A Workload-Aware DVFS Robust to Concurrent Tasks for Mobile Devices

Chengdong Lin[1,2], Kun Wang[1], Zhenjiang Li[1], Yu Pu[2]
[1]Department of Computer Science, City University of Hong Kong, China; [2]Alibaba DAMO Academy, China

ABSTRACT

Power governing is a critical component of modern mobile devices, reducing heat generation and extending device battery life. A popular technology of power governing is dynamic voltage and frequency scaling (DVFS), which adjusts the operating frequency of a processor to balance its performance and energy consumption. With the emergence of diverse workloads on mobile devices, traditional DVFS methods that do not consider workload characteristics become suboptimal. Recent application-oriented methods propose dedicated and effective DVFS governors for individual application tasks. Since their approach is only tailored to the targeted task, performance drops significantly when other tasks run concurrently, which is however common on today's mobile devices. In this paper, our key insight is that hardware meta-data, widely used in existing DVFS designs, has great potential to enable capable workload awareness and task concurrency adaptability for DVFS, but they are underexplored. We find that workload characteristics can be described in a hyperspace composed of multiple dimensions derived from these meta-data to form a novel workload contextual indicator to profile task dynamics and concurrency. On this basis, we propose a meta-state metric to capture this relationship and design a new solution, GearDVFS. We evaluate it for a rich set of application tasks, and it outperforms state-of-the-art methods.

CCS CONCEPTS

• **Human-centered computing** → **Mobile devices**; • **Software and its engineering** → **Power management**.

KEYWORDS

Mobile Devices; DVFS; Power Management

ACM MobiCom '23, October 2–6, 2023, Madrid, Spain

ACM ISBN 978-1-4503-9990-6/23/10…$15.00
https://doi.org/10.1145/3570361.3592524

ACM Reference Format:
Chengdong Lin[1,2], Kun Wang[1], Zhenjiang Li[1], Yu Pu[2]. 2023. A Workload-Aware DVFS Robust to Concurrent Tasks for Mobile Devices. In *The 29th Annual International Conference on Mobile Computing and Networking (ACM MobiCom '23), October 2–6, 2023, Madrid, Spain.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3570361.3592524

1 INTRODUCTION

With the rapid development of mobile system-on-chip (SoC), high-performance tasks such as deep learning, gaming and video processing can be run on a wide range of mobile devices, *e.g.*, smartphones, the NVIDIA Jetson series [7, 34], Odroid [8], etc. The small size of mobile devices leads to their compact architectural designs [5]. It requires efficient power governing to control the heat generation of the device. Otherwise the entire device life cycle will be greatly shortened [3]. More importantly, mobile SoCs are often battery-powered, such as in smartphones [82] or autonomous machines [6]. Power governing is also essential to ensure long battery life.

Dynamic voltage and frequency scaling (DVFS) is a popular power governing technology for mobile devices [23, 31, 38]. It adjusts the *frequency* (and voltage, but they are related due to the operating curve §2.1) of processors, such as CPUs and GPUs, at runtime to ensure that the energy those processors consume matches their workloads, thereby preventing wasted energy and reducing heat generation. Without DVFS, device battery life can be reduced by up to 75% [63].

However, due to the increasing diversity of workloads from application tasks, traditional DVFS solutions (*governors*), like *schedutil* from the Linux kernel [2], become suboptimal, as they do not take into account the workload's characteristics [41]. Thus, recent studies propose a series of *application-oriented* methods, by designing dedicated governors for various applications [21, 23, 41, 51]. The rationale is that when an application task is considered, learning and predicting its workload are more tractable, thereby optimizing the processor frequency. These methods are successful when given the workload "context" of the application, but are not ready for widespread use in practice due to the following limitation.

Their energy management is tailored only to the targeted application, making them vulnerable to application task concurrency. Efficacy drops significantly when other (especially heavy) tasks run concurrently (as extra resource consumers), but this is common in modern mobiles [53], *e.g.*, a foreground

APP with multiple background tasks in smartphones [19], simultaneous sensing and recognition tasks in augmented reality [37], and concurrent computer vision, video capture and media streaming tasks in autonomous vehicles/robots [10].

The reason for this shortcoming is that DVFS aims to allocate *judicious* energy (without wasting it) to processors to provide enough computing power and accommodate device workloads [35]. The hurdle is the complex relationship between dynamic workloads and matchable computing power. Application-oriented methods focus on relatively stable or fixed application contexts, without special consideration for task concurrency and context switching. When only one task is considered, this relationship can be bypassed by simply checking whether task-side quality of service (QoE), such as the required frame rate or latency, is achieved. However, this approach does not generalize to determine computing power in the presence of concurrent tasks, since tasks may have different QoEs. Using any QoE alone is hardly to ensure good energy efficiency in a device. In addition, even if they have the same QoE, the requirements for values are often different, which are difficult to be unified.

In the DVFS literature [35], it is known that by keeping processor **utilization** at a judicious level (*e.g.*, 80%) [27], sufficient computing power can be provided without wasting energy [33]. Utilization, as a hardware-level indicator of the processor's working state, has been used in traditional application-agnostic DVFS [2, 72, 75]. We find that by exploiting utilization, there is great potential to further achieve a DVFS design that is not only general but also *workload-aware* and *task-concurrency-tolerant* by addressing two challenges.

First, for the CPU, its utilization consists of two sub-states, *active computing* and *stalled waiting* (such as memory access) [36]. Their percentages can vary widely under different workload characteristics. However, when DVFS adjusts the CPU frequency, these two sub-states have different effects on CPU utilization and CPU energy consumption, which leads to the first problem — under various workloads with different active/stalled percentages, frequency needs to be adjusted differently to achieve a target utilization (*e.g.*, 80%), even starting from the same initial utilization value. This uncertainty is further complicated by the CPU task scheduling [74].

Second, for other accelerated processors like GPUs, the stalled-waiting sub-state has little effect, but another problem arises: when it comes to heavier workloads [17, 54, 76], such as video rendering and deep learning tasks, more computing from the GPU may be required. In this case, it is easy to be trapped in a suboptimal frequency setting for CPU and GPU when their utilization values are governed *independently*, which instead should be considered jointly. This second challenge is also observed recently [41].

In this paper, our key insight is that rich metadata of processors can be obtained during their executions. We can leverage collective views of these metadata, and the device-wide workload characteristics can be described in the *hyper-space* of multiple dimensions derived from them. This hyper-space description is then used as a contextual indicator for workload characteristics to profile task dynamics and concurrency, based on which DVFS can maintain the utilization of processors at a judicious level by choosing appropriate frequencies for effective energy management. Thus, we introduce a *general* and *workload-aware* DVFS, GearDVFS, as follows.

(1) We propose a new metric called *meta-state*, derived from the hyper-space composed of a set of utilization statistics (related to CPU and GPU) and some other hardware metadata. We introduce a novel method to extract meta-states by carefully fusing these dimensions and mining their temporal relationship to make workload-awareness possible. (2) We formulate a novel workload-aware DVFS design problem that is general and compatible with varying degrees of task concurrency. We propose a *model-free* solution by leveraging reinforcement learning (RL). Different from existing RL-based methods [41, 47], we address two unique issues by 1) incorporating meta-states into RL, and 2) addressing large action spaces for fine-grained frequency adjustments.

Experiment. We develop a GearDVFS prototype and evaluate it on NVIDIA Jetson NX, Nano, Odroid-XU3, Raspberry Pi 4B and Redmi phone for a variety of popular mobile applications, including both deep learning tasks and commercial APPs with diverse workload characteristics and concurrency. Extensive experiments show that GearDVFS outperforms the latest governor used on existing mobile devices and the state-of-the-art zTT [41], achieving up to 23.9% and 26.9% energy efficiency improvements, respectively. In addition, to show a broader utility of GearDVFS, we further test it for less task-concurrency scenarios and find that it can still bring up to 16.92% improvement. Finally, GearDVFS is lightweight and DVFS can be governed periodically about every 100 ms. GearDVFS is open source at http://geardvfs.github.io. In summary, this paper makes the following contributions:

- We propose a novel and general workload-aware DVFS to accommodate today's mobile tasks of strong dynamics and concurrency, which is also backwards compatible with traditional scenarios of less task concurrency.
- We propose a new meta-state metric, formulate the DVFS design, and introduce a novel solution by incorporating meta-states and solving its action-space issue.
- We develop a prototype and show remarkable performance gains compared to the latest methods from commercial devices and the state-of-the-art DVFS design.

2 BACKGROUND AND MOTIVATION

DVFS is a crucial technology for power governing to conserve energy and reduce heat generation of processors on mobiles.

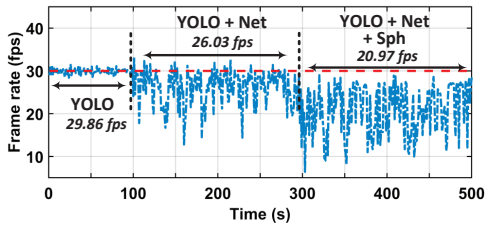

Figure 1: Frame rates achieved by zTT [41] for YOLO v3 under different workloads with the target 30 fps QoE.

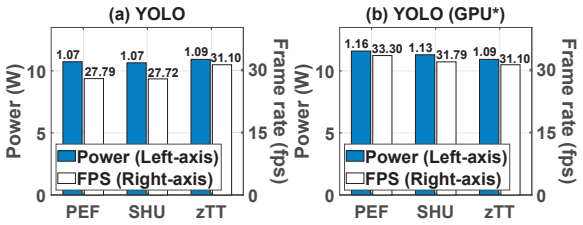

Figure 2: Average power consumption and frame rate achieved by different DVFS governors for YOLO v3, wherein *performance* (PEF) and *schedutil* (SHU) are (a) without and (b) with the optimized GPU frequency.

2.1 Background

Principle of DVFS. For any processor i (such as CPU and GPU), its energy consumption $E_i \propto f_i \times V_i^2$, where f_i and V_i are the working frequency and voltage, respectively. Moreover, the working voltage V is further proportional to f [70]. Thus, $E_i \propto f_i^3$ and DVFS only needs to adjust frequency f within an adjustable range specified in the hardware manual, according to the observation of the device execution status.

Mobile processors. In mobile devices, the CPU is the main processor for processing tasks, due to its mature programming environment, general availability and robust operational support [59, 74]. For emerging deep learning tasks, mobile GPUs can facilitate processing, which provide as much performance as CPUs [81], but cannot take full responsibility, as many operations and models are not supported by mobile GPUs [59, 79], which must fall back to CPU to run [37]. Thus, recent mobile DVFS designs [41, 51, 60] focus on the energy management of CPU and GPU, including GearDVFS.

Concurrent tasks on other platforms. On other computing platforms, such as data centers, they also handle concurrent tasks and have their own DVFS. However, since such platforms are designed to provide computing services to a large number of clients simultaneously, the volume of their concurrent tasks is huge. Therefore, their workloads are primarily characterized by more macro metrics, such as the distribution of incoming tasks, and the granularity of this distribution varying typically ranges from minutes to hours [29, 46]. Although recent mobile devices also have concurrent tasks, the task concurrency is much lower than that of these platforms. Therefore, the fine-grained workload variation of each task is important in mobile DVFS designs, which need to capture run-time workload dynamics of these tasks and the granularity is usually sub-second on mobile devices [31, 41]. Hence, their DVFS solutions are unsuitable for mobile devices.

2.2 Inefficiency of Current DVFS Designs

Application-oriented DVFS methods. As introduced before, they propose to associate a dedicated DVFS governor for one target application, which is pre-trained by considering the application workload characteristics and quality of experience (QoE), such as frame rate and latency. To understand their inefficiency in practice, we conduct a preliminary experiment with the state-of-the-art application-oriented method zTT [41]. We reuse the source code [12] released by the authors of zTT and examine the object detection task YOLO v3 [64] that was studied in the zTT paper. In this experiment, we also train zTT on a Jetson device, similar to the zTT paper [41], with the frame rate of 30 fps as the target QoE, which we hope to maintain consistently. If frame rate fluctuates wildly, the video displayed on the device becomes unstable.

Initially, when only YOLO is running, zTT can stabilize the frame rate around 30 fps (the target QoE) very well, such as 29.86 fps on average in Figure 1. Then, we consider a mobile robot scenario to initiate video clip transmission. Because these two tasks have different QoEs, their governors cannot be unified and we adopt the YOLO governor since YOLO is performed more frequently. We find that the achieved frame rate becomes less stable and the average QoE drops to 26.03 fps. Then, the robot goes further with speech recognition to interact with people detected nearby. The frame rate fluctuates significantly, dropping to an average of 20.97 fps.

Tasks may have different QoEs, such as frame rate for YOLO, transmission latency for networking, and execution latency for speech recognition. Thus, various DVFS governors of this kind may not be unified, and any individual one can hardly accommodate the overall workload dynamics well. In addition, as unveiled in the evaluation later, even if all tasks have the same QoE, their performance may still be limited.

Application-agnostic DVFS methods. This type of methods mainly govern CPU processors and are not coupled to any particular application, such as *performance* and *schedutil*. Their goal is to ensure that CPU works close to a target utilization [2, 67], like 80%. There are also independent governors for GPU, like *simple_ondemand*, which has a similar working principle. Although these methods are general and compatible to various workloads, their performance is limited most of the time [41]. We now understand their limitations empirically, which delivers useful insights to motivate our design.

Computing-intensive workloads. Deep learning usually leads to such workloads, involving both CPU and GPU to handle. We now use YOLO to examine two popular governors

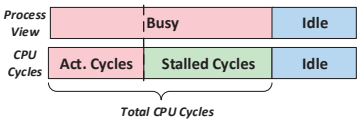

Figure 3: Active computing and stalled waiting.

performance (PEF) and *schedutil* (SHU) for such workloads. Figure 2(a) shows that PEF and SHU lead to similar power consumption compared to zTT, while their achieved frame rates are much less than 30 fps, that is, they have low QoEs.

By checking the experiment log, we find that CPU has been set to the highest frequency by PEF and SHU, while more GPU computing should be involved in this task. However, GPU is governed by *simple_ondemand* independently, which fails to achieve the cooperation by reducing CPU frequency, meanwhile increasing the GPU frequency. Therefore, we freeze the GPU governor *simple_ondemand* with an optimal GPU frequency (manually searched) for YOLO, and run PEF and SHU again. Figure 2(b) shows that their frame rates can be improved significantly, comparable to zTT. This result suggests that CPU and GPU should be governed jointly.

Memory intensive and balanced workloads. This type of workloads are also popular and mainly handled by the CPU. In such workloads, even when the CPU is busy, the device status could be very different. The busy state of the processor includes *active computing* and *stalled waiting*, which are two different sub-states, as depicted in Figure 3. As these two sub-states have completely different effects on CPU utilization and energy consumption when CPU frequency is adjusted, they pose further challenges to the DVFS design.

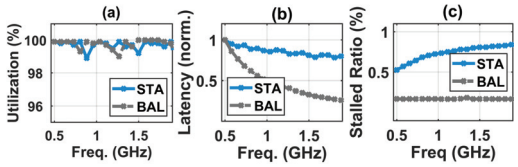

Figure 4: (a) CPU utilization, (b) normalized latency and (c) ratio of the sub-state of stalled waiting, when two different types of tasks run at different CPU frequencies.

To understand this point, we run two tasks from a popular workload benchmark dataset "SysBench" [44], where "STA" is a memory intensive task that performs frequent memory reads and writes, and "BAL" is a balanced task involving computation and memory access. We vary the CPU frequency from 500 MHz to 1.9 GHz. If we only look at the CPU utilization (as many traditional DVFS methods), Figure 4(a) shows that these two tasks exhibit similar utilization levels, which cannot be distinguished well. However, if we examine their QoEs, which are measured by the latency for these two tasks, we can see that the change of the CPU frequency has different effects on their latency — with more stalled waiting states (STA), its latency reduction (*i.e.*, QoE gain) by increasing the CPU frequency is much less than the balanced workload (BAL) in Figure 4(b). To distinguish these two tasks, we observe that if we look at the breakdown of CPU utilization, such as by plotting the ratio of stalled waiting over the busy period in Figure 4(c), these two workloads are highly distinguishable. In fact, with more dimensions of useful hardware metadata processed by an effective design, workload characteristics can be captured better (§3).

Summary. Our empirical study reveals the limitations of the existing DVFS designs. Through this study, we also find that the available metadata from the device hardware actually have great potential to overcome these limitations.

3 SYSTEM DESIGN

To be general to various workloads, GearDVFS is also designed to maintain processor utilization close to a judicious target level, while we need new techniques to achieve that.

3.1 Problem Formulation

We first formulate the workload-aware DVFS design problem in general, which provides crucial insights to inspire our method. Within a time window T_{win}, a processor utilization u can be represented as $u = \frac{T_{busy}}{T_{win}}$, where T_{busy} is the busy period of the processor in T_{win}. As discussed in §2.2, the busy period of a processor includes *active computing* (T_{act}) and *stalled waiting* (T_{sta}) two different states. Therefore, by setting $T_{win} = 1$, we can express u ($= \frac{T_{busy}}{T_{win}}$) as:

$$u = \frac{T_{busy}}{T_{win}} = \frac{T_{act} + T_{sta}}{1} = \beta \times \frac{W_{act}}{f} + T_{sta}, \qquad (1)$$

where T_{act} is *proportional* to the computation workload W_{act} and *reversely proportional* to the working frequency f [27, 75], β is a processor-relevant factor, and T_{sta} is free from the change of f. Eq. (1) suggests that u can be written as $u = a/f + b$, where $a = \beta \times W_{act}$ and $b = T_{sta}$. If a and b are known, we can set f as $\frac{a}{u^g - b}$ to achieve the target utilization u^g easily. However, in various workloads, their characteristics can be greatly different, leading to different a and b values due to two reasons. First, for different workloads, the percentages of T_{act} and T_{sta} are different, which causes different W_{act} (in a) and T_{sta} (in b) in the first place. Second, even if workload W_{act} is fixed, the computing time T_{act} is further related to a factor β, depending on the processor's task scheduling [74].

Workload characteristics. Therefore, the workload characteristics for DVFS can be modeled using the following tuple:

$$< a, b >, \text{ where } a = \beta \times W_{act} \text{ and } b = T_{sta}. \qquad (2)$$

As Figure 5 depicts, for the current frequency (f^0) and processor utilization (u^0), f needs to be adjusted to different values (f^i) to achieve the same target utilization u^g under different workload characteristics (curves C_i). In other words,

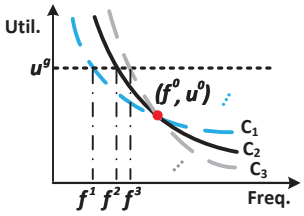

Figure 5: DVFS workload-awareness aims to predict the most likely workload curve, so that a correct frequency adjustment can be made to achieve the target utilization.

workload characteristics essentially refer to different curves ($u = a/f + b$) in Figure 5, which are time-varying by nature, *i.e.*, changing from one curve to another over the time.

Workload-awareness. Thus, *the DVFS workload-awareness aims to predict the most likely workload-characteristic curve to make a correct frequency adjustment each time*.

- For application-agnostic methods, such workload context information is unknown. So, they adopt a heuristic strategy of using a *pre-defined* curve all the time, which naturally leads to suboptimal performance.
- Application-oriented methods can choose the correct curve for the target task. But, when more concurrent tasks are run, the corresponding curve changes, explaining why they become less effective in experiments (§2).

As the workload characteristics of mobile applications become more complex and dynamic, it is difficult to directly derive the optimal frequency adjustment strategy. Therefore, we employ the exploration-and-exploitation process of reinforcement learning, a data-driven method combined with a new meta-state metric design to handle complex workload characteristics in this paper.

3.2 Design of GearDVFS Governor

For each processor i, GearDVFS aims to control the processor frequency $f_i(t)$ to maintain its utilization $u_i(t)$ close to a judicious target level u_i^g, such as 80%, which can be expressed as the following optimization problem $\boldsymbol{P}$:

$$(\boldsymbol{P}): \quad \min \ \frac{1}{T\times M}\sum_{t=1}^{T}\sum_{i=1}^{M}|u_i(t)-u_i^g|, \tag{3}$$

$$s.t. \quad f_i^{min} \le f_i(t) \le f_i^{max}, \ \forall i,t, \tag{4}$$

$$c_i(t) \le c_i^{thermal}, \ \forall i,t. \tag{5}$$

In Eq. (3), the decision variables are the frequency of each processor i and the objective of optimization is to minimize the average difference (*i.e.*, $\frac{1}{T\times M}$) between the achieved utilization $u_i(t)$ and its target level u_i^g over total M processors (such as CPU and GPU) from $t = 1$ to $t = T$, *e.g.*, $T = 50$ ms. Eq. (4) ensures that $f_i(t)$ is not beyond its feasible changing range $[f_i^{min}, f_i^{max}]$ specified in the hardware manual, and Eq. (5) requires that the temperature of each processor is within its thermal throttling $c_i^{thermal}$ to avoid overheating [41]. Before we move on, two points are noteworthy:

Why can controlling utilization save energy? When utilization is low, it means that the processor is *overpowered* for the current workload, leading to energy waste. In this case, a slight decrease of f usually does not affect task processing [21], but it can save substantial energy because the energy consumption is proportional to f^3 (§2.1). Hence, when utilization is low (*e.g.*, 30%), f is decreased to "slow down" the chip for conserving energy and reducing heat generation.

On the other hand, processors should also not be fully loaded, and a margin (*e.g.*, 20%) is allocated to tolerate workload dynamics. Otherwise, it can cause the device to be unresponsive when the extra workload needs to be handled by a fully loaded processor, resulting in poor application QoE.

Do we need to achieve u^g all the time? When the workload is extremely light or heavy, the target utilization cannot (and also should not) be achieved, which is considered and ensured in GearDVFS (by the design in §3.2.3).

RL-based solution. Inspired by recent DVFS methods [41], the problem $\boldsymbol{P}$, introduced in Eqs. (3)–(5), can be converted to a reinforcement learning (RL) problem [71] to solve, which consists of several main components: *states*, *actions*, *rewards*, and *transitional probabilities*. For DVFS, states and actions can represent the workload characteristics and frequency adjustments for each device. One action a can make the device "transit" from current state s to a new state s'. Meanwhile, a probability $p(s'|s,a)$ denotes the chance that this transition would occur, and a Q-function [68] is usually employed to quantify the reward r of this state transition. To avoid manual investigation, RL can learn an optimal policy $\pi^\star$ to automatically select the action a for each state s to maximize the expected reward in the long run through

$$(\boldsymbol{Q}): \ \max_\pi\{E[Q_\pi(s,a)]\}, \tag{6}$$

where Q-function $Q_\pi(s,a) = R_\pi(s) + \gamma\sum_{s'} p(s'|s,a)\times R_\pi(s')$, $R_\pi(s)$ is the reward achieved at state s and γ is a discounting factor. In practice, the transition probabilities $p(s'|s,a)$s are difficult to model explicitly. The advantage of using the Q-function is that it is *model free* [47], which can leverage the neural network, such as deep Q-network (DQN) [18, 80], to estimate the expected reward sum in an end-to-end manner. Details of RL can be found in [71]. After problem $\boldsymbol{Q}$ is solved, the derived $\pi^\star$ is used for power governing. However, RL is only a framework. We need to customize its states, actions and rewards, and solve unique challenges (§1) in our design.

3.2.1 GearDVFS meta-states. In RL, states refer to a set of observable situations related to the control objective. Utilization u of a processor is in a form $u = a/f + b$, where <a, b> represents workload characteristics and it should serve as the RL states. Although workload characteristics can be

formulated using <a, b>, some parameters involved cannot be obtained directly from hardware data, such as β in factor a. In addition, even if <a, b> is known, DVFS needs to predict (near future) workload characteristics instead of using their current values, because whenever the processor frequency is adjusted, that frequency is used for the next time window.

To overcome these challenges, we propose to derive a good substitute to approximate such workload contexts and enable the state construction. Our main idea is to derive new representation states, called **meta-states**, from the history of a series of raw and observable hardware data, to represent and predict complex workload characteristics. There could be other alternative ways, *e.g.*, using a linear/higher-order predictor and training it with hardware data. Since this process essentially aims to distill knowledge from the raw observation data, one advanced way from the deep learning domain [49] is to distill knowledge from a hidden *latent hyper-space* encoded by a network, based on raw observation data. In GearDVFS, we employ the popular *encoder-decoder* framework [32] and extend it with the following designs to learn meta-states.

Primary meta-states. As depicted by the top dashed box in Figure 6, the encoder first takes the raw observation $x(t)$ as input to derive an intermediate latent variable $z(t)$, where $x(t)$ is a set of raw hardware data from the device, including the following information in our current design:

- a) The set of utilization statistics, with CPU utilization for 1) active computing $u_i^{act}(t)$ and 2) stalled waiting $u_i^{sta}(t)$, and 3) GPU utilization $u_i(t)$.[1]
- b) Other *raw hardware data*: 4) frequency of each processor $f_i(t)$ and 5) processor temperatures $c_i(t)$.

Based on the obtained latent variable $z(t)$, the decoder tries to reconstruct the original $x(t)$, where the reconstructed one is denoted as $\hat{x}(t)$. Both the encoder and decoder are implemented by a lightweight multi-layer perceptron (MLP) in GearDVFS. The rationale of this encoding-decoding process is to explore a good intermediate latent space hidden inside the raw observation input $x(t)$, so that the obtained latent variable $z(t)$ is representative enough to reconstruct $x(t)$ precisely. Thus, we introduce the following mean squared error (MSE) loss to ensure the closeness between $x(t)$ and $\hat{x}(t)$:

$$\mathcal{L}_{MSE} = \frac{1}{T}\sum_{t=1}^{T} MSE(x(t), \hat{x}(t)). \tag{7}$$

On the other hand, to make the exploration of the latent space more robust, each dimension of latent variable $z(t)$ is expressed as a distribution [24] instead of a fixed value. GearDVFS adopts a simple yet effective Gaussian distribution [43], and the encoder learns and outputs the mean $\mu(t)$ and variance $\sigma^2(t)$ of the distribution $p(z(t)|x(t))$ as follows:

$$z(t)|x(t) \sim \mathcal{N}(\mu(t), \sigma^2(t)),$$

where $p(z(t)|x(t))$ is a *conditional* distribution of $z(t)$ under the observation of $x(t)$. The latent variable $z(t)$ then can be obtained by *sampling* over this distribution $p(z(t)|x(t))$. In addition, as workloads are time-varying by nature, we add a long short-term memory (LSTM) to further capture the temporal feature of workloads, which takes the input as the current $x(t)$ and $z(t)$, as well as the previous output of LSTM at $t-1$, to produce the new output, which is used as the primary meta-state, denoted as $m(t)$ in Figure 6.

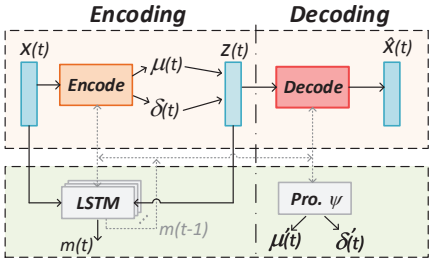

Figure 6: Illustration of the meta-state learner.

Enhanced meta-states. To avoid overfitting of $p(z(t)|x(t))$, the sampled conditional distribution $p(z(t)|x(t))$ is further regulated by the distribution $p(z(t))$ in the training, which is assumed as a normal distribution in prior studies [66]. However, this does not necessarily hold for mobile workloads. We thus use the meta-state $m(t-1)$ only to feed into a MLP network ψ (by excluding the raw observation $x(t)$) to estimate the mean and variance of $p(z(t))$ for regulating $p(z(t)|x(t))$. Since $p(z(t)|x(t))$ and $p(z(t))$ are two distributions, we use Kullback–Leibler (KL) divergence to measure their difference, leading to the second loss function:

$$\mathcal{L}_{KL} = \frac{1}{T}\sum_{t=1}^{T} KL(p(z(t)|x(t)), p(z(t))). \tag{8}$$

The sum of $\mathcal{L}_{MSE}$ and $\mathcal{L}_{KL}$ forms the final loss for learning meta-states. After the system training (§4), each raw observation $x(t)$ goes through this learner to obtain meta-state $m(t)$ first, which is then used to form RL state $S_{MS} =< m(t), x(t) >$ to trigger different actions. In current GearDVFS, we concatenate meta-state $m(t)$ and the raw hardware data $x(t)$ in constructing the RL state S_{MS} to enhance the state reliability.

3.2.2 GearDVFS actions. For DVFS, the actions are defined to set the frequency of each CPU and GPU processor. In any RL state S_{MS}, taking different actions could lead to different device statuses, which are measured by the rewards (next subsection). With the DQN method for Eq. (6), the Q-function can be instantiated by a lightweight Q-network [18, 80]. The

[1]Due to a limited number of performance monitoring counter (PMC) units, *e.g.*, three on our devices, we use them to monitor frontend stalled, backend stalled and overall cycles to obtain active computing and stalled waiting for CPU. Since GPU is mainly used for computation intensive tasks, overall utilization may already describe its working state well, and current mobile devices only report an overall utilization of GPU. When detailed GPU data become accessible in the future, they can be explicitly included in our design.

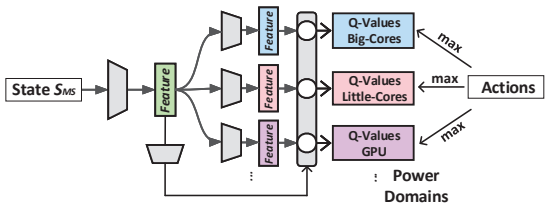

Figure 7: Illustration of the Q-network design with the parallel branches to handle different power domains.

training can teach the Q-network how to select the best action under each state S_{MS} to gain the highest reward.

The RL training is a self-exploration process. The training algorithms try all or certain actions from an action space (depending on the algorithms) to obtain the corresponding rewards [32]. After training, the Q-network simply selects the action leading to the highest reward each time.

Problem. However, we find that directly applying traditional RL training to the DVFS design encounters a crucial large action-space issue. We suppose that M processors exist and each processor i has n_i available frequency levels. In total, $\prod_{i=1}^{M} n_i$ different combinations (actions) need to be considered each time, such as 1872 actions in NVIDIA Jetsons for each state S_{MS}. If all the combinations are explored each time, the Q-network should have the same amount of output entries, where each entry outputs the corresponding reward. This naturally leads to a *wide* output structure for the Q-network, which is time-consuming to train and difficult to converge.

To overcome this issue, the existing methods usually consider only a subset of the action space each time, *e.g.*, zTT [41] considers nine (out of 1872) actions (*i.e.*, with nine output entries) each time on NVIDIA Jetson TX2 [12]. Although this simplification makes training tractable, it faces difficulty in fully exploring the hardware capability (*i.e.*, most actions are not explored yet), leading to the sub-optimal performance.

Solution. The key idea to cope with this issue in GearDVFS is that through a careful Q-network design, we can segregate the reward exploration for each processor, so that we can accomplish the exploration more from a *parallel* feature extraction, rather than increasing the output dimension. By doing so, the output dimension can be reduced from the *production* $\prod_{i=1}^{M} n_i$ to a *summation* $\sum_{i=1}^{M} n_i$, *e.g.*, reduced to 37 on Jetson TX2. In GearDVFS, we employ a branch-based structure [73] to realize this Q-network design.

As Figure 7 depicts, the RL state S_{MS} is first fed into a set of the shared feature extraction layers to extract the common features across different processors, named as the *power domains* in the figure. These common features are fed into a series of feature extraction *branches*, where each branch corresponds to one power domain (*i.e.*, a processor with frequency to be adjusted independently), such as CPU, GPU, DLA, etc. Then, the common features are mapped to the same dimensions as

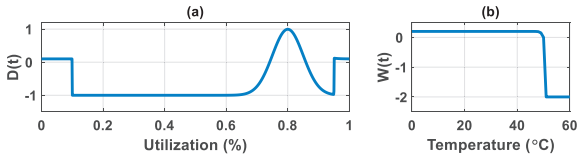

Figure 8: The reward design includes (a) a target utilization of 80% and (b) a thermal threshold of 50°C.

the features for each domain, and they are fused together to output the rewards for each branch.

This design essentially copes with the trade-off between latency and optimality of the frequency exploration. Our Q-network design explores the reward of each power domain in parallel (to speed up exploration), instead of checking all combinations with many impractical cases. Since the number of actions for each branch is small, all possible frequencies for each power domain can be explored. On the other hand, since power domains should be governed jointly, our Q-network considers all power domains to learn their features and exploits their joint features to find the maximum Q-value for each power domain. Therefore, our Q-network is essentially reshaped to fit the DVFS design to better handle this trade-off, so that all feasible frequencies of each power domain can be explored for fine-grained frequency adjustment.

3.2.3 GearDVFS rewards. The objective of our DVFS design formulation is to maintain the utilization close to a target level u^g (§3.2). Therefore, we propose a new reward function $D_i(t)$ for each processor i as follows:

$$D_i(t) = \begin{cases} \lambda, & u_i(t) \notin [u_i^{min}, u_i^{max}], \\ u + v \times e^{-\frac{(u_i(t)-u_i^g)^2}{w^2}}, & otherwise, \end{cases}$$

where the first case represents extremely light ($< u_i^{min}$) or heavy (u_i^{max}) workloads, for which using even the minimum or maximum frequency cannot achieve the target utilization u_i^g. In this case, using the minimum or maximum frequency is actually the "optimal" action, and therefore, a small positive reward λ (= 0.1 empirically) is given without penalty.

In the second case, the utilization is within $[u_i^{min}, u_i^{max}]$, and we calculate the reward based on the *distance* between $u_i(t)$ and u_i^g. The hyper-parameters u, v and w are chosen, so that when the distance is small, a positive reward, close to 1, is given, while a negative reward, close to -1, is given for a large distance. When $u_i(t)$ is near the target u_i^g, more rewards are obtained. We use an exponential function instead of a step function, to distinguish the changing direction of the reward to guide the RL training, as shown in Figure 8(a).

In addition to $D_i(t)$, we also add $W_i(t)$ proposed by zTT [41] to avoid the thermal throttling issue (Figure 8(b)), where $W_i(t) = 0.2 \cdot \tanh\{T_{thre} - T_C(t)\}$ if $T_C(t) < T_{thre}$, and $W_i(t) = -2$ if $T_C(t) \geq T_{thre}$, where $T_C(t)$ is the current device temperature and T_{thre} is the temperature threshold. The overall reward

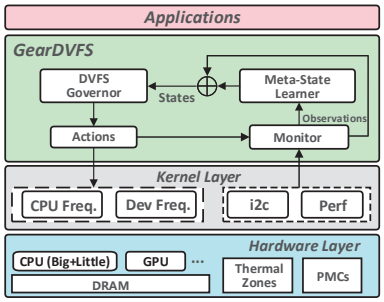

Figure 9: Illustration of the GearDVFS architecture.

$R(t)$ in current GearDVFS is $R(t) = \frac{1}{M}\sum_{i=1}^{M}(D_i(t) + W_i(t))$, where M is the total number of processors.

3.2.4 Governor updates. With the *states*, *actions*, and *rewards* designed above, GearDVFS knows how to select the action leading to the highest reward each time after the training. Finally, we discuss one practical consideration to update the GearDVFS governor. After training, GearDVFS may experience new "contexts" different from the training, which may include new workload dynamics, different temperatures, etc. The RL training is a self-exploration process (§3.2.2) that does not require ground truth, *i.e.*, we only provide sufficient execution traces for exploration [47]. Thus, RL-based solutions could leverage the actual hardware data to update the governor to adapt to new contexts. Similar to [41], we also employ fine-tuning to enable this update (*i.e.*, updating the governor with actual hardware data), as the network structure remains the same and only a few parameters have to be updated. We note that this update is optional, which can be disabled due to the privacy policy of the historical data usage.

4 IMPLEMENTATION

Test-bed. We have developed GearDVFS using Python 3.6 on NVIDIA Jetson NX. The frequency of its CPU (NVIDIA Carmel) and GPU (NVIDIA Volta) can be adjusted in the ranges of 0.1–1.9 GHz and 0.1–1.1 GHz, respectively. Moreover, we have also deployed and tested GearDVFS on Odroid-XU3, Jetson Nano and Raspberry Pi 4B for various micro-benchmark experiments. Redmi Note 9 is further used to test GearDVFS in the scenarios with less task concurrency. Figure 9 summarizes the implementation of GearDVFS, as an intermediate layer between the kernel and applications. GearDVFS contains two major modules: *meta-state learner* and *governor*. We use small MLPs only for each basic network block in these two modules to make GearDVFS lightweight. GearDVFS monitors the hardware data to derive meta-states and performs RL learning periodically, *e.g.*, 100 ms (§5.3).

For Jetson and XU3, we adopt INA3221/231 modules to monitor the device energy consumption and develop a compact Perf to access hardware data with less latency. For Pi, we use Waveshare power monitor HAT to monitor its energy consumption. On the phone, we adopt *simpleperf* from Android to access the hardware data. We further develop a launcher to start each APP from the computer through Wi-Fi by Android debug bridge (ADB) and estimate APP energy consumption more accurately using Google's battery historian service [1].[2]

Application tasks. To evaluate GearDVFS, 12 tasks from four real-world mobile applications will be used, which cover a wide spectrum of workload characteristics:

i) Self-driving (4 tasks): 1) lane detection, 2) object detection, 3) segmentation (to separate roads, pedestrians, vehicles, etc.) and 4) depth estimation. They are all deep learning tasks, which are directly from a self-driving system [10] for small autonomous cars controlled by Jetson devices [55, 69] with balanced (but more towards computing-intensive) workloads.

ii) Robot (3 tasks): 1) YOLO v3 [64], a deep learning based object recognition task; 2) video uploading, a networking task; and 3) QuartzNet [45], a deep learning based speech recognition task. Tasks 1) and 3) have more computing-intensive workloads. Task 2) has more memory-intensive workloads.

iii) UAV ground station (2 tasks): 1) YOLO v3; and 2) multi-stream video receiving from various UAVs [11]. They have computing- and memory-intensive workloads, respectively.

iv) Mobile phone APPs (3 APPs); 1) Tik Tok; 2) PlayerUnknown's Battlegrounds (PUBG), a popular 3D mobile game; and 3) Zoom. Their workloads are more computing-intensive.

System training. We deploy and reuse a single GearDVFS governor template in the device, which is trained following a standard RL training procedure [41] by using the actual workloads from the application, *e.g.*, the self-driving workloads are used to train GearDVFS if the device is installed in an autonomous vehicle. When the device is later used in a different application scenario, our governor can be reused after retraining with new workload data. On the other hand, if the application scenario is unclear initially, we can provide a default version trained with some typical workloads. Device can first use this version and updates the governor with the hardware data during actual execution for adaptation.

Q-network is converged when the reward becomes stable over epochs [41]. In current GearDVFS, we adopt the first five epochs empirically to train its meta-state learner and governor alternatively (*i.e.*, freezing one to update the other one). Thereafter, the governor is trained (with the learner fixed) until reward becomes stable (examined in §5).

5 EVALUATION

We compare the following DVFS methods in the evaluation:

[2]This service can estimate the energy consumption of each APP separately, so that other energy consumers, such as background services and wireless, will not be accounted for. Using Wi-Fi here can also avoid phone's charging if a USB cable is used, which could affect the measurement.

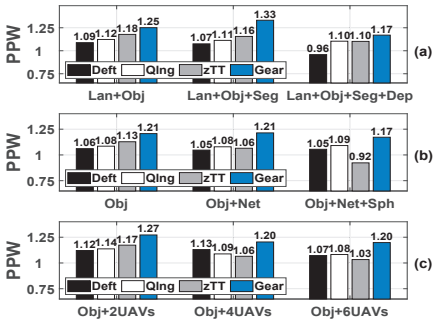

Figure 10: PPW of each method in the mobile applications of (a) self-driving, (b) robot and (c) UAV ground station under various concurrent tasks.

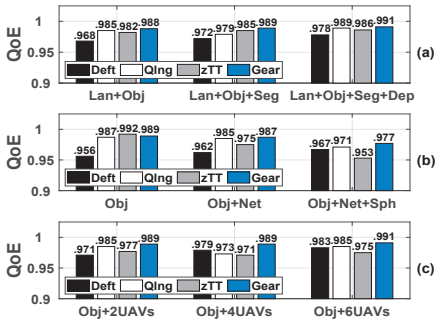

Figure 11: QoE of each method in the mobile applications of (a) self-driving, (b) robot and (c) UAV ground station under various concurrent tasks.

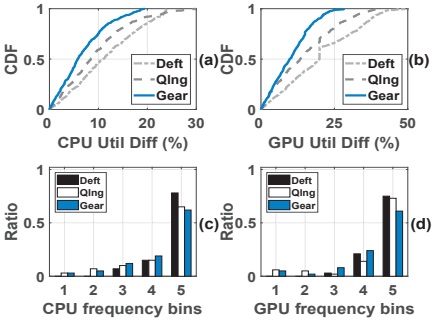

Figure 12: (a-b) CDF of utilization differences, and (c-d) distribution of frequency selections from low (1) to high (5) frequency bins.

1) Default (Deft): we use popular *schedutil* for CPU and *simple_ondemand* for GPU as *Default* in the evaluation, which are used in existing mobiles, such as NVIDIA Jetson devices.

2) Q-Learning (Qlng): recent Q-learning based method that adopts *utilization* to control the frequency of CPU only, which does not considers the workload context and employs the default *simple_ondemand* method for GPU.

3) zTT: the state-of-the-art application-oriented DVFS method [41] with the thermal throttling consideration.

4) GearDVFS (Gear): the method proposed in this paper.

Two main metrics are adopted in the evaluation:

1) Performance per watt (PPW): this is a popular metric to quantify the energy efficiency of one device [9], which is computed as the ratio between QoE and the power consumption. A larger PPW indicates better performance. This metric is intended to avoid situations, where DVFS methods inappropriately use very low operating frequencies to save energy but unduly hurt application QoE.

2) Utilization (Util): Deft, Qlng and Gear methods aim to achieve a target utilization in general, which is set to be 80% for both CPU and GPU in this experiment. For these three methods, we also investigate how close the actual utilization is achieved with respect to the target level (80%).

5.1 Overall Performance

Performance of PPW. We first investigate the overall PPW performance of four methods. As applications have different QoEs, such as frame rate and latency, we use the QoE achieved by the traditional *performance* governor as a baseline to normalize the QoE achieved by each method and then compute PPW.[3] In the evaluation, a larger PPW indicates better performance. In Figure 10(a), we first examine the four tasks from the self-driving application. In this application, lane detection (Lan) and object detection (Obj) two tasks need to execute all the time, while segmentation (Seg) and depth estimation (Dep) are selective for various vehicles of different built-in sensors [10]. As all these tasks use frame rate as QoE, we extend zTT by using the overall frame rate in its training to ensure it can work with multiple tasks. Figure 10(a) shows that both Qlng and zTT outperform Deft, and the overall PPW improvement of zTT is larger than that of Qlng. Overall, Gear further improves PPW by 15.0–23.9%, 6.0–19.6% and 6.2–15.2% compared with Deft, Qlng and zTT, respectively.

In Figure 10(b), we examine object detection (Obj), video uploading (Net) and speech recognition (Sph) for the robot application, where Obj uses frame rate as QoE and the rest two tasks use latency as QoE. As their QoEs are different, we have to choose one QoE to train zTT. We select frame rate, since Obj is performed more frequently. When concurrent tasks (with different QoEs) start to execute, zTT performs worse than Deft and Qlng. Gear can achieve the best performance in all scenarios, improving PPW by 11.1–16.0%, 7.4–12.5% and 7.0–26.9% compared with Deft, Qlng and zTT, respectively.

In Figure 10(c), we further examine the application of UAV ground station [11] to receive the transmitted video streams from multiple UAVs to detect special events or objects, such as fires or humans [11, 25]. In this experiment, we vary the number of video streams from two to six. We find that jitter between the various streams can make the overall frame rate at the device side unstable, and so Qlng and zTT sometimes perform worse than Deft. Gear can achieve the best performance, improving PPW by 6.5–13.5%, 10.7–15.6% and 8.5–16.0% compared with Deft, Qlng and zTT, respectively.

Figure 11 further shows that recent methods (Qlng, zTT and Gear) effectively improve the energy efficiency of the device, *i.e.*, without using inappropriately low operating frequencies to save energy that could hurt QoE. As different tasks may have different QoE metrics, we normalize the QoE achieved in each setting relative to the QoE achieved using the maximal

[3] If *performance* has frame rate r_0, the normalization for another method of frame rate r_1 is r_1/r_0. If *performance* has latency l_0, we normalize l_1 of another method as $\frac{1/l_1}{1/l_0}$, so that a larger normalized QoE means being better.

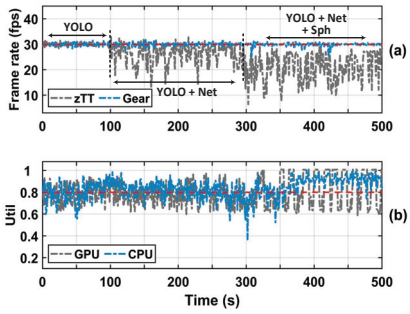

Figure 13: (a) Frame rates achieved by zTT and Gear for the robot task with a similar setting as Figure 1. (b) CPU and GPU utilization achieved by Gear.

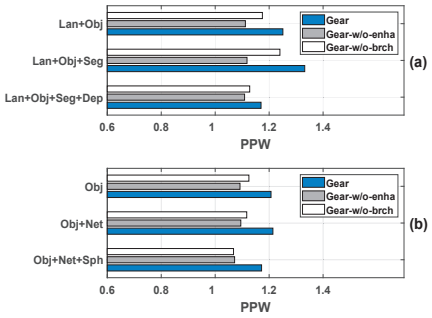

Figure 14: Ablation study for GearDVFS with several intermediate versions in the applications for (a) self-driving and (b) robot in Figure 10.

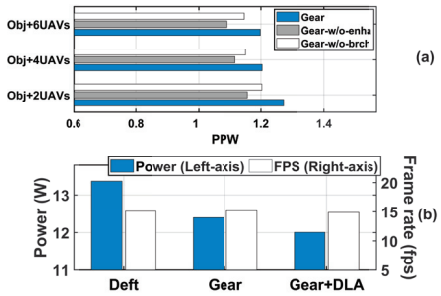

Figure 15: (a) Ablation study for GearDVFS in the UAV ground station. (b) Performance of GearDVFS by including one more power domain DLA.

operating frequency (*i.e.*, the highest QoE), so that a higher bar indicates a better QoE in Figure 11. We can see that these recent methods do not sacrifice QoE performance for higher PPW results. With an effective design, Gear can achieve both good energy efficiency in Figure 10 and QoE in Figure 11.

Performance understanding. We then understand why Gear can improve the energy efficiency in Figure 10.

CPU and GPU utilization. As the design objective of GearDVFS is to use judicious energy to achieve the target utilization level (80% in the evaluation), Figures 12 plots the fine-grained CPU and GPU utilization achieved by Gear for the tasks ("Lan+Obj+Seg" in Figure 10(a)) as an example.

Figure 12(a) shows that Gear can maintain CPU utilization close to 80%. It plots the CDF of the utilization differences (between the achieved utilization and the target 80%), which are small, *i.e.*, 5.7% on average. Similar to Gear, Deft and Qlng also aim to achieve the 80% target in the experiment (as zTT is not designed for this target, it is skipped from Figure 12), but they have a larger difference (compared to the target value). Figure 12(b) further plots the corresponding GPU utilization, in which Gear's GPU utilization difference (3.7% on average) is also less than the other two methods.

In Figure 12(c), we further divide the whole CPU frequency range into five bins from 1 to 5 (from low to high), and plot the distributions of the CPU frequency selections by these three methods. We can see that Gear tends to approach the same utilization target at lower frequencies than other two methods. This means that the relatively low energy consumption of Gear (by using lower frequencies) is sufficient for current application tasks, revealing why Gear achieves higher energy efficiency. Figure 12(d) indicates that Gear can adopt appropriate working frequencies for GPU as well.

QoE. We further understand why GearDVFS performs better than zTT. Figure 13(a) shows the QoE achieved by them in the robot application. When only YOLO is working, the frame rates of zTT and Gear are both close to 30. When networking (video uploading) is launched, the frame rate of zTT becomes less stable and the average QoE drops to 26.03 fps, while the frame rate of Gear is not affected. Then, when speech recognition is further performed, the frame rate of zTT fluctuates significantly and its average value drops to 20.97 fps, which is only slightly varied in Gear. Figure 13(b) further plots the CPU and GPU utilization achieved by GearDVFS in this application. The utilization of both processors can be effectively maintained close to 80%. After speech recognition starts to work, the overall workload is heavy. Thus, the CPU utilization is above 90%, which is a correct action.

Ablation Study. Next, we develop two intermediate system versions of GearDVFS for an ablation study to understand the efficacy of two main modules. **Gear-w/o-enha**: in this first version, we remove the enhancement designs in the meta-state learner module in §3.2.1 (we cannot remove the entire module since meta-states are required to form the RL states). This leads to a low-quality meta-state construction. **Gear-w/o-brch**: in this second version, we remove the branch-based structure for the Q-network in §3.2.2 and select only nine actions from the whole action space, similar to zTT [12].

In Figure 14 and 15(a), we show the PPW of each intermediate version for all the tasks studied in Figure 10. We also plot the performance of the original GearDVFS as a reference. For the self-driving tasks, PPW of "Gear-w/o-enha" and "Gear-w/o-brch" is degraded by 5.2–16.06% and 3.7–6.9%, respectively. The low-quality meta-states cause slightly more performance drops than the branch-based structure. For the rest tasks in robot and UAV applications, the conclusion is similar. In particular, PPW of "Gear-w/o-enha" and "Gear-w/o-brch" is degraded by 7.1–9.8% and 4.3–9.1%, respectively. The results suggest the effectiveness of each module.

5.2 Micro-benchmarks

Additional power domain. The branch-based Q-network design in GearDVFS is not only beneficial to the power governing (Figure 14), it also brings an opportunity to support additional power domains (§3.2.2). To unveil this point, we

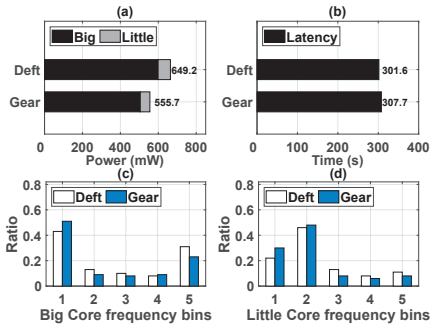

Figure 16: Experiment on XU3's CPU of big.LITTLE cores. (a) Power consumption and (b) latency to complete the same set of tasks. Frequency selection for the (c) big and (d) little cores.

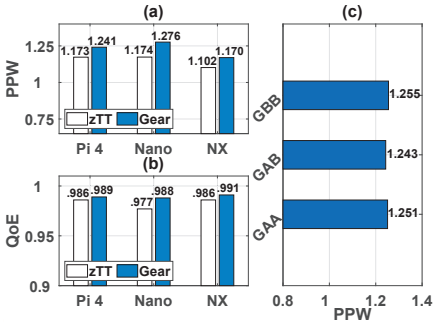

Figure 17: Performance of (a) energy efficiency and (b) QoE on different devices for the same application scenario. (c) Energy efficiency on the same type of mobile devices.

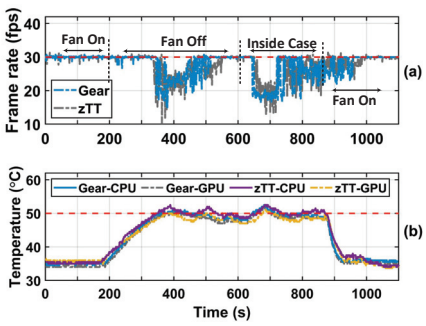

Figure 18: Updating DVFS governors when we vary ambient temperature. (a) Impacts on frame rate when the on-board fan is switched off. (b) CPU and GPU temperatures over time.

include a new power domain on Jetson NX — deep learning accelerator (DLA), which is an independent on-board module to accelerate deep learning tasks. By default, DLA is disabled. In Figure 15(b), we enable it and add one more branch in the Q-network for DLA. With this update, GearDVFS can control the frequencies of CPU, GPU and DLA. When Deft is adopted with DLA for YOLO, the average power consumption is 13.4 W. If GearDVFS controls CPU and GPU only and leaves DLA governed by its default governor, the power consumption ("Gear") can be reduced to 12.4 W without impacting the frame rate. If DLA is also governed by Gear ("Gear+DLA"), the power consumption is further reduced to 12.01 W, leading to 3.2–10.2% energy reduction.

Big.LITTLE cores. Mobile CPUs may have Big.LITTLE cores. In this experiment, we deploy GearDVFS on Odroid-XU3, which has such a CPU, and compare it with its built-in DVFS governor Deft. Figure 16(a) shows that the average power consumption of GearDVFS is improved by more than 14.4% than that of Deft, while the latency of these two methods to complete the same set of tasks is similar, as shown in Figure 16(b), which indicates that Gear does not affect the task processing. Figure 16(c)-(d) report the distributions in the frequency selections for big and little cores of two methods, where Gear can select relatively lower frequencies for both types of CPU cores to improve their energy efficiency.

Different devices. In this experiment, we examine Gear on different devices, including Raspberry Pi 4B, Jetson Nano and Jetson NX, for the same application scenario. According to the computation capabilities of each device, we run "Lan+Obj" on Raspberry Pi 4B, "Lan+Obj+Seg" on Jetson Nano, and "Lan+Obj+Seg+Dep" on Jetson NX. Figure 17(a) shows that Gear's PPW is comparable across the three devices. Due to the high workload on Jetson NX, its PPW value is slightly lower than other two. The energy efficiency of zTT has a similar trend to that of Gear, but Gear can improve PPW by 5.79–8.72%. Figure 17(b) further shows that both methods also achieve good QoE performance in this experiment.

Devices of the same type. For devices of the same type, after GearDVFS is trained on device A, it can be directly deployed to another device B. In Figure 17(c), we use two Jetson NX devices, denoted as "A" and "B", to train GearDVFS, denoted as "G". We investigate PPW for three settings of G: GAA (G is trained on A and tested on A), GAB (G is trained on A and tested on B) and GBB (G is trained on B and tested on B). The results show that the energy efficiency is similar for these three settings. If Gear needs to be deployed to another type of device, new training will be required as their hardware may have different parameters and capabilities.

Impact of temperature. In this experiment, we vary the ambient temperature to investigate GearDVFS when the feature of *runtime governor update* is enabled (§3.2.4). In Figure 18, we run YOLO at 30 fps and use the on-board fan to control the heat dissipation. Initially, the fan is on. Then, we switch it off and the temperatures of both CPU and GPU start to increase. Before temperature reaches the thermal threshold (set to 50°C), the frame rate keeps stable. After the threshold is reached, the working frequency is decreased dramatically (to avoid overheating). The frame rate thus drops, while Gear starts to adapt to this new environment and recovers frame rate shortly. Next, we introduce a challenging setting, in which we switch off the fan and cover the device by a protective case. In this extreme case, the working frequency is decreased significantly to avoid overheating and the frame rate is fluctuating around 25 fps. Finally, when the fan is switched on, the frame rate can be recovered again. Since Gear is a general DVFS design, we integrate the thermal throttling capability of state-of-the-art zTT into Gear and find that it can achieve thermal throttling performance similar to zTT in Figure 18(a-b).

Adaptation. In this experiment, we investigate the adaptation performance of GearDVFS to new scenarios. We first change

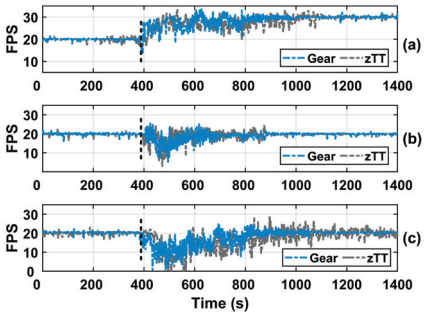

Figure 19: Adaptation performance when we (a) change the QoE requirement, (b) increase the number of tasks, and (c) start a new task at time 400.

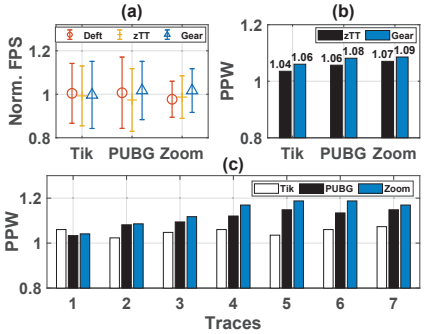

Figure 20: Experiment on a phone. (a) Frame rates; (b) PPW relative to Deft; and (c) PPW of Gear with more copies of the historical data from each APP.

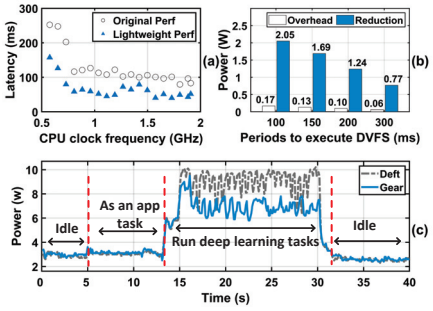

Figure 21: System overhead. (a) Latency overhead. (b) Energy overhead and reduction at different DVFS periods. (c) Power consumption over time.

the QoE requirement in Figure 19(a), where "Lan" and "Obj" from the self-driving application are executed. Their frame rates are 20 fps initially, and we change it to 30 fps for one task (*e.g.*, "Obj") at time 400. Gear takes 500 seconds to finish adaptation, about 200 seconds fewer than zTT. We then increase the number of tasks (present during training) on the device in Figure 19(b), where the number of video streams is increased from two to four at time 400 from the UAV ground station application. The results show that it takes about 490 and 500 seconds for Gear and zTT to complete the adaptation, respectively. Finally, we start a new task (not present during training) in Figure 19(c), where a mobile robot performs object detection initially. Later, speech recognition is launched, and this task was not present in DVFS training. Because speech recognition itself is a heavy task and it has a different QoE from object detection, Gear takes a longer time, about 650 seconds, to complete the adaptation, and zTT does not fully stabilize the frame rate in this new scenario.

By adaptation, Gear can be adjusted to new settings in less time than training from scratch (*e.g.*, about 35 minutes in §5.3), but it can still cause unstable QoE performance for a few minutes, and we plan to further improve it in the future.

Mobile phone. Finally, to show a broader utility of the GearDVFS design that is backwards compatible and effective to the scenarios of less workload concurrency, we test it on the Redmi phone for Tik Tok, PUBG and Zoom as introduced in §4. For each APP, it executes with concurrent background services from operating system only. We investigate the achieved frame rate and the energy efficiency for each APP. As a comparison, we also plot the performance achieved by zTT and Deft (the default governor on this phone). The average frame rates of three APPs are 60, 30 and 20 fps, respectively.

Figure 20(a) depicts the achieved frame rates, which are normalized relative to their average values for a clear illustration. The results show that the frame rates achieved by the three methods are close to each other, suggesting that zTT and Gear do not affect the task processing. Since Deft is the default DVFS method on this phone, we further plot the normalized PPW of zTT and Gear against Deft. Figure 20(b) shows that zTT improves the PPW for the three APPs by 4–7%, and Gear can further improve the PPW by 6–9%.

In this experiment, we further ask the user to keep using these three APPs, and we employ the historical execution traces of each APP to update the Gear governor. In Figure 20(c), we vary the number of such traces from 1 to 7. The result shows that with more traces, the PPW can be further increased, for example, up to 16.92% when six traces are adopted. After using six traces, we find that the improvement is marginal if we further increase the number of traces.

5.3 System Overhead

Training time. The RL training is converged when the obtained rewards become stable [41]. We have studied the change in reward during training of GearDVFS, which takes around 35 minutes (*i.e.*, 2100 seconds) to train GearDVFS from scratch before its reward becomes stable (on a PC with an Intel i7-8700K CPU). Although training time is a one-time effort before deployment, it is still meaningful to study how to further reduce it in the future, *e.g.*, by using multi-threaded execution for training and more powerful machines.

Memory occupancy and execution latency. When GearDVFS is executed, its memory footprint is only 0.4 MB. Due to its lightweight design, it also executes efficiently. With our optimized Perf profiler (§4) to accelerate hardware data access, the latency for GearDVFS to collect data and make each DVFS decision is reduced to be less than 200 ms (mostly 100 ms) at different frequencies in Figure 21(a), so that DVFS can be governed periodically about every 100 ms.

Power consumption. Finally, we measure the power consumption of GearDVFS on Jetson NX using onboard INA3221 power monitor. As a benchmark, we also show the power consumption of the device when using its default governor,

schedutil. In Figure 21(c) when Deft and Gear are used, the average power consumption in idle state is 2.88 W and 2.99 W, respectively. Then, we start another copy of Gear (as an application task), and the average power consumption increases only to 3.05 W (by Deft) and 3.16 W (by Gear), indicating that Gear itself consumes little energy, about 170 mW. Next, we run deep learning tasks for the same period of time in both methods. The device using Deft consumes an average of 9.06 W, which is reduced by 22.6% to 7.01 W with Gear.

When the DVFS period increases (DVFS is performed less frequently), Figure 21(b) shows that the energy overhead of GearDVFS itself decreases, *e.g.*, from 0.17 W to 0.06 W, but the reduction of energy consumption also decreases when heavy tasks are executed, *e.g.*, from 2.05 W to 0.77 W. Thus, short DVFS periods of GearDVFS can lead to more energy reduction during the execution of application tasks.

6 RELATED WORK

DVFS on computers. This type of DVFS design is usually not coupled to any particular application. Typical examples come from the Linux kernel [42], which follow a set of frequency adjustment rules, but their performance is suboptimal when the workload is complex [41]. Hence, many other rule-based methods are proposed to learn workloads to improve DVFS design. For example, a workload-driven DVFS is introduced in [36], which is mainly for computers and some features (*e.g.*, P-States) are not available on mobiles. The authors [15] classify several pre-defined concurrent workloads, and the authors [13] further propose a per-core power model to improve DVFS performance. However, rule-based methods can only capture a few pre-defined workload types, which become less effective in practice. Hence, recent studies start to explore learning-based methods, *e.g.*, using basic [26, 77] and deep [52] Q-learning, which can be accelerated by programmable logic [47]. On the other hand, research [40] also exploits advanced hardware features, *e.g.*, thread-level parallelism (TLP), to better capture workload characteristics.

However, such advanced hardware features are computationally expensive and not suitable for mobiles. On the other hand, the processors on computers (*e.g.*, CPU and GPU) are often separate components, use separate resources (*e.g.*, memory), and are governed independently by existing designs. Instead, the processors are integrated into the same SoC in mobiles, with unified memory access and shared resources [56], which should be jointly governed by the mobile DVFS.

Application-oriented DVFS methods. In mobile devices, studies have explored various application-oriented schemes with dedicated DVFS governors. For instance, the authors in [20, 39, 65, 83] propose such designs to benefit web browsing. A series of customized designs can also optimize energy efficiency for mobile games by governing the achieved frame rate [22, 51, 61, 62]. The designs in [21, 23, 50, 57, 60] further consider the more general multi-media applications with a frame rate QoE but beyond games. NeuOS [16] is optimized for DNN-driven tasks. The state-of-the-art zTT [41] makes a novel contribution to this category of designs recently, by considering not only QoE but also ensuring zero thermal throttling. However, the application-oriented methods are vulnerable to task concurrency and workload dynamics.

CPU task scheduling. DVFS has the following relationship with CPU task scheduling. CPU task scheduling determines which tasks are allocated to which cores [4, 14], and DVFS determines the operating frequencies of each core based on the result of CPU scheduling. The DVFS solutions used in commercial mobile devices or proposed in recent studies [51, 74] mainly follow this layered design paradigm. On the other hand, some manufacturer-specific schedulers have been proposed, *e.g.*, MPDecision [48]. When the device workload is light, they can switch off some processor cores directly. Such scheduling is usually performed on a larger time frame, and it could be integrated into DVFS to comprehensively optimize both short-term (fine-grained) frequency adjustment and long-term on/off switching for processor cores in the future.

Assorted learning and scheduling technologies. Recent theoretical studies have been conducted to address the issue in which the RL states are not observable [30, 32, 49]. Our design is inspired by these works, but we propose new designs for DVFS, including 1) a dedicated network to derive the meta-states from a hidden latent space [24]; 2) a new RL-based DVFS governor with our designs tailored for the RL states, actions and rewards; and 3) the branch-based Q-network [73] to ensure a fine-grained frequency adjustment. On the other hand, recent works have also studied workload characteristics and frequency scheduling of CPUs and GPUs in cloud or data centers [28, 58, 78], which however are orthogonal to the mobile device solutions (§2.1) studied in this paper.

7 CONCLUSION

This paper has proposed a novel DVFS design called GearDVFS to accommodate task concurrency and workload dynamics for mobile devices. In designing GearDVFS, we have identified the reason why the recent DVFS solution is not effective, and introduced a new and effective meta-state metric. With this metric, the workload characteristics can be described, based on which we enable a workload-aware DVFS design by addressing two unique challenges. We have developed a prototype to show valuable performance gains.

ACKNOWLEDGEMENT

We sincerely thank the anonymous shepherd and reviewers for their helpful review comments. Zhenjiang Li is the corresponding author.

REFERENCES

[1] Battery historian. https://github.com/google/battery-historian.

[2] Cpu-freq governor. https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt.

[3] Edge ai market. https://www.marketsandmarkets.com/Market-Reports/edge-ai-hardware-market-158498281.html.

[4] Energy aware scheduling. https://www.kernel.org/doc/html/next/scheduler/sched-energy.html.

[5] Energy consumption issue. https://www.marketsandmarkets.com/Market-Reports/embedded-system-market-98154672.html.

[6] Jetson projects. https://developer.nvidia.com/embedded/community/jetson-projects.

[7] Nvidia jetson. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems.

[8] Odroid. https://www.hardkernel.com/.

[9] Performance per watt. https://en.wikipedia.org/wiki/Performance_per_watt.

[10] Self-driving-ish computer vision system. https://github.com/iwatake2222/self-driving-ish_computer_vision_system.

[11] Self-driving-ish computer vision system. https://github.com/kn1ghtf1re/Hermes-Deepstream.

[12] Source code of ztt. https://github.com/ztt-21/zTT.

[13] Bilge Acun, Kavitha Chandrasekar, and Laxmikant V Kale. Fine-grained energy efficiency using per-core dvfs with an adaptive runtime system. In *Proceedings of IGSC*, 2019.

[14] Mario Bambagini, Mauro Marinoni, Hakan Aydin, and Giorgio Buttazzo. Energy-aware scheduling for real-time systems: A survey. *ACM Transactions on Embedded Computing Systems*, 2016.

[15] Karunakar Reddy Basireddy, Eduardo Weber Wachter, Bashir M Al-Hashimi, and Geoff Merrett. Workload-aware runtime energy management for hpc systems. In *Proceedings of HPCS*, 2018.

[16] Soroush Bateni and Cong Liu. Neuos: A latency-predictable multi-dimensional optimization framework for dnn-driven autonomous systems. In *Proceedings of USENIX ATC*, 2020.

[17] Romil Bhardwaj, Zhengxu Xia, Ganesh Ananthanarayanan, Yuanchao Shu, Nikolaos Karianakis, Kevin Hsieh, Paramvir Bahl, and Ion Stoica. Ekya: Continuous learning of video analytics models on edge compute servers. In *Proceedings of USENIX NSDI*, 2022.

[18] Sidartha AL Carvalho, Daniel C Cunha, and Abel G Silva-Filho. Autonomous power management in mobile devices using dynamic frequency scaling and reinforcement learning for energy minimization. *Elsevier Microprocessors and Microsystems*, 2019.

[19] Xiaomeng Chen, Abhilash Jindal, Ning Ding, Yu Charlie Hu, Maruti Gupta, and Rath Vannithamby. Smartphone background activities in the wild: Origin, energy drain, and optimization. In *Proceedings of ACM MobiCom*, 2015.

[20] Yonghun Choi, Seong-hoon Park, and Hojung Cha. Optimizing energy efficiency of browsers in energy-aware scheduling-enabled mobile devices. In *Proceedings of ACM MobiCom*, 2019.

[21] Yonghun Choi, Seonghoon Park, and Hojung Cha. Graphics-aware power governing for mobile devices. In *Proceedings of ACM MobiSys*, 2019.

[22] Yonghun Choi, Seonghoon Park, Seunghyeok Jeon, Rhan Ha, and Hojung Cha. Optimizing energy consumption of mobile games. *IEEE Transactions on Mobile Computing*, 2021.

[23] Po-Kai Chuang, Ya-Shu Chen, and Po-Hao Huang. An adaptive on-line cpu-gpu governor for games on mobile devices. In *Proceedings of IEEE ASP-DAC*, 2017.

[24] Junyoung Chung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron C Courville, and Yoshua Bengio. A recurrent latent variable model for sequential data. In *Proceedings of NeurIPS*, 2015.

[25] Ashutosh Dhekne, Ayon Chakraborty, Karthikeyan Sundaresan, and Sampath Rangarajan. Trackio: Tracking first responders inside-out. In *Proceedings of USENIX NSDI*, 2019.

[26] Sai Manoj Pudukotai Dinakarrao, Arun Joseph, Anand Haridass, Muhammad Shafique, Jörg Henkel, and Houman Homayoun. Application and thermal-reliability-aware reinforcement learning based multi-core power management. *ACM Journal on Emerging Technologies in Computing Systems*, 2019.

[27] Xing Fu, Khairul Kabir, and Xiaorui Wang. Cache-aware utilization control for energy efficiency in multi-core real-time systems. In *Proceedings of IEEE ECRTS*, 2011.

[28] Juncheng Gu, Mosharaf Chowdhury, Kang G Shin, Yibo Zhu, Myeongjae Jeon, Junjie Qian, Hongqiang Liu, and Chuanxiong Guo. Tiresias: A gpu cluster manager for distributed deep learning. In *Proceedings of USENIX NSDI*, 2019.

[29] Lin Gu, Deze Zeng, Ahmed Barnawi, Song Guo, and Ivan Stojmenovic. Optimal task placement with qos constraints in geo-distributed data centers using dvfs. *IEEE Transactions on Computers*, 2014.

[30] David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In *Proceedings of NeurIPS*, 2018.

[31] Jawad Haj-Yahya, Mohammed Alser, Jeremie Kim, A Giray Yağlıkçı, Nandita Vijaykumar, Efraim Rotem, and Onur Mutlu. Sysscale: Exploiting multi-domain dynamic voltage and frequency scaling for energy efficient mobile processors. In *Proceedings of ACM/IEEE ISCA*, 2020.

[32] Dongqi Han, Kenji Doya, and Jun Tani. Variational recurrent models for solving partially observable control tasks. In *Proceedings of ICLR*, 2020.

[33] Junyeong Han and Sungyoung Lee. Performance improvement of linux cpu scheduler using policy gradient reinforcement learning for android smartphones. *IEEE Access*, 2020.

[34] Rui Han, Qinglong Zhang, Chi Harold Liu, Guoren Wang, Jian Tang, and Lydia Y Chen. Legodnn: block-grained scaling of deep neural networks for mobile vision. In *Proceedings of ACM MobiCom*, 2021.

[35] Sodam Han, Yonghee Yun, Young Hwan Kim, and Seokhyeong Kang. Proactive scenario characteristic-aware online power management on mobile systems. *IEEE Access*, 2020.

[36] Ranjan Hebbar and Aleksandar Milenković. An experimental evaluation of workload driven dvfs. In *Proceedings of ACM/SPEC ICPE*, 2021.

[37] Joo Seong Jeong, Jingyu Lee, Donghyun Kim, Changmin Jeon, Changjin Jeong, Youngki Lee, and Byung-Gon Chun. Band: coordinated multi-dnn inference on heterogeneous mobile processors. In *Proceedings of ACM MobiSys*, 2022.

[38] Abhilash Jindal and Y Charlie Hu. Experience: developing a usable battery drain testing and diagnostic tool for the mobile industry. In *Proceedings of ACM MobiCom*, 2021.

[39] Wonwoo Jung, Chulkoo Kang, Chanmin Yoon, Donwon Kim, and Hojung Cha. Devscope: a nonintrusive and online power analysis tool for smartphone hardware components. In *Proceedings of IEEE/ACM/IFIP CODES+ISSS*, 2012.

[40] Heba Khdr, Santiago Pagani, Muhammad Shafique, and Jörg Henkel. Thermal constrained resource management for mixed ilp-tlp workloads in dark silicon chips. In *Proceedings of DAC*, 2015.

[41] Seyeon Kim, Kyungmin Bin, Sangtae Ha, Kyunghan Lee, and Song Chong. ztt: learning-based dvfs with zero thermal throttling for mobile devices. In *Proceedings of ACM MobiSys*, 2021.

[42] Young Geun Kim, Joonho Kong, and Sung Woo Chung. A survey on recent os-level energy management techniques for mobile processing units. *IEEE Transactions on Parallel and Distributed Systems*, 2018.

[43] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *Proceedings of ICLR*, 2014.

[44] Alexey Kopytov. Sysbench manual. *MySQL AB*, 2012.

[45] Samuel Kriman, Stanislav Beliaev, Boris Ginsburg, Jocelyn Huang, Oleksii Kuchaiev, Vitaly Lavrukhin, Ryan Leary, Jason Li, and Yang Zhang. Quartznet: Deep automatic speech recognition with 1d time-channel separable convolutions. In *Proceedings of IEEE ICASSP*, 2020.

[46] Paul J Kuehn and Maggie Mashaly. Dvfs-power management and performance engineering of data center server clusters. In *Proceedings of IEEE WONS*, 2019.

[47] Eunji Kwon, Sodam Han, Yoonho Park, Jongho Yoon, and Seokhyeong Kang. Reinforcement learning-based power management policy for mobile device systems. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2021.

[48] Ohchul Kwon, Wonjae Jang, Giyeon Kim, and Chang–Gun Lee. Accurate thermal prediction for nans (n-app n-screen) services on a smart phone. In *Proceedings of IEEE SIES*, 2018.

[49] Alex Lee, Anusha Nagabandi, Pieter Abbeel, and Sergey Levine. Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model. In *Proceedings of NeurIPS*, 2020.

[50] Gwangmin Lee, Seokjun Lee, Geonju Kim, Yonghun Choi, Rhan Ha, and Hojung Cha. Improving energy efficiency of android devices by preventing redundant frame generation. *IEEE Transactions on Mobile Computing*, 2018.

[51] Xianfeng Li and Gengchao Li. An adaptive cpu-gpu governing framework for mobile games on big. little architectures. *IEEE Transactions on Computers*, 2020.

[52] Xiao Li, Lin Chen, Shixi Chen, Fan Jiang, Chengeng Li, and Jiang Xu. Power management for chiplet-based multicore systems using deep reinforcement learning. In *Proceedings of ISVLSI*, 2022.

[53] Robert LiKamWa and Lin Zhong. Starfish: Efficient concurrency support for computer vision applications. In *Proceedings of ACM MobiSys*, 2015.

[54] Sicong Liu, Bin Guo, Ke Ma, Zhiwen Yu, and Junzhao Du. Adaspring: Context-adaptive and runtime-evolutionary deep model compression for mobile applications. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2021.

[55] Biswadip Maity, Saehanseul Yi, Dongjoo Seo, Leming Cheng, Sung-Soo Lim, Jong-Chan Kim, Bryan Donyanavard, and Nikil Dutt. Chauffeur: Benchmark suite for design and end-to-end analysis of self-driving vehicles on embedded systems. *ACM Transactions on Embedded Computing Systems*, 2021.

[56] Sparsh Mittal. A survey on optimized implementation of deep learning models on the nvidia jetson platform. *Journal of Systems Architecture*, 2019.

[57] Nachiappan Chidambaram Nachiappan, Praveen Yedlapalli, Niranjan Soundararajan, Anand Sivasubramaniam, Mahmut T Kandemir, Ravi Iyer, and Chita R Das. Domain knowledge based energy management in handhelds. In *Proceedings of IEEE HPCA*, 2015.

[58] Amy Ousterhout, Joshua Fried, Jonathan Behrens, Adam Belay, and Hari Balakrishnan. Shenango: Achieving high cpu efficiency for latency-sensitive datacenter workloads. In *Proceedings of USENIX NSDI*, 2019.

[59] Jongsoo Park, Maxim Naumov, Protonu Basu, Summer Deng, Aravind Kalaiah, Daya Khudia, James Law, Parth Malani, Andrey Malevich, Satish Nadathur, et al. Deep learning inference in facebook data centers: Characterization, performance optimizations and hardware implications. *arXiv preprint arXiv:1811.09886*, 2018.

[60] Jurn-Gyu Park, Nikil Dutt, and Sung-Soo Lim. Ml-gov: A machine learning enhanced integrated cpu-gpu dvfs governor for mobile gaming. In *Proceedings of IEEE/ACM ESTIMedia*, 2017.

[61] Anuj Pathania, Qing Jiao, Alok Prakash, and Tulika Mitra. Integrated cpu-gpu power management for 3d mobile games. In *Proceedings of ACM/EDAC/IEEE DAC*, 2014.

[62] Anuj Pathania, Santiago Pagani, Muhammad Shafique, and Jörg Henkel. Power management for mobile games on asymmetric multi-cores. In *Proceedings of IEEE/ACM ISLPED*, 2015.

[63] Ivan Ratković, Nikola Bežanić, Osman S Ünsal, Adrian Cristal, and Veljko Milutinović. An overview of architecture-level power-and energy-efficient design techniques. *Elsevier Advances in Computers*, 2015.

[64] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[65] Jie Ren, Ling Gao, Hai Wang, and Zheng Wang. Optimise web browsing on heterogeneous mobile platforms: a machine learning based approach. In *Proceedings of IEEE INFOCOM*, 2017.

[66] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of ICML*, 2014.

[67] Claudio Scordino, Luca Abeni, and Juri Lelli. Energy-aware real-time scheduling in the linux kernel. In *Proceedings of ACM SAC*, 2018.

[68] Rishad A Shafik, Sheng Yang, Anup Das, Luis A Maeda-Nunez, Geoff V Merrett, and Bashir M Al-Hashimi. Learning transfer-based adaptive energy minimization in embedded systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2015.

[69] Shuyao Shi, Jiahe Cui, Zhehao Jiang, Zhenyu Yan, Guoliang Xing, Jianwei Niu, and Zhenchao Ouyang. Vips: real-time perception fusion for infrastructure-assisted autonomous driving. In *Proceedings of of ACM MobiCom*, 2022.

[70] D Suleiman, M Ibrahim, and I Hamarash. Dynamic voltage frequency scaling (dvfs) for microprocessors power and energy reduction. In *Proceedings of ICEEE*, 2005.

[71] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[72] Zhenheng Tang, Yuxin Wang, Qiang Wang, and Xiaowen Chu. The impact of gpu dvfs on the energy and performance of deep learning: An empirical study. In *Proceedings of ACM e-Energy*, 2019.

[73] Arash Tavakoli, Fabio Pardo, and Petar Kormushev. Action branching architectures for deep reinforcement learning. In *Proceedings of AAAI*, 2018.

[74] Manni Wang, Shaohua Ding, Ting Cao, Yunxin Liu, and Fengyuan Xu. Asymo: scalable and efficient deep-learning inference on asymmetric mobile cpus. In *Proceedings of ACM MobiCom*, 2021.

[75] Xiaorui Wang, Xing Fu, Xue Liu, and Zonghua Gu. Power-aware cpu utilization control for distributed real-time systems. In *Proceedings of IEEE RTAS*, 2009.

[76] Yanwen Wang, Jiaxing Shen, and Yuanqing Zheng. Push the limit of acoustic gesture recognition. *IEEE Transactions on Mobile Computing*, 2020.

[77] Zhe Wang, Zhongyuan Tian, Jiang Xu, Rafael KV Maeda, Haoran Li, Peng Yang, Zhehui Wang, Luan HK Duong, Zhifei Wang, and Xuanqi Chen. Modular reinforcement learning for self-adaptive energy efficiency optimization in multicore system. In *Proceedings of ASP-DAC*, 2017.

[78] Qizhen Weng, Wencong Xiao, Yinghao Yu, Wei Wang, Cheng Wang, Jian He, Yong Li, Liping Zhang, Wei Lin, and Yu Ding. Mlaas in the wild: Workload analysis and scheduling in large-scale heterogeneous gpu clusters. In *Proceedings of USENIX NSDI*, 2022.

[79] Juheon Yi and Youngki Lee. Heimdall: mobile gpu coordination platform for augmented reality applications. In *Proceedings of ACM MobiCom*, 2020.

[80] Huanhuan Zhang, Anfu Zhou, Jiamin Lu, Ruoxuan Ma, Yuhan Hu, Cong Li, Xinyu Zhang, Huadong Ma, and Xiaojiang Chen. Onrl: improving mobile video telephony via online reinforcement learning. In *Proceedings of ACM MobiCom*, 2020.

[81] Qiyang Zhang, Xiang Li, Xiangying Che, Xiao Ma, Ao Zhou, Mengwei Xu, Shangguang Wang, Yun Ma, and Xuanzhe Liu. A comprehensive benchmark of deep learning libraries on mobile devices. In *Proceedings of ACM WWW*, 2022.

[82] Pengfei Zhou, Yuanqing Zheng, and Mo Li. How long to wait? predicting bus arrival time with mobile phone based participatory sensing. In *Proceedings of ACM MobiSys*, 2012.

[83] Yuhao Zhu, Aditya Srikanth, Jingwen Leng, and Vijay Janapa Reddi. Exploiting webpage characteristics for energy-efficient mobile web browsing. *IEEE Computer Architecture Letters*, 2012.