2025 IEEE/ACM International Conference On Computer Aided Design (ICCAD) | 979-8-3315-1560-7/25/$31.00 ©2025 IEEE | DOI: 10.1109/ICCAD66269.2025.11240765

EarDVFS: Environment-Adaptable RL-based DVFS for Mobile Devices

Jaeheon Kwak*
Ajou University
Republic of Korea
jhkwak@ajou.ac.kr

Sangeun Oh
Korea University
Republic of Korea
sangeunoh@korea.ac.kr

Jinkyu Lee
Sungkyunkwan University
Republic of Korea
jinkyu.lee@skku.edu

Insik Shin
KAIST and *Fluiz*
Republic of Korea
ishin@kaist.ac.kr

Abstract— **Dynamic Voltage and Frequency Scaling (DVFS) is a key technology for enhancing power efficiency in computing devices. However, conventional DVFS methods struggle with the unique demands of mobile devices. Recent reinforcement learning (RL)-based approaches address this by tailoring to mobile-specific thermal and workload characteristics. Yet, these solutions make frequency adjustments that ignore device-specific configurations calibrated by vendors, neglect the impact of ambient and non-processor components—such as battery, display, and integrated circuits—that significantly affect processor thermal management, and rely on fixed environment-dependent parameters, limiting adaptability across different environments. To address these limitations, we propose EarDVFS, an environment-adaptable RL-based solution that employs proactive throttling to combine the strengths of traditional and RL-based methods, considers the temperatures of ambient and non-processor components for better thermal management, and features an environment-robust RL parameter design. Extensive experiments across varying ambient temperatures, devices, and workloads demonstrate that EarDVFS consistently enhances power efficiency by an average of 21.6% and up to 49.6% compared to default DVFS while maintaining performance. Furthermore, we conduct comprehensive ablation studies on the action, state, and reward elements of our RL model, confirming that each element significantly contributes to EarDVFS's adaptability and effectiveness across diverse thermal environments.**

I. Introduction

Advancements in mobile processors have enabled mobile devices to perform a broader range of tasks with speed and performance. However, more powerful processors necessitate improved management of processor temperature, power consumption, and performance. Yet, the limited battery capacity and constrained cooling solutions of mobile devices make this management challenging.

Currently, mobile devices adjust processor frequency through Dynamic Voltage and Frequency Scaling (DVFS) and thermal throttling to manage the power efficiency, temperature, and performance of processors. Device vendors embed carefully engineered device-specific configurations into the default system, taking into account physical characteristics of each device to ensure reliable performance in universal use cases. However, these traditional approaches are limited by their reliance on predetermined rules, making them less effective at handling mobile devices' unique characteristics, particularly their sensitivity to application-specific workloads and thermal conditions.

To address these shortcomings, reinforcement learning (RL)-based DVFS methods have recently emerged [1]–[3]. These methods manage processor frequencies based on application-specific workload patterns and implement more comprehensive thermal management. Despite their promise, our analysis reveals several critical limitations in these RL-based methods: (**L1**) operating based on fixed device-agnostic parameters while disregarding device-specific configurations that directly affect power efficiency and performance; (**L2**) failing to consider ambient temperature variations and non-processor components that significantly impact thermal management strategies; and (**L3**) relying on fixed environment-dependent parameters, such as target temperature, constrains adaptability across different conditions.

*This work was conducted while the author was at KAIST.

To address these limitations, we propose three core solutions that establish the foundation of our EarDVFS (<u>E</u>nvironment-<u>A</u>daptable <u>R</u>einforcement Learning-based DVFS):

- Proactive throttling that sets frequency boundaries instead of direct frequency adjustment, maintaining RL-based benefits while leveraging device-specific configurations (**S1**),
- Exploiting battery-related temperatures that stably track ambient temperature variations and thermal impact from non-processor components (e.g., battery, display, integrated circuits) (**S2**), and
- Environment-robust RL design that avoids using environment-dependent parameters (**S3**).

Our solution builds upon Soft Actor-Critic (SAC) [4], which provides significant advantages over traditional DQN-based [5] methods, particularly in handling extensive state and action spaces. Through in-depth experiments comparing EarDVFS with existing solutions across various environmental conditions, we demonstrate that EarDVFS consistently delivers robust performance while improving power efficiency by an average of 21.6% and up to 49.6%.

The main contributions of this work are as follows:

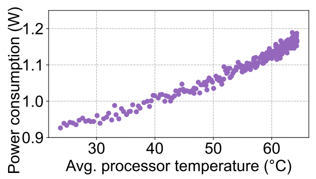
- We clarify the limitations of existing methods through systematic experiments and deep systems understanding (§II and §III).
- We propose solutions to the identified limitations and integrate them into the EarDVFS design (§III and §IV).
- We employ SAC to enable diverse action exploration and handle large action and state spaces (§IV).
- We rigorously evaluate EarDVFS in diverse environments using a stringent testbed, demonstrating its superior adaptability and efficiency (§V and §VI).
- We conduct an ablation study to validate the individual impact of our key design elements on EarDVFS's behavior (§VI).

II. Background

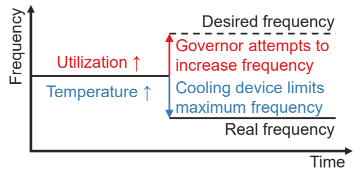
A. DVFS Principles

DVFS is a technique that dynamically adjusts voltage–frequency levels to enhance system performance and power efficiency. The voltage–frequency levels are determined by algorithms called *governors*, which make decisions based on system's workload.

Today, most mobile systems use `schedutil` as their default governor. It adjusts the CPU frequency (and also voltage) levels to maintain a target CPU utilization. The utilization is calculated by the Completely Fair Scheduler [6], [7] as the sum of the running time of runnable tasks and recent utilization of non-runnable tasks. The `schedutil` monitors the current utilization and frequency to adjust the new frequency to reach the target CPU utilization (e.g., 80%). Each vendor sets their own target CPU utilization values based on their device/processor characteristics for optimal performance in general workloads. Therefore, arbitrary frequency adjustments without considering these device-specific configurations could compromise system performance and efficiency.



(a) Power consumption and the average temperature of processors.



Frequency
Desired frequency
Utilization ↑
Governor attempts to increase frequency
Temperature ↑
Cooling device limits maximum frequency
Real frequency
Time

(b) Throttling can override the frequency decisions set by governors.

Fig. 1. Effects of processor temperature variations and throttling mechanisms.

B. Necessity and Principles of Throttling

Throttling is a technique that lowers processor temperature through frequency limitation. In mobile systems, which lack cooling mechanisms, throttling is essential for both safety and power efficiency.

Thermal Effects on Power Efficiency. High temperatures diminish power efficiency. To investigate this effect, we run a WebGL rendering application [8] on a smartphone at a fixed frequency and measure power consumption and processor temperature. Fig. 1(a) shows power consumption and the average processor temperature. As the average processor temperature increases, power consumption rises, reaching 22.3% higher at 60°C compared to 30°C. This rise in power consumption is attributed to additional power loss from elevated leakage current and electrical resistance within the processor as temperatures rise [9]. Therefore, effective thermal management of the processor is essential for maintaining power efficiency.

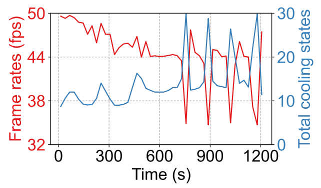
Throttling Principles. In Linux, a framework called the cooling device manages throttling. This framework sets the *cooling state*, a value indicating the throttling intensity applied to the processor, based on the processor-specific temperature threshold and temperature trends. Notably, the value of cooling state is determined independently for each processor (e.g., big core cluster and GPU). Whenever the cooling state value increases by one, the maximum frequency level decreases by one level. For example, if the cooling state of a big cluster reaches its maximum, it will only operate at its minimum frequency. Throttling can overrule governors; even if utilization is high and the governor attempts to increase the frequency, high temperatures will cause throttling to override the desired frequency to a lower one, as shown in Fig. 1(b). Thus, understanding processor behavior requires attention to both governors and throttling.

III. Establishment of Design Principles

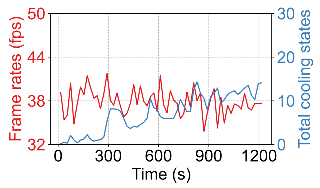
In this section, we identify the limitations of RL-based DVFS methods and propose solutions through preliminary experiments conducted with the Google Pixel 6 under ambient temperature control.

A. Inefficiency of Existing Systems and RL-based Solutions

Current mobile devices make narrow-sighted throttling decisions and are unable to optimize governors for application-specific workload patterns. Governors operate based on predefined rules without considering workload information, and a cooling device throttles a processor solely based on the temperature of a single processor.

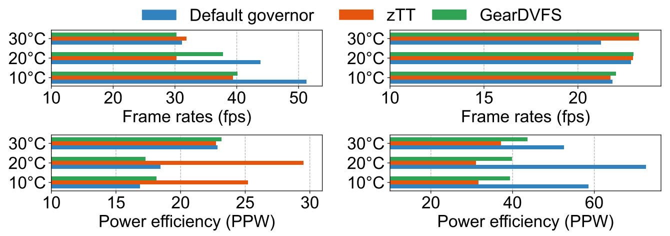


(a) Default governor exhibits frame rate and cooling state fluctuations.



(b) GearDVFS shows a stable behavior, eliminating the fluctuations.

Fig. 2. Frame rates and total cooling states of processors during 3D rendering.



(a) When running WebGL. (b) When running Skype.

Fig. 3. Frame rates and PPW of the default governor, zTT, and GearDVFS across different application types and ambient temperature levels.

This limitation results in undesirable behaviors such as performance fluctuations. Fig. 2(a) shows the frame rates and total cooling states of processors when a smartphone runs WebGL rendering [8] at an ambient temperature of 20°C, configured with 10,000 fish. After 600 seconds, both frame rates and cooling states begin to fluctuate. These fluctuations arise from a repeating cycle: (1) processors operate at high frequencies, causing temperatures to rise; (2) rising temperatures increase the cooling state, triggering throttling that reduces frequencies, frame rates, and temperatures; (3) reduced temperatures lower the cooling state, allowing the governor to increase frequencies again. These frequency fluctuations can result in underclocking and core reassignments [10], degrading both performance and power efficiency. If governors consider workload characteristics and cooling mechanisms use broader information, these issues could be avoided.

To this end, recent studies have explored reinforcement learning (RL)-based DVFS approaches for mobile devices [1]–[3]. Kim et al. introduced zTT [1], which prevents throttling and maintains performance by regulating processor temperature and application-specific frame rate within target thresholds. Similarly, Lin et al. proposed GearDVFS [2], which considers workload characteristics to drive CPU utilization to a target level while simultaneously keeping target temperature. Gong et al. developed LOTUS [3], a DVFS framework for DNN-based object detection. They train per-application RL agent to make frequency decisions based on workload patterns and adjust frequencies to keep processor temperatures below the target while considering the temperatures of all processors simultaneously. This can mitigate the fluctuations, as shown in Fig. 2(b), but it also exhibits degradation in both frame rates and power efficiency, revealing remaining limitations. Rule-based approaches such as CRAVE [11] and QUAREM [12] have also been proposed, but they suffer from limited adaptability due to their rule-based nature and lack sufficient consideration of thermal aspects.

B. Limitations of RL-based DVFS Methods and Solution

Despite the advancements of RL-based DVFS methods, our experiments reveal their suboptimal performance in diverse environments. To evaluate their environment-adaptability, we conduct experiments using two applications with contrasting computing loads: WebGL Aquarium (high load) and Skype (low load). We compare the default governor (`schedutil`), zTT, and GearDVFS under varying ambient temperatures (10°C, 20°C, and 30°C).

Fig. 3 illustrates the performance (frame rates) and power efficiency (performance per watt, PPW) of each DVFS methods. At 30°C, RL-based methods show better performance than the default governor due to its broaden-sighted thermal management. However, in other environments, they may overly limit performance or result in lower power efficiency. We now identify three core limitations **L1–L3** that contribute to the performance degradation and propose corresponding solutions **S1–S3** to address these challenges.

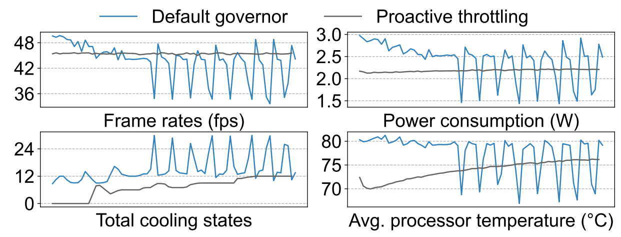


Fig. 4. Applying proactive throttling mitigates fluctuations.

L1: Ignoring Device-Specific Configurations. A significant limitation of RL-based DVFS methods is their rigid operation focused solely based on device-agnostic predetermined parameters (e.g., target utilization, frame rates, or temperature thresholds).

We identify that ignoring device-specific settings in frequency decision is problematic. GearDVFS successfully alleviates frequency fluctuations; however, it shows a notable decrease in PPW and frame rates compared to the default governor, with average reductions of 13.9% and 6.3% respectively, as shown in Fig. 3.

This performance degradation stems from GearDVFS's misaligned target utilization. Using `ftrace` [13], we reverse-engineer `schedutil`'s target utilization by analyzing how it determines its next frequency based on current frequency and utilization. This analysis confirms that the Pixel 6's actual target CPU utilization ranging from 15-50%, substantially deviating from GearDVFS's fixed 80%. Since these device-specific parameters are calibrated considering the physical characteristics of the processor, ignoring these parameters yields suboptimal performance. This issue is inherent to current RL-based methods [1]–[3], as they make frequency decisions completely independent of these device-specific configurations.

S1: Proactive Throttling. While it would be beneficial for RL-based DVFS to learn each device's unique configurations, these values are often fragmented across devices or not publicly accessible [14], [15]. Instead, inspired by the concept of throttling, we propose *proactive throttling*—a method that constrains the maximum frequency in advance of throttling.

Proactive throttling does not directly determine processor frequency but limits maximum frequency boundaries. Within these boundaries, the default governor makes the final frequency decision. This approach preserves the benefits of leveraging the device-specific configurations inherent to the default governor.

To verify the effectiveness of proactive throttling, we compare the fluctuation scenario using the default governor as it is, and that using the default governor with proactive throttling. The maximum frequencies of the big, mid, little clusters, and GPU are limited to 1.83, 1.33, 1.33, and 0.4 GHz, respectively, which are the determined through experiments. Fig. 4 shows frame rates, power consumption, total cooling states, and average processor temperatures during a 30-minute WebGL application run. Proactive throttling not only eliminates fluctuations but also maintains average frame rates and reduces average power consumption, cooling state, and processor temperature by 10.3%, 51.7% and 4.9%, respectively.

Then, how can we find the optimal maximum frequency for proactive throttling? In Section IV, we will propose a method to determine these values through RL. This approach preserves both the advantages of RL—including workload-dedicated optimization and broader-sighted thermal management—while maintaining the benefits of device-specific configurations inherent to the default governor.

L2: Insufficiency of Considering Processor Temperature Alone. Existing RL-based DVFS methods rely solely on processors' temperature for thermal management, overlooking other critical factors, i.e., the temperature of ambient and non-processor components.

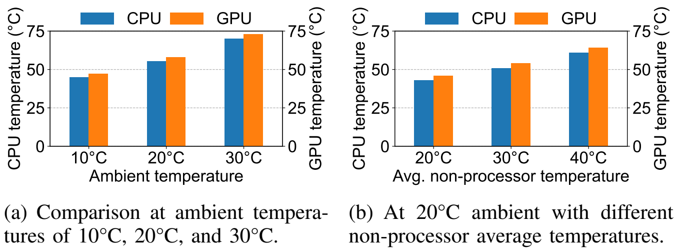


Fig. 5. Processor temperature differences in a fixed-frequency 3D rendering.

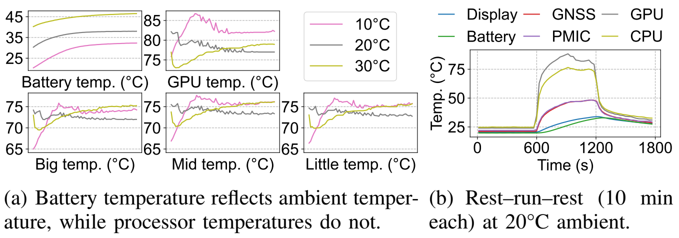


Fig. 6. Temperatures during 3D rendering with the default governor.

Our experiments confirm their substantial impact on processor temperature. In a test running a GPU-intensive 3D rendering application [16] for 10 minutes at fixed minimum frequencies shows that raising the ambient temperature from 10°C to 30°C increases CPU/GPU temperatures from 45.0°C/47.2°C to 70.0°C/73.0°C, respectively, as shown in Fig. 5(a).

Moreover, when maintaining the ambient temperature at 20°C, increasing the average temperature of non-processor components—-including battery, display, power management integrated circuit (PMIC), and global navigation satellite system (GNSS) chip—-from 20°C to 40°C caused CPU and GPU temperatures to climb from 42.8°C/45.8°C to 60.9°C/64.0°C, respectively, as shown in Fig. 5(b).

These findings show that ambient temperature and thermal states of non-processor components highly influence the temperatures of processors. Since these factors vary with usage environment and workload, adaptive and responsive processor thermal management must explicitly account for both. However, most mobile systems lack dedicated ambient temperature sensors and consist of many thermally interactive components, making it difficult to infer ambient conditions or the influence of other components solely from processor temperatures. This limitation hinders RL-based approaches from effectively leveraging such information in thermal management.

S2: Leveraging Battery Temperature. We propose using the battery as a reliable indicator not only of ambient temperature but also of the thermal impact from non-processor components. Among various smartphone components, the battery is the single largest in terms of volume, allowing its temperature to more accurately reflect external thermal variations.

To confirm this, we run a 30-minute GPU-intensive 3D rendering application [16] and monitor the temperatures of the big, mid, and little CPU clusters, GPU, and battery. As shown in Fig. 6(a), while processor temperatures fluctuate unpredictably, battery temperature remains closely tied to ambient conditions.

Additionally, battery temperature effectively captures real-time changes in ambient temperature. Fig. 7 presents frame rates, total cooling states, the average temperature of processors, and battery temperature while running a 3D rendering application [16] under the default governor. We alter the ambient temperature every 10 minutes between 10°C, 30°C, and back to 10°C. Although performance,

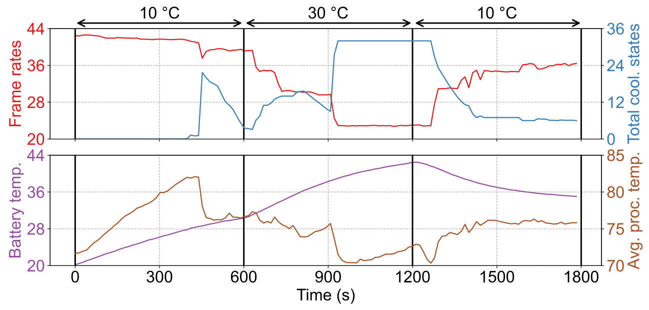


Fig. 7. Frame rates, total cooling states, and the temperatures of battery and processors during 3D rendering at 10°C–30°C–10°C ambient (10 min each).

cooling states, and processor temperatures do vary, it is difficult to infer ambient temperature fluctuations solely from these metrics. By contrast, battery temperature exhibits distinct inflection points at 600 and 1,200 seconds, quickly indicating changes in ambient temperature and demonstrating its usefulness as a dynamic thermal indicator.

Furthermore, the battery has the highest heat capacity among all individual components. In an experiment where the device is idle for 10 minutes, then run a 3D rendering application [16] for 10 minutes, followed by another 10-minute idle period, we measure the temperatures of the CPU, GPU, battery, display, PMIC, and GNSS chip. As illustrated in Fig. 6(b), the battery's temperature rises and falls more gradually than any other component, confirming it has the greatest heat capacity and accumulates/dissipates heat at a slower rate. Due to its large size, volume, and heat capacity the battery serves as the most reliable indicator of the overall thermal state among non-processor components.

Although one could theoretically monitor the temperatures of other non-processor components—often dozens in total—incorporating all of them into the state would inflate the model's size and hinder learning efficiency. Moreover, other non-processor components, having lower heat capacities, tend to be dominated by the processor's heat generation and thus offer only limited additional insight. Consequently, focusing on the battery temperature alone provides the best balance between accuracy and model efficiency for thermal management.

L3: Fixed Environment-Dependent Parameters. Existing RL-based DVFS solutions are governed by environment-dependent variables such as target temperature and frame rates. For example, their problem formulation and reward functions aim to maintain processor temperature below target temperatures [1]–[3]. However, they use fixed values for these environment-dependent parameters like target temperature regardless of environmental conditions.

Our experiments reveal that using fixed environment-dependent parameters further reduces adaptability. For instance, WebGL achieves average frame rates exceeding 30 fps at 10°C regardless of the method used, but fails to maintain this rate at 30°C, as shown in Fig. 3(a). In particular, at 30°C, since the cooling device's throttling always takes higher priority over any governor, no governor can achieve an average of 30 fps. Despite this, zTT rigidly maintains its target frame rate, demonstrating the fundamental limitation of fixed parameter values.

Similarly, target temperature settings in RL-based DVFS solutions are flawed. As discussed with Fig. 5(a), processor temperature varies with ambient temperature. However, RL-based DVFS approaches maintain fixed target temperatures regardless of environmental conditions, inherently limiting their adaptability.

S3: Environment-Robust RL Design. To overcome these limitations, we propose an environment-robust RL-based DVFS design that avoids using parameters with environment-dependent optimal values. By eliminating such parameters in problem formulation, as well as in the state, action, and reward design, the RL agent can operate without constraints imposed by environmental conditions.

Unlike previous approaches [1]–[3], we design our problem formulation to exclude temperature-related constraints. Additionally, we design a new reward function using only minimal environment-robust parameters. These changes allow our RL agent to remain adaptable and robust across a wide range of environmental scenarios. Detailed aspects of this design will be elaborated in the following sections.

IV. Design of EarDVFS

In this section we design EarDVFS, addressing the limitations of existing approaches (**L1–L3**) with our proposed solutions (**S1–S3**).

A. Problem Formulation

The goal of a DVFS governor is to maximize performance and minimize power consumption. Let $Q(t)$ and $P(t)$ denote the application's quality of experience (QoE), user perspective performance, and the power consumption of all processors, respectively, at a given time t. Based on **S3** and [1], our optimization problem **P1** to maximize governor's objectives over time T is formulated as follows:

$$(\mathbf{P1}): \quad \max_{\pi} \frac{1}{T} \sum_{t}^{T} \left(min(\alpha, Q(t)) + \frac{\beta}{P(t)} \right) \tag{1}$$

In Eq. (1), π denotes the policy for adjusting the frequencies of processors. α is the QoE value providing maximum utility to the user, and $\min(\alpha, Q(t))$ limits the QoE to this maximum, indicating that user utility does not increase beyond this level. β is the weight for trade-off between performance and power consumption, with lower values favoring performance.

Recent works [1]–[3] solve **P1**-style problems by converting them into model-free RL problems. Model-free RL requires defining the *action, state,* and *reward*. In RL-based DVFS, processor frequency adjustments, workload and system status, and the self-evaluation of frequency decisions correspond to the action, state, and reward, respectively. Its goal is to design the action, state, and reward effectively to derive the optimal π through learning.

Existing RL-based DVFS studies [1]–[3] have used value-based RL methods like DQN [5], which learn a Q-function representing expected cumulative reward for taking an action in a given state. However, value-based RL struggles to estimate the Q-function when the state and action spaces are large. This drawback does not align with our objective of leveraging extensive information such as processor and battery temperature and cooling state.

To address this, we adopt Soft Actor-Critic (SAC) [4], [17]. In this approach, *Actor* decides actions based on a policy and learns to maximize expected rewards, while *Critic* estimates expected rewards to help refine the policy. SAC incorporates two key innovations: 1) maximizing both expected rewards and policy entropy for diverse action exploration, and 2) its Actor-Critic structure enables stable learning even with large state and action spaces. This design makes the policy more robust and adaptable across various environments.

B. Action, State, and Reward Design

To solve problem **P1**, we design EarDVFS's three components: action, state, and reward based on **S1**, **S2**, and **S3**, respectively.

Action Design. Previous RL-based DVFS methods focused on finding an optimal π to directly adjust processor frequencies. However, these approaches disregard device-specific configurations in frequency decision (**L1**). To this end, we design EarDVFS's action to limit the maximum processor frequency, implementing proactive throttling from **S1**. By employing proactive throttling, as explained in

Section III-B, EarDVFS preserves the device-specific configurations of the default governor while also exploiting the strengths of RL-based DVFS approaches. Hence, we redefine policy π in Eq. (1) by π' that denotes the policy for proactive throttling, which limits upper bound of the frequency range.

Additionally, EarDVFS enables efficient control of fine-grained frequencies through its continuous action space. This is a significant advancement over DQN-based approaches, which struggle with large action spaces since their discrete action space requires output networks to scale with all possible action combinations.

The advantages of our continuous action space become evident when comparing with existing approaches. For instance, the limited action space sizes ($\prod_{i=1}^{M} a_i$, with M processors and actions a) of zTT and GearDVFS—-3 and 37, respectively—-force zTT to adjust all processor frequencies simultaneously and GearDVFS to build separate networks for each processor. In contrast, EarDVFS's action space covers all available frequency levels 11, 14, 17, and 12 for the big, mid, little CPU clusters and GPU, resulting in $\prod_{i=1}^{M} a_i = 31,416$. Despite EarDVFS considers all possible action combinations, neither its model size nor learning efficiency is compromised, which will be discussed in Section VI.

State Design. The state space of EarDVFS consists of 30 parameters: (1) thermal management signals (battery temperature, wireless charging module temperature, cooling states), (2) performance indicators (frame rates $Q(t)$, total processor power consumption $P(t)$, PPW, reward), and (3) processor statuses (temperature, previous action, operating frequency, utilization, power consumption for each of big, mid, little CPU clusters and GPU).

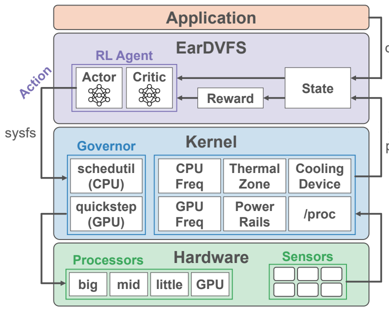
Following **S2**, we particularly incorporate battery-related temperatures, battery and wireless charging module temperatures as key indicators of the thermal environment. These parameters provide stable ambient condition signals, enabling better environmental adaptation than using processor temperature alone. The cooling states complement this by offering additional insights on thermal management.

The performance indicators ($Q(t)$, $P(t)$, PPW) serve as indicators for evaluating DVFS effectiveness, while reward value guiding the learning process. The processor statuses are essential for understanding workload characteristics, per-processor load, and behavior. The previous action and operating frequency allow the RL agent to understand how its decisions impact the actual operating frequencies, while utilization metrics help capture the workload characteristics [2].

Reward Design. The optimal values for α and β in Eq. (1) inevitably vary across different systems, devices, applications, and users. Existing methods employ extensive parameters such as target temperature, frame rates, and utilization in their reward functions to approximate these values. However, our experiments in Section III-B revealed that the fixed use of those parameters is fundamentally flawed (**L3**) as their optimal values vary depending on environmental conditions.

Following **S3**, we design the reward function of EarDVFS to solve the problem **P1** in Eq. (1) based on understanding of EarDVFS's action. Instead of attempting to find new optimal values for α and β, we design our reward function to enhance the default DVFS that are already designed to be generally compatible across environments. Our reward function considers two aspects: 1) maintaining acceptable performance despite proactive throttling's inherent restrictions (for $Q(t)$), and 2) improving power efficiency by guiding proper processor temperature (for $P(t)$). We propose the following reward function:

$$reward(t) = Q(t) * \left[\gamma + \left(\frac{M - C(t)}{M}\right)^{\lambda}\right] \quad (2)$$



(a) Implementation overview.

(b) Portable refrigerator.

Fig. 8. EarDVFS implementation details.

where $C(t)$ is the total cooling states, M is a constant value representing the maximum of the total cooling state, γ is a small positive constant to prevent division by zero, and λ controls the impact of the cooling state on the reward.[1] The $Q(t)$ increases reward for higher QoE, ensuring that proactive throttling does not excessively degrade performance. For power efficiency, we utilize cooling states which are already calibrated for each device. The $M - C(t)$ in the numerator increases the reward as $C(t)$ decreases, promoting lower processor temperatures to improve power efficiency. Additionally, it suppresses fluctuations in $C(t)$, preventing frequency fluctuations that could reduce power efficiency.

V. Implementation

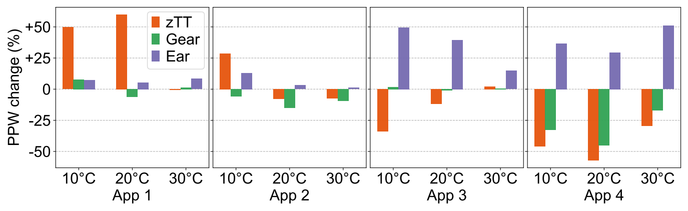
We implement EarDVFS on Google Pixel 6 device (Android 12, Kernel 5.10), featuring a Google Tensor GS101 with big/mid/little CPU clusters (2.80/2.25/1.80 GHz max) and GPU (858 MHz max). Fig. 8(a) shows an overview of EarDVFS implementation. EarDVFS operates independently from applications and default governors. Every 500 ms, EarDVFS updates its states and calculates the reward. Based on this information, the actor and critic networks perform learning, where the actor determines actions and the critic updates value estimations. The actions set maximum processor frequencies through `sysfs` [18], while default governors operate independently within these frequency limits. We update the states of RL agent through `procfs` [19] and `sysfs` (temperatures, utilization, cooling states, frequencies), Android on-device power rails monitor [20] (power), and `dumpsys` [21] SurfaceFlinger command (frame rates).

All experiments are conducted in a controlled environment inside the portable refrigerator as shown in Fig. 8(b). The refrigerator features both cooling and heating capabilities to maintain consistent internal temperature. Note that governor behaviors in this closed environment match those in open spaces. To ensure reproducible results, we wait for the device to reach thermal equilibrium before each experiment, initiating experiment only when the battery temperature stabilizes at 20°C, 30°C, and 40°C for ambient temperatures of 10°C, 20°C, and 30°C, respectively.

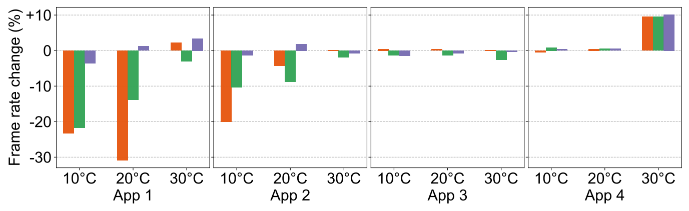
EarDVFS is implemented with lightweight neural networks to minimize overhead. Actor and Critic networks utilize four and three fully-connected layers respectively. EarDVFS's RL agent runs on a desktop (Python 3.10, PyTorch 1.12) equipped with an Intel i9-10900K CPU, communicating with the phone via `adb` [22]. Due to the compact network sizes, each action decision on the desktop takes less than 100 ms. We also implement EarDVFS on Pixel 6 with PyTorch Mobile [23][2], which achieves less than 1 ms latency by eliminating

[1]We set $\gamma = 1$ and $\lambda = 1/2$ through experiments.

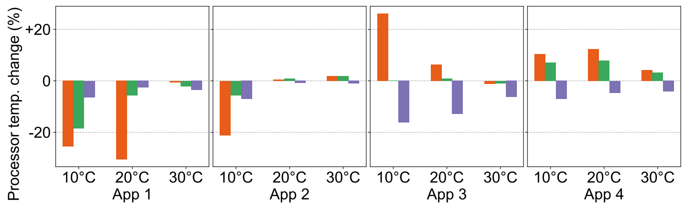
[2]Since PyTorch Mobile currently does not support on-device RL training yet, we port the desktop-trained model to Pixel 6.



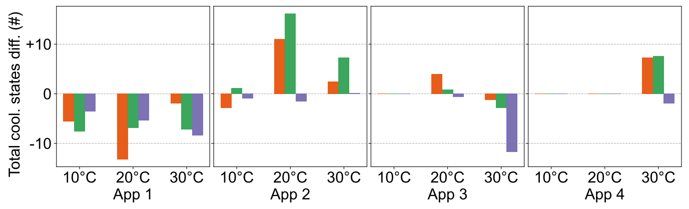
(a) Rate of change in average power efficiency (PPW).



(b) Rate of change in average frame rates.



(c) Rate of change in average processor temperature.



(d) Change of average total cooling state levels.

Fig. 9. Overall results corresponding to ambient temperature and application compared to the default governor.

wire communication overhead from `adb`. The implementation of EarDVFS is available online[3].

VI. Evaluation

A. *Evaluation Setup*

We evaluate our proposed EarDVFS (denoted as **Ear**) against three existing DVFS governors: the default governor (denoted as **Deft**) of Pixel 6 (`schedutil` for CPU and `quickstep` for GPU), **zTT** [1], and GearDVFS [2] (denoted as **Gear**). We utilize the open sources of zTT and GearDVFS for their implementations. The evaluation uses four applications representing different processor load patterns:

- **App 1.** WebGL Aquarium [8]: A CPU-intensive 3D rendering application executed on Chrome (version 130.0.6998.39), configured with 10,000 fish.
- **App 2.** Seascape Benchmark [16] (version 2.0.7): A GPU-intensive application that benchmarks GPU performance.
- **App 3.** Genshin Impact [24] (version 5.5.0): A popular mobile game with intensive CPU and GPU utilization, running at 30 fps cap with medium graphics settings.
- **App 4.** Skype [25] (version 8.138.0.213): A video conferencing application with a low processor load, sharing video during the test to simulate a meeting.

Each evaluation runs for 30 minutes under controlled ambient temperatures (10°C, 20°C, and 30°C). The experiments begin with no prior training to assess the adaptability of each governor across varying environments.

B. *Results*

We evaluate the overall performance and run-time behavior of the four governors in environments with fixed ambient temperature, changing temperature conditions, and on different devices.

Overall Performance. To investigate environmental adaptability, we test four governors across multiple environments. We evaluate how newly proposed DVFS methods (zTT, GearDVFS, and EarDVFS) perform compared to the default governor which is designed for general workloads, in terms of power efficiency (PPW), frame rates (QoE), processor temperatures (across big, mid, little CPU clusters and GPU), and total cooling states. Fig. 9 shows the average changes in these metrics when running App 1–4 for 30 minutes at ambient temperatures of 10°C, 20°C, and 30°C.

We first observe that EarDVFS consistently demonstrates improvements across diverse scenarios. As shown in Figs. 9(a), (c), and (d), it robustly enhances PPW, with an average improvement of 21.6%, while reducing average processor temperatures and total cooling states for all applications and under all ambient conditions. Moreover, as shown in Fig. 9(b), EarDVFS maintains robust QoE, showing average increases of 0.77% in frame rates and an average decrease of only 0.90% where frame rate reductions occur. Remarkably, EarDVFS achieves notable performance in App 3, delivering a 49.6% enhancement in PPW with only a 1.46% decrease in frame rates at 10°C. This is particularly impressive given App 3's high power consumption resulting from intensive CPU and GPU utilization. These improvements come from reducing the average processor temperature by 13% and alleviating frequency fluctuations.

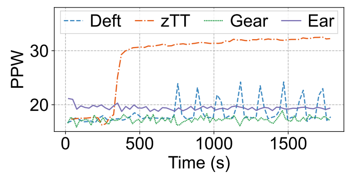
On the other hand, zTT and GearDVFS exhibit inconsistent performance across different environments. Both methods partially succeed in thermal management, highlighting the advancement of RL-based DVFS approaches. In particular, they perform well in thermal management for App 1. At an ambient temperature of 10°C, zTT and GearDVFS reduce processor temperatures by 25.5% and 18.5%, respectively. However, they struggle in other scenarios because their target temperatures are not well-suited for different application workloads. Furthermore, their performance in other metrics is generally suboptimal.

zTT improves PPW in App 1 at 10°C and 20°C, and App 2 at 20°C. However, these gains come at the cost of the significant QoE degradation. To meet its target temperature, zTT excessively reduces QoE, even failing to maintain its target frame rates. This behavior is undesirable and will be examined in detail alongside the run-time behavior shortly. As for GearDVFS, it exhibits the weakest overall performance, rarely outperforming the default governor in terms of both PPW and QoE. This underperformance is attributed to its bad target CPU and GPU utilization values, discussed in Section III-B.

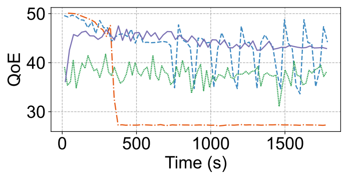
Run-Time Behavior. We now examine the run-time behavior of four DVFS methods to understand their performance in detail. We analyze their behaviors by running App 1 at 20°C for 30 minutes, measuring previously observed four overall performance metrics, CPU big cluster frequency and utilization, and normalized loss and reward values throughout learning. The results are shown in Fig. 10.

Figs. 10(a)–(d) and (g) displays that the big cluster frequency significantly impacts overall performance metrics due to App 1's CPU-intensive workload. The default governor shows noticeable frequency fluctuation after 600 seconds, yielding fluctuation in the overall performance metrics. In contrast, EarDVFS successfully eliminates

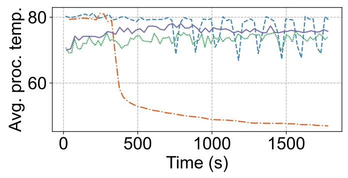
[3]https://github.com/0jaehunny0/EarDVFS



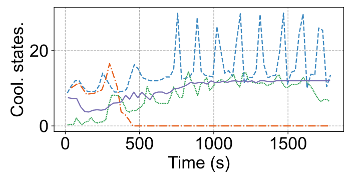
(a) Power efficiency (PPW).



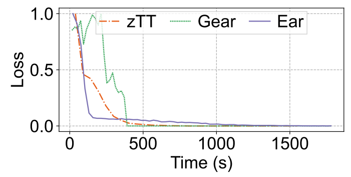
(b) Frame rates (fps).



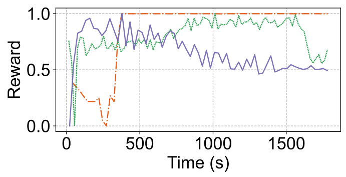
(c) Avg. processor temperature (°C).



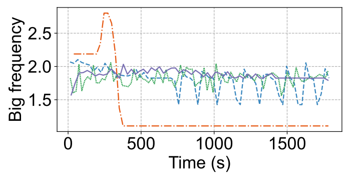
(d) Total cooling states (#).



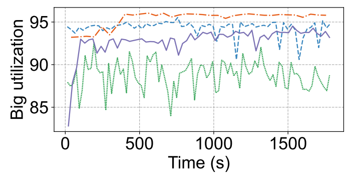
(e) Normalized loss.



(f) Normalized reward.



(g) Big cluster frequency (GHz).



(h) Big cluster utilization (%).

Fig. 10. The 30-minute run-time behaviors of four governors while running App 1.

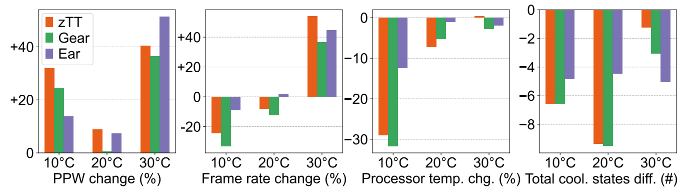


Fig. 11. Overall performance results of four DVFS governors on Pixel 7.

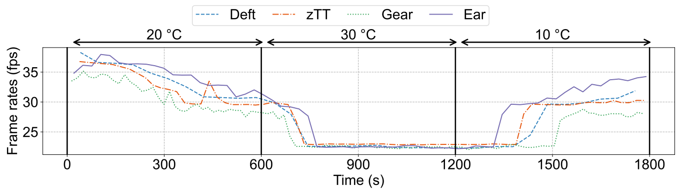


Fig. 12. Frame rates for four DVFS methods during real-time ambient temperature changes (20°C → 30°C → 10°C) while running App 2.

these fluctuations as intended through our reward design and proactive throttling mechanism. Notably, after 1,000 seconds when learning and the impact of heat generation stabilizes, its frequency variations are almost entirely eliminated, exhibiting the desirable frequency behavior we discussed in Section III.

zTT initially maintains high frequency and QoE but subsequently adjusts the big cluster frequency too low to meet its target temperature, as shown in Figs. 10(b), (c), and (g). This excessive frequency reduction stems from the zTT's small action space, resulting in coarse-grained frequency choices. While this approach achieves high PPW and its target temperature maintenance, yielding reward to increase, it fails to keep its target frame rates. One might argue this is due to an inadequate target temperature setting, but our additional experiments reveal similar issues even with targeting 55°C and 75°C.

Fig. 10(h) illustrates that GearDVFS maintains its big cluster utilization closest to 80%, which aligns with its target utilization. However, this approach fails to demonstrate its effectiveness. While GearDVFS maintains lower big cluster frequency compared to default governor and EarDVFS, resulting in lower QoE, it fails to achieve better PPW than either of them. This is likely due to its reliance on a suboptimal fixed target utilization. As Pixel 6's default governor employs variable target utilization rather than a fixed 80%, simple parameter adjustment is insufficient to achieve better performance.

To understand how RL-based DVFS methods learn, we compare the normalized loss and reward of zTT, GearDVFS, and EarDVFS (Critic), as shown in Figs. 10(e) and (f). With SAC, EarDVFS efficiently learns and nearly converges within two minutes despite its large action and state spaces.

Regarding reward, unlike zTT and GearDVFS, which increase rewards over time, EarDVFS shows a reward decrease as the device heats up. This coincides with frame rate reductions and increased cooling states. This reward decrease reflects the challenging conditions and aligns with the reward function of EarDVFS, which utilizes the cooling state as a parameter.

Generalization Test. To verify that EarDVFS's benefits generalize across different devices, we conduct additional experiments on a Google Pixel 7, which has different hardware characteristics from the Pixel 6 used in our primary evaluations. We compare all four governors (default, zTT, GearDVFS, and EarDVFS) running App 1 under three ambient temperatures (10°C, 20°C, and 30°C). Note that hyperparameter values used in Pixel 6 experiments are maintained.

As shown in Fig. 11, the overall performance trends on Pixel 7 are similar to those observed on Pixel 6. However, all three RL-based approaches shows better performance at an ambient temperature of 30°C. This suggests that the default governor on Pixel 7 exhibits poorer performance in high-temperature environments, while the RL-based solutions effectively overcome these limitations.

Across all temperature conditions, EarDVFS achieves superior power efficiency (PPW) and QoE (frame rates) compared to other governors, with average improvements of 24.1% and 12.6% respectively over the default one. These results demonstrate that EarDVFS can be effectively generalized to devices with different processors and internal designs without device-specific reconfiguration.

Real-Time Ambient Temperature Changes. To evaluate the adaptability of DVFS governors, we conduct real-time ambient temperature variation tests. We run App 2 for 30 minutes while changing the ambient temperature every 10 minutes (20°C → 30°C → 10°C). We focus on frame rates to determine whether the governors can maintain performance despite these dynamic temperature changes.

As shown in Fig. 12, EarDVFS demonstrates superior adaptability to changing thermal environments. When transitioning from 20°C to 30°C at 10 minutes, all other DVFS methods exhibit earlier frame rate drops. In contrast, EarDVFS maintains more stable frame rates. These results are attributable to EarDVFS's better CPU frequency adjustment. In GPU-intensive workloads, excessively high CPU frequencies generate heat without improving QoE, and EarDVFS manages CPU frequencies most effectively in this regard. Until the 600-second mark, the default governor's little core cluster frequency is 68.9% higher than that of EarDVFS. This superior frequency management by EarDVFS reduces heat generation from CPU, delaying throttling as much as possible. After the transition from 30°C to 10°C at 20

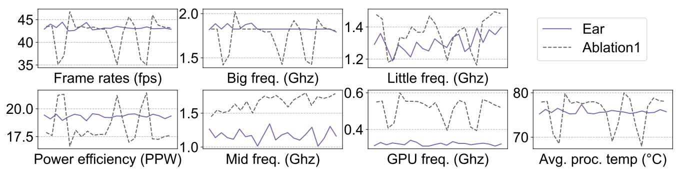


Fig. 13. Comparison of performance and behavior between EarDVFS and Ablation1 (without proactive throttling).

minutes, EarDVFS quickly adapts to the lower ambient temperature environment, exhibiting the fastest QoE improvement. This is due to both better thermal management during the 30°C period and faster adaptation to the changed ambient temperature.

C. Ablation Study

To validate the effectiveness of our proposed design elements, we conduct a comprehensive ablation study by systematically removing key elements from EarDVFS. We analyze how each element—proactive throttling (in Action), battery temperature (in State), and cooling state (in Reward)—contributes to the DVFS behavior and the overall performance. All experiments are conducted on a Pixel 6 device running App 1, which is the best scenario to discover processor frequency/temperature fluctuation.

Action–Proactive Throttling. We first evaluate the impact of our proactive throttling mechanism by creating a variant (`Ablation1`) that directly adjusts processor frequencies instead of setting frequency upper boundaries. As shown in Fig. 13, removing the proactive throttling results in not only significant processor temperature/frequency fluctuation but also the overall performance degradation.

Most notably, the frequency adjustment of Ablation1 produces intense fluctuation in the big CPU cluster, demonstrating the critical role of proactive throttling in suppressing fluctuations. The direct frequency control fails to account for device-specific parameters calibrated by vendors, leading to less efficient operation and average frame rates and PPW decreased by 1.2% and 5.4%, respectively. The performance differences indicate that our proactive throttling approach (S1) effectively leverages device-specific configurations while maintaining the benefits of the RL-based DVFS method.

State–Battery Temperature. We next examine the significance of battery temperature monitoring by creating a variant (`Ablation2`) that does not include battery temperature in state parameters.

As shown in Fig. 14, across different ambient temperature environments, Ablation2 consistently shows lower QoE and PPW compared

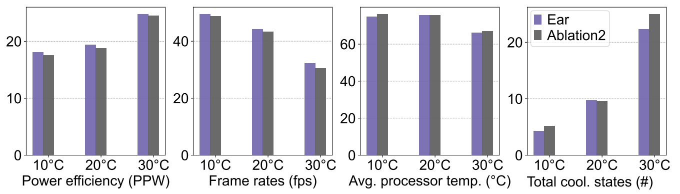


Fig. 14. Performance comparison between EarDVFS and Ablation2 (without battery temperature) at different ambient temperatures.

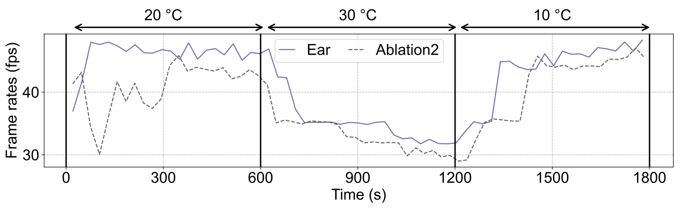


Fig. 15. Comparison of frame rates between EarDVFS and Ablation2 real-time ambient temperature changes (20°C → 30°C → 10°C).

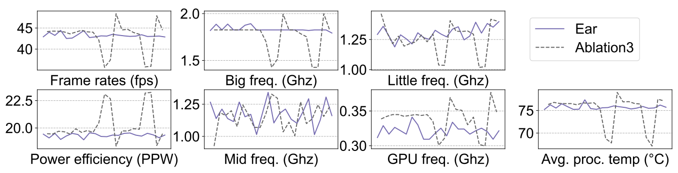


Fig. 16. Comparison of performance and behavior between EarDVFS and Ablation3 (without cooling state).

to EarDVFS, while maintaining higher temperature and cooling states. From these experiments, we observe that battery temperature information enables the RL agent to responsively grasp environmental context, impacting the initial learning phase. Fig. 15 reveals the most significant limitation of Ablation2: its inability to adapt to environmental changes. When ambient temperature changes from 20°C to 30°C and then to 10°C, Ablation2 shows delayed and inadequate responses to these transitions. The average frame rates of Ablation2 are 8.3% lower than EarDVFS despite having nearly identical PPW, validating our approach of incorporating battery temperature (S2) as a key state component for sensing environmental conditions, enabling superior adaptation to variable thermal environments.

Reward–Cooling State. Finally, we evaluate the impact of cooling state in our reward function by implementing a variant (`Ablation3`) that excludes cooling state from the reward calculation. The reward function of Ablation3 is simplified to:

$$reward(t) = Q(t) \tag{3}$$

As shown in Fig. 16, Ablation3 exhibits notable fluctuations in processor temperature and frequencies. In EarDVFS, the cooling state value in the reward function helps prevent increases in cooling states, thereby suppressing fluctuations. Without this element, Ablation3's cooling states fluctuate, causing processor frequencies and temperatures to fluctuate as well—an undesirable behavior as explained in Section III. However, the intensity of these fluctuations is lower than in Ablation1, suggesting that proactive throttling contributes more significantly to suppressing fluctuations than the cooling state in the reward function. Nevertheless, these results confirm that a reward function just omitting environment-dependent parameters is overly simplistic, while our environment-robust reward design (S3), which incorporates the cooling state, enables better adaptation across diverse environments and reduces fluctuations.

VII. Conclusion

This paper reveals critical limitations in current RL-based DVFS methods and presents EarDVFS, an environment-adaptable RL-based DVFS solution featuring proactive throttling, battery temperature-based sensing, and environment-robust design. Our extensive evaluation across diverse devices, ambient temperatures and application types demonstrates that EarDVFS achieves consistent power efficiency improvements of up to 49.6% while maintaining robust performance, demonstrating its ability to adapt to diverse environments.

Acknowledgement

This work was supported by the National Research Foundation of Korea (NRF) through a grant funded by the Korea government (MSIT) (RS-2024-00438248, RS-2024-00347516, RS-2025-00522352) and the Ministry of Education (RS-2024-00451982). It was also supported by IITP (RS-2020-II201819, RS-2025-02305705, RS-2023-00232728), TIPA (RS-2024-00447529), and K-Startup (20144069).

References

[1] S. Kim, K. Bin, S. Ha, K. Lee, and S. Chong, "Ztt: learning-based dvfs with zero thermal throttling for mobile devices," in *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*, 2021, pp. 41–53.

[2] C. Lin, K. Wang, Z. Li, and Y. Pu, "A workload-aware dvfs robust to concurrent tasks for mobile devices," in *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking*, 2023, pp. 1–16.

[3] Y. Gong, Y. Wu, Z. Zhan, P. Zhao, L. Liu, C. Wu, X. Tang, and Y. Wang, "Lotus: learning-based online thermal and latency variation management for two-stage detectors on edge devices," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, ser. DAC '24. New York, NY, USA: Association for Computing Machinery, 2024. [Online]. Available: https://doi.org/10.1145/3649329.3657310

[4] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*. PMLR, 2018, pp. 1861–1870.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[6] T. L. K. D. Community, "Schedutil The Linux Kernel documentation — docs.kernel.org," https://docs.kernel.org/scheduler/schedutil.html.

[7] C. S. Pabla, "Completely fair scheduler," *Linux Journal*, vol. 2009, no. 184, p. 4, 2009.

[8] Greggman and H. Engines, "WebGL Aquarium — webglsamples.org," https://webglsamples.org/aquarium/aquarium.html.

[9] N. S. Kim, T. Austin, D. Baauw, T. Mudge, K. Flautner, J. S. Hu, M. J. Irwin, M. Kandemir, and V. Narayanan, "Leakage current: Moore's law meets static power," *computer*, vol. 36, no. 12, pp. 68–75, 2003.

[10] S. Park, J. Park, D. Shin, Y. Wang, Q. Xie, M. Pedram, and N. Chang, "Accurate modeling of the delay and energy overhead of dynamic voltage and frequency scaling in modern microprocessors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 5, pp. 695–708, 2013.

[11] D. Mukherjee, S. Hachem, J. Bao, C. Madsen, T. Ma, S. Ghose, and G. Agha, "Crave: Analyzing cross-resource interaction to improve energy efficiency in systems-on-chip," in *Proceedings of the Twentieth European Conference on Computer Systems*, 2025, pp. 59–75.

[12] S. Isuwa, S. Dey, A. P. Ortega, A. K. Singh, B. M. Al-Hashimi, and G. V. Merrett, "Quarem: maximising qoe through adaptive resource management in mobile mpsoc platforms," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 21, no. 4, pp. 1–29, 2022.

[13] Google, "Use ftrace — Android Open Source Project — source.android.com," https://source.android.com/docs/core/tests/debug/ftrace.

[14] ——, "kernel/sched/cpufreq_schedutil.c - kernel/gs - Git at Google — android.googlesource.com," https://android.googlesource.com/kernel/gs/+/refs/heads/android-gs-raviole-5.10-android12-qpr3/kernel/sched/cpufreq_schedutil.c.

[15] ——, "Develop kernel code for GKI — Android Open Source Project — source.android.com," https://source.android.com/docs/core/architecture/kernel/kernel-code.

[16] NatureApps, "Seascape Benchmark - GPU test - Apps on Google Play — play.google.com," https://play.google.com/store/apps/details?id=com.nature.seascape&pcampaignid=web_share.

[17] S. Huang, R. F. J. Dossa, C. Ye, J. Braga, D. Chakraborty, K. Mehta, and J. G. Araújo, "Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms," *Journal of Machine Learning Research*, vol. 23, no. 274, pp. 1–18, 2022. [Online]. Available: http://jmlr.org/papers/v23/21-1342.html

[18] P. Mochel, "The sysfs filesystem," in *Linux Symposium*, vol. 1. The Linux Foundation San Francisco, CA, USA, 2005, pp. 313–326.

[19] E. Mouw, "Linux kernel procfs guide," *Delft University of Technology*, 2001.

[20] Google, "Power Profiler — Android Studio — Android Developers — developer.android.com," https://developer.android.com/studio/profile/power-profiler.

[21] ——, "dumpsys — Android Studio — Android Developers — developer.android.com," https://developer.android.com/tools/dumpsys.

[22] ——, "Android Debug Bridge (adb) — Android Studio — Android Developers — developer.android.com," https://developer.android.com/tools/adb.

[23] T. L. Foundation, "Home — pytorch.org," https://pytorch.org/mobile/home/.

[24] miHoYo Network Technology Co. Ltd., "Genshin Impact - Step into a Vast Magical World for Adventure — genshin.hoyoverse.com," https://genshin.hoyoverse.com/.

[25] Microsoft, "Skype," https://www.skype.com/.