*, ser. IPSN 16. IEEE Press, 2016.

[42] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, “Pruning filters for efficient convnets,” 2017.

[43] S. Hong and H. Kim, “An Analytical Model for a GPU Architecture with Memory-level and Thread-level Parallelism Awareness,” in *Proc. 36th International Symposium on Computer Architecture (ISCA)*, 2009.

[44] S. S. Baghsorkhi, M. Delahaye, S. J. Patel, W. D. Gropp, and W. mei W. Hwu, “An Adaptive Performance Modeling Tool for GPU Architectures,” in *Proc. 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, 2010.

[45] Y. Zhang and J. D. Owens, “A Quantitative Performance Analysis Model for GPU Architectures,” in *Proc. 17th International Symposium on High Performance Computer Architecture (HPCA)*, 2011.

[46] G. Wu, J. L. Greathouse, A. Lyashevsky, N. Jayasena, and D. Chiou, “GPGPU Performance and Power Estimation Using Machine Learning,” in *Proc. 21st IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2015.

[47] V. Hirvisalo, “On static timing analysis of gpu kernels,” in *14th International Workshop on Worst-Case Execution Time Analysis*, ser. OpenAccess Series in Informatics (OASIcs), 2014.