zTT: Learning-based DVFS with Zero Thermal Throttling for Mobile Devices

Seyeon Kim
KAIST
seyeon625@kaist.ac.kr

Kyungmin Bin
Seoul National University
kmbin@snu.ac.kr

Sangtae Ha
University of Colorado Boulder
sangtae.ha@colorado.edu

Kyunghan Lee
Seoul National University
kyunghanlee@snu.ac.kr

Song Chong
KAIST
songchong@kaist.edu

ABSTRACT

DVFS (dynamic voltage and frequency scaling) is a system-level technique that adjusts voltage and frequency levels of CPU/GPU at runtime to balance energy efficiency and high performance. DVFS has been studied for many years, but it is considered still challenging to realize a DVFS that performs ideally for mobile devices for two main reasons: i) an optimal power budget distribution between CPU and GPU in a power-constrained platform can only be defined by the application performance, but conventional DVFS implementations are mostly application-agnostic; ii) mobile platforms experience dynamic thermal environments for many reasons such as mobility and holding methods, but conventional implementations are not adaptive enough to such environmental changes. In this work, we propose a deep reinforcement learning-based frequency scaling technique, zTT. zTT learns thermal environmental characteristics and jointly scales CPU and GPU frequencies to maximize the application performance in an energy-efficient manner while achieving zero thermal throttling. Our evaluations for zTT implemented on Google Pixel 3a and NVIDIA JETSON TX2 platform with various applications show that zTT can adapt quickly to changing thermal environments, consistently resulting in high application performance with energy efficiency. In a high-temperature environment where a rendering application with the default mobile DVFS fails to keep producing more than a target frame rate, zTT successfully manages to do so even with 23.9% less average power consumption.

CCS CONCEPTS

• **Human-centered computing** → **Ubiquitous and mobile computing systems and tools**; • **Software and its engineering** → **Power management**.

KEYWORDS

Mobile devices; DVFS; Deep reinforcement learning

MobiSys '21, June 24-July 2, 2021, Virtual, WI, USA

ACM ISBN 978-1-4503-8443-8/21/06…$15.00
https://doi.org/10.1145/3458864.3468161

ACM Reference Format:
Seyeon Kim, Kyungmin Bin, Sangtae Ha, Kyunghan Lee, and Song Chong. 2021. zTT: Learning-based DVFS with Zero Thermal Throttling for Mobile Devices. In *The 19th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '21), June 24-July 2, 2021, Virtual, WI, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3458864.3468161

1 INTRODUCTION

With the advent of mobile processors integrating CPU and GPU, high-performance tasks such as deep learning, gaming and image processing are running on mobile devices. To fully exploit CPU and GPU's capability on mobile devices, we need to utilize their processing capability as much as possible. However, it is challenging due to the nature of mobile devices whose users are sensitive to battery consumption and device temperature. Many researchers have studied techniques enabling energy-efficient operations in mobile processors, mostly at managing the temperature and power consumption below predefined thresholds.

DVFS (Dynamic Voltage and Frequency Scaling) is a technique that reduces heat generation and power consumption from the circuit by adjusting CPU or GPU voltage-frequency levels at runtime. To best utilize its benefits, many DVFS techniques [6, 9, 17, 20, 21, 24, 29, 33, 52] have been developed for mobile processors. Still, it is known challenging to implement a DVFS that performs ideally for mobile devices. There exist several reasons behind this difficulty.

First, conventional DVFS implementations stay mostly in the operating system kernel, thus becoming application-agnostic. However, applications and their demands can only define the optimal power distribution between CPU and GPU in a power-constrained platform. For instance, it is more efficient to allocate more resources to GPU than CPU when processing graphic tasks. When running mobile games, since CPU and GPU demands vary from game to game, their power distribution needs to change accordingly. Allocating more power than is necessary to either CPU or GPU will unnecessarily increase the power consumption and the system temperature. Therefore, to provide the best performance to mobile devices, it is essential to distribute the power budget judiciously among the processors by incorporating the actual performance of applications (i.e., application QoE).

Second, most DVFS implementations are not free from overheating problem [39, 44], especially on mobile devices. As mobile devices do not have active cooling methods such as fan control, if the device temperature goes above the level that TDP (thermal design power) can handle, thermal throttling occurs, which significantly degrades application performance. Figure 1 shows an

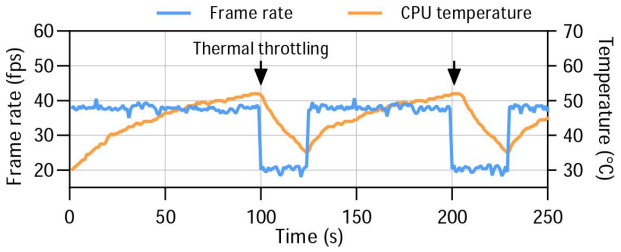


Figure 1: CPU temperature and frame rate while rendering a video. When overheated, CPU significantly lowers clock frequency by thermal throttling.

example of thermal throttling that happens to reduce heat generation when the temperature goes beyond a predefined threshold. To avoid critical frame drops, it is necessary for the device to predict and manage its temperature in advance. However, unlike desktops and servers, there is no one-size-fits-all approach since mobile devices experience complex thermal environments for several reasons, such as user mobility, holding methods, and external temperature [22]. It is also not straightforward to determine the impact of internal heat generation from processors to the device temperature as CPU and GPU are thermally coupled [31, 37]. As a result, most DVFS techniques [32, 36, 40, 42, 48] relying on a predefined temperature prediction model would not work properly for mobile devices. Furthermore, recent supervised learning-based approaches [8, 11, 45, 53] only give the adaptation ability to previously trained environments. Thus, their thermal management in mobile settings has no performance guarantee. To overcome the problems mentioned above, a new design of DVFS that continuously learns the application performance as well as environmental changes and adapts to those changes is required. In this paper, we introduce zTT (zero thermal throttling), a deep reinforcement learning (DRL)-based DVFS that quickly learns and adapts to application performance and environmental changes.

To realize zTT, we propose three techniques listed below based on our experimental observations on mobile platforms. First, we design a new DVFS framework that adjusts CPU/GPU voltage-frequency levels and learns the thermal throttling boundaries with the cooling capability and the temperature trend in a given environment. Second, we use DRL to handle the high-dimensional action and state spaces for learning. These spaces include CPU/GPU clock frequencies, temperatures, and power consumption levels. Third, we minimize the adaptation time to an environmental change, thus improving the convergence speed of DRL by leveraging the ideas from transfer learning and using sample copies from the past data.

We evaluate zTT in various mobile applications, including deep learning applications, on two mobile platforms, NVIDIA JETSON TX2 and Google Pixel 3a. From the experiments, we verify that zTT achieves higher performance in frames per second (FPS) with lower power consumption compared to state-of-the-art DVFS implementations. We also verify that zTT successfully prevents thermal throttling even in dynamic environments with abrupt temperature changes.

The main contributions of this paper are four-fold:

- We analyze how the performance and temperature characteristics of a mobile device change with applications and environmental changes through experiments and advocate the potential of learning-based DVFS.
- We formulate an optimization problem for a mobile device's power and thermal management as an MDP (Markov decision problem) that can handle the trade-off between application performance and power consumption with constraints on temperature limit.
- We propose zTT, a new practical DRL-based DVFS mechanism that adapts to applications and environments without any prior model or training and adapts quickly with historical data via transfer learning.
- We implement zTT on two mobile platforms, NVIDIA JETSON TX2 and Google Pixel 3a, and experimentally validate that zTT maximizes QoE of an application while adapting to environmental changes.

2 BACKGROUND

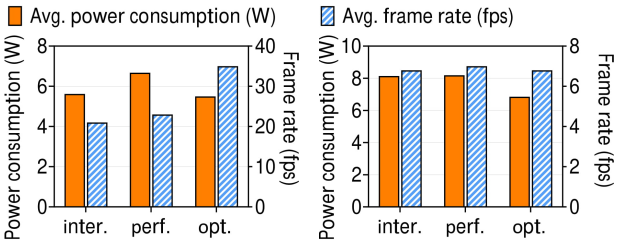
2.1 CPU and GPU DVFS Techniques

The power consumption of a processor, including CPU or GPU, is determined by its utilization and voltage-frequency (VF) level. So adjusting the VF level according to the workload makes the use of resources more efficient. DVFS is a well-known technique that dynamically adjusts the VF level of a processor for energy efficiency and thermal management. The VF scaling algorithm of a DVFS implementation, namely governor, is mainly built by the processor manufacturer and is controlled by the operating system. For instance, `ondemand` and `interactive` governors [7], which are default governors in Linux, adjust CPU frequency based on the predefined CPU utilization levels. Linux kernel also utilizes `simple_ondemand` governor for GPU, which is a simpler version of `ondemand`. The existing governors are known to ensure stable performance and reduce power consumption, but two limitations hinder them from being ideal. First, they do not take the performance of applications into account. Therefore, conventional utilization-based controls are not guaranteed to provide optimal performance to applications. Second, given a limited power budget, having separate governors for CPU and GPU make it hard to jointly utilize resources more efficiently.

2.2 Thermal Issues on Mobile Devices

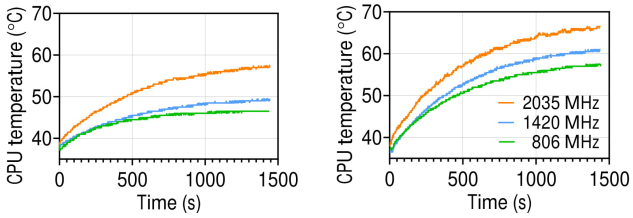
As the capabilities of mobile devices evolve, mobile processors become more powerful and denser within small package spaces.

The heat generated in such a compact area is tough to manage, especially when performing heavy computations. Furthermore, the introduction of 5G networks using high-frequency, high-speed mmWave puts the devices at a higher risk of overheating. Thermal management ensuring no severe performance degradation in such a situation is becoming more critical. Using a fan for heat management is very common in most computing platforms, but most mobile devices use heat sinks (or heat pipes) for heat dissipation, given the limited thin space. Unfortunately, even with heat sinks of proper sizes, mobile devices are not free from overheating. The mobile device's temperature characteristics are affected by various factors such as environmental temperature, holding method, protective case, and applications [22], thermal coupling among processors and battery [50] and the material of the surface in contact. Despite



(a) Video rendering (b) YOLO

Figure 2: Average power consumption and frame rate for video rendering and image detection application (YOLO v3). Two default CPU DVFS techniques (`interactive`, `performance`) and `optimal` frequency scaling are used for comparison.



(a) Video rendering (b) YOLO

Figure 3: CPU temperatures of JETSON TX2 with different CPU clock frequencies. CPU temperature shows a significant difference by applications: a) video rendering with H.264 codec and b) image detection application YOLO v3.

substantial research efforts on the thermal management of mobile devices [5, 30], these factors leave the problem of overheating in mobile devices unsolved. As a result, thermal throttling, a compulsory thermal management technique, is still widely used. It drastically reduces the processor clock frequency when the processor temperature rises above a certain threshold, leading to an immediate temperature drop. Thermal throttling is powerful but far from being ideal because it hurts application performance and user experience. Implementing a DVFS that can produce maximal application performance while persistently keeping the device temperature within its operational range is desirable.

3 OBSERVATION

We conducted preliminary measurements to understand the inefficiency of existing DVFS schemes and present how the performance and temperature characteristics change with the application and environment. All experiments were conducted on NVIDIA JETSON TX2 running on Linux 16.04 (kernel 4.4) and Google Pixel 3a running on Android OS 9.0 (kernel 4.9). The measurements show that there is an enormous opportunity to improve energy efficiency while preventing thermal throttling through performance and temperature prediction by learning their characteristics.

3.1 Inefficiency of Existing DVFS Schemes

Application performance on mobile devices depends heavily on both CPU and GPU and how their clock frequencies are controlled. In the early stage of mobile platforms, the CPU had to handle most operations while GPU was only assigned for graphics-related tasks. However, general-purpose GPU (GPGPU) enabled mobile devices to run applications requiring heavy computation such as deep learning, mobile game, and real-time image processing with GPU. In these applications, task assignment to CPU and GPU is a black box to the software developers even though it is critical to coordinate CPU and GPU for optimal performance.

Figure 2 shows the average FPS and total power consumption of JETSON TX2 while rendering a video with H.264 codec and YOLO v3 [1] [34] with different DVFS governors. We use `simple_ondemand` governor for GPU and three CPU frequency scaling techniques. The `Interactive` and `Performance` governors are default governors in Linux. `Optimal` is a virtual governor simulated to improve energy efficiency based on our measurements. For video rendering, `Performance` sets the CPU clock at the maximum, but it gets little frame rate improvement compared to `Interactive`, at the cost of 21.3% more power consumption. `Optimal`, on the other hand, boosts the GPU clock slightly and achieves even higher frame rate compared to `Performance` while consuming power similar to `Interactive`. For running YOLO, both `interactive` and `performance` governors have similar power consumption and frame rates while the `optimal` governor consumes 19.5% less power for the same frame rate level by properly lowering the CPU clock. This gap of the existing DVFS techniques comes from the fact that they do not take the performance characteristics of applications into account, and DVFS governors for CPU and GPU do not operate collaboratively. Our observations imply that if it can predict which resource would be the bottleneck for the application performance, it would be possible to control CPU and GPU much more energy-efficiently.

[1] A deep-learning based object detection application

3.2 Thermal Characteristics

Several factors determine the temperature of a processor. First, the higher the heat generation of the mobile processor, the higher the overall temperature. Figure 3 (a) shows the CPU temperature of JETSON TX2 with different CPU clock frequencies while rendering a video. Higher frequencies consume more power, resulting in generating more heat. Even with the same clock frequency setting, the power consumption varies depending on processor utilization, implying that temperature characteristics can vary from application to application. Figure 3 (b) shows that even with the same clock frequency, the YOLO application demanding higher processor utilization makes the device's temperature higher.

Second, processor temperature is affected by thermal coupling between mobile processors such as big.Little CPU cores and GPU cores. In the case of mobile devices such as smartphones, the heat generated by one processor significantly affects the others because mobile processors are densely packed in a compact space. So it is common for them to share a heat sink. Figure 4 shows the steady-state temperature of two mobile devices with different GPU clock frequencies with a fixed CPU frequency. Figure 4 shows evidence that the mobile CPU and GPU are thermally coupled. This coupling

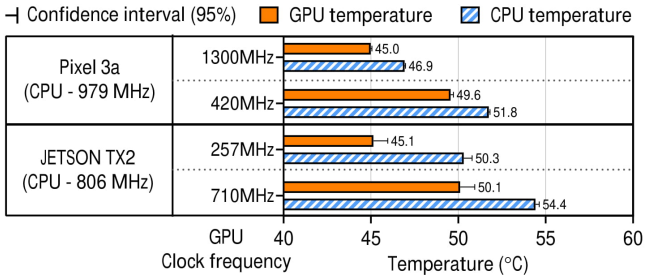


Figure 4: CPU temperature increases when the GPU clock frequency increases due to CPU-GPU thermal coupling even if CPU clock frequency remains at the same level.

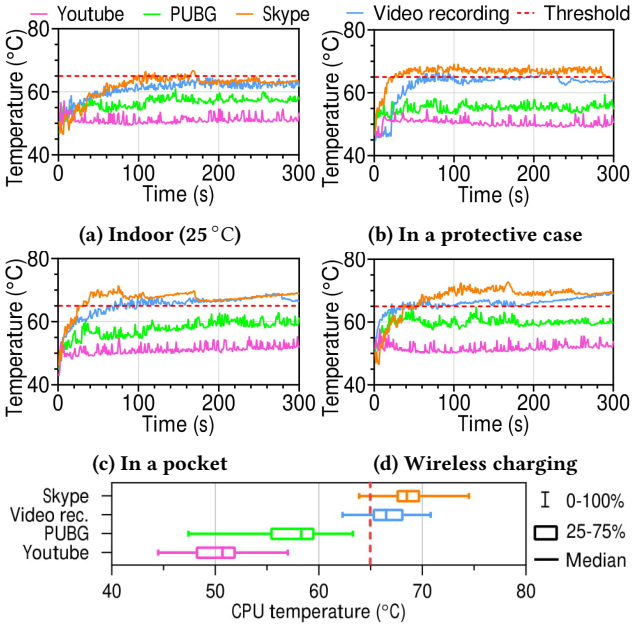
can be loose or tight, depending on the type and structure of the mobile devices.

Finally, the thermal characteristics of mobile devices are sensitive to environmental changes. To quantitatively study the effect of the environment on the device temperature, we experimented with applications running on the Pixel 3a's CPU/GPU under various settings (Figure 5). As shown in Figure 5 (a), the temperature can be managed reasonably in a low-temperature environment, but managing the temperature becomes much more difficult in situations where the phone is in a protective case or in a pocket (Figures 5 (b) and (c)). Figure 5 (d) shows that the heat generated from charging also affects CPU temperature. In particular, Skype and video recording cause thermal throttling(>65 °C). As summarized in Figure 5 (e), the temperature characteristics can vary significantly by applications and by environments, and overheating indeed happens in real-life situations.

4 zTT DESIGN

We present the design of zTT, an effective DRL-based DVFS system that quickly learns and adapts to application performance and environmental changes without incurring thermal throttling. Figure 6 illustrates the purpose and impact of learning in zTT. The lattice points within the total power budget curve for a mobile device [2] represent all available CPU/GPU power consumption combinations. The graph shows that the better the cooling, the more combinations are usable, thus providing better performance for an application. To find out the best possible combination at the moment, zTT learns the environment and application performance. Environment learning is to learn about environmental changes that frequently happen over time and by device mobility. Even with the same CPU/GPU clock frequency combinations, different environments can lead to varying temperatures of mobile processors. Therefore, learning from the environment means predicting how the temperature will change in the current situation when using a given CPU/GPU clock frequency combination. Application learning is to learn application performance characteristics per application over CPU/GPU resource requirements. Performance lines from two different applications (i.e., CPU-intensive and GPU-intensive applications) are conceptually drawn in Figure 6, and these are what zTT learns.

[2] The total power budget curve of a mobile processor is defined by CPU and GPU power consumption points from which no more increase can be allowed by TDP (thermal design power) of the mobile device. This curve is nonlinear in practice as in Figure 6.



(e) Application-specific steady-state temperature for four environments (a), (b), (c) and (d).

Figure 5: CPU temperature according to environment and application when the clock frequency of CPU/GPU is fixed. The red lines indicate the threshold of thermal throttling.

To realize zTT that can achieve both learning goals in a single framework, we below present our step-by-step design for our optimization problem formulation, its transformation to MDP, and its transformation to model-free RL and DRL in detail.

4.1 Problem Formulation

We start by formulating our optimization problem that incorporates the observations in Section 3.

We define a utility function $U(t)$ to represent the application performance as user QoE at time t. This utility function can be interpreted differently by users and by applications [51]. Although there are no golden rules for defining user QoE, many existing studies define the QoE as guaranteeing application performance beyond a certain level (e.g., required frame rates for video/gaming applications) [9, 13, 38]. We also use $U(t)$ for video and gaming applications to ensure their frame rates above a target frame rate at time t. The power consumption $P(t)$ is defined by the total instantaneous power consumption from CPU and GPU at time t. Then, the problem of maximizing $U(t)$ and minimizing $P(t)$ on average for T while guaranteeing to avoid thermal throttling becomes as follows:

$$\textbf{(P0):} \max_{\pi} \frac{1}{T}\sum_{t=1}^{T}\{U(t)+\frac{\beta}{P(t)}\} \quad (1)$$

$$\text{s.t.} \quad T_C(t) \le T_{C,th}, \forall t \quad (2)$$

$$T_G(t) \le T_{G,th}, \forall t \quad (3)$$

In Eq. (1), π denotes the set of available policies controlling the CPU/GPU frequency combinations from $t = 1$ to $t = T$, ($f_C(1)$,

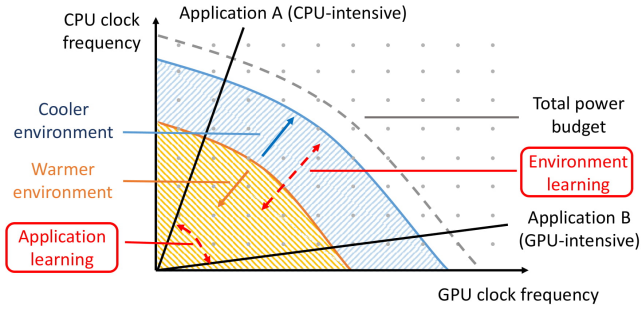


Figure 6: The purpose of application and environment learning. The range of CPU/GPU clocks expands in a cooler environment within the total power budget.

$f_G(1), \ldots, f_C(T), f_G(T))$, where $f_C(t)$ and $f_G(t)$ denote the configured frequency of CPU and GPU at time t, respectively. β is a trade-off weight. For instance, a user who prefers to maximize application performance regardless of the power consumption may set β closely to 0. A larger β gives more weight to the power consumption. Constraints (2) and (3) are hard-bounds for zero thermal throttling that enforces the CPU and GPU temperatures, $T_C(t)$ and $T_G(t)$ to be within their threshold temperatures, $T_{C,th}(t)$ and $T_{G,th}(t)$. The threshold temperatures depend on the chipset specifications, and getting overheated beyond these temperatures may damage the chipsets.

MDP Transformation. The problem **P0** can be converted to MDP **M0** $:=< S, A, R, p, \gamma >$ consisting of state, action, reward, transition probability, and discount factor as in Table 1, and its Bellman optimality equation follows Eq. (4).

Table 1: State, Action and Reward for zTT's MDP, M0

$s(t) \in S$	$f_C(t), f_G(t), T_C(t), T_G(t), P_C(t), P_G(t), x(t)$
$a(t) \in A$	$f_C(t), f_G(t)$
$r(t) \in R$	$U(t) + \frac{\beta}{P(t)} + W(t)$

$$V^{\pi^*}(s) = \max_a \{R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^{\pi^*}(s') \quad (4)$$

The objective of **M0** is to find an optimal policy π^* satisfying Eq. (4), where function $V^{\pi^*}(s)$ is the value function of state s for an optimal policy π^*. In order to handle the constraints, Eq. (2) and (3), of **P0**, a reward compensation function, $W(t)$, is added in the reward R. Setting $W(t) = 0$ makes the optimal solution of **M0** the same as **P0** for $\gamma = 1$. We will discuss our novel reward design with $W(t)$ in the next Section 4.2.

Transformation to Model-free RL. Solving **M0** is challenging because it is practically impossible to precisely model the transitions between temperatures, with the configured CPU and GPU frequencies, and the MDP actions of **M0**. Thus by replacing value function $V(s)$ with Q-function $Q(s,a)$, we find the optimal solution of **M0** through solving **M1**:

$$\textbf{(M1):} \max_\pi \mathbb{E}_\pi [Q_\pi(s,a)] \quad (5)$$

Q-function in **M1** is defined by the sum of expected reward for each action taken at a state. Q-learning is a popular model-free RL algorithm that does not require probabilistic modeling for state transitions, which is essential for an MDP. Although Q-learning is known to find an optimal policy providing a series of actions maximizing the expected reward, it has a known limitation in its size of the state and action spaces such as an infinite number of states in **M0** due to having continuous values (e.g., power consumption, temperature). Deep Q-Network (DQN) [25], a popular DRL algorithm, approximates Q-function with DNN to address the state and action size issue of Q-learning. To exploit the idea of DQN, we customize our state and actions spaces and reward function for **M0**. We detail this in section 4.2.

Furthermore, to deal with practical scenarios in which the environment and application change frequently, we provide adaptability to zTT by adopting recent advances in DQN to **M1** in a customized manner. We detail this in section 4.3.

4.2 State, Action and Reward Design

zTT solving **M1** with DQN can adapt to application performance and environmental changes while conventional RL methods will not work due to unique constraints such as hard bounds on thermal limits. Below we describe how we customize DQN for zTT by redefining state, action, and reward for **M1**.

4.2.1 *State.*

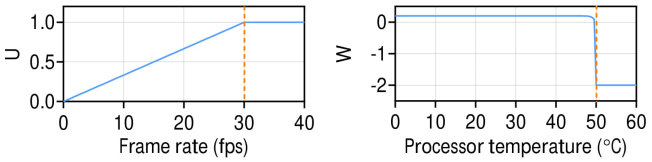
Our new states are defined by a tuple containing seven parameters: $(f_C(t), f_G(t), T_C(t), T_G(t), P_C(t), P_G(t), x(t))$. Targeting video and gaming applications, we observe the frame rate at time t, $x(t)$, and the power consumption of CPU/GPU at time t, $P_C(t)$ and $P_G(t)$, to manage the total power consumption $P(t) = P_C(t) + P_G(t)$. To keep an eye on overheating, temperatures of CPU and GPU, $T_C(t)$ and $T_G(t)$, are measured at every t. We also monitor the clock frequency of CPU/GPU, $f_C(t)$ and $f_G(t)$, to help approximate resource usage.

4.2.2 *Action.*

We define the set of actions to be the clock frequency of CPU and GPU. Within the threshold temperatures of CPU and GPU, actions are taken using a modified ϵ-*greedy* method. With probability ϵ, zTT's action does exploration, and with probability $1 - \epsilon$, it does exploitation. When the temperature goes close to the threshold, cool-down action, detailed below, is used to lower the temperature and prevent overheating.

Exploration and exploitation. We let *exploration* take uniform random actions over the entire clock frequency range. At the beginning of the training, the sample collection phase kicks in for batch training, so the only *exploration* is taken to collect samples. After minimum required samples are collected, *exploration* is performed much less, with the probability of ϵ. In the exploitation phase, an action $a(t)$ that maximizes $Q(s(t), a(t))$ is selected given state $s(t)$ at every t.

Cool-down action. In conventional ϵ-*greedy*, the exploration is done without considering its result. Therefore, an adversarial action that results in a negative reward can be chosen. This adversarial action is considered important as it expands sampling for training, but it can cause thermal throttling in our case. To avoid such critical happenings, we let zTT randomly select one of the clock frequencies lower than the current clock frequency when approaching the



(a) Reward function U by the frame rate of an application **(b) Reward function W by processor temperature**

Figure 7: Reward function example when the target FPS is 30 and threshold temperature is 50 °C while rendering a video. User QoE is 1 when the target frame rate is achieved. Reward by temperature becomes negative if it exceeds the threshold temperature.

threshold temperature, ensuring that the device stays under the throttling threshold. This approach prevents drastic performance degradation from thermal throttling, thus keeping the application performance much more stable during adaptation. When the temperature decreases through the cool-down action, the entire range of clock frequencies is searched through exploration. Note that cooling down with a single fixed low clock in thermal throttling results in severe sample imbalance, leading to inefficiency compared to our approach.

4.2.3 *Reward.*

We propose a dedicated reward design for zTT, which introduces $W(t)$ interacting with $U(t)$ and $P(t)$ as in Table 1. In cases where frame rates are critical for the application QoE (e.g., video and gaming apps), zTT defines the utility function as in Eq. 6. Eq. 6 is designed to have a larger utility for a better frame rate $x(t)$ until it gets saturated by an application-specific target frame rate, X_t. For video rendering, Figure 7 (a) shows that if the perceived FPS exceeds X_t (e.g., 30 in this example), we assume that the user satisfaction would not improve further[3] while consuming more power.

$$U(t) = \begin{cases} 1, & if\, x(t) \geq X_t \\ \frac{x(t)}{X_t}, & otherwise \end{cases} \tag{6}$$

Regarding the temperature, zTT must learn how to predict whether the temperature of the next moment would exceed the threshold or not. Therefore, with $W(t)$, we design a negative reward for those actions that result in temperature threshold violations. In Figure 7 (b), we exemplify $W(t)$ given that the threshold temperature is 50 °C. When the temperature is below the threshold, $W(t)$ gives a constant positive reward, while $W(t)$ is given a sharp drop in its reward toward a non-negligible negative value near the threshold temperature. Eq. 7 shows this design for CPU, which can be applied to GPU as well. zTT evaluates $W(t) = W_C(t) + W_G(t)$. We use $tanh(\cdot)$ instead of a step function to reflect the small thermal budget near the threshold.

$$W_C(t) = \begin{cases} \lambda \cdot tanh(T_{C,th} - T_C(t)), & if\, T_C(t) < T_{C,th} \\ -10 \cdot \lambda, & otherwise \end{cases}^4 \tag{7}$$

[3] Using $U(t)$ increasing gradually beyond X_t in our framework is also possible.
[4] We set $\lambda = 0.2$ through our experiments.

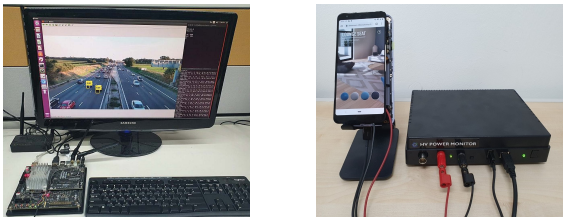
Algorithm 1: zTT Algorithm

Constants: $T_{C,th}, T_{G,th}, X_t$
Initialize: Replay memory D to capacity $|D|$, Neural network θ, Target network $\theta^- = \theta$ and $\epsilon = 1$
Initialize: Q-function with θ
$s(1) = (f_C(1), f_G(1), T_C(1), T_G(1), P_C(1), P_G(1), x(1))$
for $t = 1, 2, ..., T$ **do**
  /* $\epsilon-greedy$ method or cool-down actions */
  Get action $a(t)$
  Observe $s(t+1)$
  Calculate reward $r(t)$
  Append sample $(s(t), a(t), r(t), s(t+1))$ in D
  if $r(t) \leq 0$ *or* $r(t) \geq 1$ **then**
    /* Additional Copy of sample to highlight it */
    Append copy $(s(t), a(t), r(t), s(t+1))$ in D
  /* Batch-training with the target network technique */
  if *more than* $|M|$ *samples are collected* **then**
    Sample random minibatch M of (s, a, r, s') from D
    /* Locking θ parameter during the updates. */
    for $(s, a, r, s) \in M$ **do**
      Set $y = r + \gamma \max_{a'} Q(s', a'; \theta^-)$
    Perform gradient descent on $\sum_{(s,a,s',y)\in M}(y - Q(s, a; \theta))^2$ with respect to θ
    Every C steps, $\theta^- = \theta$
    Every k steps, reset learning rate
    ϵ decays

4.3 Providing Adaptability to DQN

States and actions were carefully defined to solve our **M1** with DQN as in Section 4.2. We design the DQN for zTT to be a fully-connected neural network. At the beginning of learning, zTT takes random actions and stores $(s(t), a(t), r(t), s(t+1))$ samples in its replay memory. When the minimum required samples are collected, it performs batch training and approximates its Q-function. To improve the stability and convergence of DQN, we adopt transfer learning and use sample copies from historical data. In DQN, during the updates of the RL equations, the Q-function of our interest (i.e., target function) can change due to the changes made to the underlying DNN, making the training difficult. The target network technique locks the target function parameters during the updates and replaces them with the latest network's values every few steps.

To improve the diversity of samples, the probability of exploration, ϵ, starts from 1 and decreases gradually over time. By doing so, zTT learns about environments and applications, and it gets the ability to predict thermal headroom (i.e., thermal margin to the threshold temperature). However, this does not sufficiently provide the adaptation ability to zTT because a DQN algorithm is originally designed to solve MDP, where its model does not change over time. That is, the algorithm assumes that the transition probabilities between states do not change in the MDP. In our problem **M1**, the transition probability of MDP can change as environments and applications vary. To overcome this limit, we adopt two more techniques to implement the learning in zTT that enables adaptation to varying environments or applications.



(a) NVIDIA JETSON TX2 (b) Google Pixel 3a

Figure 8: The testbed setups for NVIDIA JETSON TX2 and Pixel 3a with a Monsoon power monitor.

Transfer learning. We develop a transfer learning-based approach[5] for our DQN algorithm. The idea behind this is that even if the environment changes, only a few of the entire neural network parameters will change because the structure of the model might be substantially similar. We verify in Section 6 that our DQN-based algorithm works reasonably well in changing environments with mostly the same neural network parameters, including the same input and output layers, and small updated parameters.

Using sample copies for faster convergence. When the environment changes, the mobile device's thermal characteristics change accordingly: there may exist two different situations. First, if the situation changes to worse heat dissipation, the thermal budget may suffer from satisfying QoE for the application. Second, the situation can change to benefit from better cooling. In this case, applications are given the opportunities to exploit higher clock frequencies, thus better QoE probabilistically. Samples collected from these two types of environmental changes should be emphasized in training and adaptation so that those changes can later be adopted more quickly. To do so, we append a set of additional copies of those unique samples to the replay memory to highlight them during the batch training. Additionally, to prevent the learning rate from stagnating because the gradient descent algorithm decreases it over time for convergence, we reset the learning rate periodically. Thus, even when the environment changes, it is possible to prevent being trapped in the previous environment's optimal point. These techniques increase the sample efficiency in the event of environmental changes, leading to quicker training and better adaptation ability. How zTT works is detailed in Algorithm 1.

5 IMPLEMENTATION

We implement zTT with Python 3.5 and Keras 2.2 [16] on Google Pixel 3a and NVIDIA JETSON TX2 (Figure 8). zTT controls the CPU and GPU clocks of these devices whose specifications are shown in Table 2. The CPU of JETSON TX2 has a clock frequency range of 0.3 GHz to 2.0 GHz. The big and LITTLE CPU of Pixel 3a has a range of 0.3GHz to 1.9GHz and 0.3GHz to 1.7GHz, respectively.

Figure 9 shows an overview of zTT implementation. zTT agent runs separately from other mobile applications. DQN in zTT agent takes action based on our modified ϵ-*greedy* method or do exploration to cool-down as described in Section 4.2.2. The agent then calculates the reward, collects samples of the observed state, and puts them into the replay memory. After that, DQN is trained with

[5] Transfer learning is a technique that improves learning a predictive function for a new sample using a predictive function that has already been trained on prior samples.

Table 2: Specifications of JETSON TX2 and Pixel 3a.

Device	JETSON TX2	Pixel 3a
CPU	NVIDIA Denver2 + ARM Cortex-A57	ARM Cortex-A55(LITTLE) + ARM Cortex-A75(big)
GPU	NVIDIA Pascal GPU	Adreno 615
Memory	8GB DDR4	4GB LPDDR4X
OS	Ubuntu 16.04	Android 9.0 Pie

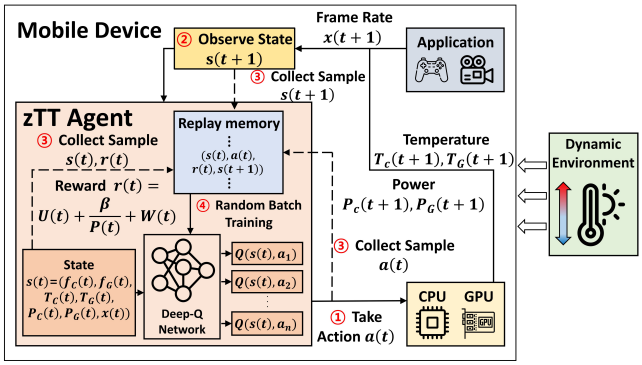


Figure 9: Overview of zTT implementation

random batch sampling from the replay memory. These procedures are repeated until the predefined number of iterations is over.

To observe the state $s(t)$, JETSON TX2 is programmed with Python 3.5 to monitor its power consumption with INA3221 module [47] [6], temperatures, [7], clock frequencies of CPU and GPU with built-in sensors which are accessible through `sysfs` [26] in the Linux kernel. We also monitor the clock frequencies of big.LITTLE CPU and GPU, temperatures through `sysfs` in Android kernel on Pixel 3a. We measure the power consumption of Pixel 3a by using a Monsoon digital power monitor [27] as in Figure 8.

For our experiment on video rendering, we use OpenCV2 [28] to measure the frame rate to learn and control application QoE. For a 3D rendering application, we use WebGL Aquarium [15] running on JETSON TX2 and measure the frame rate through the JavaScript function `requestAnimationFrame()`, which is called when the web browser renders every frame. For the Google Pixel 3a, we measure the frame rate of applications by collecting the logs generated by the `SurfaceFlinger` [2] Android system service. Detailed implementation settings and source code are available online [8].

6 EVALUATION

We experiment zTT on JETSON TX2 and Google Pixel 3a to see how well zTT adapts to various real-world applications and environmental changes while preventing overheating and saving energy. The details of experimented applications are summarized in Table 3.

[6] Read "/sys/bus/i2c/drivers/ina3221x/[Devicelabel]/in_power_input" 100 times per second and average them.
[7] Read "/sys/devices/virtual/thermal/thermal_zone[label]/temp"
[8] https://github.com/ztt-21/zTT

Table 3: Experimented applications and devices.

Application	Description	Device
Aquarium [15]	WebGL-based 3D object rendering	JETSON TX2
YOLO [35]	Deep learning-based object detection	JETSON TX2
Video rendering	Rendering a video with OPENCV2	JETSON TX2
Showroom VR [23]	WebGL-based 3D object rendering	Pixel 3a
Skype [43]	Video call	Pixel 3a
Call of duty 4 [1]	3D Mobile game	Pixel 3a

6.1 Evaluation Setup

For JETSON TX2, `interactive` and `simple_ondemand` governor are used for CPU and GPU respectively, and we denote them to `default`. Those governors are default governors in recent Linux kernels and are known to reflect application loads measured from processor usage statistics to control the CPU frequency. For the Pixel 3a phone, `schedutil`, the default CPU governor from Android Pie is used. The `schedutil` governor also controls the clock frequency based on the processor usage statistics while utilizing the Energy-Aware Scheduling (EAS) algorithm [3], which has been applied to the latest Android devices. EAS improves energy efficiency by allocating adequate CPU cores (e.g., big and LITTLE) using its CPU energy consumption model. The default governor for GPU is `msm_adreno_tz` governor, which is based on ondemand governor but focuses more on performance. These settings are `default` for Pixel 3a.

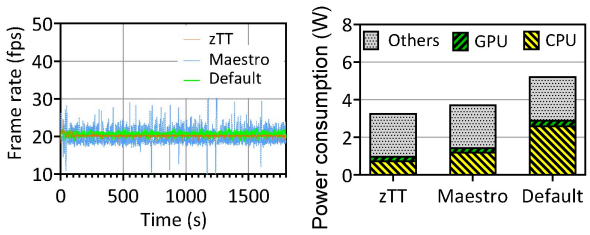
`Maestro` [38] is a recent DVFS mechanism that aims at guaranteeing QoS of mobile apps under thermal constraints by preventing thermal throttling. It proactively detects the QoS level of an application, which is susceptible to thermal throttling. It then performs the PI (proportional-integral) control-based DVFS with thread mapping to meet the QoS level. Maestro is the current state-of-the-art in thermal limit-aware DVFS mechanisms and is used for our comparison.

To set the target frame rate in the `default` setting, the `waitKey()` function of OPENCV2 was used for video rendering. For the YOLO, we used a tiny-YOLO model which is a light-weight version of the YOLO to meet the target frame rate of 15 fps approximately. For the Aquarium benchmark, we adjusted the number of fishes on the screen. In the case of android apps in this paper, it aims to operate at a refresh rate of 60 Hz without any additional settings.

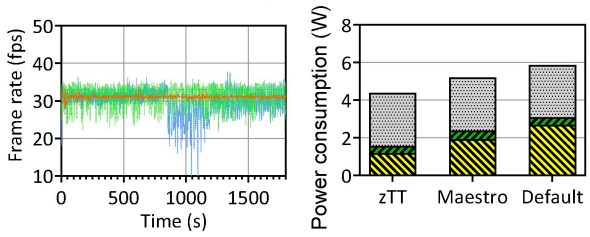
6.2 Learning Application QoE

We first evaluate whether zTT can learn an application and adapt to its performance requirements. To focus on learning applications, we rule out the environmental effects by conducting experiments in a heavily-cooled environment.

Figure 10 shows the FPS and power consumption for video rendering by OPENCV2. We test the target FPS of 20 and 30. We compare zTT with `Maestro` and `default` on JETSON TX2. zTT performs very stably while minimizing power consumption by

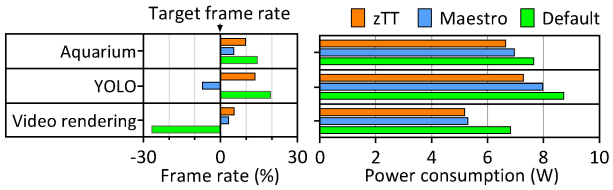


(a) Setting target FPS at 20

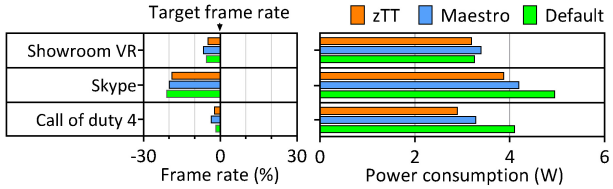


(b) Setting target FPS at 30

Figure 10: Frame rate and total power consumption when rendering a video with different target frame rates with sufficient cooling. For comparison, `Maestro` and `default` are used for JETSON TX2.



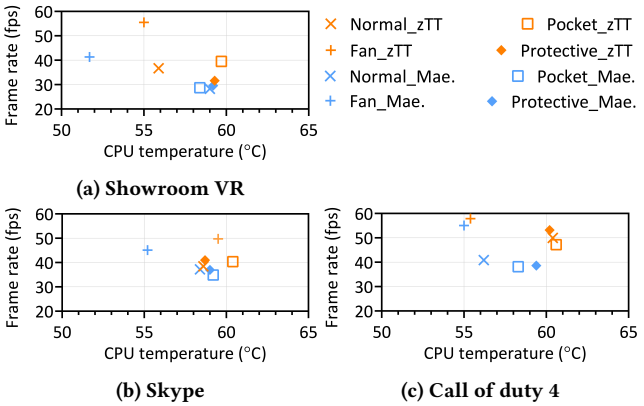
(a) JETSON TX2 runs each app with different target FPS (20 FPS for Aquarium, 15 FPS for YOLO, and 30 FPS for video rendering). zTT saves most power consumption over `Maestro` and `default` while achieving the target FPS.



(b) Pixel 3a runs each app targeting 60 FPS, which is physically not achievable. While minimizing FPS degradation, zTT saves more power consumption.

Figure 11: Frame rate and total power consumption for mobile apps running on JETSON TX2 and Pixel 3a.

efficiently using the CPU and GPU resources. The `Maestro` and `default`, on the other hand, experience difficulties in utilizing the specific resource requirements. The `default` scales up the clock frequency, but it still fails in achieving 20 or 30 FPS consistently. `Maestro` saves power consumption but shows instability in attaining the required frame rates. We can see that zTT makes similar performance compared to `default` governor with stricter performance guarantee while saving power, on average, 37.4% and 23.9% for the target FPS of 20 and 30, respectively.

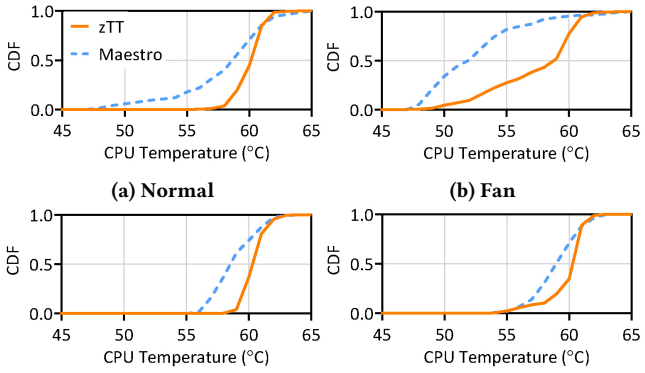


(a) Showroom VR

(b) Skype

(c) Call of duty 4

Figure 12: The average frame rate and CPU temperature of zTT and `Maestro` in four environmental settings for each application. zTT achieves a higher frame rate under the same environment with higher CPU temperature within the target temperature.



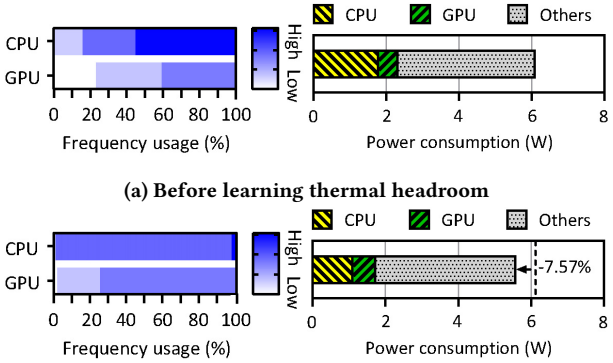
(a) Normal

(b) Fan

(c) Pocket

(d) Protective case

Figure 13: The CDF of the average CPU temperature in three applications under the same environment. zTT shows higher CPU temperature distribution across all environmental settings.



(a) Before learning thermal headroom

(b) After learning thermal headroom

Figure 14: Frequency usage and power consumption before and after predicting thermal headroom when rendering a video indoors (25 °C). After learning the thermal headroom, zTT lowers the CPU clock frequency to reduce power consumption, thereby decreasing the steady-state temperature.

We also check the performance of zTT and the others with several real-world mobile applications requiring different target FPSes on JETSON TX2. As shown in Figure 11 (a), zTT guarantees the target frame rate for three applications and saves more power over `Maestro` and `default`. After that, we slightly change the experiment to know each governor's behavior when each application sets the target FPS very high on Pixel 3a. This experiment aims to see how each governor behaves when the objective is not achievable due to the platform limitation. The target FPS of three applications is set to 60 FPS, which is not achievable on Pixel 3a. In Figure 11 (b), compared to the others, we can see that zTT tries to meet the target frame rate as closely as possible to minimize QoS degradation while reducing the total power consumption as much as possible.

6.3 Learning Static Environments

We evaluate zTT with three applications (Table 3) in four different environmental settings (i.e., normal, fan, pocket, protective case) on the Pixel 3a device. We compare zTT with `Maestro`. We set the threshold CPU temperature at 65 °C. In Figure 12, zTT achieves higher FPS compared to `Maestro` within the CPU temperature threshold of 65 °C for all cases. It indicates that zTT can more aggressively utilize thermal budget than `Maestro`; it increases the CPU clock aggressively as long as the temperature is still within the threshold. Figure 13 shows this aggressive utilization of the thermal budget. zTT shows higher temperature distribution than `Maestro` in different environments. It reveals that zTT can learn the environment and adaptively increase CPU or GPU frequency under a given thermal budget to achieve higher user QoE while `Maestro` shows relatively lower thermal budget utilization and QoE.

In Figure 14, we see how zTT controls the frequency of CPU and GPU before and after learning the thermal headroom. After learning, zTT figures out that the current policy can cause overheating and thus lowers CPU clock frequency to stabilize the temperature below the threshold, resulting in the reduction in power consumption by 7.5% without causing any thermal throttling.

6.4 Learning Changing Environments

We verified that zTT can learn the thermal characteristics of a static environment with application resource requirements in Sections 6.2 and 6.3. However, unlike the servers and PCs located in a static position, mobile devices are carried to different environments. We first evaluate the impact of transfer learning, sample copies, and convergence on improving zTT's adaptation speed. We then show the experimental results from the cases with environmental changes.

6.4.1 *Transfer Learning.*

Learning a new environment from zero knowledge is not only inefficient but also degrades application QoE. To make zTT more efficient, we use the transfer learning technique explained in Section 4.3 that boosts the adaptation speed. We evaluated the impact of transfer learning in reducing adaptation time on JETSON TX2. Figure 15 shows the adaption time improvement with transfer learning

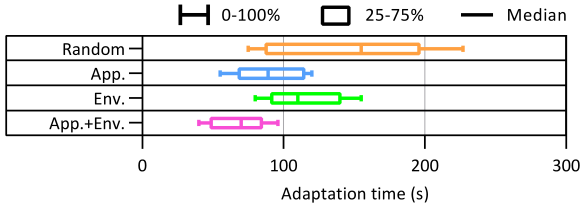
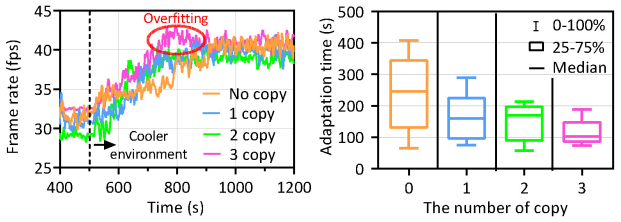


Figure 15: Adaptation time with transfer learning (no information (random), the app alone, the environment alone, and both app and environment).



(a) Frame rate **(b) Adaptation time**

Figure 16: Adaptation speeds according to the number of sample copies. While rendering a video, we change the environment from indoors to cooler outdoors. After adaptation, zTT makes a new policy with higher clocks and improves the frame rate.

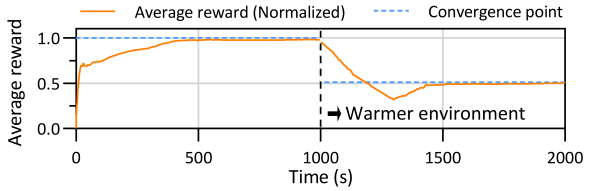


Figure 17: Average reward by zTT with video rendering application on JETSON TX2. Convergence points are updated when the environment changes.

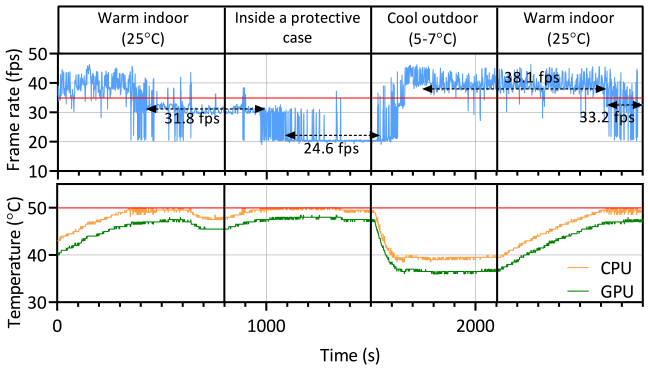


Figure 18: Frame rate and temperature of JETSON TX2 rendering a video while experiencing a number of environmental changes.

with additional knowledge (i.e., the application alone, the environment alone, and both application and environment). It shows that transfer learning reduces more than half compared to the baseline of random sampling, indicating that the learning time reduces significantly with additional knowledge.

6.4.2 Using Sample Copies for Faster Convergence.

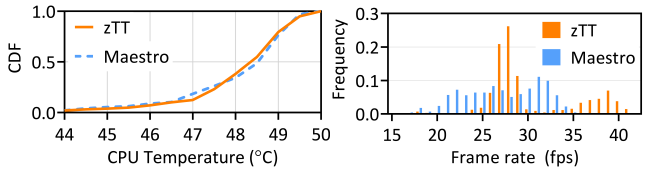
Replicating specific samples in the replay memory increases sample efficiency but is known to cause overfitting due to the so-called sample imbalance problem. Thus, we check how the number of sample copies affects adaptation when there exist environmental changes. For the rendering application, we vary the number of sample copies. Figure 16 (a) shows the frame rate when the environment changes from a warm indoor to a cool outdoor environment with the target of 35 FPS. As the number of sample copies increases, adaptation gets faster. However, for three or more copies, we observe that overfitting causes excessive resource usage. Figure 16 (b) shows the time when the frame rate controlled by zTT becomes stabilized at the target of 35 FPS. As shown, even with only one sample copy, the adaptation time reduces by half. We observe that using one or two sample-copies makes zTT sufficiently efficient in adapting to new environments.

6.4.3 Convergence.

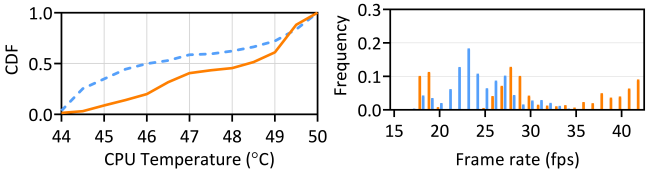
Deep Q-learning is considered converged when there are no more changes in Q-values or average reward over epochs. At such a point, deep Q-learning is usually evaluated with average rewards [4]. In Figure 17, we monitored the average reward for zTT over time while rendering a video on JETSON TX2. The dotted line is the convergence point measured at static environments with sufficient training phase. After initial training, the average reward asymptotically converges to the dotted line. At 1000 (s), experiencing environmental change, the average reward for the new environment follows an updated convergence point as incoming reward changes.

6.4.4 Case Study of Environmental Changes.

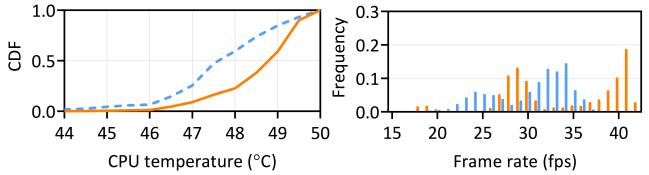
We let a mobile device experience three environmental changes: 1) warm indoors (25 °C), 2) cool outdoors (5-7 °C), and 3) inside a protective case. Figure 18 shows three environmental changes while rendering a video in JETSON TX2. Target FPS is set to 35, and the threshold temperature is set to 50 °C. When the thermal headroom is insufficient due to environmental changes (i.e., warm indoors to inside a protective case, or cool outdoors to warm indoors), overheating prevention through a cool-down action occurs around the threshold temperature. After learning the samples obtained from overheating prevention, zTT predicts the thermal headroom in the new environment and creates a new policy with lower frequencies. On the other hand, in a situation where the thermal headroom becomes more generous (i.e., a device being inside a protective case to cool outdoors), zTT seeks to improve the application QoE through exploration. In such a case, if the target FPS is satisfied, a new frequency scaling policy is created to save more power. We can see that zTT can adapt to the new environment and utilize resources efficiently while avoiding thermal throttlings. Figure 19 represents how aggressively the thermal budget will be used by zTT in a periodic and extremely changing environment. An external portable fan was periodically turned on and off at a distance of 0.8m from JETSON TX2 to create a substantial environmental change



(a) The portable fan is turned off.



(b) The portable fan is turned on and off every 10 minutes.



(c) The portable fan is turned on and off every 3 minutes.

Figure 19: CDFs of CPU temperature (left) and histogram of frame rate (right) when rendering a 60-minute video on JETSON TX2. The external environment changes periodically with a portable fan.

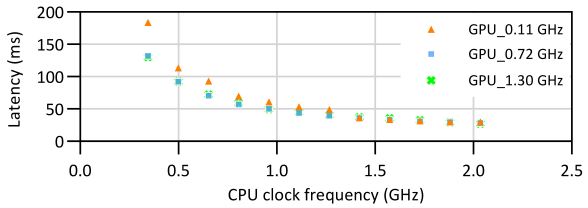


Figure 20: Latency to decide action with zTT according to CPU and GPU clock frequencies.

in thermal condition while rendering a 60 minutes-video targeting 35 FPS. As shown in Figure 19 (a), when the fan is off, the thermal budget is insufficient to hit 35 FPS, but zTT overall shows higher frame rates than `Maestro`. In Figure 19 (b) and (c), the portable fan is turned on and off every 3 and 10 minutes, respectively. As the environment changes, `Maestro` stays longer at lower temperatures and cannot fully utilize the thermal budget, while zTT recognizes the environmental change and uses the thermal budget aggressively. Therefore, zTT achieves higher frame rates stably without being overheated thanks to its ability to adapt to environmental changes.

6.5 Overhead

We evaluate the overhead of zTT in JETSON TX2. The average power consumption of running the zTT algorithm ranges from 45 mW to 150 mW. Considering that the power consumption for running an app in JETSON TX2 is between 5W to 9W as in Figure 11 (a), this overhead is acceptable given its advantage in power saving. Note that we have not applied any neural network compression techniques to our zTT implementation meaning that there is room for additional power saving. We also measure the time for zTT to decide an action. As shown in Figure 20, determining an action took about 183 ms when the GPU and CPU clock frequencies were the lowest and about 30 ms when their clock frequencies were the highest. Note that the real-time state parameters collected by zTT should not change while deciding an action. If the measured temperature changes while determining an action, it can confuse learning and slow down the convergence. Since we use a 1-second average for the state parameters such as temperature, power consumption, and frame rate in our zTT implementation, the latency overhead of zTT (<200ms) is practically acceptable.

7 DISCUSSION

Reducing initial learning time. Training from scratch, zTT takes exploration steps, which can cause QoE fluctuations. In practice, there is no need for much exploration as many samples for representative cases can be pre-populated or collected on mobile devices on the fly. Furthermore, using a pre-trained neural network, training time also can be significantly reduced. The meta-learning approach [12, 14] that can create a generalized neural network applicable to various environments with few samples can substantially reduce the steps for retraining. Once mobile device manufacturers or OS providers collect sufficient samples from users, they might create the general neural network with meta-learning. This neural network can easily be integrated into zTT and used as a basis for quick adaptation. Another approach is context-based acceleration for initial learning. Context change can be recognized using application characteristics and various sensors such as temperature sensors, GPS or illuminance sensors. If context change is recognized, the base Q-network to be used for initial learning can be prefetched according to the context. Thus, learning time can be greatly reduced more than learning from scratch.

zTT for multiple concurrent applications. Mobile devices rarely operate multiple apps concurrently in the foreground, especially with apps with graphical output, while a PC or server can run multiple apps together. Even when background apps run, for energy-efficiency, Android and iOS provide most computation cycles to the foreground app while freezing or dozing apps in the background. Therefore, even when multiple apps are running concurrently, zTT can focus on the QoE of the app running in the foreground. When the foreground app is switched which has already been trained, trained Q-network can be utilized for the app. In the case of an app that runs for the first time, it follows the transfer learning-based approach as we discussed.

Generality. zTT requires user-defined QoE per app (e.g., 60 fps for action game, 30 fps for video call) while the existing default governors are application-agnostic. We believe that this is not a big limitation for applicability, but rather a natural research direction. For example, the latest android app developers support frame rate setting (30 or 60 fps [10], five levels from Low to Extreme [46]) to give users choices about thermal concerns, graphics quality, or power savings in a heuristic manner while it still overutilizes the resources due to the nature of hardware-based DVFS. Therefore, zTT can provide an optimized solution to app developers, which can be packaged into multiple Q-networks when publishing apps even with thermal freedom. Note that Q-networks for zTT are very lightweight.

Table 4: Summary of the related works

	QoE	Thermal	Processor type	Environment	Approach	Platform
Ours	FPS	✓	CPU, GPU	✓	Deep-RL	Development board, Smartphone
Prakash et al. [33]	FPS	✓	CPU, GPU	X	Q-learning	Development board
Sahin et al. [38]	FPS	✓	CPU	X	PI-control	Development board
Park et al. [29]	FPS	X	CPU, GPU	X	Lookup table	Development board
Gupta et al. [17], Pathania et al. [32]	FPS	X	CPU, GPU	X	Modeling-based	Development board
Bhat et al. [6], Wang et al. [49]	Execution time	✓	CPU, GPU	X	Modeling-based	Development board
Ren et al. [36]	Web loading time	X	CPU, GPU	X	Modeling-based	Development board
Choi et al. [9]	Frame rendering time	X	CPU, GPU	X	Modeling-based	Smartphone

8 RELATED WORK

There have been many kinds of research to manage the energy efficiency and temperature of mobile devices considering the application performances. We summarize the related works in Table 2.

Application-aware approach. To use CPU and GPU resources energy-efficiently, we have to predict resource requirements of applications, but using process states such as CPU utilization is far from the ideal. Various methods have been proposed to understand application requirements better. Choi et al. [9] designed an overlaid CPU/GPU governor by adjusting maximum CPU frequency and minimum GPU frequency based on frame load prediction to improve Android graphics pipeline and user QoE. Park et al. [29] suggested the CPU/GPU frequency capping method to improve game applications' energy efficiency. It constructs a lookup table consisting of CPU and GPU cost by training many gaming applications. Yang et al. [52] used individual user profiles to improve energy efficiency by controlling CPU clocks and learning user experience of an application. Ren et al. [36] adjusted CPU/GPU clocks and allocation of tasks by predicting how much and which core to use to improve web browsing performance and energy efficiency.

DVFS with concerns on the thermal limit. Recent research explored various techniques to efficiently use resources as long as they do not exceed a physically defined thermal limit. Bhat et al. [6] and Gupta et al. [17] proposed a predictive thermal and power management system by developing a model based on extensive measurements to estimate the total power budget within the temperature threshold. Prakash et al. [33] suggested a thermal model that avoids thermal throttling while running a high-performance game based on processor utilization and clocks, and Pathania et al. [32] designed power and performance models for a similar purpose. Sharifi et al. [40] used the physical information of processors to build a thermal model and developed a predictive task scheduler. Sahin et al. [38] classified applications into throttling-susceptible continuous computations and latency-sensitive bursty tasks. According to the type of applications, they adopt a different policy based on the power profile to reduce thermal throttling duration. Wang et al. [49] proposed a framework for optimizing the execution time of collaborative CPU-GPU computing with thermal constraints based on the predictive models. However, there are no studies that attempt to predict future temperature budgets in consideration of various environments.

RL for power and thermal management. Mobile devices are prone to temporal and spatial variations of environments. This uncertainty makes power and thermal control more complicated. RL can be a potential tool for solving this complex problem. Iranfar et al. [19] proposed a Q-learning based power and thermal management algorithm for frequency scaling and thread allocation by limiting state-action space. Gupta et al. [18] proposed a Deep Q-learning methodology to optimize the power management in dynamically changing workloads at runtime. Carvalho et al. [41] presented a Q-learning based online power management method that does not require any prior knowledge of workload. Still, no work uses DRL on mobile devices to address both environmental changes and application QoE.

9 CONCLUSIONS

As mobile devices evolve for high-performance tasks, energy efficiency and thermal management become more critical. This paper formulated a trade-off problem between power consumption and application performance with hard constraints on the thermal threshold. We found that this formulation is challenging to be solved with a single approximate model due to diverse application performance requirements and varying environments affecting thermal conditions. We tackle this challenge in an adaptive and predictive manner by proposing a deep reinforcement learning-based dynamic frequency scaling algorithm, zTT. zTT learns application requirements and thermal characteristics of environments using samples collected in real-time and improves energy efficiency while preventing overheating. We verified that zTT works successfully in static and dynamic environments with various applications running on NVIDIA JETSON TX2 and Pixel 3a devices.

ACKNOWLEDGMENTS

This research was supported in part by Samsung Research Funding & Incubation Center of Samsung Electronics (SRFC-TD2003-01) and the Engineering Research Center Program through the National Research Foundation of Korea (NRF) funded by the Korean Government MSIT (NRF-2018R1A5A1059921). Kyunghan Lee is the corresponding author of this work.

REFERENCES

[1] Activision Publishing, Inc., Tencent Games Co., Ltd. Call of duty: Mobile. https://www.callofduty.com/mobile, 2019.
[2] Android. SurfaceFlinger. https://source.android.com/devices/graphics/surfaceflinger-windowmanager, 2020.
[3] ARM Developer Community, Quentin Perret. Energy Aware Scheduling (EAS) in Linux 5.0. https://community.arm.com/developer/ip-products/processors/b/processors-ip-blog/posts/energy-aware-scheduling-in-linux, 2019.

[4] Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research 47* (2013), 253–279.

[5] Bhat, G., Gumussoy, S., and Ogras, U. Y. Power and thermal analysis of commercial mobile platforms: Experiments and case studies. In *Proceedings of IEEE Design, Automation & Test in Europe Conference & Exhibition* (2019), pp. 144–149.

[6] Bhat, G., Singla, G., Unver, A. K., and Ogras, U. Y. Algorithmic optimization of thermal and power management for heterogeneous mobile platforms. *IEEE Transactions on Very Large Scale Integration Systems 26*, 3 (2017), 544–557.

[7] Brodowski, D., and Golde, N. Linux cpufreq governors.

[8] Chen, Z., Stamoulis, D., and Marculescu, D. Profit: priority and power/performance optimization for many-core systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 37*, 10 (2017), 2064–2075.

[9] Choi, Y., Park, S., and Cha, H. Graphics-aware power governing for mobile devices. In *Proceedings of ACM International Conference on Mobile Systems, Applications, and Services* (2019), pp. 469–481.

[10] Corporation, D. Cookie run : Kingdom, 2021.

[11] Dinakarrao, S. M. P., Joseph, A., Haridass, A., Shafique, M., Henkel, J., and Homayoun, H. Application and thermal-reliability-aware reinforcement learning based multi-core power management. *ACM Journal on Emerging Technologies in Computing Systems 15*, 4 (2019), 33.

[12] Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400* (2017).

[13] Gong, F., Ju, L., Zhang, D., Zhao, M., and Jia, Z. Cooperative dvfs for energy-efficient hevc decoding on embedded cpu-gpu architecture. In *Proceedings of Design Automation Conference* (2017), pp. 1–6.

[14] Gong, T., Kim, Y., Shin, J., and Lee, S.-J. Metasense: few-shot adaptation to untrained conditions in deep mobile sensing. In *Proceedings of ACM Conference on Embedded Networked Sensor Systems* (2019), pp. 110–123.

[15] Google. WebGL Aquarium. http://webglsamples.org/aquarium/aquarium.html, 2009.

[16] Gulli, A., and Pal, S. *Deep learning with Keras*. Packt Publishing, 2017.

[17] Gupta, U., Ayoub, R., Kishinevsky, M., Kadjo, D., Soundararajan, N., Tursun, U., and Ogras, U. Y. Dynamic power budgeting for mobile systems running graphics workloads. *IEEE Transactions on Multi-Scale Computing Systems 4*, 1 (2017), 30–40.

[18] Gupta, U., Mandal, S. K., Mao, M., Chakrabarti, C., and Ogras, U. Y. A deep q-learning approach for dynamic management of heterogeneous processors. *IEEE Computer Architecture Letters 18*, 1 (2019), 14–17.

[19] Iranfar, A., Shahsavani, S. N., Kamal, M., and Afzali-Kusha, A. A heuristic machine learning-based algorithm for power and thermal management of heterogeneous mpsocs. In *Proceedings of IEEE/ACM International Symposium on Low Power Electronics and Design)* (2015), pp. 291–296.

[20] Isuwa, S., Dey, S., Singh, A. K., and McDonald-Maier, K. Teem: Online thermal- and energy-efficiency management on cpu-gpu mpsocs. In *Proceedings of IEEE Design, Automation & Test in Europe Conference & Exhibition* (2019), pp. 438–443.

[21] Kadjo, D., Ayoub, R., Kishinevsky, M., and Gratz, P. V. A control-theoretic approach for energy efficient cpu-gpu subsystem in mobile platforms. In *Proceedings of Design Automation Conference* (2015), p. 62.

[22] Kang, S., Choi, H., Park, S., Park, C., Lee, J., Lee, U., and Lee, S.-J. Fire in your hands: Understanding thermal behavior of smartphones. In *Proceedings of ACM International Conference on Mobile Computing and Networking* (2019), pp. 13:1–13:16.

[23] Little Workshop. WebVR Showroom. https://showroom.littleworkshop.fr/, 2017.

[24] Mei, X., Wang, Q., and Chu, X. A survey and measurement study of gpu dvfs on energy conservation. *Digital Communications and Networks 3*, 2 (2017), 89–100.

[25] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *Nature 518*, 7540 (2015), 529.

[26] Mochel, P. The sysfs filesystem.

[27] Monsoon-solutions. High Voltage Power Monitor. http://www.msoon.com/LabEquipment/PowerMonitor/, 2019.

[28] opencv dev team. Opencv2.4 documentation. OPENCV2.4.https://docs.opencv.org/2.4.13.6, 2019.

[29] Park, J.-G., Hsieh, C.-Y., Dutt, N., and Lim, S.-S. Synergistic cpu-gpu frequency capping for energy-efficient mobile games. *ACM Transactions on Embedded Computing Systems 17*, 2 (2018), 45:1–45:23.

[30] Paterna, F., and Rosing, T. Š. Modeling and mitigation of extra-soc thermal coupling effects and heat transfer variations in mobile devices. In *Proceedings of IEEE/ACM International Conference on Computer-Aided Design* (2015), pp. 831–838.

[31] Paterna, F., and Rosing, T. v. Modeling and mitigation of extra-soc thermal coupling effects and heat transfer variations in mobile devices. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design* (2015), pp. 831–838.

[32] Pathania, A., Irimiea, A. E., Prakash, A., and Mitra, T. Power-performance modelling of mobile gaming workloads on heterogeneous mpsocs. In *Proceedings of Design Automation Conference* (2015), pp. 1–6.

[33] Prakash, A., Amrouch, H., Shafique, M., Mitra, T., and Henkel, J. Improving mobile gaming performance through cooperative cpu-gpu thermal management. In *Proceedings of Design Automation Conference* (2016), p. 47.

[34] Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. You only look once: Unified, real-time object detection. In *Proceedings of IEEE conference on computer vision and pattern recognition* (2016), pp. 779–788.

[35] Redmon, J., and Farhadi, A. Yolov3: An incremental improvement. *arXiv* (2018).

[36] Ren, J., Wang, X., Fang, J., Feng, Y., Zhu, D., Luo, Z., Zheng, J., and Wang, Z. Proteus: network-aware web browsing on heterogeneous mobile systems. In *Proceedings of ACM International Conference on emerging Networking EXperiments and Technologies* (2018), pp. 379–392.

[37] Sahin, O., and Coskun, A. K. Providing sustainable performance in thermally constrained mobile devices. In *Proceedings of ACM/IEEE Symposium on Embedded Systems for Real-Time Multimedia* (2016), pp. 72–77.

[38] Sahin, O., Thiele, L., and Coskun, A. K. Maestro: Autonomous qos management for mobile applications under thermal constraints. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 38*, 8 (2018), 1557–1570.

[39] Sekar, K. Power and thermal challenges in mobile devices. In *Proceedings of ACM International Conference on Mobile Computing & Networking* (2013), pp. 363–368.

[40] Sharifi, S., Krishnaswamy, D., and Rosing, T. Š. Prometheus: A proactive method for thermal management of heterogeneous mpsocs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 32*, 7 (2013), 1110–1123.

[41] Shen, H., Tan, Y., Lu, J., Wu, Q., and Qiu, Q. Achieving autonomous power management using reinforcement learning. *ACM Transactions on Design Automation of Electronic Systems 18*, 2 (2013), 1–32.

[42] Singla, G., Kaur, G., Unver, A. K., and Ogras, U. Y. Predictive dynamic thermal and power management for heterogeneous mobile platforms. In *Proceedings of IEEE Design, Automation & Test in Europe Conference & Exhibition* (2015), pp. 960–965.

[43] Skype Inc. Skype. https://skype.com/, 2003.

[44] Suh, H., Shahriaree, N., Hekler, E. B., and Kientz, J. A. Developing and validating the user burden scale: A tool for assessing user burden in computing systems. In *Proceedings of ACM CHI conference on human factors in computing systems* (2016), pp. 3988–3999.

[45] Tan, Y., Liu, W., and Qiu, Q. Adaptive power management using reinforcement learning. In *Proceedings of IEEE/ACM International Conference on Computer-Aided Design-Digest of Technical Papers* (2009), pp. 461–467.

[46] Tencent. Pubg mobile, 2018.

[47] Texas Instruments. High-side measurement, shunt and bus voltage monitor with i2c- and smbus-compatible interface. https://www.ti.com/product/INA3221, 2016.

[48] Walker, M. J., Diestelhorst, S., Hansson, A., Das, A. K., Yang, S., Al-Hashimi, B. M., and Merrett, G. V. Accurate and stable run-time power modeling for mobile and embedded cpus. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 36*, 1 (2016), 106–119.

[49] Wang, S., Ananthanarayanan, G., and Mitra, T. Optic: Optimizing collaborative cpu–gpu computing on mobile devices with thermal constraints. *IEEE transactions on computer-aided design of integrated circuits and systems 38*, 3 (2018), 393–406.

[50] Xie, Q., Kim, J., Wang, Y., Shin, D., Chang, N., and Pedram, M. Dynamic thermal management in mobile devices considering the thermal coupling between battery and application processor. In *Proceedings of IEEE/ACM International Conference on Computer-Aided Design* (2013), pp. 242–247.

[51] Yan, K., Zhang, X., Tan, J., and Fu, X. Redefining qos and customizing the power management policy to satisfy individual mobile users. In *Proceedings of IEEE/ACM International Symposium on Microarchitecture* (2016), pp. 1–12.

[52] Yang, L., Dick, R. P., Memik, G., and Dinda, P. Happe: Human and application-driven frequency scaling for processor power efficiency. *IEEE Transactions on Mobile Computing 12*, 8 (2012), 1546–1557.

[53] Zhang, Q., Lin, M., Yang, L. T., Chen, Z., and Li, P. Energy-efficient scheduling for real-time systems based on deep q-learning model. *IEEE Transactions on Sustainable Computing 4*, 1 (2017), 132–141.