IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received November 4, 2018, accepted December 8, 2018, date of publication December 17, 2018,
date of current version January 11, 2019.

Digital Object Identifier 10.1109/ACCESS.2018.2887099

Power- and Time-Aware Deep Learning Inference for Mobile Embedded Devices

WOOCHUL KANG[1], (Member, IEEE), AND JAEYONG CHUNG[2], (Member, IEEE)
[1]Department of Embedded Systems Engineering, Incheon National University, Incheon 22012, South Korea
[2]Department of Electronic Engineering, Incheon National University, Incheon 22012, South Korea

Corresponding author: Jaeyong Chung (jychung@inu.ac.kr)

This work was supported in part by the Basic Science Research Program through the National Research Foundation of Korea (NRF) through the Ministry of Education under Grant NRF-2016R1D1A1B03934266, and in part by the Incheon National University Research Grant in 2018.

ABSTRACT Deep learning is a state-of-the-art approach that provides highly accurate inference for many cyber-physical systems (CPS) such as autonomous cars and robots. Deep learning inference often needs to be performed locally on mobile and embedded devices, rather than in the cloud, to address concerns such as latency, power consumption, and limited bandwidth. However, existing approaches have focused on delivering ''best-effort'' performance to resource-constrained mobile embedded devices, resulting in unpredictable performance under highly variable environments of CPS. In this paper, we propose a novel deep learning inference runtime, called DeepRT, that supports multiple QoS objectives simultaneously against unpredictable workloads. In DeepRT, the multiple inputs/multiple outputs (MIMO) modeling and control methodology is proposed as a primary tool to support multiple QoS goals including the inference latency and power consumption. DeepRT's MIMO controller coordinates multiple computing resources, such as CPUs and GPUs, by capturing their close interactions and effects on multiple QoS objectives. We demonstrate the viability of DeepRT's QoS management architecture by implementing a prototype of DeepRT. The evaluation results demonstrate that, compared with baseline approaches, DeepRT can support the desired inference latency as well as power consumption for various deep learning models in a highly robust manner.

INDEX TERMS Deep learning, DVFS, feedback control, embedded systems, low power, power-awareness, Quality-of-Service, QoS, real-time.

I. INTRODUCTION

In recent years, deep learning has emerged as a state-of-the-art approach that provides robust and highly accurate inference for many intelligent systems [1]. For instance, visual scene-understanding enabled by complex deep neural networks is actively used by many cyber-physical systems (CPS) such as camera-based surveillance, home automation devices, cognitive-assistance wearables, and autonomous vehicles [2], [3]. In many CPS applications, deep learning inference needs to be performed locally on mobile embedded devices, rather than in the cloud to address concerns such as latency, power consumption, limited bandwidths, and privacy [4]. However, since mobile embedded devices have limited resources and their operating environments are highly dynamic [5], it is very challenging to support predictable inference performance. For instance, when a self-driving vehicle drives from a highway to crowded city areas, it needs to perform additional computation such as detecting pedestrians, which might incur sudden increase of inference latency and power consumption due to potential resource contention. For many CPS, the unpredictability of performance might incur problems, ranging from degraded user experience to compromised safety to unexpectedly shorter lifetime. However, many previous approaches have focused on optimizing the performance of deep learning inference for resource-constrained embedded devices either by compressing deep learning models [6], [7] or by exploiting hardware accelerators [8], [9]. These approaches provide significant performance gains, but they are not aware of performance requirements of applications and provide only 'best-effort' performance in terms of inference latency, power consumption, etc.

In this paper, we propose a deep learning inference runtime, called DeepRT [10], that provides predictable inference performance for CPS. In particular, DeepRT's novel QoS management architecture supports multiple goals simultaneously, e.g., inference latency and power consumption, by exploiting MIMO (Multiple Inputs/Multiple outputs) modeling and control techniques [11]. Deep learning inference requires collaboration between heterogeneous computing resources such as CPUs and GPUs, and the proposed MIMO-based QoS management architecture captures close interactions between such heterogeneous computing resources and their effect on multiple QoS objectives.

To show the viability, we have implemented and evaluated the proposed QoS management architecture by extending Caffe [12], a popular open-source deep learning framework. The evaluation results demonstrate that DeepRT's approach to MIMO-based QoS management is significantly more effective than baseline approaches. In particular, DeepRT can support both target inference latency and power consumption simultaneously in a highly robust and efficient manner against unpredictable workloads.

The rest of this paper is organized as follows. Section II gives an overview of DeepRT. Section III presents the architecture of DeepRT and the details of feedback control mechanism of DeepRT. Section IV shows the details of evaluation settings and evaluation results. The related work is presented in Section V. Finally, Section VI concludes the paper and discusses future work.

II. OVERVIEW OF DEEPRT

A. BACKGROUND ON DEEP LEARNING INFERENCE

Deep learning is a machine learning algorithm to solve intuitive problems such as recognizing spoken words or objects in images by applying a cascade of layers of mathematical transformation units. Deep learning *models* define the structure of these layers and their related parameters or weights. The parameter values of the models are *learned* through the process called *training*. During the training phase, batches of hundreds of input data is traversed forward and backward through the layers to update the parameter values until the desired inference accuracy is obtained. The training phase is often performed off-line at the cloud because this phase is highly computation-intensive and might take a long time (e.g., a few days or weeks.) Once a deep learning model is trained, it can be deployed to various systems such as self-driving cars, smartphones, surveillance cameras, and the cloud. The deployed deep learning models are used by applications to *infer* a situation for given sensor inputs. This step is called the *inference* phase. During the inference phase, the input data from sensors such as cameras are transformed through the sequence of layers of the model to generate output. Since the transformation at each layer requires highly data-parallel computation, GPUs that are found in most modern computing hardware are actively exploited for deep learning inference. Figure 1 depicts the structure of an inference task that requires the collaboration between the CPU and the GPU devices. Once an inference task is invoked with

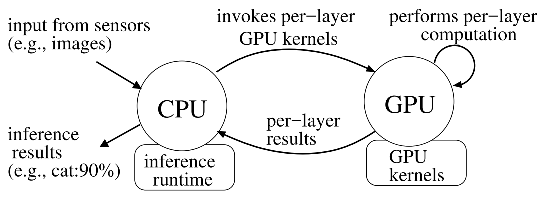


FIGURE 1. Collaboration of CPU and GPU for deep learning inference.

a specified deep learning model M, each layer of M is executed sequentially by invoking the layer's corresponding GPU kernels. This whole process of transforming sensor inputs to inference results is often managed by a deep learning *inference runtime*, such as TensorRT [13] and Caffe [12], that provides a set of optimized implementation of inference operations defined in the model.

Low inference latency and power efficiency are two primary performance metrics during the inference phase. For example, self-driving cars need to complete perception-action cycles on the order of a few tenth of a second. If such deadlines are not supported, users might have negative experience of the application, or the safety in safety-critical systems can be compromised. In many battery-powered devices, power-efficiency is as important as the low latency because they may have to operate without recharging for days or months. For example, current self-driving cars consume around 2.5 kilowatts for making self-driving decisions, and this massive power consumption is becoming a problem that limits driving distances [14].

B. SERVICE MODEL OF DEEPRT

DeepRT is a deep learning inference runtime that supports predictable inference service for applications exploiting deep learning models [10]. DeepRT has been designed for soft real-time applications that require predictable inference latency and power consumption. An application can request DeepRT to initiate an inference task with a deep learning model M and a related Quality-of-Service (QoS) specification Q. The QoS specification Q is defined as a tuple $Q = \langle D, P \rangle$, in which D is a deadline, or desired inference latency, and P is a desired power consumption.

In CPS, inference tasks are usually periodic. For instance, inference tasks periodically sample data from sensors and provide data to DeepRT for timely inference service. Deadline D is an end-to-end latency required to complete each activation of the inference task, or processing all layers in the model M. The (relative) deadline of periodic inference tasks are usually set to their periods because each activation of the inference tasks needs to be finished before their next period. In DeepRT, soft deadline semantics are applied, in which inference tasks are still worth even if they miss their deadlines. For example, inference results for wearable cognitive-assistance systems is still worth even if they miss their deadlines [15]. Soft deadline semantics have been chosen since most inference tasks of CPS run in unpredictable open environments, where the complete knowledge of task sets or timing constraints are not available. The purpose of DeepRT is not to completely avoid deadline misses, but to support QoS management that minimizes the deadline misses at runtime while achieving predictable power consumption.

III. POWER- AND TIME-AWARE INFERENCE

In this section, we discuss DeepRT's core mechanism to support predictable inference latency and power consumption simultaneously.

A. QoS METRICS

Many real-time systems use the *deadline miss ratio* as a primary QoS metric for performance monitoring. The deadline miss ratio shows the ratio of tardy inference tasks to the total number of inference tasks. For instance, mobile applications such as speech-to-text or translation need to complete the inference under a stringent low latency, e.g., 200ms. If the desired deadline miss ratio of the speed-to-text using deep learning is 0.99, then 99% of inference tasks are supposed to meet 200ms deadlines.

However, it turns out that the deadline miss ratio is not appropriate for DeepRT because the task invocations of inference applications are usually low compared to typical real-time applications. For instance, typical frame rates of wearable cognitive-assistance systems are between 5Hz and 12.5Hz [15]. With such a low number of service invocations, the deadline miss ratio has a wide confidence interval and this might lead to unstable statistical interpretation. To address this problem, DeepRT controls the QoS based on the average *tardiness* of inference tasks. For each inference task, we define *tardiness* as a metric to monitor the timeliness of a task:

$$tardiness = \frac{\text{actual inference latency}}{\text{target inference latency}}. \tag{1}$$

When an inference task is being delayed, then its tardiness becomes greater than 1. Conversely, when the tardiness is less than 1, it means that the task is completed earlier than the deadline.

Another QoS metric, which might pose conflicting requirements, is power consumption. In modern processors, the speed of processors, such as CPUs and GPUs, is controlled by their operating clock frequencies. With this capability, we can reduce the ratio of tardy inference tasks at the cost of increased power consumption. For example, doubling the clock frequency of a processor can reduce the latency of tasks running on the processor by up to half. Increasing the processor clock frequency also drops the processor utilization. In many real-time systems, scheduling policies, such as rate monotonic and EDF (Earliest Deadline First), can guarantee no deadline misses if the utilization of the processor is under certain bounds [16]. However, one disadvantage of increasing processor frequency to meet the deadlines is the increased power consumption. As shown in Equation 2, the power consumption is proportional to the processor frequency f:

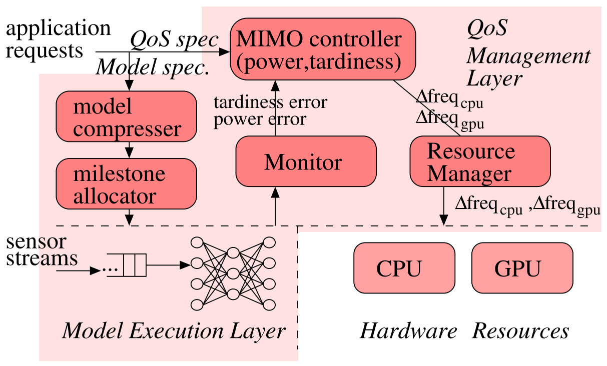


FIGURE 2. DeepRT's QoS management architecture.

$$power(V, f) = C \times V^2 \times f, \tag{2}$$

where C represents a device specific constant, and V indicates the supply voltage [17]. Since V increases proportionally with f, the combination of V and f has a cubic impact on total power consumption. Therefore, a proper processor frequency should be chosen, either statically or dynamically, to support the desired deep learning inference latency under the specified power consumption range.

B. QoS MANAGEMENT ARCHITECTURE

Figure 2 shows the QoS management architecture of DeepRT. DeepRT consists of two major components: the *QoS management layer* and the *model execution layer*. A deep learning inference task is invoked with a service request from applications that specifies a deep learning model M and a set of QoS requirements Q. The *model execution layer* is responsible for typical jobs of a deep learning runtime; it passes the input from sensors to the cascade of layers of the given deep learning model M to generate final inference results. The *QoS management layer* continuously monitors the performance of inference tasks running in the model execution layer, and controls the underlying hardware resources to support the desired QoS requirements Q.

The *model compressor* in the QoS management layer first checks if the model M needs to be compressed at runtime. If the footprint of M is no less than the available memory, M is compressed to reduce its memory footprint.[1] For precise monitoring of the tardiness of inference tasks, milestones are inserted at several chosen layers of model M. Whenever either these milestones are passed or the inference task is completed, its tardiness is reported to the *monitor*. The *monitor* computes the QoS errors, or the differences between the desired and the monitored QoS metrics, i.e., the tardiness error and the power consumption error of inference tasks. The *MIMO feedback controller* computes the clock frequency adaptation of CPU and GPU devices to reduce these QoS

[1]The model compression algorithm is out of scope of this paper. Readers are referred to our previous work [10].

TABLE 1. Notations used in the feedback control loop.

Notation	Meaning
$ref_{\{tard\|pwr\}}$	Target tardiness and power consumption, respectively
$tardiness(k)$	Average inference tardiness measured at the kth period
$power(k)$	Average power consumption measured at the kth period
$e_{tard}(k)$	$ref_{tard} - tardiness(k)$
$e_{pwr}(k)$	$ref_{pwr} - power(k)$
$freq_{cpu}(k)$	The frequency of CPU at the kth period
$freq_{gpu}(k)$	The frequency of GPU at the kth period
$\Delta freq_{cpu}(k)$	Desired adaptation of CPU frequency at the kth period
$\Delta freq_{gpu}(k)$	Desired adaptation of GPU frequency at the kth period

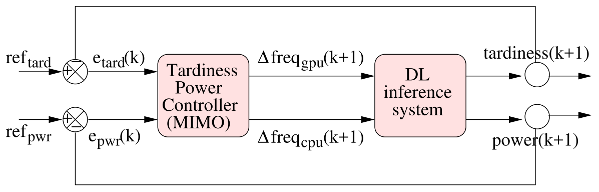


FIGURE 3. MIMO control of tardiness and power.

errors. Finally, the *resource manager* makes a schedule to enforce the requested clock frequency adaptation.

1) FEEDBACK CONTROL PROCEDURE

The goal of QoS management is to support the end-to-end latency of inference tasks and the power consumption close to the target deadline D and the target power consumption P, respectively. To achieve this goal, we might consider scheduling tasks a priori. However, because the environment of CPS is highly dynamic and unpredictable, it is very hard, if not impossible, to have accurate knowledge of related tasks and their resource requirements. For instance, scheduling policies of some resources, such as GPUs, are usually unknown [18].

To address this problem, we propose the *Multiple Inputs/Multiple Outputs* (MIMO) feedback control architecture shown in Figure 3, in which multiple QoS goals are handled dynamically by controlling heterogeneous processors, e.g., CPUs and GPUs. The overall MIMO feedback control procedure is as follows:

1) At the k-th monitoring instant, the average tardiness errors $e_{tard}(k)$ and the average power consumption error $e_{pwr}(k)$ are computed for inference tasks.
2) Based on $e_{tard}(k)$ and $e_{pwr}(k)$, the MIMO controller computes the control signals $\Delta freq_{cpu}(k + 1)$ and $\Delta freq_{gpu}(k + 1)$ for CPU and GPU devices, respectively.
3) Given CPU's and GPU's current frequencies $freq_{cpu}(k)$ and $freq_{gpu}(k)$, the frequencies until the $(k+1)$th monitoring instant are set to $freq_{cpu}(k)+\Delta freq_{gpu}(k+1)$ and $freq_{gpu}(k)+\Delta freq_{gpu}(k+1)$ for CPU and GPU devices, respectively.
4) The model execution layer of DeepRT enforces $\Delta freq_{cpu}(k + 1)$ and $\Delta freq_{gpu}(k + 1)$ using available discrete clock frequencies of CPU and GPU devices, respectively.
5) Wait until the $(k + 1)$th monitoring instant; the time period between the kth and the $(k + 1)$th monitoring instants is the $(k + 1)$th *monitoring period*.
6) Repeat the above steps in 1)- 5) for continuous QoS management.

2) MILESTONES AND MONITORING PERIOD

The monitoring period of the feedback control loop determines the rate of feedback control. The shorter the monitoring period is, the more frequent and timely control of QoS can be provided at the cost of increased overheads. Therefore, the monitoring period should be set considering the trade-off between the timeliness of QoS control and its overhead.

One challenge in setting the monitoring period of DeepRT is the diversity of deep learning models. For instance, the inference latency of small models such as *LeNet* [19] is just a few milliseconds in a mobile device while deeper models such as *GooLeNet* [20] have more than several hundreds milliseconds inference latency in the same environment. If an inference latency is longer than the monitoring period, the feedback control decision for the next monitoring period cannot be made properly due to the unavailability of the tardiness information at the monitoring instant. To handle this problem, DeepRT instruments milestones into several chosen layers for progress monitoring of deep learning models. Each milestone is associated with an internal deadline as follows:

$$\frac{\text{profiled latency to the milestone}}{\text{profiled end-to-end inference latency}} \times \text{deadline } D \qquad (3)$$

Whenever a milestone is passed during the inference phase, its tardiness is calculated according to the internal deadline. For example, Figure 4 shows the latency to the n-th layer of GooLeNet during the inference. If the monitoring period is 200ms, the QoS control cannot be provided at every monitoring instant because the profiled end-to-end inference latency using the GooLeNet model is about 207ms, and, hence, the tardiness information cannot be provided to the MIMO controller. Therefore, at least one milestone needs to be inserted for proper feedback control. Since the profiled latency to the 40th layer is about 103.5ms, if the target deadline D is 200ms, one milestone can be inserted at the 40th layer with the internal deadline of $100ms (= \frac{103.5ms}{207ms} \times D)$. Whenever the 40th layer is passed during the inference, the tardiness of the inference task is computed using this internal deadline, resulting in proper feedback control in every monitoring period.

C. FEEDBACK CONTROL LOOP DESIGN

In this section, we take a systematic approach to designing a MIMO feedback control loop for the simultaneous control of tardiness and power consumption of inference tasks.

1) SYSTEM MODELING

The first step in designing a feedback control loop is to build a system model by mathematically quantifying the effect of control inputs on the monitored QoS metrics. As discussed in

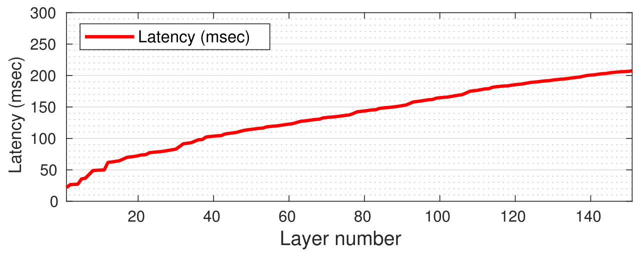


FIGURE 4. **Inference latency to n-th layer of GoogLeNet.**

Section III-B, we are interested in supporting the tardiness of inference tasks and the power consumption simultaneously. To support two QoS goals simultaneously, we need at least two control inputs that can affect the QoS metrics in an effective and meaningful manner [11]. Given *tardiness* and *power* as two QoS metrics, we can easily choose the clock frequencies of two heterogeneous processors involved in inference as two primary tuning knobs to control the tardiness and the power consumption of inference tasks because, as shown in Figure 1, both *tardiness* and *power* of deep learning inference are determined by the close collaboration between CPUs and GPUs. While CPUs are responsible for managing the whole forward process of inference tasks such as staging in/out data between the layers of a deep learning model, GPUs perform computation-intensive parallel transformation of each layer. We choose to use a linear time-invariant MIMO model, shown in Equation 4, to capture these interactions between multiple control inputs ($freq_{cpu}$ and $freq_{gpu}$) and multiple system outputs (*tardiness* and *power*).

$$\begin{bmatrix} tardiness(k+1) \\ power(k+1) \end{bmatrix} = \mathbf{A} \cdot \begin{bmatrix} tardiness(k) \\ power(k) \end{bmatrix} + \mathbf{B} \cdot \mathbf{u}(k), \quad (4)$$

where $\mathbf{u}(k)$ is a vector representation of control inputs:

$$\mathbf{u}(k) = \begin{bmatrix} freq_{gpu}(k) \\ freq_{cpu}(k) \end{bmatrix} \quad (5)$$

The parameters **A** and **B** reflect the modeled system's dynamics. The model shows that the next system state $\begin{bmatrix} tardiness(k+1) \; power(k+1) \end{bmatrix}^T$ is determined by the current state $\begin{bmatrix} tardiness(k) \;\; power(k) \end{bmatrix}^T$ and the control inputs $\mathbf{u} = \begin{bmatrix} freq_{gpu}(k) \;\; freq_{cpu}(k) \end{bmatrix}^T$. Because the model has two control inputs and two system outputs, both **A** and **B** are 2×2 matrices. To obtain the parameters **A** and **B**, we might consider *first principles* of deep learning inference systems. However, deep learning inference systems are very complex artifact that has complex interaction between the inference runtime and various hardware resources. Therefore, we choose to use *system identification*, instead of first principles. System identification is a statistical approach to build mathematical models of dynamic systems from empirical data. During actual system identification, various combinations of two control inputs are applied to the system to stimulate the dynamics of the modeled system. For system identification, we choose to use *CaffeNet* model on a Jetson TK1 mobile device. [2] Figure 5 shows the behavior of DeepRT during the system identification. The model parameters **A** and **B** can be obtained by performing regression analysis using this empirical data. In our study, the obtained model parameters are as follows:

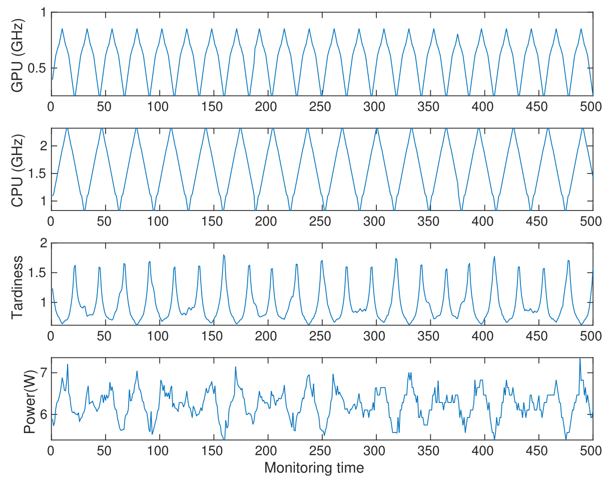


FIGURE 5. **System identification.**

$$\mathbf{A} = \begin{bmatrix} 0.638 & -0.047 \\ 0.529 & 0.154 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} -0.391 & -0.027 \\ 1.744 & 0.488 \end{bmatrix}$$

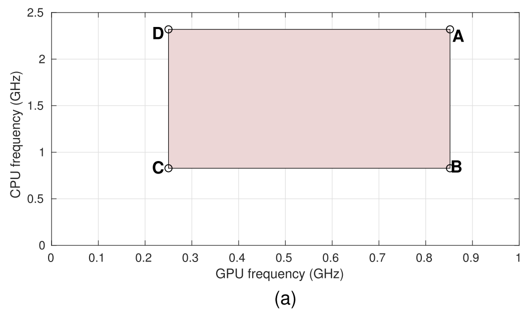
Since matrix **B** quantifies the effect of inputs to the system dynamics, it needs to be analyzed carefully. It should be noted that $freq_{gpu}(k)$'s weight on *tardines* is -0.391 while $freq_{cpu}(k)$'s is only -0.027. This implies that $freq_{gpu}(k)$ has about 14 times higher impact on the tardiness of inference tasks than $freq_{cpu}(k)$. In contrast, $freq_{gpu}(k)$'s weight on *power* is only about four times of $freq_{cpu}(k)$'s.; the weight of $freq_{gpu}(k)$ on *power* is 1.744 while $freq_{cpu}(k)$'s is 0.488. This implies that $freq_{cpu}(k)$ has relatively higher impact on power consumption of inference tasks. We found that various deep learning models (*LeNet*, *CaffeNet*, and *GooLeNet*) manifest very similar dynamics.

2) VALIDATION AND ANALYSIS OF SYSTEM MODEL

Once a system model is obtained, its validity must be evaluated by comparing the model's predictions with actual experimental data. The accuracy metric R^2 $(= 1 - \frac{variance(\text{experimental value - predicted value})}{variance(\text{experimental value})})$ is commonly used to quantify the accuracy of a model. The R^2 value of 1.0 suggests a perfect fit, and a model is considered acceptable for control purposes if $R^2 \geq 0.8$ [11]. In our study, R^2 for the tardiness and the power consumption is 0.84 and 0.80, respectively, and, hence, our model is accurate enough for control purposes.

A valid model of a system can be used to predict the behavior of the system. For example, the system model can be used to investigate the system's *controllability* that tells if there exists some input sequence to drive the system to any

[2]The details of evaluation testbed is discussed in Section IV.



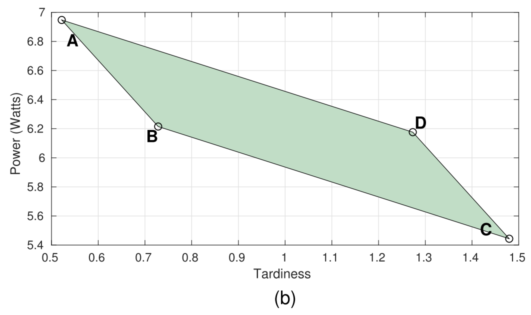


FIGURE 6. Feasible control regions. (a) Range of feasible inputs. (b) Predicted range of feasible outputs.

target state. The *controllability* of a system can be tested with a controllability matrix C shown in Equation 6.

$$C = \begin{bmatrix} \mathbf{A}^{n-1}\mathbf{B} & \mathbf{A}^{n-2}\mathbf{B} & \ldots & \mathbf{AB} & \mathbf{B} \end{bmatrix} \tag{6}$$

A linear time-invariant system is controllable if and only if controllability matrix C is invertible [11]. In our modeled system, the rank of C with the given system parameters $\mathbf{A}$ and $\mathbf{B}$ is 2, and, hence, the target system is controllable. This result implies that any state of $\begin{bmatrix} tardiness & power \end{bmatrix}^T$ can be reachable by applying some sequence of control inputs $\mathbf{u}(0), \mathbf{u}(1), \ldots, \mathbf{u}(n-1), \mathbf{u}(n)$. However, since control inputs $\mathbf{u}(k) = \begin{bmatrix} freq_{gpu}(k) & freq_{cpu}(k) \end{bmatrix}^T$ have limits in real systems, regions of feasible outputs need to be analyzed. For example, Figure 6-(a) illustrates the range of CPU and GPU clock frequencies in a NVIDIA Jetson TK1 embedded board. The solid parallelogram in Figure 6-(b) is the corresponding feasible region predicted by the MIMO model in Equation 4. Each labeled corner in Figure 6-(a) is displayed in Figure 6-(b) with the same letter. A combination of *power* and *tardiness* that does not lie within the parallelogram in Figure 6-(b) is not feasible because either one or both of $freq_{cpu}$ and $freq_{gpu}$ is out of feasible ranges.

3) CONTROLLER DESIGN

Once we have a valid MIMO approximation model, a MIMO controller can be designed for the target system, DeepRT. For its robustness and simplicity, we choose to exploit the proportional integral (PI) control function given by

$$\mathbf{u}(k) = -\mathbf{K}\begin{bmatrix} \mathbf{e}(k) \\ \mathbf{e}_I(k) \end{bmatrix} = -\begin{bmatrix} \mathbf{K}_P & \mathbf{K}_I \end{bmatrix}\begin{bmatrix} \mathbf{e}(k) \\ \mathbf{e}_I(k) \end{bmatrix} \tag{7}$$

TABLE 2. Hardware of the testbed.

CPU	ARM Cortex-A15 (quad-core), 0.051-2.32 GHz (22 levels)
GPU	Kepler GPU with 192 CUDA cores 0.072-0.852 GHz (15 levels)
Memory	2 GBytes shared between CPU and GPU
Storage	16 GBytes eMMC Memory

where $\mathbf{K}_P$ and $\mathbf{K}_I$, respectively, are proportional and integral control gains. At each k-th monitoring instant, the controller calculates the control input $\mathbf{u}(k)$ by monitoring the QoS error $\mathbf{e}(k)$ and the integrated QoS error $\mathbf{e}_I(k)$:

$$\mathbf{e}(k) = \begin{bmatrix} e_{tard}(k) \\ e_{pwr}(k) \end{bmatrix} = \begin{bmatrix} ref_{tard} - tardiness(k) \\ ref_{pwr} - power(k) \end{bmatrix}, \tag{8}$$

$$\mathbf{e}_I(k) = \begin{bmatrix} e_{I,tard}(k) \\ e_{I,pwr}(k) \end{bmatrix} = \begin{bmatrix} e_{I,tard}(k-1) + e_{tard}(k) \\ e_{I,pwr}(k-1) + e_{pwr}(k) \end{bmatrix}. \tag{9}$$

The characteristics of the controller, such as settling time and maximum overshoot, are determined by the control gains $\mathbf{K}_P$ and $\mathbf{K}_I$. To get proper control gains for MIMO systems, *LQR (linear quadratic regulator)* technique is typically used. LQR is a method to get optimal control gains by focusing on the trade-off between control efforts and control errors. Typically, computer tools such as MATLAB command *dlqr* is used. For more details on the LQR method, readers are referred to [11].

IV. EVALUATION

In this section, we evaluate our implementation of DeepRT and compare it with a state-of-the-art implementation of deep learning inference runtime.

A. IMPLEMENTATION AND TESTBED

We have implemented DeepRT by extending *Caffe* [12], which is a popular open source deep learning framework. Deep learning frameworks, such as Caffe and TensorFlow [21], provide the implementation of efficient deep learning inference functionalities, such as optimized GPU kernels for various deep learning layers. However, they do not support QoS. Though our implementation is based on Caffe, the proposed MIMO control mechanism of DeepRT is framework-neutral and can be easily adapted to other deep learning frameworks such as TensorFlow [21].

In our testbed, DeepRT runs on a NVIDIA Jetson TK1 mobile platform. Table 2 summarizes the specification of the Jetson TK1. As shown in Table 2, the clock frequencies of CPU/GPU of Jetson TK1 have a dozens of discrete levels. For fine-grained and smooth control of CPU/GPU clock frequencies, the model execution layer of DeepRT exploits the *pulse width modulation* (PWM) technique [22] that emulates arbitrary clock frequencies by switching between several discrete frequency levels. The operating system of Jetson TK1 is Ubuntu 14.04 Linux, and it supports several CPU DVFS policies through DVFS governors [23]. The default DVFS governor is *on-demand*, in which the clock frequencies

TABLE 3. Baseline approaches.

Open	Vanilla Caffe with a default CPU DVFS governor. GPU frequency is set to the maximum.
MIMO	DeepRT supporting MIMO control of tardiness and power
$SISO_{max}$	DeepRT supporting SISO control of tardiness with GPU. CPU frequency is set to the maximum.
$SISO_{open}$	DeepRT supporting SISO control of tardiness with GPU. CPU frequency is managed by a default DVFS governor.

TABLE 4. Deep learning models.

	Number of parameters	Number of conv. layers	Number of FC layers	Number of total layers
CaffeNet	60.96M	5 (20.8%)	3 (12.5%)	24
GoogLeNet	6.99M	57 (37.5%)	1 (0.6%)	152

of CPUs are adjusted according to the utilization of CPUs. Unlike CPUs, Jetson K1 does not provide DVFS governors for GPUs, and, hence, their clock frequency must be set manually by users.

During the evaluation, the power/energy consumption of the Jetson TK1 is monitored with a Yokogawa WT310E power meter. The real-time power/energy measurements are reported to the Jetson TK1 via USBTMC protocol.

B. BASELINES AND EVALUATION GOALS

The objectives of the performance evaluation are 1) to determine if DeepRT can support multiple QoS goals simultaneously under various unpredictable conditions and 2) to test the robustness of DeepRT against unpredictable workload changes.

We evaluate DeepRT with the baselines shown in Table 3. *Open* represents a state-of-the-art approach to deep learning inference. In *Open*, inference tasks run on the original Caffe deep learning framework with the default CPU DVFS governor, which is *on-demand*. The frequency of GPUs is set to the maximum because GPUs of Jetson TK1 does not provide adaptive DVFS governors. *MIMO* is our approach supporting both the target tardiness and the power consumption of inference tasks via simultaneous control of CPU and GPU devices. Both $SISO_{max}$ and $SISO_{open}$ support the target tardiness of inference tasks through *Single Input/Single Output* (SISO) control of the GPU clock frequency. Therefore, both $SISO_{max}$ and $SISO_{open}$ are not aware of power consumption. In $SISO_{max}$, the frequency of CPU is set to the maximum. In contrast, the CPU frequency of $SISO_{open}$ is adjusted dynamically by the default *on-demand* CPU DVFS governor.

To verify the effectiveness of DeepRT in various deep learning models, two representative, but very contrasting, models shown in Table 4 are considered. *CaffeNet* is a slightly modified variant of *AlexNet* [24]. CaffeNet represents memory-intensive deep learning models that have a large number of parameters. Even though *CaffeNet* has only 24 layers, it has three large fully-connected layers that occupy 96% of its 61 million parameters. *CaffeNet* model's parameters occupy 243 MBytes of storage. Due to this large number

TABLE 5. DeepRT's Tasks.

	Scheduling	Priority	Period
Control task	round robin	10 (real-time)	500ms
Inf. task	FIFO (CPU), unknown (GPU)	5 (CPU, real-time), NA (GPU)	Table 6
misc. task	round robin	0 (non-real-time)	non-periodic

TABLE 6. Inference task's periods and QoS goals.

	Period	Deadline (D)	Target power (P)
CaffeNet	0.08 sec	0.05 sec	5.5W
GoogLeNet	0.20 sec	0.15 sec	6.3W

of parameters, inference tasks with *CaffeNet* model need to perform frequent high-bandwidth memory operations. In contrast, *GoogLeNet* represents highly computation-intensive deep learning models [20]. Even though *GoogLeNet* has deep 152 layers, the model requires only 7 million parameters and takes up 53 MBytes of storage. Most layers of *GoogLeNet* are computationally demanding convolution layers. Both *CaffeNet* and *GoogLeNet* take 224 $\times$ 224 images as input and classify them into 1000 classes.

For the evaluation of DeepRT, tasks shown in Table 5 are involved. Inference tasks of DeepRT perform actual execution of a given deep learning model. Inference tasks are periodic and, as shown in Figure 1, they run both in CPU and GPU devices. Inference tasks are activated periodically to process periodic sensor streams. As shown in Table 6, different periods and QoS goals are assigned to different models, considering their primary applications and resource requirements. For example, the period of *CaffeNet* is determined to achieve 12.5 frames/second, which is required for semantic segmentation of real-time camera images [25]. The target latency, or relative deadline, of the inference task is set to be shorter than its period in order to reserve slack time for non-real-time tasks that perform aperiodic jobs such as processing I/O and GUIs. Because many CPS applications must process sensor inputs in real time without buffering, the batch size of inference tasks is set to one. The control task is responsible for QoS management. The control task is activated periodically every 500ms, and adapts the clock frequencies of CPU and GPU devices according to the monitored inference tardiness and power consumption.

C. AVERAGE PERFORMANCE

In this set of experiments, we investigate if DeepRT can support QoS goals under various conditions. In each experiment, the performance is observed for at least 300 monitoring periods and its average with 99% confidence interval is reported.

1) VARYING TARGET TARDINESS

In this experiment, the target tardiness of inference tasks are varied from 0.7 to 1.3 while the target power consumption is fixed.

Figure 7 shows the results for *CaffeNet*, in which the target power consumption is fixed at 5.5W. As shown

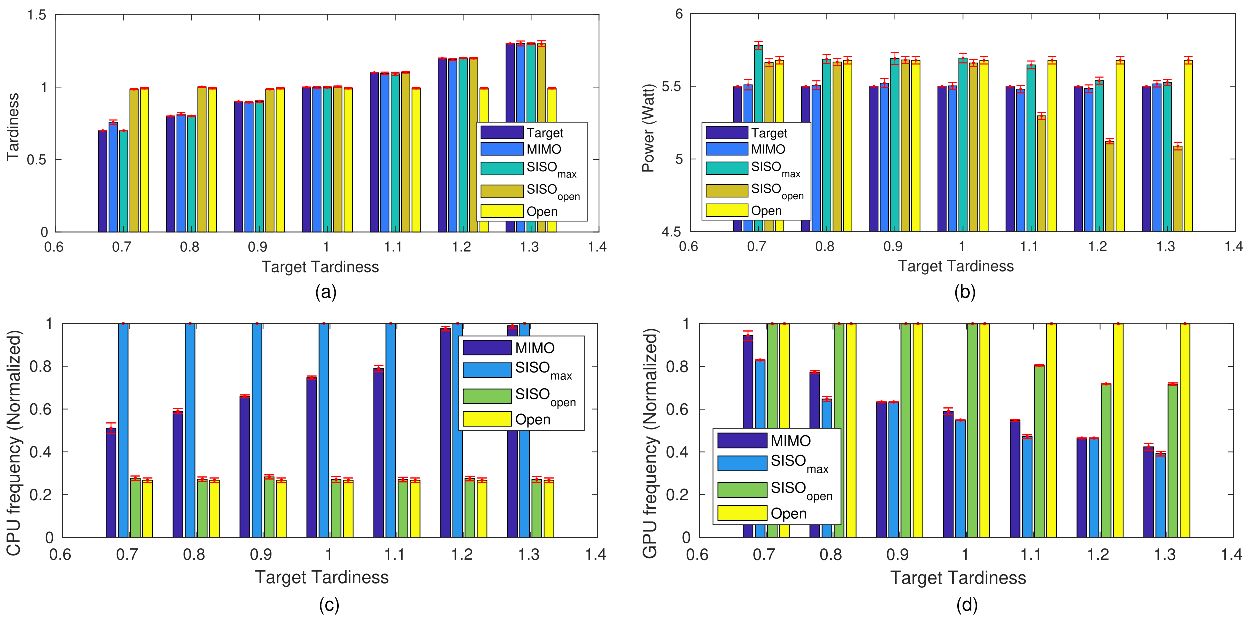


FIGURE 7. Varying target tardiness (CaffeNet). (a) Tardiness. (b) Power. (c) CPU frequency. (d) GPU frequency.

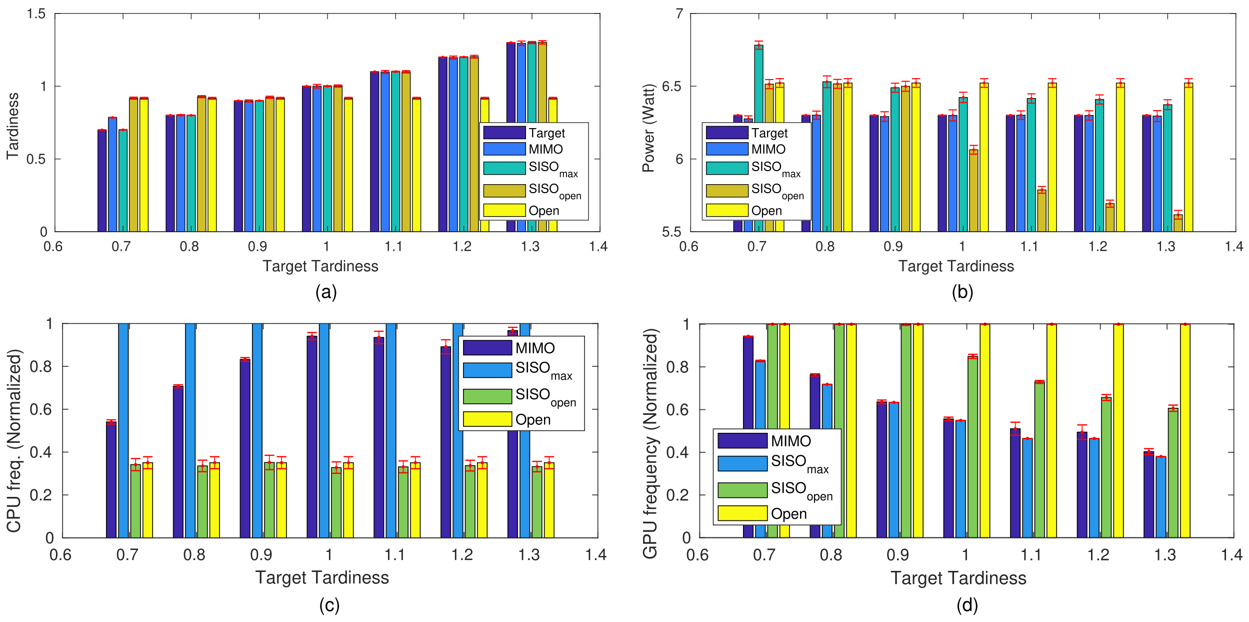


FIGURE 8. Varying target tardiness (GooLeNet). (a) Tardiness. (b) Power. (c) CPU frequency. (d) GPU frequency.

in Figure 7-(a), both *MIMO* and $SISO_{max}$ satisfy the target tardiness very closely. Because *Open* does not support QoS, its tardiness is not affected by varying target tardiness. $SISO_{open}$ is supposed to support the target tardiness through SISO control of tardiness, but this experiment shows that it cannot support the target tardiness. For instance, $SISO_{open}$'s tardiness deviates from the target tardiness by 42% when the target tardiness is 0.7. This is incurred by two conflicting feedback controllers of $SISO_{open}$. In $SISO_{open}$, the GPU clock frequency is controlled by DeepRT's SISO controller, while the CPU clock frequency is controlled by the *on-demand* DVFS governor. Because both controllers are not aware of each other, they cannot make coordinated control decisions, failing to converge to the target tardiness.

Figure 7-(b) shows the power consumption in the same experiment. It shows that the target power consumption is closely supported by *MIMO* while other approaches do not support the target power consumption. This is because the approaches other than *MIMO* are not aware of the power consumption of inference tasks. For instance, $SISO_{max}$'s power consumption varies from 5.78W to 5.52W as the target tardiness varies from 0.7 to 1.3. In contrast, *MIMO* maintains the power consumption at 5.5W despite the varying target tardiness. As shown in Figures 7-(c) and -(d), *MIMO* supports both the target tardiness and the power consumption by simultaneously adjusting the clock frequencies of CPU and GPU devices. This simultaneous and coordinated adaptation of CPU and GPU clock frequencies is achieved by the system model in Equation 4 within the feasible control region shown in Figure 6.

However, as shown in Figure 7-(a), the tardiness of *MIMO* begins to deviate from the target tardiness as the target tardiness is set to be away from 1.0, For example, when the target tardiness is 0.7, the tardiness of *MIMO* is 7% off the target. This is because one of tuning knobs begins to reach its saturation point as the target tardiness moves further away from 1.0. For instance, as shown in Figure 7-(d), the GPU clock frequency is almost at its maximum when the target tardiness is 0.7, and, hence, the clock frequency cannot be further increased to achieve the QoS goals.

Figure 8 shows the result when the same experiment is performed for the *GooLeNet* model. For *GooLeNet*, the target latency and the power consumption of inference tasks are set to 0.15 seconds and 6.3W, respectively. Though *GooLeNet* and *CaffeNet* have very different model structures, we can observe similar results for both deep learning models. As shown in Figures 8-(a) and (b), *MIMO* closely supports the varying target tardiness of inference tasks while maintaining 6.3W power consumption.

2) VARYING TARGET POWER CONSUMPTION

In this experiment, the target power consumption of inference tasks is varied while the target tardiness is fixed at 1.0. Since only *MIMO* is aware of power consumption, only *MIMO* is evaluated for *CaffeNet* and *GooLeNet* models.

Figure 9 shows the result for *CaffeNet*, in which the target power consumption is varied from 5.1W to 5.8W while the target deadline is fixed at 0.05 seconds. In Figures 9-(a) and (b), we can see that *MIMO* closely supports both the target tardiness and the target power consumption while the target power consumption is in between 5.2W and 5.6W. Figure 9-(c) shows that both tunning knobs, i.e., $freq_{cpu}$ and $freq_{gpu}$, are adjusted to support both QoS goals simultaneously. However, if the target power consumption is outside the range between 5.2W to 5.6W, both tardiness and power consumption begin to deviate from the QoS goals. For instance, when the target power consumption is 5.1W, the tardiness and power consumption, respectively, are 1.01 and 5.158W, which are +1% and +1.1% deviations from the respective QoS goals. On the other hand, when the target power consumption is 5.8W, the tardiness and power consumption, respectively, are 0.974 and 5.65W, which is −2.6% and −2.6% different from each QoS goal. These results are incurred because one or both of the tuning knobs, i.e., $freq_{cpu}$ and $freq_{gpu}$, are saturated at both extremes. For instance, in Figure 9-(c), when the target power consumption is 5.7W,

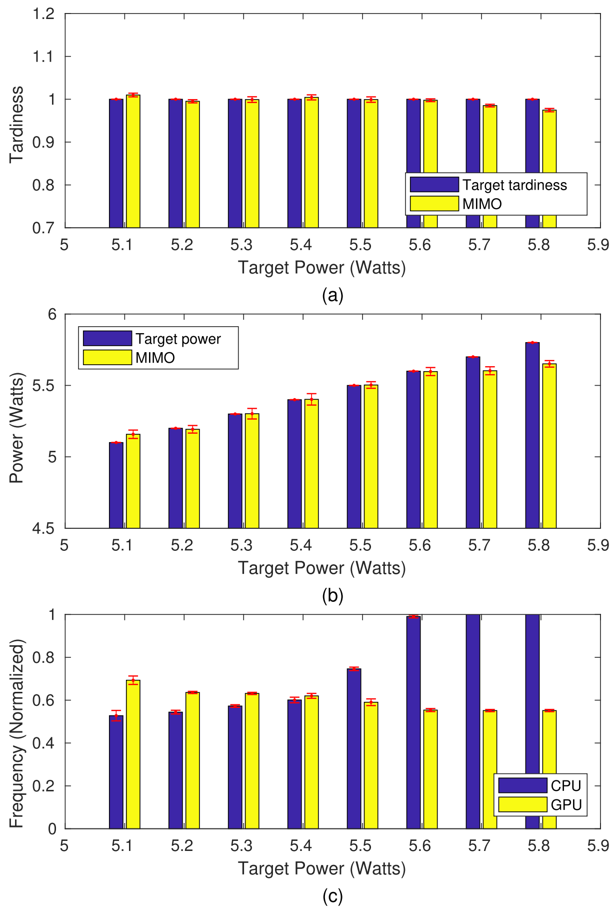


FIGURE 9. **Varying target power (CaffeNet). (a) Tardiness. (b) Power. (c) CPU/GPU frequency.**

the CPU frequency is already at the maximum, and, hence, it is impossible to further increase its frequency. As shown in Figure 6, the combination of target tardiness and power consumption is achievable only within the feasible input region. Once one of inputs is saturated, we need to change QoS goals accordingly.

Figure 10 shows the result for *GooLeNet*, in which the target power consumption is varied from 5.8W to 6.6W while the target inference latency is set to 0.2 seconds. Although the model structure of GooLeNet is deeper and more computationally intensive than CaffeNet, we can observe similar results. As shown in Figures 10-(a) and (b), as far as both clock frequencies of CPUs and GPUs are within the feasible input region, *GooLeNet* closely supports both QoS goals while the desired power consumption is varied.

D. TRANSIENT PERFORMANCE AND ROBUSTNESS

Average performance is not enough to demonstrate the performance of dynamic systems, such as DeepRT, since the performance can change severely in a short time period. For example, consider a situation where a person with a wearable cognitive-assistance device walks to a more crowded area. This change in the physical world might result in sudden increase of workloads to process more objects.

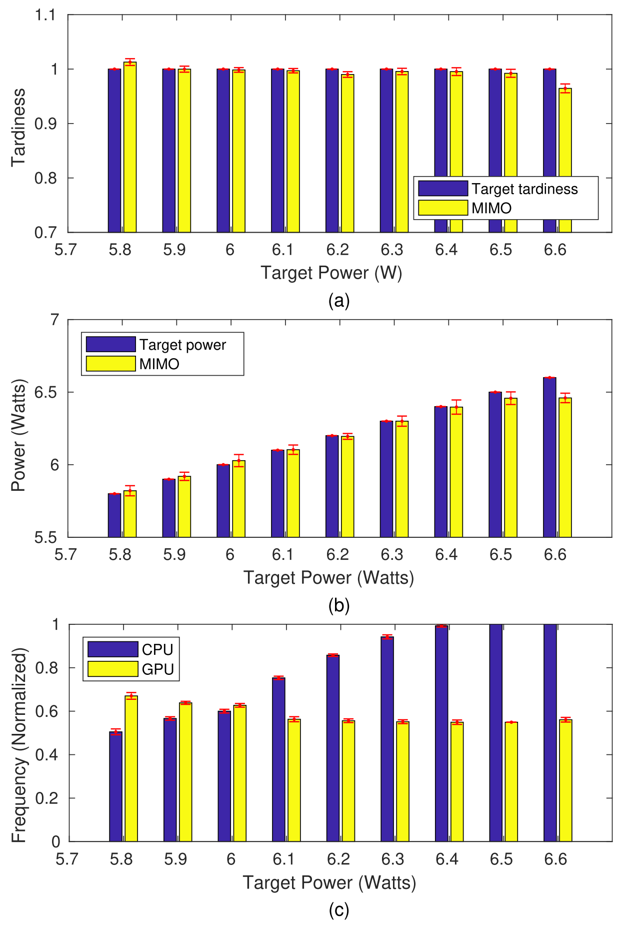


FIGURE 10. Varying target power (GoogLeNet). (a) Tardiness. (b) Power. (c) CPU/GPU frequency.

In this experiment, we introduce sudden change of workloads to observe the transient behavior of DeepRT and its robustness against unexpected disturbances. At the 100th monitoring instant, a periodic disturbance thread is activated and continues until the 200th monitoring instant. The disturbance thread performs matrix multiplication. The size of the matrix is arbitrarily changed from 40×40 to 120×120 to make the workload unpredictable. The period of the disturbance thread is set to 10ms so that it frequently interrupts inference tasks.

Figure 11 shows the transient behavior of *MIMO* when the sudden disturbance is applied to interrupt the execution of the inference task that runs *GooLeNet*. Until the 100th monitoring period, both the target tardiness and the power consumption are closely supported by adjusting the clock frequencies of CPU and GPU devices. As the disturbance is applied at the 100th monitoring instant, the tardiness increases immediately to 1.18. However, the tardiness is stabilized quickly to the target tardiness in the next monitoring period. Similarly, at the 100th monitoring instant, the power consumption increases suddenly to 6.51W, which is 3.3% higher than the QoS goal. However, the power consumption is stabilized to have less than 2% deviation from the QoS goal within one monitoring period. Figure 11 also shows the clock frequencies of GPU and CPU devices during the experiment. We can observe that the QoS goals are supported by adjusting the clock frequencies of both devices simultaneously. For instance, at the 100th monitoring instant, the clock frequencies of GPU and CPU devices are adjusted by +36% and −46%, respectively, to handle the change of workloads.

Figure 11-(b) shows the transient behavior of $SISO_{max}$ in the same experiment. We can observe that $SISO_{max}$ closely satisfies the target tardiness even in the presence of sudden disturbance. For instance, when the disturbance is applied at the 100th monitoring instant, $SISO_{max}$'s tardiness is stabilized within 2 monitoring periods. However, since $SISO_{max}$ is not aware of power consumption, its power consumption increases to 6.95W in the presence of disturbance. As shown in Figure 11-(b), $SISO_{max}$ adjusts only GPU clock frequency to support the target tardiness, so unlike *MIMO*, its power consumption cannot be controlled.

V. RELATED WORKS

There has been significant effort to develop highly efficient and light-weight deep learning inference methods for mobile and embedded devices. Several hardware accelerators have been proposed for efficient deep learning inference on resource-constrained edge devices [7], [8]. They showed significant improvements both in terms of inference latency and energy consumption, but a major drawback is their inflexibility and limited scope of application. Several software-based approaches also have been proposed [13], [26]. *DeepX*, for example, is a software accelerator that decomposes a monolithic deep learning model into unit-blocks for efficient execution on heterogeneous local device processors [26]. There has been also significant effort in hand-crafting small footprint deep learning models [27], [28] or compressing trained models for resource-constrained mobile devices [29]–[31]. Despite these large body of work, none of these approaches support predictable inference performance. Because they are not aware of applications' QoS, they only support unpredictable 'best-effort' performance. Our previous work on DeepRT supports predictable inference latency via feedback control [10]. However, our previous work supports only one QoS metric, and does not consider power consumption as one of primary QoS metrics.

Control theory is one of the most widely used mathematical tool to tune various aspects of dynamic systems, and it has been applied actively to many software systems such as web servers [32], databases [33], and approximate computing frameworks [34]. However, these control-theoretic solutions of previous studies are not directly applicable to our work because they do not consider specific requirements and constraints arising from deep learning inference. A key research question in deep learning inference is how to support multiple QoS goals when various heterogeneous processors are involved. To the best authors' knowledge, no previous work has addressed this question directly. In this work, we propose a MIMO control method as a primary tool that supports

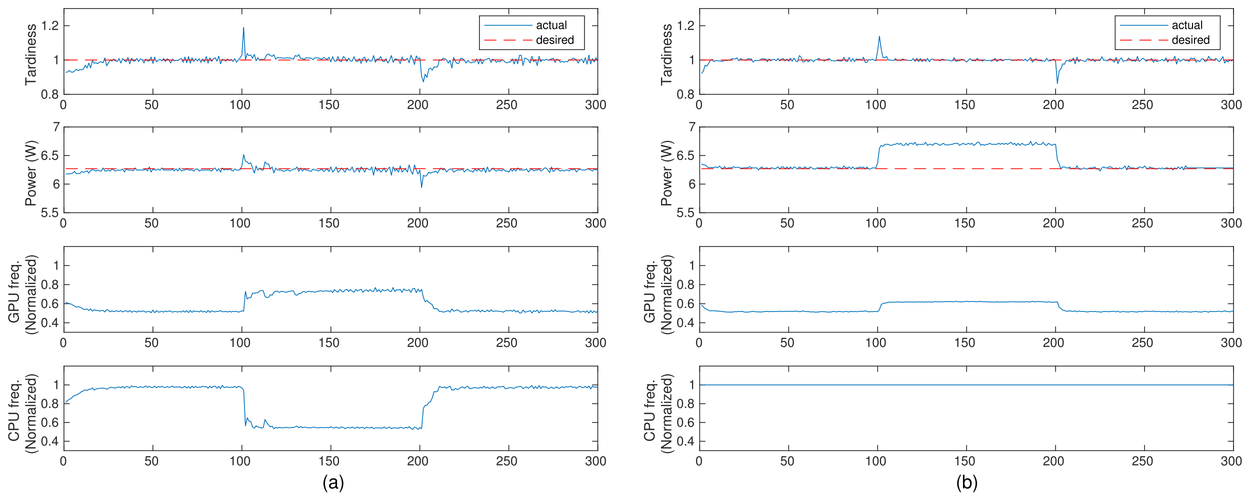


FIGURE 11. **Transient performance (GoogLeNet). (a) *MIMO*. (b) *SISO*$_{max}$.**

multiple QoS goals simultaneously by controlling heterogeneous processors.

DVFS is a primary tuning knob to control the speed of various processors. Since voltage/frequency scaling has a cubic effect on power consumption, there has been a large body of research on exploiting DVFS to support the timeliness of tasks in an energy-efficient manner [35], [36]. The use of DVFS for real-time tasks was theoretically studied first by Yao *et al.* [37]. In their work, they assume an ideal processor that can scale to any continuous speed. Fu *et al.* proposed to use *pulse width modulation* (PWM) techniques to map the desired average power to a series of discrete frequency levels [36]. DeepRT also exploits PWM to emulate continuous frequency levels using discrete frequency levels. Despite the maturity of CPU DVFS, there has been little study on GPU DVFS [38]. Some results demonstrated that applying CPU DVFS strategies to GPUs could be ineffective [39], [40]. Though Deep learning inference tasks require close coordination of CPUs and GPUs, no previous works have addressed the problem of coordinating DVFS of CPUs and GPUs. DeepRT's MIMO controller simultaneously adjusts the speed of CPU and GPU devices to support both timeliness and power consumption of inference tasks.

VI. CONCLUSIONS

This paper introduces DeepRT, a deep learning inference runtime that coordinates underlying heterogeneous resources to support predictable inference latency and power consumption simultaneously. DeepRT demonstrates a control-theoretic solution to supporting predictable inference performance. In particular, the MIMO modeling/control methodology is proposed as a systematic tool that captures the potential interactions between heterogeneous processors and QoS metrics. We have demonstrated the viability of the proposed QoS management approach by implementing an experimental prototype. The results show that DeepRT can support the desired inference latency as well as power consumption for various deep learning models in a highly robust manner.

For future work, we would like to enhance DeepRT in several different directions. First, the feedback controller of DeepRT will be extended to include more QoS metrics such as inference accuracy. By pruning low-weight connections of deep learning models, the computational overhead can be reduced significantly. However, model-pruning usually reduces the inference accuracy. We plan to study a runtime mechanism that makes a systematic negotiation between inference accuracy and computational overheads. Second, we are interested in supporting adaptive feedback control that allows a controller itself can be adjusted online. We believe that adaptive control approaches can significantly increase the generality and robustness of DeepRT's QoS management architecture.

REFERENCES

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org

[2] F. Falcini, G. Lami, and A. M. Costanza, "Deep learning in automotive software," *IEEE Softw.*, vol. 34, no. 3, pp. 56–63, May/Jun. 2017.

[3] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello. (2016). "ENet: A deep neural network architecture for real-time semantic segmentation." [Online]. Available: https://arxiv.org/abs/1606.02147

[4] W. Shi and S. Dustdar, "The promise of edge computing," *Computer*, vol. 49, no. 5, pp. 78–81, 2016.

[5] Y.-F. Tu and G.-J. Hwang, "The roles of sensing technologies and learning strategies in library-associated mobile learning: A review of 2007–2016 journal publications," *Int. J. Mobile Learn. Org.*, vol. 12, no. 1, pp. 42–54, 2018.

[6] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," *CoRR*, 2015. [Online]. Available: http://arxiv.org/abs/1510.00149

[7] S. Han *et al.*, "EIE: Efficient inference engine on compressed deep neural network," in *Proc. 43rd Int. Symp. Comput. Archit. (ISCA)*, 2016, pp. 243–254.

[8] Z. Du *et al.*, "ShiDianNao: Shifting vision processing closer to the sensor," in *Proc. ACM/IEEE 42nd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2015, pp. 92–104.

[9] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proc. 44th Annu. Int. Symp. Comput. Archit. (ISCA)*, New York, NY, USA, 2017, pp. 1–12.
[10] W. Kang and J. Chung, "DeepRT: predictable deep learning inference for cyber-physical systems," *Real-Time Syst.*, pp. 1–13, Jul. 2018, doi: 10.1007/s11241-018-9314-y.
[11] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, *Feedback Control of Computing Systems*. Hoboken, NJ, USA: Wiley, 2004.
[12] Y. Jia *et al.* (2014). "Caffe: Convolutional architecture for fast feature embedding." [Online]. Available: https://arxiv.org/abs/1408.5093
[13] (2017). *NVIDIA TensorRT*. [Online]. Available: https://developer.nvidia.com/tensorrt
[14] J. Stewart. (Feb. 2018). Self-driving cars use crazy amounts of power, and it's becoming a problem. Wired. [Online]. Available: https://www.wired.com/story/self-driving-cars-power-consumption-nvidia-chip/
[15] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for VM-based cloudlets in mobile computing," *IEEE Pervasive Comput.*, vol. 8, no. 4, pp. 14–23, Oct. 2009.
[16] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, 1973.
[17] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," *ACM SIGOPS Oper. Syst. Rev.*, vol. 35, no. 5, pp. 89–102, 2001.
[18] T. Amert, N. Otterness, M. Yang, J. H. Anderson, and F. D. Smith, "GPU scheduling on the NVIDIA TX2: Hidden details revealed," in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, Dec. 2017, pp. 104–115.
[19] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.
[20] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2015, pp. 1–9.
[21] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. OSDI*, vol. 16. 2016, pp. 265–283.
[22] Y. Fu, N. Kottenstette, C. Lu, and X. D. Koutsoukos, "Feedback thermal control of real-time systems on multicore processors," in *Proc. 10th ACM Int. Conf. Embedded Softw. (EMSOFT)*, New York, NY, USA, 2012, pp. 113–122.
[23] V. Pallipadi and A. Starikovskiy, "The ondemand governor," in *Proc. Linux Symp.*, vol. 2, 2006, pp. 215–230.
[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.
[25] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2015, pp. 3431–3440.
[26] N. D. Lane *et al.*, "DeepX: A software accelerator for low-power deep learning inference on mobile devices," in *Proc. 15th ACM/IEEE Int. Conf. Inf. Process. Sensor Netw. (IPSN)*, Apr. 2016, pp. 1–12.
[27] A. G. Howard *et al.* (2017). "MobileNets: Efficient convolutional neural networks for mobile vision applications." [Online]. Available: https://arxiv.org/abs/1704.04861
[28] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50× fewer parameters and < 0.5 MB model size." [Online]. Available: https://arxiv.org/abs/1602.07360
[29] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," in *Proc. Brit. Mach. Vis. Conf.*, 2014. [Online]. Available: http://www.bmva.org/bmvc/2014/papers/paper073/
[30] Y. Gong, L. Liu, M. Yang, and L. Bourdev. (2014). "Compressing deep convolutional networks using vector quantization." [Online]. Available: https://arxiv.org/abs/1412.6115
[31] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 2285–2294.
[32] C. Lu, T. F. Abdelzaber, J. A. Stankovic, and S. H. Son, "A feedback control approach for guaranteeing relative delays in Web servers," in *Proc. 7th IEEE Real-Time Technol. Appl. Symp.*, Taipei, Taiwan, 2001, pp. 51–62, doi: 10.1109/RTTAS.2001.929865.
[33] W. Kang and J. Chung, "Energy-efficient response time management for embedded databases," *Real-Time Syst.*, vol. 53, no. 2, pp. 228–253, 2017, doi: 10.1007/s11241-016-9264-1.
[34] H. Hoffmann, "JouleGuard: Energy guarantees for approximate applications," in *Proc. 25th Symp. Oper. Syst. Princ. (SOSP)*, New York, NY, USA, 2015, pp. 198–214.
[35] J.-J. Chen and C.-F. Kuo, "Energy-efficient scheduling for real-time systems on dynamic voltage scaling (DVS) platforms," in *Proc. 13th IEEE Int. Conf. Embedded Real-Time Comput. Syst. Appl.*, Aug. 2007, pp. 28–38.
[36] X. Fu and X. Wang, "Utilization-controlled task consolidation for power optimization in multi-core real-time systems," in *Proc. IEEE 17th Int. Conf. Embedded Real-Time Comput. Syst. Appl. (RTCSA)*, vol. 1, Aug. 2011, pp. 73–82.
[37] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced cpu energy," in *Proc. 36th Annu. Symp. Found. Comput. Sci.*, Oct. 1995, pp. 374–382.
[38] X. Mei, Q. Wang, and X. Chu, "A survey and measurement study of GPU DVFS on energy conservation," *Digit. Commun. Netw.*, vol. 3, no. 2, pp. 89–100, 2017.
[39] D. H. K. Kim, C. Imes, and H. Hoffmann, "Racing and pacing to idle: Theoretical and empirical analysis of energy optimization heuristics," in *Proc. IEEE 3rd Int. Conf. Cyber-Phys. Syst., Netw., Appl.*, Aug. 2015, pp. 78–85.
[40] Y. Abe, H. Sasaki, S. Kato, K. Inoue, M. Edahiro, and M. Peres, "Power and performance characterization and modeling of GPU-accelerated systems," in *Proc. IEEE 28th Int. Parallel Distrib. Process. Symp.*, May 2014, pp. 113–122.

WOOCHUL KANG (S'09–A'09–M'14) received the Ph.D. degree in computer science from the University of Virginia, in 2009. He was a Senior Researcher with the Electronics and Telecommunications Research Institute, South Korea, from 2000 to 2004 and from 2009 to 2012, and a Post-Doctoral Research Associate with the University of Illinois at Urbana–Champaign, USA, from 2012 to 2013. He is currently an Associate Professor with the Department of Embedded Systems Engineering, Incheon National University, Incheon, South Korea. His research interests include real-time systems, distributed middleware, feedback control of computing systems, and deep learning for embedded systems.

JAEYONG CHUNG received the B.S. degree in electrical engineering from Yonsei University, Seoul, South Korea, in 2006, and the M.S. and Ph.D. degrees in electrical and computer engineering from the Department of Electrical and Computer Engineering, The University of Texas at Austin, Austin, in 2008 and 2011, respectively. He was with the Strategic CAD Lab, Intel, and the IBM Thomas J. Watson Research Center during the summers of 2008 and 2010, respectively. From 2011 to 2013, he was with the Design Compiler Team at Synopsys, Inc., Mountain View, CA, USA. He is currently an Associate Professor with the Department of Electronic Engineering, Incheon National University, Incheon, South Korea. His current research interests include neuromorphic systems and deep learning.