Coordinated Batching and DVFS for DNN Inference on GPU Accelerators

Seyed Morteza Nabavinejad, Sherief Reda, *Senior Member, IEEE,* and Masoumeh Ebrahimi, *Senior Member, IEEE*

Abstract—Employing hardware accelerators to improve the performance and energy-efficiency of DNN applications is on the rise. One challenge of using hardware accelerators, including the GPU-based ones, is that their performance is limited by internal and external factors, such as power caps. A common approach to meet the power cap constraint is using the Dynamic Voltage Frequency Scaling (DVFS) technique. However, the functionally of this technique is limited and platform-dependent. To tackle this challenge, we propose a new control knob, which is the size of input batches fed to the GPU accelerator in DNN inference applications. We first evaluate the impact of batch size on power consumption and performance of DNN inference. Then, we introduce the design and implementation of a fast and lightweight runtime system, called BatchDVFS. Dynamic batching is implemented in *BatchDVFS* to adaptively change the batch size, and hence, trade-off throughput with power consumption. It employs an approach based on binary search to find the proper batch size within a short period of time. Combining dynamic batching with the DVFS technique, *BatchDVFS* can control the power consumption in wider ranges, and hence, yield higher throughput in the presence of power caps. To find near-optimal solution for long-running jobs that can afford a relatively significant profiling overhead, compared with *BatchDVFS* overhead, we also design an approach, called BOBD, that employs Bayesian Optimization to wisely explore the vast state space resulted by combination of the batch size and DVFS solutions.Conducting several experiments using a modern GPU and several DNN models and input datasets, we show that our *BatchDVFS* can significantly surpass the techniques solely based on DVFS or batching, regarding throughput (up to 11.2x and 2.2x, respectively), while successfully meeting the power cap.

Index Terms—deep neural networks, GPU accelerator, power consumption, throughput, batch size, dynamic voltage frequency scaling

1 Introduction

Nowadays, DNNs can achieve state-of-the-art results in various applications such as speech recognition [1], computer vision [2], and natural language processing [3]. Due to high computational complexity and resource demand of large and complex DNNs, a large body of research has focused on designing and implementing hardware accelerators for training and inference phases of such DNNs. Different hardware platforms such as manycore systems [4], [5], GPUs [6], [7], FPGAs [8], [9], and ASICs [10], [11] are employed to design sophisticated accelerators. The GPU accelerator is a popular option for improving the performance of DNNs due to its programmability and scalability features and the proven promising results. While in theory, a GPU accelerator can exploit its maximum power capacity when executing applications, in a real-world setup, the available power budget is capped by an external agent. The reason is that different components of a computing platform such as CPU, memory modules, storage, network interface, and hardware accelerators (e.g., GPU accelerator), usually share the power source. Since the maximum power delivered by the power source is less than the sum of the power capacity of all components in the platform, the real power share of each component is less than its nominal power capacity [12]. Consuming more power by a component than its allocated share means power shortage in other components. The power cap can severely degrade the performance of DNNs on GPU accelerators, and hence, one should employ proper techniques and methods to address this challenge.

Employing dynamic voltage frequency scaling (DVFS) it the common practice to manage the power usage within the power cap in processors such as CPUs and GPUs. A large body of research has studied power management in GPU accelerators and have proposed various approaches based on DVFS [13], [14], [15]. While many GPU vendors have developed user-friendly interfaces for DVFS management (e.g., nvidia-smi by Nvidia [16]), the application of those interfaces is restricted due to several obstacles. First, the DVFS levels accessible on different GPUs are limited to the ones designed by the GPU vendor, and the end-user cannot set the DVFS to a level beyond the available ones. Consequently, the power consumption of the GPU accelerator is also limited to a set of certain values. Second, increasing the DVFS level does not certainly lead to performance improvement. An application (e.g., DNN) that is running on GPU should also be able to fully utilize the available resources such as streaming multiprocessors (SMs); otherwise, DVFS increases the working frequency of GPU streaming multiprocessors without any benefit in the performance of the application.

To address the aforementioned shortages of the conventional DVFS technique in GPU-based DNN inference accelerators, we introduce a new control knob, which is the batch size. Batching is a common approach for improving the throughput of DNN inference and has been studied in previous works [17], [18]. Batching helps improve resource utilization, and hence, throughput by processing a batch of inputs instead of an individual input. We show that the larger batches can lead to higher throughput, but it also leads to higher resource utilization, and consequently, higher power consumption. Therefore, batch size (BS) can be employed as a new control knob for managing the power consumption of DNN inference, and to trade-off throughput with power. Leveraging this control knob, we design and implement a lightweight runtime system called *BatchDVFS*[1]. It can determine the batch size for each time epoch consid-

- *S. M. Nabavinejad is with the School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran*
- *S. Reda is with School of Engineering, Brown University, Providence, RI*
- *M. Ebrahimi is with KTH Royal Institute of Technology, Sweden*
  E-mail: nabavinejad@ipm.ir, sherief_reda@brown.edu, mebr@kth.se

1. source code available at: https://github.com/nabavinejad/BatchDVFS

ering the power cap of the GPU accelerator, such that the throughput of the DNN inference is maximized. *BatchDVFS* is based on binary search, and hence, can select the proper batch size in a fraction of time. *BatchDVFS* can dynamically change the batch size during the execution of applications, with no interrupt. Furthermore, *BatchDVFS* leverages the DVFS technique to manage the power consumption in a wider range and finer-grain, that is beyond the sole capabilities of batching or DVFS techniques alone.

The *BatchDVFS* is designed for making immediate decisions to avoid power cap violation at the beginning of the job, or when the system has a dynamic power cap that is changed time to time. Therefore, it puts less emphasize on the optimality of the solutions and is more focused on finding a valid solution as quick as possible. Hence, it can render the performance of job low, especially when the power cap is constant for a long period of time for long-running jobs. To address this challenge, we design another offline approach that uses Bayesian Optimization (BO) [19], [20]. This approach, which we call *Bayesian Optimization for coordinating BS and DVFS (BOBD)*, can find the combination of batch size and DVFS that leads to near-optimal solution, but with a significant time overhead. Hence, it cannot be used instead of *BatchDVFS*, but rather complements it.

We conduct various experiments to evaluate the efficacy of *BatchDVFS*. We employ a P40 GPU as the accelerator, several well-studied image classification DNNs as workloads, and two popular image datasets as the input of DNN inference. The experimental results show that *BatchDVFS* can maintain the power cap while improving the throughput by up to 11.2x and 2.2x compared with two other approaches that only use one of DVFS or batching techniques, respectively.

The main contributions of the paper are as follows:

- We study the impact of batching on throughput, resource utilization, and power consumption of DNN inference on a GPU accelerator. We change the batch size to understand how different batch sizes affects the aforementioned parameters.
- We implement dynamic batch sizing for image classification DNNs that enables us to change the batch size of DNN inference on the fly, with least possible overhead and without any interrupt.
- We design and implement a lightweight runtime system called *BatchDVFS* that can find the proper batch size in a few steps using binary search. Leveraging the advantages of both adaptive batching and DVFS technique, *BatchDVFS* tries to keep the power consumption lower than the power cap while maximizing the throughput as measured by the number of inputs processed per second.
- To find near-optimal solution for long-running jobs that can afford a relatively significant profiling overhead, compared with *BatchDVFS* overhead, we design a BOBD approach that leverages Bayesian Optimization to wisely explore the vast state space resulted by combination of batch size and DVFS, in pursue of finding a solution with higher throughput than *BatchDVFS* solution.

The rest of the paper is organized as follows: we discuss the motivation behind our study in Section 2. Our proposed approach is introduced in Section 3. We evaluate our approach and present results of experiments in Section 4. Finally, we summarize the related works in Section 5 and conclude the paper in Section 6.

2 Motivation

Batching is a common approach widely studied and employed in previous works to improve the throughput of DNN inference. In this approach, the input data is fed to the DNN for inference in the form of batches, instead of individually. This approach especially works for the GPU-based DNN accelerators, since it can leverage the intrinsic parallel architecture of GPU to significantly improve the throughput. One parameter that can be tuned when using this approach is batch size. In this section, we aim to show the impact of the batch size on power and performance of DNN inference. We employ three DNNs, namely Inception [21], ResNet [22], and MobileNet [23] and two image datasets as input with 10,000 images, one from the ImageNet dataset [24] and the other from the Caltech-256 dataset [25]. We deploy the three DNNs on an Nvidia P40 GPU in sequence, and repeat that several times to process the two datasets. In each iteration, we double the batch size and monitor the throughput (image per second), as well as the power consumption and resource utilization of the GPU (later in Section 4, we discuss the experimental setup in more detail).

The throughput and maximum power consumption are shown in Fig. 1, which reveals the strong relationship between the batch size and power and throughput. With increasing the batch size, both throughput and maximum power consumption increase. From the results, we can conclude that larger batch size can yield higher throughput, and hence, it is beneficial to use large batch sizes. However, since the high batch size leads to high power consumption, we need to control the power when a power cap is applied on the system. In this work, we target the maximum throughout while respecting the power cap. To reach the maximum throughout, the batch size can be increased at the cost of higher power consumption. However, it can be observed that with a proper choice of the batch size and minimal loss of throughout, more energy-efficiency can be achieved. This investigation is part of our future work.

To go further into the details, in Fig. 2, we depict the power consumption and Streaming Multiprocessors (SMs) utilization over time for one of the DNNs (ResNet with the ImageNet dataset) for BS = 256. We observe that the utilization and power consumption of SMs are significantly fluctuating. As presented in this figure, when the input batch is being prepared for execution, the power consumption is low and SMs utilization is zero because no computation is happening. But when input execution starts, spikes happen on the SMs utilization, and consequently, consumed power. These spikes determine the maximum power consumption.

The conventional approach to manage the power consumption is DVFS, which is designed and implemented in almost all the processors, including the GPU we have used. Our GPU supports a wide range of frequency levels, from 544 MHz to 1531 MHz. To study the effectiveness of the DVFS technique under different batch sizes, we apply ten DVFS levels (including the lowest and highest ones) on various DNNs under different batch sizes for the ImageNet dataset. The results depicted in Fig. 3 clearly indicate that the dynamic range of power consumption covered by DVFS is limited to batch size. For small batch sizes (e.g., MobileNet-batch size 1), setting the DVFS on the maximum level can increase the power up to a certain limit. As the batch size increases (e.g., MobileNet-batch size 256), the DVFS can reach higher power consumption, but the lower bound of the power consumption cannot be reduced more

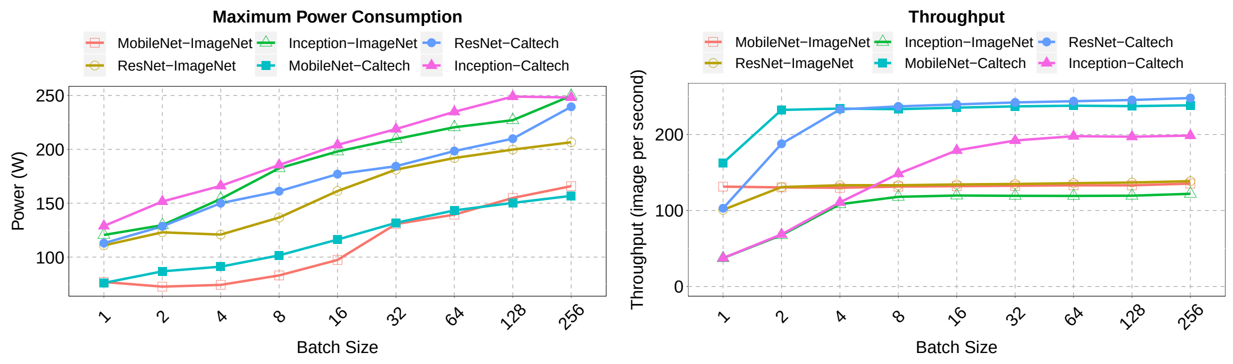

Fig. 1. Impact of the batch size on throughput and power consumption of DNN inference on the P40 GPU.

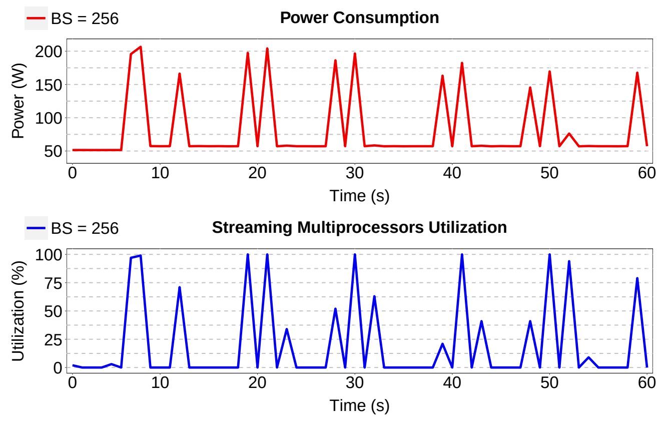

Fig. 2. Relationship between resource utilization of GPU and its power consumption.

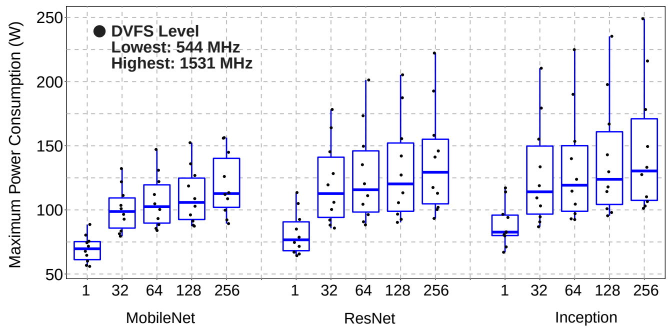

Fig. 3. Dynamic power range of DVFS under various batch sizes.

than a certain threshold, even when the lowest DVFS level is applied.

From the results, we can conclude that the combination of DVFS and a constant batch size might significantly limit the dynamic power range and throughput of DNN inference on GPU accelerators. Having a small batch size and relaxed power cap, DVFS alone cannot fully utilize the available power capacity, and consequently, the throughput would be dramatically less than the achievable throughput by a larger batch size. On the other hand if a large batch size is selected to achieve high throughput, a tight power cap cannot be met by DVFS, and it leads to serious power cap violation. To address this challenge, we propose to use the combination of dynamic batch size and DVFS to maximize the throughput, while meeting the power cap.

3 Methodology

3.1 Problem Statement

The problem that we aim to address in this work is as follows: Given a deployed inference application on the GPU and an external power cap (P_{cap}), we identify the batch size (BS) and DVFS levels to maximize throughput, as measured by the number of inputs processed per second, subject to the power cap. Both throughput and power consumption of DNN inference application are a function of the batch size and DVFS, in addition to other parameters such as input dataset, temperature of GPU, etc. The objective function is to maximize the throughput over the course of time (T) while meeting the power cap.

$$Maximize \quad \frac{1}{T}\sum_{t=1}^{T} Throughput^{t} \tag{1}$$

s.t.

$$Power^{t} \leq P_{cap} \tag{2}$$

3.2 BatchDVFS

A key prerequisite to have a runtime system that can effectively decide on the batch size during the execution of DNNs is implementing dynamic batch sizing in the applications. By default, the batch size can be determined only when the application is submitted to the GPU accelerator. Once the application is deployed and started the execution, the batch size cannot be changed any longer. The conventional approach to change the batch size is to terminate the running instance and launch a new one with a different batch size, which imposes significant delay or reduces the average throughput. To mitigate this challenge, we implement dynamic batch sizing by modifying a few lines of code. The changes are straightforward and do not degrade the programmability of the DNNs. Besides, it imposes almost no notable overhead on latency or throughput. Dynamic batch sizing enables us to change the batch size on the fly without any interruption to the flow of DNN inference. The summary of changes applied on the code used for inference can be found on the code repository of the paper. These changes are implemented in the code that is used in the inference phase to load the DNN model and prepare and send the images to the inference graph of the model. When preparing the images, we can determine the batch size and change it if required as well, and then send the images and batch size to the graph. Note that our approach requires no modification to TensorFlow library. Having dynamic batch sizing implemented, we can proceed to the design and implementation of our runtime system, *BatchDVFS*.

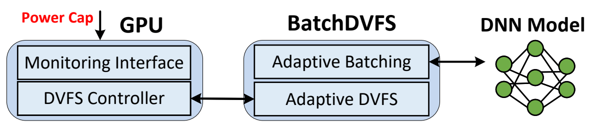

Fig. 4. Overall flow of *BatchDVFS*

The design objective of *BatchDVFS* is to maximize the average throughput while considering the power cap. To achieve this objective, *BatchDVFS* dynamically changes the batch size over the course of time. In the design of *BatchDVFS* we take into account the observations presented in Section 2. The first key observation is that both maximum power consumption and throughput increase with the batch size. Therefore we can assume that those parameters are sorted in ascending order with respect to the batch size. Having them sorted, *BatchDVFS* can employ during runtime a pseudo binary search approach to efficiently search the state space in a few steps and find the most suitable batch size. Since the time complexity of the binary search is *O (log n)*, the time overhead of *BatchDVFS* is negligible.

The second observation is the spiking nature of the power consumption (see Fig. 2), which is used in the design of *BatchDVFS*. Considering maximum power consumption, which occurs during the utilization spikes, it is essential to make sure the power consumption does not surpass the power cap during those spikes. Thereby, after changing the batch size, *BatchDVFS* waits for a few batches to be processed to profile the power consumption of a few number of utilization spikes, and then decide based on that. The last observation used in the design of *BatchDVFS* is the impact of DVFS on the power consumption. From Fig. 3, we see that while the DVFS alone cannot cover a wide range of power consumption, combining it with different batch sizes can effectively handle the power consumption.

Given the aforementioned observations, the overall flow of *BatchDVFS* can be described as follow, also presented in Algorithm 1 and Fig. 4. *BatchDVFS* starts with a default batch size (1 in our experiments). After processing a few batches (for three seconds in our experiments) and profiling the power consumption (line 1), it compares the profiled power data with the power cap. If the maximum power consumption is equal or less than power cap and greater than or equal to a coefficient of it ($\alpha \times P_{cap}$), then the current batch size is proper enough to maintain the power cap and maximize the throughput, and no further action is needed (lines 5-6). We have set $\alpha = 0.8$ in the experiments. Considering a period for power consumption, instead of an absolute equality (e.g., max(PowerReading) == P_{cap}) helps system to achieve an stable state. If we consider an absolute equality, then the system should constantly change the batch size because even batches with the same size yield various maximum power consumption that might be less or more than power cap. Finally, these excessive batch size changes can lead to enormous instances of power cap violation.

If the maximum power consumption is less than $\alpha \times P_{cap}$, it means there is room to improve throughput by increasing the batch size. Therefore, *BatchDVFS* sets the batch size equal to the batch size in the middle of the current batch size and the largest possible batch size. If the current batch size is the largest one, *BatchDVFS* starts increasing the DVFS to further improve the throughput. If DVFS is set to its highest level, then no further throughput improvement is possible through combination of batching and DVFS (lines 7-12). If the maximum power consumption is greater than the power cap, *BatchDVFS* sets the batch size as the one in the middle of the lowest possible batch size and the current batch size. If the current batch size is the smallest one, no further batch size reduction is possible. At this time, *BatchDVFS* starts reducing the DVFS to further decrease the power and meet the power cap. If the DVFS has reached its lowest level, the power cap cannot be met in the current state (lines 13-23). We give priority to batch size because of the finer granularity it provides to control the power consumption over DVFS. When we change the DVFS level, we observe a significant change in power consumption, which is due to limited levels of DVFS. However, a wider range of values can be employed for batch size that helps to manage the power in smaller steps.

Algorithm 1 BatchDVFS

```
1:  Input: $P_{cap}$, SB(1:N):Set of available batch sizes in ascending order
2:  minBS, initialBS, currentBS = 1, maxBS = N, BS = SB(initialBS), PowerReading = []
3:  while True do
4:      PowerReadng.append(monitorPower(inference(BS)))
5:      if $\alpha \times P_{cap} \leq$ max(PowerReading) $\leq P_{cap}$ then
6:          Continue with BS
7:      if max(PowerReading) $< \alpha \times P_{cap}$ then
8:          minBS = currentBS
9:          currentBS = ceil($\frac{minBS+maxBS}{2}$)
10:         BS = SB(currentBS)
11:         if BS == MAXBS then # MAXBS = 256 in our work
12:             Start increasing the DVFS
13:     if max(PowerReading) $> P_{cap}$ then
14:         if currentBS == minBS then
15:             maxBS= currentBS, minBS = 1,
16:             currentBS = floor($\frac{minBS+maxBS}{2}$)
17:             BS = SB(currentBS)
18:         else
19:             maxBS = currentBS
20:             currentBS = floor($\frac{minBS+maxBS}{2}$)
21:             BS = SB(currentBS)
22:         if BS == MINBS then # MINBS = 1 in our work
23:             Start reducing the DVFS
```

Since *BatchDVFS* continues processing the input data even when it is searching for proper batch size, it does not impose any time or resource overhead on the system. However, the throughput may slightly degrade when the *BatchDVFS* is searching, or the power cap might be violated for a short period of time. The other design feature of *BatchDVFS* is the continuous batch size adjustment. Upon finding a suitable batch size it does not stop the procedure, but continues monitoring the power consumption. It restarts batch size adjustment again if it detects power cap violation or throughput improvement opportunities that might happen due to changes in the power cap or power consumption of the DNN. The power cap might be changed by an external power controller, and the power consumption of DNN can be affected by parameters such as variation in input data.

3.3 Bayesian Optimization for Near-Optimal Solution

The *BatchDVFS* approach introduced in section 3.2 is a lightweight runtime system that can adjust the batch size and DVFS in an online manner, and hence, is appropriate for making immediate decisions to avoid power cap violation when the job starts execution, or when the power cap is changed during execution of the job. However, this approach, as expected, cannot guarantee finding the optimal or near-optimal solution with regard to the throughput while meeting the power cap. It can render the performance of the job low, especially when the power cap is constant for an extended period of time, and the job is long-running.

To address this challenge, we design and implement an offline approach leveraging Bayesian Optimization (BO) that has been employed in other works as well [19], [20], [26], [27]. This approach, which we call *Bayesian Optimization* *for coordinating BS and DVFS (BOBD)*, can find the combination of batch size and DVFS that leads to an optimal or near-optimal solution, but with a significant time overhead. Hence, it cannot be accounted as a runtime approach as of *BatchDVFS*, but rather, it complements it.

The *BatchDVFS* is designed to act as a runtime approach for making decisions as quickly as possible without interrupting jobs to avoid power cap violation when a job starts execution or when the system has a dynamic power cap that fluctuates frequently. Therefore, it puts less emphasize on the optimality of the configurations it selects. This can lead to degraded performance, in particular when the power cap is constant for a long period of time for long-running jobs. *BOBD* can address this shortage of *BatchDVFS* by finding the optimal or near-optimal solution, and hence improving the performance, albeit with higher overhead. When the system is stable and it is expected to be at the same state for a long time, the *BOBD* can start searching for a solution that is better than that of *BatchDVFS*. Upon finding such a solution, the batch size and DVFS found by *BatchDVFS* can be replaced by the outcome of *BOBD*.

While overhead of *BOBD* approach is significant compared to *BatchDVFS*, it still imposes much lower overhead compared to an exhaustive approach that aims to find the optimal solution by testing all the possible combinations of batch size and DVFS. In the following, we first briefly introduce BO, and then explain the design and implementation of our *BOBD* approach.

3.3.1 Bayesian Optimization Essentials

The objective function (Throughput) and the constraint function (Power) in (1) and (2) cannot be easily modelled with respect to BS and DVFS, and thus their changes cannot be predicted as BS and DVFS change. However, we can observe them at sample points through test runs. To solve this optimization problem, we can use Bayesian Optimization (BO). The illustrative example in Fig. 5 shows how BO works. BO models the unknown functions, e.g., Throughput (BS, DVFS) and Power (BS, DVFS), with a Prior function that is a stochastic process by using the samples taken from those functions. The estimated function is updated after taking each sample, in addition to the confidence interval that shows the difference between the estimation and the real function. The sample points are selected sequentially by employing a pre-defined function called acquisition function. It identifies the sample point that can better improve the estimated function, compared with other sample points.

3.3.2 Bayesian Optimization Framework Settings

For the Prior function, the stochastic process used to estimate the unknown functions, we employ Gaussian process, a common option for BO [29]. Selecting this Prior function implies that the unknown functions we desire to estimate are a sample from Gaussian process. The true function f is estimated with a surrogate model f'. In f', the output of the function is a random variable, instead of a real value, that estimates the possible output of the function f for a certain input pair (BS and DVFS). For the first few samples, the Prior function faces a high level of uncertainty regarding the estimate of the function f. As more samples are taken, the level of uncertainty decreases, that translates to more accurate estimation. Flexibility of Gaussian process allows to end up having an accurate estimation of the function f by taking enough samples. The number of samples needed to approach to the true function depends on the similarity of that function with Gaussian process. We can employ

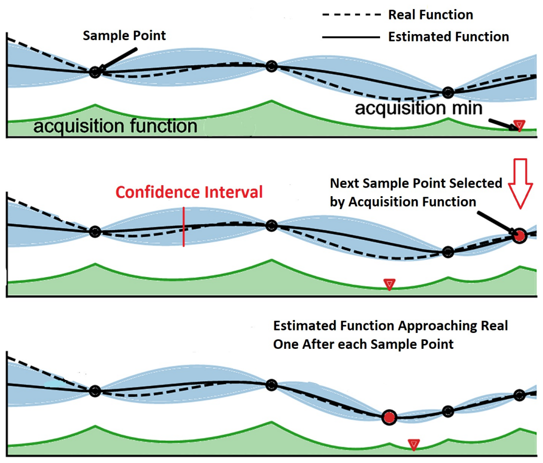
Real Function
Estimated Function
Sample Point
acquisition function
acquisition min
Confidence Interval
Next Sample Point Selected by Acquisition Function
Estimated Function Approaching Real One After each Sample Point

Fig. 5. Illustrative example to show how BO works (adapted with modification from [28]). Acquisition function finds the configuration corresponding as global extremum (minimum in this example) and determines it as the next sample point.

other Prior functions than Gaussian process for a specific DNN, but that decreases the level of the generality of the Prior function for a broader range of DNNs [26]. Moreover, Gaussian process is the only option with affordable computational demand for large scale problems.

For the acquisition function, we have three options [29]: 1) Probability of Improvement (PI) that selects the sample point that can most likely maximize the improvement likelihood over the current best result, 2) Gaussian Process Upper Confidence Bound (GP-UCB) that selects the sample point with narrowest uncertainty region for function minimization, and 3) Expected Improvement (EI) which selects the sample point that can maximize the expected improvement over the current best result. The EI method is the most popular option over the other possible options and does not require self-tuning [29]. The EI takes into account the estimated function (f') resulted by the Prior function up to current selected sample points. It also considers the best (highest in our work) value obtained for the objective function from samples tested until current sample points. Then, it examines the remaining input pairs by the estimated function f' to find the objective value for each of them. The one that can maximize the objective function improvement over the best value found until now is selected as the next input pair test sample and is evaluated to find the real value of objective function for it. Then, this new sample point and its objective value is fed to Gaussian process, along with the previous sample points and their values, to update the estimated function f'. This loop is repeated until BO reaches the maximum number of sample points.

3.3.3 BOBD Architecture

The overall architecture of *BOBD* is depicted in Fig. 6.

Sampler and Profiler. This module interacts with DNN application, GPU accelerator, and BO Engine, and is responsible for their coordination. It receives the sample point from the BO engine, and set the BS of DNN and the DVFS of GPU accordingly. During the execution of DNN application on GPU, this module gathers the info on maximum power consumption and throughput of the DNN and sends them back to the BO engine when the execution is finished, as the objective and constraint, based on the specifications that user has provided. For adjusting the DVFS of the GPU, nvidia-smi tool is used [16].

BO Engine. For Bayesian Optimization engine, we use the Spearmint [30] framework. Spearmint supports our

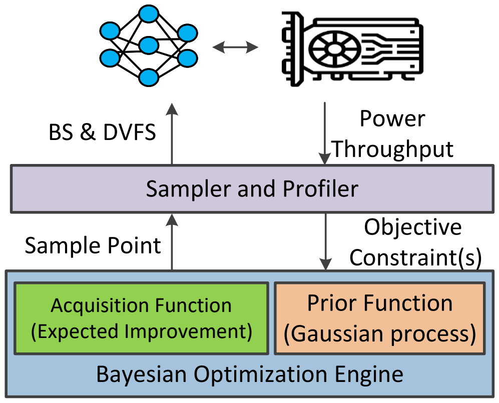

Fig. 6. Overall architecture of BOBD.

choices for the Prior function (Gaussian process) and acquisition function (EI). BO engine submits the sample points (i.e., pairs of BS and DVFS) to the Sampler and Profiler module one by one. It starts with a base pair of BS and DVFS as the first sample point (BS= 1 and DVFS = 544 MHz). After the test sample is launched, completed, and profiled by the Sampler and Profiler module, BO engine receives the resulted power and throughput. Then, it updates the estimation for unknown functions using the results of sample points by Gaussian process, and determines the next sample point using the EI function. It continues until reaching the maximum number of samples, or until no further improvement is possible, which one is sooner.

4 Evaluation

4.1 Evaluation Setup

Hardware Platform. We employ a dual-socket Xeon server equipped with two E5-2680 v4 Xeon chips each with 28 cores running at 2.4 GHz and 128 GB of DDR4 memory. Ubuntu 16.04 with kernel 4.4 is installed on the server with CUDA 10.0 and TensorFlow 1.15. A Tesla P40 GPU Accelerator is installed in the server that is based on Nvidia Pascal architecture and has 3840 CUDA cores and 24 GB GDDR5 memory, and its maximum power limit is 250 W. We employ 10 DVFS levels for GPU in our experiments: 544 MHz, 632 MHz, 734 MHz, 835 MHz, 949 MHz, 1063 MHz, 1189 MHz, 1303 MHz, 1430 MHz, and 1531 MHz. Please note we can only control the frequency of GPU. The GPU driver automatically adjusts the voltage according to the selected frequency. To understand the overhead of DVFS management, we have changed the DVFS level for 100 times and measured its average time, which is 100 ms for the GPU used in the experiments. The overhead of managing DVFS is included in the overall overhead of the system. Since the DVFS level is not changed as frequent as the batch size in our BatchDVFS approach, its overhead has a negligible impact on total overhead.

DNN Models and Datasets. We choose sixteen DNNs with different characteristics such as size and computational complexity to show the applicability of *BatchDVFS* on a wide variety of DNNs [31]. The specifications of the DNNs are presented in Table 1. We have two image datasets, one from ImageNet [24] which is a popular dataset that is widely used in other works, and the other one is Caltech-256 [25] which is collected by researchers from the California Institute of Technology. The workload used in the experiments consists of thirty jobs shown in Table 2. The power cap for each job is a number between 50 W (the minimum power when loading a model on GPU) and 250 W (the maximum power capacity of GPU).

TABLE 1
Lists of DNNs Used in the Experiments

DNN (Abbreviation)	Reference	# Parameters	Computational Complexity (Mega FLOPs)
Inception-V1 (IncV1)	[2]	6.6 M	13.22
Inception-V2 (IncV2)	[32]	11.2 M	22.33
Inception-V3 (IncV3)	[21]	23.8 M	54.25
Inception-V4 (IncV4)	[33]	42.7 M	91.94
Mobilenet-V1-1 (MobV1-1)	[23]	4.2 M	8.42
Mobilenet-V1-05 (MobV1-05)	[23]	1.3 M	2.64
Mobilenet-V1-025 (MobV1-025)	[23]	0.5 M	0.92
Mobilenet-V2-1 (MobV2-1)	[34]	3.4 M	6.94
Mobilenet-V2-14 (MobV2-14)	[34]	6.9 M	12.12
NASNET-Large (NAS-L)	[35]	88.9 M	177.11
NASNET-Mobile (NAS-M)	[35]	5.3 M	10.50
PNASNET-Large (PNAS-L)	[36]	86.1 M	171.76
PNASNET-Mobile (PNAS-M)	[36]	5.1 M	10.06
ResNet-V2-50 (ResV2-50)	[22]	25.6 M	51.00
ResNet-V2-101 (ResV2-101)	[22]	44.5 M	88.88
ResNet-V2-152 (ResV2-152)	[22]	60.2 M	120.08

TABLE 2
Specification of Jobs Used in the Experiments

	DNN	Dataset	Power Cap (W)		DNN	Dataset	Power Cap (W)
1	IncV1	ImagNet	131	16	IncV1	CalTech	125
2	IncV2	ImagNet	123	17	IncV2	CalTech	192
3	IncV3	ImagNet	228	18	IncV3	CalTech	80
4	IncV4	ImagNet	84	19	IncV4	CalTech	245
5	MobV1-1	ImagNet	145	20	MobV1-1	CalTech	98
6	MobV1-05	ImagNet	89	21	MobV1-05	CalTech	120
7	MobV1-025	ImagNet	112	22	MobV1-025	CalTech	92
8	MobV2-1	ImagNet	89	23	MobV2-1	CalTech	208
9	MobV2-14	ImagNet	164	24	MobV2-14	CalTech	152
10	NAS-L	ImagNet	83	25	NAS-M	CalTech	135
11	NAS-M	ImagNet	159	26	PNAS-L	CalTech	86
12	PNAS-M	ImagNet	246	27	PNAS-M	CalTech	136
13	ResV2-50	ImagNet	143	28	ResV2-50	CalTech	198
14	ResV2-101	ImagNet	167	29	ResV2-101	CalTech	236
15	ResV2-152	ImagNet	237	30	ResV2-152	CalTech	79

Batch Size Range. The range of batch sizes that *BatchDVFS* and Clipper can select is from 1 to 256 (range: 1 to 256). While we have used this range in the experiments, any other set with an arbitrary size can also be used. Since the time complexity of *BatchDVFS* is *O (log n)*, it can handle large sets and find a suitable batch size with the least number of tries and within a reasonable time. The upper bound of the batch size set (i.e., 256 in this work) depends on the memory capacity of the GPU accelerator. Very big batch sizes lead to out of memory (OOM) error. Therefore we used a very fast offline profiling that needs to be executed only once for a new DNN to find the upper bound of the batch size. During the offline profiling, we start from a small batch size (e.g., 16), and each time we add a constant amount to this value (e.g., 16). We only need to execute one batch with a batch size to see if the OOM error happens or not. Hence, it would be very fast. We continue until OOM error happens, and the last successful batch size is the biggest one we can have.

Systems Compared. We compare our approach against two other methods: PIT [13] which employs DVFS to manage the power consumption, and Clipper [37] which leverages the Batching:

1) PIT [13] considers changing the DVFS starting from the least amount and increasing it step by step until reaching the power cap. Since we a have limited set of DVFS levels in our setup (ten levels), we believe that the same routine can efficiently handle the DVFS management. Hence, for the

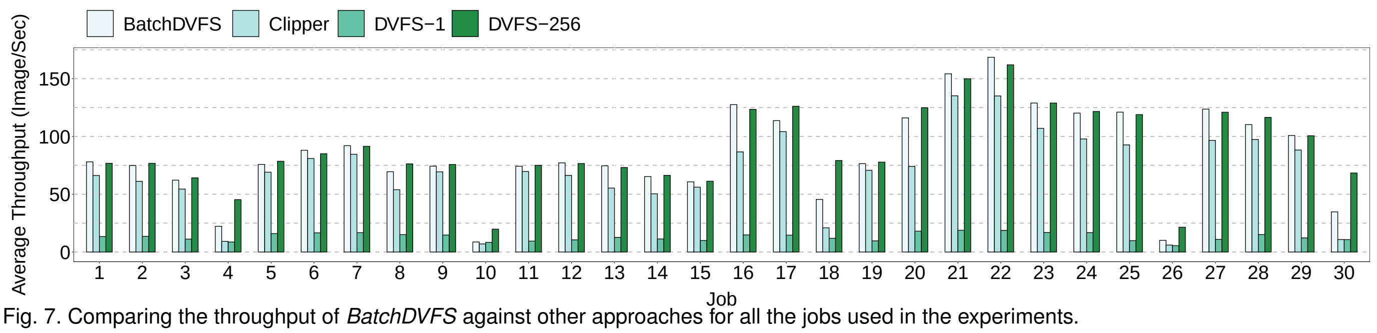

Fig. 7. Comparing the throughput of *BatchDVFS* against other approaches for all the jobs used in the experiments.

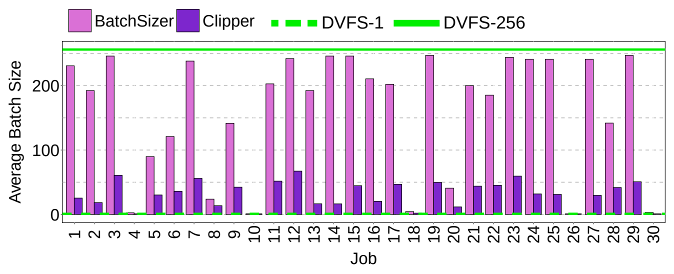

Fig. 8. The average batch size employed by *BatchDVFS* and Clipper to process each job in comparison with constant value selected by DVFS-1 and DVFS-256

sake of comparison, we implement a DVFS-based approach that uses the same routine. We implement two versions of this approach, each with a different batch size: DVFS-1 and DVFS-256. The batch sizes are selected as follows: the smallest possible batch size (one) to have control over the power consumption, and the largest batch size (256) to achieve high throughput. Similar to *BatchDVFS*, both of these approaches start their work with frequency set at the middle of the DVFS levels (1063 MHz). We do not call this approach PIT because the PIT leverages quantized version of the DNNs as well, while we do not use quantization in DVFS-1 and DVFS-256 approaches.

2) Clipper [37]: this approach originally uses the batch size to manage the latency, but we modify it such that it considers the power cap instead of latency. Clipper employs an additive-increase-multiplicative-decrease (AIMD) scheme to find the optimal batch size that maximizes the throughput, while meeting the latency SLO. We tune Clipper such that it starts from batch size one and additively increases the batch size by a fixed amount (steps of four in this work) until the power consumption exceeds the power cap. At this point, Clipper performs a small multiplicative back-off and reduces the batch size by 10%. Clipper does not employ DVFS and it is controlled by internal DVFS controller of the GPU. Similar to *BatchDVFS*, Clipper starts with batch size of one.

For the sake of fairness, in the implementation of both DVFS and Clipper, the $\alpha = 0.8$ coefficient is considered for comparing the power consumption and the power cap, similar to the *BatchDVFS*. Moreover, when the batch size is changed by Clipper or the frequency level is changed by DVFS-based approaches, waiting for three seconds to see the power spikes is considered similar to the *BatchDVFS* design.

4.2 Experimental Results

First, we present the throughput results in Fig. 7. *BatchDVFS* can improve the throughput in all the jobs compared with Clipper and DVFS-1, while having close (and in some cases lower) throughput results compared with DVFS-256. It improves the throughput by up to 2.2x and 11.2x (which is 35% and 5.3x, on average) compared with Clipper and DVFS-1, respectively. To understand the reason behind these improvements, we show the average batch size and DVFS selected by all the approaches for each job in Fig. 8 and Fig. 9, respectively. DVFS-1 has the fixed batch size of one, while the AIMD mechanism of Clipper needs some time to adjust the batch size, and consequently, none of them can efficiently leverage the throughput improvement obtained by large batch sizes. On the other hand, they try to increase the throughput by hiring higher DVFS levels, and hence, both of them have higher average frequency compared with *BatchDVFS*. But the higher DVFS level cannot compensate the absence of large batches, and consequently, they end up having lower throughput than *BatchDVFS*.

Comparing the *BatchDVFS* with DVFS-256, we see that either the throughput improvement is not that significant, or even it is significantly lower than DVFS-256 in jobs such as 4 and 10. The results shown in Fig. 8 shows that for these jobs the average batch size of *BatchDVFS* is significantly smaller than the DVFS-256 (which is 256). Therefore, it is expected to see higher throughput for DVFS-256. To understand the root of this difference between throughput of DVFS-256 and *BatchDVFS*, we present the detailed results of power, batch size, and frequency for a few jobs in Fig. 10, Fig. 11, and Fig. 12, respectively. Since the batch size of DVFS-1 and DVFS-256 is constant, we have not plotted their lines in Fig. 11.

Fig. 10 indicates that for the jobs in which DVFS-256 obtains higher throughput than *BatchDVFS* (i.e., Fig. 10(a): job 4, Fig. 10(d): job 26), it also completely violates the power cap (the same is true for jobs 10, 18, and 30, but we have not plotted the results for them due to lack of space). We see that while *BatchDVFS* only violates the power cap at the beginning of some jobs when it is adjusting the batch size, DVFS-256 completely fails to maintain the power cap and violates it during the entire execution time. For example, in Fig. 10(a), DVFS-256 detects the power cap violation, and since its batch size is constant, it tries to reduce the power consumption by adjusting DVFS to its lowest possible level, see Fig. 12 (a). However, even the lowest DVFS level cannot reduce the power consumption below power cap, and consequently, DVFS-256 suffers from serious power cap violation. On the other hand, *BatchDVFS* immediately detects the power cap violation after, and hence, reduces it to maintain the power consumption (see Fig. 11(a)). In addition to batch size, it adjusts the DVFS and decreases it by one level (unlike DVFS-256 that reduces it to the lowest level) and successfully manages the power consumption with the combination of batch size and DVFS.

Observing the behavior of *BatchDVFS* and Clipper regarding batch size reveals the impact of different batch size adjustment mechanisms on their performance. For example in Fig. 11(b), *BatchDVFS* increases the batch size faster than

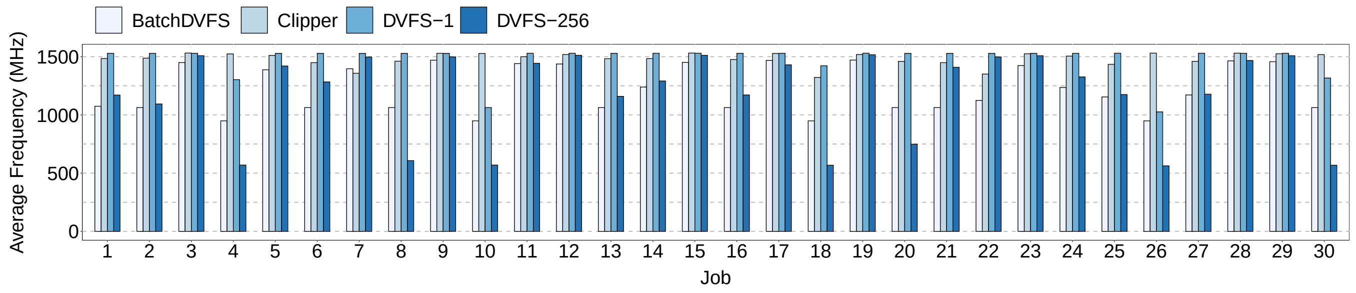

Fig. 9. Illustrating the average frequency selection of *BatchDVFS* and other approaches.

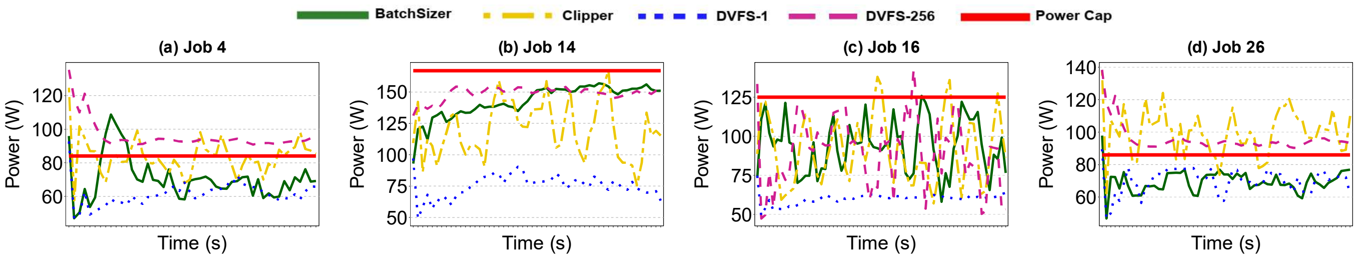

Fig. 10. Showing the power behavior of *BatchDVFS* and other approaches for several jobs.

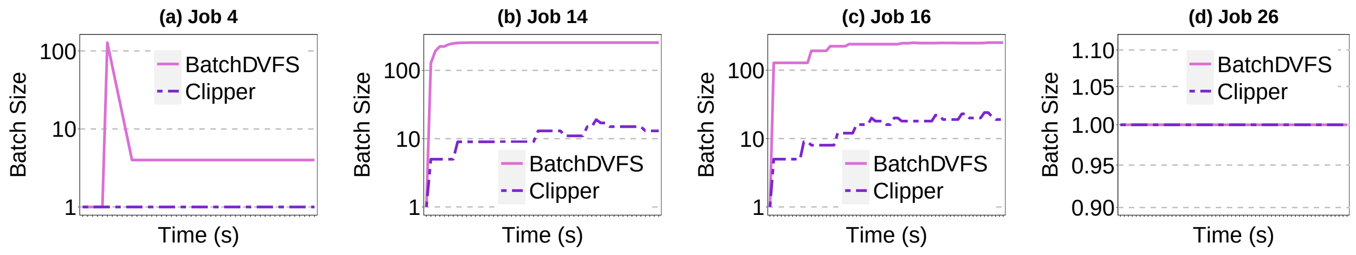

Fig. 11. Batch size management by *BatchDVFS* and Clipper to meet the power cap. These are the same jobs plotted in Fig. 10. The Y axis is shown in base-10 log scale to better distinguish the results for two approaches

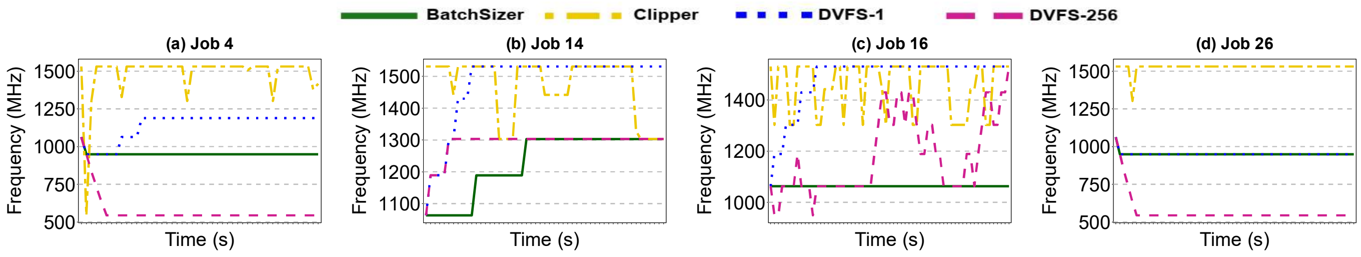

Fig. 12. DVFS adjustment by different approaches for jobs with various power cap. These are the same jobs plotted in Fig. 10.

Clipper, and hence, can better exploit the opportunity for increasing the throughput. When the power cap is tight, e.g., job 26 - Fig. 10(d), both *BatchDVFS* and Clipper employ the lowest batch size to manage the power consumption (see Fig. 11(d)). But unlike *BatchDVFS*, Clipper cannot successfully meet the power cap. The root of this failure lies in the DVFS management mechanism of these approaches. *BatchDVFS* has an adaptive DVFS controller, and therefore for job 26, Fig. 12(d), the controller decides to keep the frequency at a low level to maintain the power consumption below the power cap. However, the default controller of GPU that is used by Clipper, tends to employ high DVFS levels to increase the throughput. Therefore we see that even with batch size of one, Clipper is not able to meet the power cap and violates it frequently. Finally, we see in DVFS-1 that the low batch size helps to successfully meet the power cap. However, its extremely conservative batch size (one) severely restricts its performance, and hence, it always has very low throughput compared with all the other approaches.

From the results, we can conclude that hiring either one of the two control knobs i.e., batch size or DVFS, limits the achievable dynamic power range, and consequently, might lead to either low throughput or power cap violation. Hence, it is essential to combine them together to have a flexible and efficient online power management system.

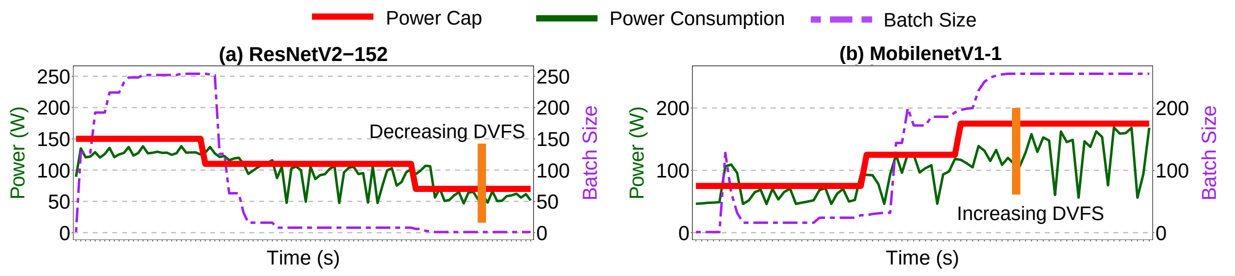

Fig. 13. Illustrating the dynamic behavior of *BatchDVFS* in the presence of changing power cap. As can be seen, *BatchDVFS* can swiftly adapt itself by the new applied power cap via dynamically changing the batch size and DVFS.

4.2.1 *Sensitivity Analysis*

The power cap applied to a system is usually subject to change over the course of time. As we mentioned in Section 3.2, *BatchDVFS* can adapt itself to changes in power cap and start adjusting the batch size again in the presence of such changes. To study this feature of *BatchDVFS*, we conduct a separate set of experiments where the power cap is dynamic. In the experiments, we have two jobs that start execution with a certain power cap, and the applied power cap changes two times during execution: 1) ResNetV2-152 (Fig. 13(a)): the power cap is initialized with 150 W, but is decreased by steps of 40 W (150 W, 110 W, 70 W). 4) MobilenetV1-1 (Fig. 13(b)): the power cap starts from 75 W, and increases by steps of 50 W up to 175 W (75 W, 125 W, 175 W). The results clearly indicate the success of *BatchDVFS* in adapting itself with the power cap. As the power cap changes, *BatchDVFS* employs smaller or larger batch sizes to manage the power consumption. When the batch size reaches its minimum or maximum value, *BatchDVFS* leverages DVFS to further increase the performance (MobilenetV1-1) or decreases the power consumption (ResNetV2-152).

4.3 Comparing BatchDVFS and BOBD

We use BOBD to find the solution for all the 30 jobs listed in Table 2. Note that since BOBD does not guarantee optimality of the output, we cannot claim that the solutions are optimal. In Fig. 14, we compare the throughput of *BatchDVFS* against BOBD for all the jobs. On average, BOBD improves the throughput by 15% compared with *BatchDVFS*, and the maximum improvement happens in job 4 by 55%. These results imply that if enough time is available for profiling the job and taking some samples from it, using Bayesian Optimization can lead to better solutions. But, for an decision online, we need a runtime system similar to *BatchDVFS*, and BOBD can not be employed in such cases. We can conclude that these approaches can complement each other and be used together to achieve highest possible throughput, while avoiding power cap violation.

Finally, to better understand how BOBD works, in Fig. 15 we depict the power and throughput of 20 sample points selected by *BOBD* for job 4. The horizontal dashed red line shows the power cap of this job. By testing the first sample point, BOBD can find a valid solution that meets the power cap. After that, it aims to find better valid solutions that can yield higher throughput. While evaluating the sample points, BOBD tries to explore the edges of state space (pairs of BS and DVFS) to understand the impact of each BS and DVFS on objective (throughput) and constraint (power cap), which leads to some invalid solutions (e.g., 2, 5, 6). Eventually, it tends to select sample points that yield power consumption close to power cap and tries to find a valid solution there that obtains high throughput.

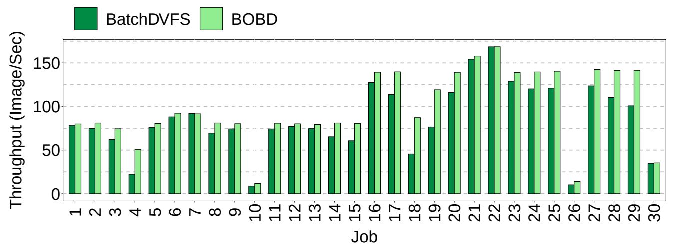

Fig. 14. Comparing the throughput of *BatchDVFS* against *BOBD*.

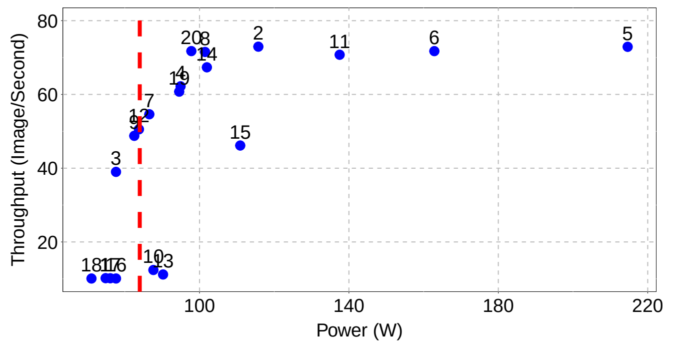

Fig. 15. The throughput and power of sample points selected by BOBD.

4.4 BatchDVFS and Co-location

In the design and implementation of *BatchDVFS*, we considered the case when only one job is running on the GPU and all the resources belong to it. Since co-location is a common practice to improve the resource utilization and energy-efficiency of GPU accelerators, we conduct preliminary experiments to understand the performance of *BatchDVFS* when several jobs are co-running together. We use four DNNs (InceptionV3, InceptionV4, ResNetV2-101, and ResNetV2-152) and a constant power cap, 120 W. First, we deploy the DNNs one by one (single execution) and apply the *BatchDVFS* to observe their throughput. Then, we deploy all of them on the GPU together and apply the *BatchDVFS* to observe their throughput again, but this time when they are co-located. The throughput result is shown in Fig. 16.

Since *BatchDVFS* is not designed for the co-location case, no mechanism is embedded in it to orchestrate the co-located jobs with each other. It only coordinates the BS and DVFS of each job individually without having any feedback or info from other jobs. The only thing considered is the power cap of the GPU. Consequently, we see that while the overall throughput of co-location (13.2 + 10.3 + 59.2 + 11.1 = 93.8) is higher than each single execution throughput, the amount of throughput reduction of each individual job, compared with its single execution throughput, is not uniform. One of the jobs (ResNetV2-101) employs large BS

(BS = 64) to increase its throughput to the level of single execution, which consumes high power. Consequently, the other ones can only use small BS (BS = 1) as there is no room for more power consumption due to power cap. So, they suffer from serious throughput reduction compared with single execution. Note that the job that employs large BS is determined randomly during the execution (e.g., based on which one starts execution sooner or decides to employ larger BS sooner), as it changes when we repeat the experiment. We can clearly see that *BatchDVFS* fails to fairly determine the BS of each co-located job, such that their throughput reduction would be proportional to their single execution throughput. Furthermore, while in our experiment the DVFS was not changed by any of the jobs, each of them is able to set it independently in its own favor.

We conclude that *BatchDVFS* needs to be extended to be able to manage the co-located jobs, efficiently. A central unit should monitor the throughput of each individual job and the impact of BS on it, as well as the overall GPU power consumption. Then, it should determine the BS of each job accordingly to maximize the overall throughput, while meeting the fairness. One possible design of the central unit can be as follows: At first, the BS of all the jobs should be increased by the same step all together (if it is determined that there is room for higher power consumption with respect to power cap). Then, the throughput of each individual job should be compared with its previous throughput. At the next step, if we decide to increase the BS (due to more room for increasing the power consumption) or decrease it (to avoid power cap violation), we can change the BS of each job proportionally with respect to its throughput sensitivity to the BS, instead of changing it for all the jobs uniformly. If the BS is decided to be increased, the job that can most benefit from larger BS regarding throughput should experience a higher BS increase than others. On the other hand, if the BS should be decreased, the BS of that job should be less decreased than others. In other words, the BS of all the jobs will be changed, and one cannot consume all the power capacity solely; however, the amount of BS change varies from job to job. In this way, while we try to maximize the overall throughput, we manage to change the BS of all the jobs, and hence, meet some degrees of fairness.

Moreover, this unit should determine the DVFS of the GPU, as it affects all the SMs, and hence, performance of all the jobs. Therefore, each job should not be able to set the DVFS level of the GPU, independently. The central unit can change the DVFS after the BS of all the jobs is set to the maximum or minimum value, and there is no room for managing the power consumption through changing the BS.

4.5 Discussion

An obstacle we have faced during conducting the experiments in the real-world setup is the profiling interval of sensor powers embedded in the GPU. The power consumption of GPU is sampled every one second, and hence, it is the narrowest interval we can have for monitoring the power consumption of the GPU when running the DNNs. This one-second sampling interval sometimes leads to missing the power consumption spikes. Therefore *BatchDVFS* (and other approaches as well) might not be able to see the impact of a change it has made in the batch size or frequency. For example, after changing the batch size from 10 to 20, it might not be able to measure how much the maximum power consumption has been increased because the power sensors where not able to catch that maximum power consumption. Consequently, *BatchDVFS* might decide to further increase the batch size, while the current value is proper and larger batch sizes can lead to power cap violation. We believe that having the sensors with higher sampling frequency can help our approach to yield even better results.

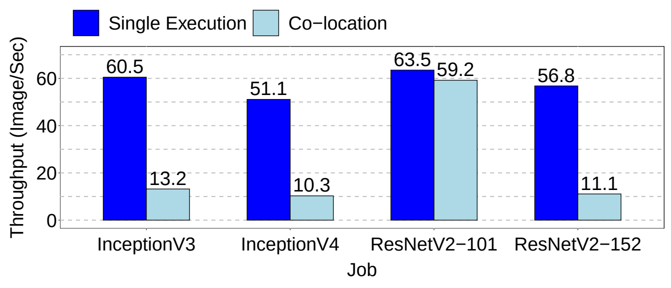

Fig. 16. Impact of co-location on performance of *BatchDVFS*

Batching can be applied in inference or training phase to achieve varying goals, such as accelerating the training process or reducing the inference time. The batch size that we consider in this work is employed in the *inference* phase of DNNs, and thus it has no effect on the accuracy of the results. This contrasts with the batching techniques employed in the *training* phase e.g., by mini-batch gradient descent approach, which may affect the accuracy of the trained model.

5 Related Work

Dynamic Voltage Frequency Scaling (DVFS) is a common technique for managing power and performance of processors such as GPUs and has been explored in a large body of research [13], [14], [15], [38], [39]. Komoda *et al.* [40] have focused on CPU-GPU systems and designed a power capping mechanism that leverages DVFS and task mapping simultaneously. Their approach seeks to adjust the frequency of CPU and GPU with respect to the task mapped to each of them. To achieve this goal, they modeled the power and performance of the system considering the task mapping and DVFS level to prevent power cap violation or load imbalance. Jiao *et al.* [14] employ the ability of modern GPUs to host several concurrent kernels to improve the power and performance. They designed and implemented several power-performance models to determine the suitable combination of kernels to be deployed on the GPU. Moreover, they adjust the frequency of cores and the memory system to improve the performance per watt of the GPU. Guerreiro *et al.* [15] employ a classification approach to categorize the GPU applications based on the impact of DVFS on their performance and obtain representative models. Leveraging the obtained models, they estimated the impact of various DVFS settings on the performance and power of new applications and tune the DVFS accordingly.

GreenMM [38] targeted the energy consumption of matrix multiplication, a prevailing operation of DNNs. They employed GPU undervolting without reducing the frequency to decrease the power consumption. To mitigate the impact of aggressive undervolting on fault rate, GreenMM introduced Algorithm-Based Fault Tolerance (ABFT). Tang *et al.* [39] conducted an extensive set of experiments to understand the impact of DVFS on performance and energy consumption of several DNNs. To this end, they executed four DNNs on three GPU accelerators. PIT [13] is another approach that employed DVFS along with reduced-precision instructions supported by new GPUs to manage the power consumption of DNN inference on GPU accelerators. It first deploys the reduced-precision model of DNN on the GPU, and if needed, it adjusts the GPU frequency with its dedicated procedure. That procedure starts from the lowest DVFS level and increases it as much as possible to improve the performance, while meeting the power cap. None of these approaches employ an adaptive batching mechanism to leverage its benefits for power management and throughput maximization.

Studying the impact of batching on DNN inference and employing it to increase the throughput has attracted researchers from both academia and industry [17], [18], [41], [42], [43], [44], [45]. Some studies indicate the benefits of batching for throughput and energy consumption of DNN inference on GPUs [41], [46], but it also elongates the latency of DNN as well. Pervasive CNN (P-CNN) [44] leveraged large batch sizes for throughput-oriented tasks to maximize their throughput while reach energy-efficiency for GPU. To select the proper batch size for such tasks, P-CNN takes into account the GPU memory and makes sure that the batch size is not excessively large to encounter an out-of-memory error. On the other hand, P-CNN selects small batch sizes for latency-critical tasks to avoid elongated response time. Clipper, a system for online ML services, [37] formed batches from concurrent streams of prediction inputs for processing to take advantage of the benefits of batching. Its additive-increase-multiplicative-decrease (AIMD) mechanism sets the batch size adaptively to find the proper size that maximizes the throughput, while assuring the latency constraint is met. These approaches have mostly focused on inference latency and usually do not consider the power cap of the hardware platform, and consequently, they do not employ the DVFS technique. *BatchDVFS* addresses the aforementioned shortcomings of these approaches.

Employing pruning and quantization to improve the power/performance of DNNs on hardware accelerators is widely studied in previous works. Various weight pruning [47], [48], [49], node pruning [50], [51], and filter pruning [52], [53] approaches are among the ones proposed to reduce the redundant connections in a DNN, and consequently, reduce the computation demand. Another widely explored research direction for improving the power and performance of DNN inference is quantizing the DNN parameters to lower numerical precision (e.g., 16-bit floating point or 8-bit integer), and thus leveraging the reduced-precision instructions supported by hardware accelerators. [13], [54], [55], [56], [57]. The pruned and quantized DNNs still can benefit from our approach to further improve the performance of DNN inference on hardware accelerators.

6 Conclusion

In this work, we explored the possibility of using batch size as a new control knob for managing the power consumption of GPU-based DNN accelerators. The batch size can control the resource utilization of GPU, and consequently, its power consumption. Combining this new control knob and DVFS technique, we designed a runtime system called *BatchDVFS* that aims to maximize the throughput while meeting the power cap. The results show that it can outperform the Clipper approach that solely relies on batch size, and DVFS-1 and DVFS-256 approaches that employ a constant batch size and manage the power by adjusting the GPU's frequency. We also designed another approach called *BOBD* that aims to find better solutions than *BatchDVFS*, but with significant time overhead. The batch size is orthogonal to previous power management and performance improvement techniques for DNNs such as quantization and pruning, and hence, can be used in conjunction with them to yield more promising results.

Acknowledgment

This paper is an extension of our prior work published in [58]. M. Ebrahimi is partially supported by grants STINT (MG2018-8007) and VR (2016-05140). S. Reda is partially supported by NSF grant 1814920 and DoD ARO grant W911NF-19-1-0484.

References

[1] T. N. Sainath, A.-r. Mohamed, B. Kingsbury, and B. Ramabhadran, "Deep convolutional neural networks for lvcsr," in *IEEE ICASSP*, 2013, pp. 8614–8618.

[2] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *CVPR*, 2015, pp. 1–9.

[3] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural info. processing sys.*, 2014, pp. 3104–3112.

[4] M. Hosseini and T. Mohsenin, "Binary precision neural network manycore accelerator," *ACM Journal on Emerging Technologies in Computing Systems*, 2020.

[5] N. Wu, L. Deng, G. Li, and Y. Xie, "Core placement optimization for multi-chip many-core neural network systems with reinforcement learning," *ACM Transactions on Design Automation of Electronic Systems*, vol. 26, no. 2, pp. 1–27, 2020.

[6] H. Shen, L. Chen, Y. Jin, L. Zhao, B. Kong, M. Philipose, A. Krishnamurthy, and R. Sundaram, "Nexus: a gpu cluster engine for accelerating dnn-based video analysis," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, 2019, pp. 322–337.

[7] P. Hill, A. Jain, M. Hill, B. Zamirai, C.-H. Hsu, M. A. Laurenzano, S. Mahlke, L. Tang, and J. Mars, "Deftnn: Addressing bottlenecks for dnn execution on gpus via synapse vector elimination and near-compute data fission," in *MICRO'17*, 2017, pp. 786–799.

[8] Y. Guan, H. Liang, N. Xu, W. Wang, S. Shi, X. Chen, G. Sun, W. Zhang, and J. Cong, "Fp-dnn: An automated framework for mapping deep neural networks onto fpgas with rtl-hls hybrid templates," in *FCCM'17*. IEEE, 2017, pp. 152–159.

[9] W. Jiang, E. H.-M. Sha, X. Zhang, L. Yang, Q. Zhuge, Y. Shi, and J. Hu, "Achieving super-linear speedup across multi-fpga for real-time dnn inference," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 18, no. 5s, pp. 1–23, 2019.

[10] E. Qin, A. Samajdar, H. Kwon, V. Nadella, S. Srinivasan, D. Das, B. Kaul, and T. Krishna, "Sigma: A sparse and irregular gemm accelerator with flexible interconnects for dnn training," in *HPCA'20*. IEEE, 2020, pp. 58–70.

[11] N. Samimi, M. Kamal, A. Afzali-Kusha, and M. Pedram, "Res-dnn: A residue number system-based dnn accelerator unit," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 2, pp. 658–671, 2019.

[12] R. Azimi, C. Jing, and S. Reda, "Powercoord: A coordinated power capping controller for multi-cpu/gpu servers," in *International Green and Sustainable Computing Conference (IGSC)*, 2018, pp. 1–9.

[13] S. M. Nabavinejad, H. Hafez-Kolahi, and S. Reda, "Coordinated dvfs and precision control for deep neural networks," *IEEE Computer Architecture Letters*, vol. 18, no. 2, pp. 136–140, 2019.

[14] Q. Jiao, M. Lu, H. P. Huynh, and T. Mitra, "Improving gpgpu energy-efficiency through concurrent kernel execution and dvfs," in *CGO'15*. IEEE, 2015, pp. 1–11.

[15] J. Guerreiro, A. Ilic, N. Roma, and P. Tomás, "Dvfs-aware application classification to improve gpgpus energy efficiency," *Parallel Computing*, vol. 83, pp. 93–117, 2019.

[16] "Nvidia system management interface," https://developer.download.nvidia.com/compute/DCGM/docs/nvidia-smi-367.38.pdf, accessed: November 27, 2021.

[17] Y. Shen, M. Ferdman, and P. Milder, "Escher: A cnn accelerator with flexible buffering to minimize off-chip transfer," in *FCCM'17*. IEEE, 2017, pp. 93–100.

[18] S. Zhang, W. Li, C. Wang, Z. Tari, and A. Y. Zomaya, "Dybatch: Efficient batching and fair scheduling for deep learning inference on time-sharing devices," in *CCGRID'20*. IEEE, 2020, pp. 609–618.

[19] T. Patel and D. Tiwari, "Clite: Efficient and qos-aware co-location of multiple latency-critical jobs for warehouse scale computers," in *HPCA*, 2020, pp. 193–206.

[20] Q. Li, B. Li, P. Mercati, R. Illikkal, C. Tai, M. Kishinevsky, and C. Kozyrakis, "Rambo: Resource allocation for microservices using bayesian optimization," *IEEE CAL*, vol. 20, no. 1, pp. 46–49, 2021.

[21] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *CVPR'16*, 2016, pp. 2818–2826.

[22] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *ECCV'18*. Springer, 2016, pp. 630–645.

[23] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” *arXiv preprint arXiv:1704.04861*, 2017.

[24] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, “Imagenet large scale visual recognition challenge,” *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[25] G. Griffin, A. Holub, and P. Perona, “The caltech-256: Caltech technical report,” *vol*, vol. 7694, p. 3, 2007.

[26] O. Alipourfard, H. H. Liu, J. Chen, S. Venkataraman, M. Yu, and M. Zhang, “Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics,” in *NSDI*, 2017, pp. 469–482.

[27] S. M. Nabavinejad and S. Reda, “Bayestuner: Leveraging bayesian optimization for dnn inference configuration selection,” *IEEE Computer Architecture Letters*, 2021.

[28] L. L. Grado, M. D. Johnson, and T. I. Netoff, “Bayesian adaptive dual control of deep brain stimulation in a computational model of parkinson’s disease,” *PLoS computational biology*, vol. 14, no. 12, 2018.

[29] J. Snoek, H. Larochelle, and R. P. Adams, “Practical bayesian optimization of machine learning algorithms,” *Advances in neural information processing systems*, vol. 25, 2012.

[30] “Spearmint,” https://github.com/HIPS/Spearmint, accessed: November 27, 2021.

[31] N. Silberman and S. Guadarrama. (2016) Tensorflow-slim image classification model library. [Online]. Available: https://github.com/tensorflow/models/tree/master/research/slim

[32] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *arXiv preprint arXiv:1502.03167*, 2015.

[33] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, “Inception-v4, inception-resnet and the impact of residual connections on learning,” in *AAAI Conference on Artificial Intelligence*, 2017.

[34] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “Mobilenetv2: Inverted residuals and linear bottlenecks,” in *CVPR’18*, 2018, pp. 4510–4520.

[35] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, “Learning transferable architectures for scalable image recognition,” in *CVPR’18*, 2018, pp. 8697–8710.

[36] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, “Progressive neural architecture search,” in *ECCV’18*, 2018, pp. 19–34.

[37] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica, “Clipper: A low-latency online prediction serving system,” in *NSDI’17*, 2017, pp. 613–627.

[38] H. Zamani *et al.*, “Greenmm: energy efficient gpu matrix multiplication through undervolting,” in *ICS*, 2019, pp. 308–318.

[39] Z. Tang *et al.*, “The impact of gpu dvfs on the energy and performance of deep learning: An empirical study,” in *e-Energy ’19*, 2019, pp. 315–325.

[40] T. Komoda, S. Hayashi, T. Nakada, S. Miwa, and H. Nakamura, “Power capping of cpu-gpu heterogeneous systems through coordinating dvfs and task mapping,” in *ICCD’13*, 2013, pp. 349–356.

[41] Z. Fang, T. Yu, O. J. Mengshoel, and R. K. Gupta, “Qos-aware scheduling of heterogeneous servers for inference in deep neural networks,” in *ACM CIKM’17*, 2017, pp. 2067–2070.

[42] Y. Chen, J. Emer, and V. Sze, “Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks,” in *ISCA’16*, 2016, pp. 367–379.

[43] X. Tang, P. Wang, Q. Liu, W. Wang, and J. Han, “Nanily: A qos-aware scheduling for dnn inference workload in clouds,” in *HPCC/SmartCity/DSS’19*. IEEE, 2019, pp. 2395–2402.

[44] M. Song, Y. Hu, H. Chen, and T. Li, “Towards pervasive and user satisfactory cnn across gpu microarchitectures,” in *HPCA’17*. IEEE, 2017, pp. 1–12.

[45] A. Ali, R. Pinciroli, F. Yan, and E. Smirni, “Batch: machine learning inference serving on serverless platforms with adaptive batching,” in *SC’20*. IEEE Computer Society, 2020, pp. 972–986.

[46] Y. Inoue, “Queueing analysis of gpu-based inference servers with dynamic batching: A closed-form characterization,” *Performance Evaluation*, p. 102183, 2020.

[47] Y. Guo, A. Yao, and Y. Chen, “Dynamic network surgery for efficient dnns,” *arXiv preprint arXiv:1608.04493*, 2016.

[48] M. A. Rumi, X. Ma, Y. Wang, and P. Jiang, “Accelerating sparse cnn inference on gpus with performance-aware weight pruning,” in *PACT’20*, 2020, pp. 267–278.

[49] W. Niu, X. Ma, S. Lin, S. Wang, X. Qian, X. Lin, Y. Wang, and B. Ren, “Patdnn: Achieving real-time dnn execution on mobile devices with pattern-based weight pruning,” in *ASPLOS’20*, 2020, pp. 907–922.

[50] T. He, Y. Fan, Y. Qian, T. Tan, and K. Yu, “Reshaping deep neural network for fast decoding by node-pruning,” in *IEEE ICASSP’14*, 2014, pp. 245–249.

[51] A. Ashiquzzaman, L. Van Ma, S. Kim, D. Lee, T.-W. Um, and J. Kim, “Compacting deep neural networks for light weight iot & scada based applications with node pruning,” in *Artificial Intelligence in Information and Communication*, 2019, pp. 082–085.

[52] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, “Pruning filters for efficient convnets,” *arXiv preprint arXiv:1608.08710*, 2016.

[53] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, “Pruning convolutional neural networks for resource efficient inference,” *arXiv preprint arXiv:1611.06440*, 2016.

[54] G. Yuan, X. Ma, C. Ding, S. Lin, T. Zhang, Z. S. Jalali, Y. Zhao, L. Jiang, S. Soundarajan, and Y. Wang, “An ultra-efficient memristor-based dnn framework with structured weight pruning and quantization using admm,” in *ISLPED’19*, 2019, pp. 1–6.

[55] S. M. Nabavinejad, L. Mashayekhy, and S. Reda, “Approxdnn: Incentivizing dnn approximation in cloud,” in *CCGRID’20*. IEEE, 2020, pp. 639–648.

[56] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, “Haq: Hardware-aware automated quantization with mixed precision,” in *CVPR’19*, 2019, pp. 8612–8620.

[57] M. Imani, M. Samragh, Y. Kim, S. Gupta, F. Koushanfar, and T. Rosing, “Rapidnn: In-memory deep neural network acceleration framework,” *arXiv preprint arXiv:1806.05794*, 2018.

[58] S. M. Nabavinejad, S. Reda, and M. Ebrahimi, “Batchsizer: Power-performance trade-off for dnn inference,” in *Proceedings of the Asia and South Pacific Design Automation Conference*, 2021, pp. 819–824.

Seyed Morteza Nabavinejad is a Post-Doctoral Research Fellow at the School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran. He received the B.Sc. degree in Computer Engineering from Ferdowsi University of Mashhad and the M.Sc., and Ph.D. degrees from Sharif University of Technology in 2011, 2013, and 2018, respectively. He has published several peer reviewed papers in venues such as *IEEE Transactions on Cloud Computing*, *IEEE Transactions on Big Data*, *DATE*, and *SAC*. His research interests include big data processing, cloud computing, green computing, approximate computing, and energy aware data centers

Sherief Reda received the Ph.D. degree in Computer Science and Engineering from the University of California, San Diego in 2006. He is currently a Full Professor with the School of Engineering, Brown University, Providence, RI. His research interests include energy-efficient computing, thermal-power sensing and management, low-power design techniques, and design automation. He has over 120 publications in peer-reviewed conferences and journals with several of them receiving best paper nominations and awards. He serves as an associate editor for IEEE Transactions on Computer-Aided Design. He is a senior member of IEEE.

Masoumeh (Azin) Ebrahimi received a Ph.D. degree with honors from University of Turku, Finland in 2013 and MBA jointly from the University of Turku and EIT-ICT School in 2015. She is currently an Associate Professor at KTH Royal Institute of Technology, Sweden and an Adjunct Professor at the University of Turku, Finland. She has led several national and international projects such as EU-MarieCurie-Vinnova, Academy of Finland, and Vetenskapsrådet (VR), STINT, and SSF. Her scientific work contains more than 100 publications, including journal articles, conference papers, book chapters, edited proceedings, and edited special issue of journal. She actively acts as a guest editor, organizer, and program chair in different venues and conferences. Her main areas of interest include interconnection networks and neural network accelerators. She is a member of HiPEAC and senior member of IEEE.