2022 IEEE International Performance, Computing, and Communications Conference (IPCCC) | 978-1-6654-8018-5/22/$31.00 ©2022 IEEE | DOI: 10.1109/IPCCC55026.2022.9894314

KylinTune: DQN-based Energy-efficient Model for Browser in Mobile Devices

Hao Xu, Long Peng†, Xiaodong Liu, Menglin Zhang, Jun Ma, Jie Yu, Zibo Yi
College of computer, National University of Denfence Technology, Changsha, China
Emails: {xuhao19, penglong, liuxiaodong}@nudt.edu.cn, zhangmenglin@yhkylin.cn, {majun, yj, yizibo14}@nudt.edu.cn

***Abstract*—Browser is a key application for mobile devices and its power management is significant given that mobile devices are power-sensitive. Currently, dynamic voltage and frequency scaling (DVFS) and energy-aware scheduling (EAS) techniques have been implemented in mobile devices for energy savings. However, it is still challenging to achieve an energy-efficient mobile browser due to the varied content of webpages that need different resources to fetch, parser, render, etc. An ideal power governor should adjust CPU frequency dynamically according to webpage characteristics, but the current governor is configured statically and webpage-agnostic. To address the above issues, we propose KylinTune, an energy-efficient model for mobile browsers. The KylinTune is based on Deep-Q Network (DQN), a reinforcement learning technique. KylinTune learns from the browser runtime and adjusts CPU frequency to an optimal execution speed for a specific webpage based on EAS. We apply KylinTune to the Chromium browser on Google Pixel2 XL and evaluate it on the top 100 popular websites. Experimental results show that KylinTune achieves 14.51%-24% energy savings in different loading environments, with trivial quality of service (QoS) degradation.**

***Index Terms*—Web browser, Energy efficiency, Deep reinforcement learning**

I. Introduction

With the popularization of mobile devices, they could replace desktop devices in a lot of scenarios, such as gaming, video playing, and even deep learning. Browser is a key application for mobile devices, as users rely on it to get information from the internet. However, mobile devices have limited power compared to desktop devices, which makes the trade-off between quality of service (QoS[1]) and energy consumption much more significant.

Dynamic voltage and frequency scaling (DVFS) is a default technique in Android mobile devices. DVFS adjusts the CPU frequency dynamically to achieve energy-efficient. Similar to DVFS, previous studies [1], [2] proposed schemes to adjust the CPU frequency for the energy efficiency of the browser process. However, these techniques or schemes directly adjust the CPU frequency of mobile devices, which could bring side effects to other running processes. Energy-aware scheduling (EAS), a technique that integrates task scheduler and CPU frequency scaling, provides CPU scheduler the ability to be aware of energy consumption. EAS adjusts the execution speed for a specific process and is applied to the latest Android devices. However, our observation shows that the default static configuration of EAS over-optimizes browser processes and generates extra energy consumption. Some researchers [3] proposed a browser energy-efficient scheme that combines EAS and Q-table, a reinforcement learning (RL) method, to adjust the browser process execution speed separately from other processes. While the Q-table method has limits on memory usage and webpage characteristics identification, which has bad effects on model accuracy. In this paper, we propose KylinTune, a model that adjusts EAS parameters dynamically based on the Deep-Q Network (DQN). KylinTune achieves energy efficiency delicately with trivial QoS (PLT) degradation.

To realize KylinTune, we propose three techniques based on our experimental observation. Firstly, after modifying the browser engine, we design a model to make KylinTune capable of collecting the browser runtime states of each webpage. The states consist of webpage features[2], network state, webpage load state, and energy consumption. Secondly, by using the DQN, KylinTune is able to learn from the runtime states of the browser and selects the optimal action for a browser to save energy consumption with acceptable QoS degradation. Thirdly, instead of directly adjusting the CPU frequency, we scale the execution speed of the browser process based on the EAS technique. We put three types of processes of the mobile browser into a new cgroup[3], and dynamically adjust the `schedtune.boost` value of the cgroup based on the DQN, which avoids side effects on other processes.

We have implemented KylinTune in the open-source Chromium browser [4] (version 84.0.4122.0) on Google Pixel2 XL. We evaluate KylinTune on the top 100 websites ranked by `alexa.com` [5]. We compare KylinTune against the default EAS configuration under WiFi and 3G cellular network environment. Furthermore, we evaluate KylinTune under the condition of cold loading (load without browser cache) and hot loading (load with browser cache), respectively. Experimental results show that KylinTune saves more energy consumption over the default EAS with trivial QoS degradation. We also compare our approach to Q-Table and the results show that the DQN mechanism has advantages in memory usage and model accuracy.

† indicates corresponding author.

[1]In this paper QoS mainly refers to page load time (PLT).

[2]We select five web features that have most significant impacts on PLT based on the previous studies [2].

[3]cgroup indicates control group, a mechanism for process controlling in Linux system.

The main contributions of this paper are listed as follows:

- We observe that the default EAS technique is not energy-efficient for mobile browsers and reveal a need for applying the RL algorithm to energy savings of mobile browsers.
- We adapt the RL algorithm to our problem scope and create a scheme that could handle a trade-off between energy consumption and the QoS of the browser.
- We propose KylinTune, a DQN-based model that could learn mobile browser runtime environment and select the optimal browser process execution speed to save energy consumption dynamically with trivial QoS degradation. To the best of our knowledge, we are the first to apply the DQN to solve the mobile browser energy consumption problem.
- We implement KylinTune on the Chromium browser with the Pixel2 XL device. Evaluation results show that KylinTune outperforms the default EAS in energy saving with trivial QoS degradation.

II. Background and Motivation

A. CPU Frequency Scaling Techniques

DVFS governors. DVFS in Android devices provides several governors that could dynamically adjust CPU frequency: `interactive` raises the CPU frequency when users touch the screen of the device; `powersave` sets the CPU frequency as low as possible to save power; `performance` sets the CPU frequency at the highest level to guarantee application performance; `userspace` provides a CPU frequency scaling interface for an application in user mode. However, these governors have limitations in the case of browser energy optimization. Firstly, they are not aware of webpage characteristics. Therefore, the control of the governor could not guarantee optimal performance for a specific webpage, which may generate extra energy consumption. Secondly, these governors directly adjust CPU frequency, which may have a side effect on other active processes.

Energy-aware scheduling. EAS [6], combining CPU task scheduler with DVFS, provides a new frequency governor `schedutil`. The `schedutil` governor directly utilizes the CPU loading information from the task scheduler instead of sampling methods taken by other DVFS governors, and it reacts more quickly. With the `schedutil`, each CPU has its "capacity" and several DVFS operating points (OPP), and each process or task has its "util". When the "capacity" of the CPU is able to absorb the "util" of the task, the OPP of the CPU remains unchanged. If the "capacity" is bigger than the current OPP after the "util" is added to the CPU, the CPU raises the OPP to absorb the "util". As a result, the CPU frequency will be set at a higher level. How to define the "util" of a task? In the design of the EAS, schedtune is a subsystem of cgroup. There are three default schedtune cgroups in Android system: `top-app`, `foreground`, and `background`. Each schedtune cgroup contains a `cgroup.procs` file that keeps all the process IDs belonging to the cgroup and a `schedtune.boost` file that determines the "util" of the process or task. The value of `schedtune.boost` scales from -100 to 100 and a larger boost value indicates a larger "util".

B. Workflow of the Web Browser

At first, the browser submits the requested resource to the server, and the server returns HTML, JavaScript, and CSS resources. HTML would be parsed to generate a DOM tree and each tag in the HTML (e.g. $< P >$, $< div >$) corresponds to a node in the DOM tree. Webpage characteristics, including the number of the DOM nodes, the depth of the DOM tree, etc., affect the PLT. The DOM tree, together with the parsed CSS elements, is traversed to generate a render flow tree. The render tree is parsed by the render engine in the next stage and finally is displayed on the screen with the work of GPU.

Generally, a browser consists of three kinds of processes: browser process, render processes, and GPU process. The browser process is the main process of the browser, which is responsible for the management of all three processes. Each render process corresponds to one tab in the browser and is responsible for converting resources into a webpage. The GPU process issues GPU commands to the GPU and displays the webpage on the screen. The whole workflow of the browser is shown in Figure 1.

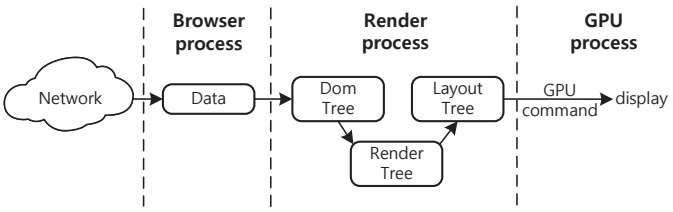


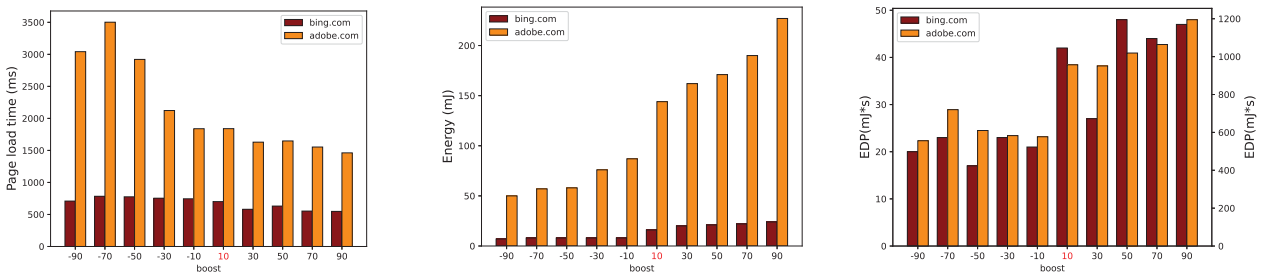
Fig. 1. Workflow of the Web browser.

C. Motivation Example

We observe two websites `bing.com` and `adobe.com` loading process in a browser to evaluate the default EAS. Our test is executed on Google Pixel2 XL, which is an EAS-enabled platform equipped with eight big/LITTLE Kyro 280 CPU and Adreno 540 GPU. The network environment is WiFi, of which the uplink bandwidth, downlink bandwidth, and delay are 15Mbps, 30Mbps, and 5ms, respectively. The browser in this test is the open-source Chromium (version 84.0.4122.0).

The evaluation metrics contain: load time, energy consumption and energy delay product (EDP) – calculated as load time $\times$ energy consumption – a useful metric to reflect on the speed of energy-efficient circuits [2], [7], [8]. In this test, we create a new cgroup `bprocess` and write all the Chromium processes IDs into the `cgroup.procs` file of `bprocess`, and then we set various `schedtune.boost` values.

The evaluation results are shown in Figure 2. `bing.com` and `adobe.com` have significantly different performances in each evaluation metric because of the difference in web



(a) Page load time (b) Energy consumption (c) EDP

Fig. 2. The page load time (a), energy consumption (b), EDP (c) when loading *bing.com* and *adobe.com*. The default *schedtune.boost* value of the browser is set to 10.

contents. The default cgroup of the active Chromium process is `top-app`, and the `schedtune.boost` value of which is 10. As the `schedtune.boost` value increases, the load time of two websites decreases. However, the load time keeps stable when the boost value reaches a certain point. On the other hand, a shorter page load time is not necessary for users given that proper load time is enough, and a shorter load time could generate extra energy consumption. The energy consumption increases together with the boost value. There is a surge when boost value is set to 10, which is the default value the EAS. With regard to EDP, `bing.com` and `adobe.com` are the best at the boost value -10 and -50, respectively, indicating that different websites have different optimal settings to achieve a balance between QoS and energy consumption in an EAS-based system.

We learn from the evaluation results that the default `schedtune.boost` value for the browser processes is not set properly, and different website has different optimal boost value. There is a need for tuning `schedtune.boost` based on website workload.

III. KylinTune

In this section, we present RL in detail and explain how it can be applied to the energy efficiency problem of mobile browsers. Following that, we provide a formulation for our problem that is adaptive to RL. Finally, we illustrate the implementation of KylinTune.

A. Call for Reinforcement Learning

In RL, an agent gets knowledge by interacting with the environment and performs actions with the consideration of consequences in the future. The RL method is a perfect solution to our problem as the CPU frequency of the browser processes set at a time would affect the PLT and the energy consumption that are only available in the future.

An RL model consists of elements listed as follows: **Environment**, a subject interacting with the agent, corresponds to mobile devices in this paper; **Agent**, a subject that performs actions, corresponds to the browser. The principle of RL is the Markov decision process (MDP). MDP consists of state space (a set of all states), action space (a set of all actions), state transition, reward, and discount factor. **State** is an overview of the environment at each moment; **action** is what the agent performs to interact with the environment; **state transition** means the agent transits from the previous state to the next state; **reward** indicates a value given by the environment after the agent performs an action and instructs how the agent performs actions in next learning episode; **discount factor** is related to a concept named return which would be presented next. The overview of RL is shown in Figure 3. In each learning episode, the agent takes action depending on the current state and reward, then the agent transits to the next state and the RL model generates a new reward.

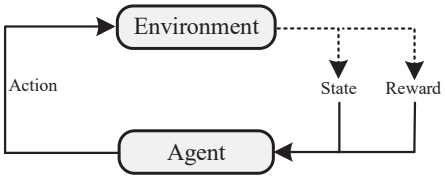


Fig. 3. The overview of RL. The agent interacts with the environment by taking action, and then the environment returns a reward and the agent transits to next state.

All the rewards got by the agent from the start to the end are noted as R_1, $\cdots$, R_t, $\cdots$, R_n. The return U_t is the accumulated sum of all the rewards from now (time t) on, which is defined in Eq. (1), and $\gamma \in [0, 1]$ indicates the discount factor. A higher γ indicates a bigger weight of the future rewards.

$$U_t = R_t + \gamma \cdot R_{t+1} + \gamma^2 \cdot R_{t+2} + \cdots + \gamma^n \cdot R_n \quad (1)$$

The agent takes the action that maximizes the expectation of return U_t at time t, which means the agent only concerns about the future rewards. At time t, the agent takes action a_t and the current state is s_t, Q-function $Q(s_t, a_t)$ (Eq. (2)) indicates the expectation of U_t when the agent takes the best policy $\pi^\star$. In other words, based on a_t and s_t, no matter what policy the agent takes in the future, expectation of U_t would not exceed the value of $Q(s_t, a_t)$. To get the largest expectation of U_t, the agent should take the action a_t which maximizes $Q(s_t, a_t)$ when the state is s_t at time t.

Based on Eq. (1) and Eq. (2), we could deduce the optimal Bellman equation Eq. (3), in which $\mathcal{A}$ means action space. In Eq. (3), $Q(s_t, a_t)$ indicates the expectation of U_t and $\underset{A\in\mathcal{A}}{max}Q(S_{t+1}, A)$ indicates the expectation of U_{t+1}.

$$Q(s_t, a_t) = \underset{\pi}{max}\,\mathbb{E}[U_t \mid S_t = s_t, A_t = a_t] \tag{2}$$

$$Q(s_t, a_t) = \mathbb{E}_{S_{t+1}}[R_t + \gamma \cdot \underset{A\in\mathcal{A}}{max} Q(S_{t+1}, A) | S_t = s_t, A_t = a_t] \tag{3}$$

After Monte Carlo approximation for Eq. (3), we obtain Eq. (4). We utilize Eq. (4) to train the deep learning network. The training process is presented in Section III-C.

$$Q(s_t, a_t) \approx r_t + \gamma \cdot \underset{a\in\mathcal{A}}{max}\, Q(s_{t+1}, a) \tag{4}$$

B. Design of State, Action, and Reward

1) State: In our design, a state in the RL model consists of 8 parameters. S_{size} is the size of the webpage, S_{node} is the number of the DOM tree nodes, S_{depth} is the depth of the DOM tree, $S_{\#style}$ is the number of the style attributes in HTML, and $S_{\#link}$ is the number of link tags in HTML. We refer to previous work [2] to select the above five web features of one webpage, which are the most significant features relating to PLT. S_{net} is the network environment state, and we classify it into seven categories, which are listed in Table I. S_{load} is the webpage loading state, which is either the on-load stage or the after-load stage. S_{power} is the power consumption generated until now from the moment that the webpage starts loading.

TABLE I
NETWORK ENVIRONMENT.

Type	Uplink bandwidth	Downlink bandwidth	Delay
Regular 2G	50kbps	100kbps	1000ms
Good 2G	150kbps	250kbps	300ms
Regular 3G	300kbps	550kbps	500ms
Good 3G	1.5Mbps	5.0Mbps	100ms
Regular 4G	1.0Mbps	2.0Mbps	80ms
Good 4G	8.0Mbps	15.0Mbps	50ms
WiFi	15Mbps	30Mbps	5ms

2) Action: Schedtune is a type of cgroup. The `schedtune.boost` file provides an interface for each schedtune cgroup to adjust the execution speed of the specific process. We select the value of `schedtune.boost` ranging from -90 to 90 and take 20 as an interval, so we get ten actions.

In DQN, we perform the ϵ-greedy algorithm to take action, which is a conventional method in RL. The algorithm is divided into **exploration** and **exploitation**. We perform exploration at ϵ probability to take random action, aiming to collect various samples. Exploitation is performed at the 1-ϵ probability to get the action that could maximize the Q-function value, i.e., the optimal action.

3) Reward: The value of a reward is closely related to the webpage loading state. In the on-load state, the reward is influenced by energy consumption. We put a negative factor before the energy consumption, instructing the agent to consume less energy. To make the value of reward positive, we add a positive constant to it. In the after-load state, the reward is influenced by energy consumption and PLT. We put negative factors before them and add another positive constant to the reward. The detailed design of the reward is shown in Algorithm 1.

C. Detailed Algorithm of KylinTune

The Algorithm 1 reveals the KylinTune algorithm. At the beginning of each episode, we get the current state s and the next state s'. The action a is got by the ϵ-greedy algorithm and the reward r is calculated with S_{load}. When the current webpage is loading, r is related to energy consumption. We set a negative factor β_1 before $energy$, which means more energy consumption would get less reward score. Aiming to avoid the reward always being negative, we add a positive constant $const_1$ to the reward. When the webpage is loaded, r is related to energy consumption and PLT, so we set a negative factor β_1 before $energy$ and a negative factor β_2 before PLT, indicating that less energy consumption and less PLT are encouraged. Similarly, we add another positive constant $const_2$ to the reward to adjust the value.

After getting a sample (s, a, r, s'), we add it into a replay memory D. Experience replay is a significant technique in RL, which could break the correlation of two adjacent samples and reuse experience to accelerate convergence. Replay memory D stores samples that contain the current state, current action, next state, and reward. Besides, we copy some samples that are accidental and more important ($r > 10$ or $r < -10$) into the replay memory, as more attention should be given to accidental samples.

When the replay memory is full of samples, we randomly select M batches from D to train the DQN model. During the training process at time t, the prediction value is the output of the neural network (Q-function value), while the target value y is $r + \gamma \cdot \underset{a\in\mathcal{A}}{max}\, Q(s_{t+1}, a)$. We perform gradient descent on the neural network parameters θ. To overcome bootstrap in the training process [9], we utilize a target network technique. The target network is another neural network, which is the same as the existing neural network except for parameters θ. During the process of updating θ of the neural network, target value y is calculated by parameters θ^- of the target network, prediction value (Q-function value) is calculated by parameters θ of the neural network. After training for a while, we update the parameters of the target network from θ^- to θ.

D. Implementation

Figure 4 is an overview of KylinTune implementation. We implement KylinTune in Chromium (version 84.0.4122.0) on Google Pixel2 XL.

We modify the Chromium engine to observe the state about web page characteristics. During the on-load stage, we get

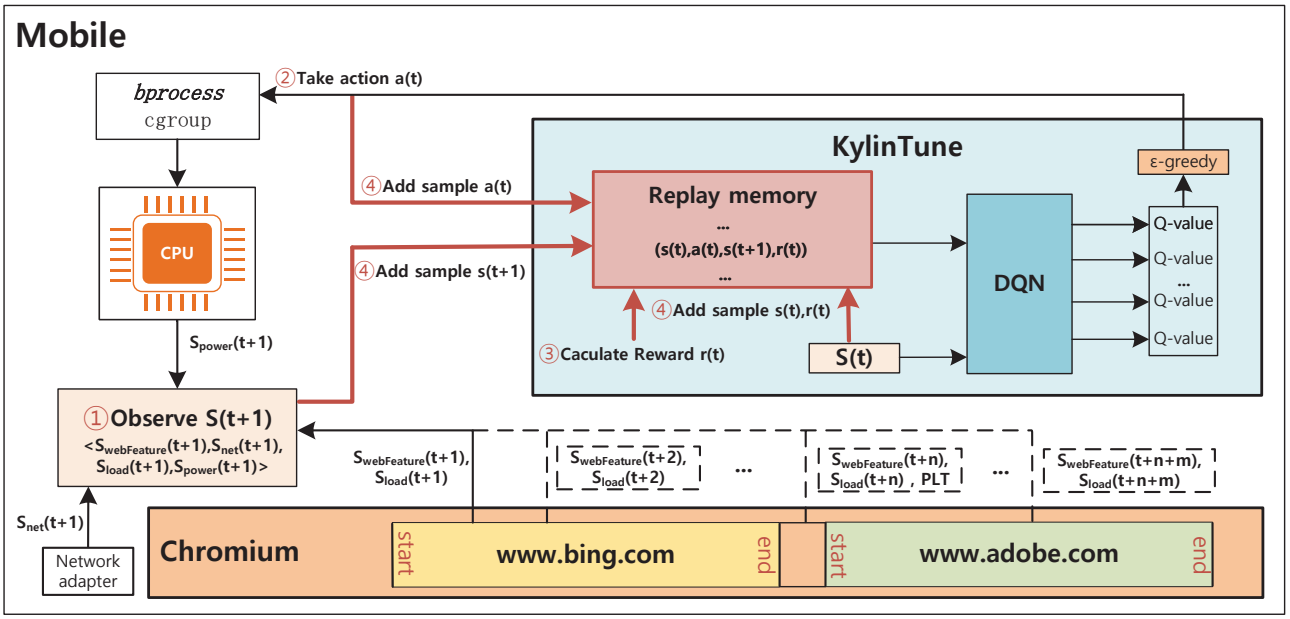


Fig. 4. The overview of KylinTune implementation. $S_{webFeature}$ consists of S_{size}, S_{node}, S_{depth}, $S_{\#style}$ and $S_{\#link}$. State in the dotted line means state in the future.

$S_{webFeature}$ (consisting of S_{size}, S_{node}, S_{depth}, $S_{\#style}$ and $S_{\#link}$) and S_{load} (indicating whether webpage is on loading); During the after-load stage, we get $S_{webFeature}$, S_{load} and PLT. We measure the time between the `NavigationStart` event and the `LoadEventEnd` event in the Chromium to get PLT. PLT only exists after the webpage has been loaded. We use Netem [10] to simulate network environment and different network corresponds to different S_{net}. The Monsoon Power Monitor [11] is used to get the power consumption state S_{power} for the whole mobile device (Figure 5).

Fig. 5. The Pixel2 XL device connected with the Monsoon Power Monitor.

The agent in DQN takes actions with the ϵ-greedy algorithm. After calculating the reward, a sample is added to the replay memory. Next, DQN starts to be trained by samples added to the replay memory when the memory is full of samples.

At the action stage of DQN, we modify the `init.rc` file of the Android open-source code and revise the max cgroup number to create a new cgroup `bprocess`. We put the IDs of three kinds of Chromium processes (browser process, render process, and GPU process) into the `cgroup.procs` file of `bprocess`. When the DQN takes an action, the `schedtune.boost` of `bprocess` would be set ranging from -90 to 90 at a 20 interval.

IV. Evaluation

A. Evaluation Setup

We implement KylinTune in Chromium browser (version 84.0.4122.0) on Google Pixel2 XL, the device specifications are listed in Table II.

The default CPU frequency governor in Pixel2 XL is `schedutil`, which is based on the EAS system. We evaluate KylinTune using the top 100 websites according to Alexa [5] in May 2022. These websites have large varieties (webpage sizes of these websites range from 3 KB to 3.1 MB and the number of DOM nodes ranges from 31 to 3940). The cumulative distribution function (CDF) of webpage size and the number of DOM nodes are shown in Figure 6. We utilize Sitespeed.io [12] tool to autoload websites. Before each loading evaluation of the 100 websites, we empty the browser cache. We set the screen brightness at the lowest level for all experiments to decrease its effect on energy consumption.

B. Energy Savings

We evaluate KylinTune with the default EAS in WiFi network and 3G cellular network environment. And then we evaluate KylinTune with EAS in hot loading of WiFi environment: we load the top 100 websites first and reload them without emptying the cache. In this scenario, resources of websites are still in the cache and DNS lookup results are also still in the local DNS cache [13].

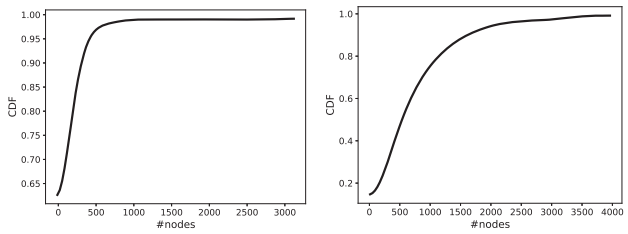
Algorithm 1: KylinTune algorithm

Constants: $\beta_1, \beta_2, const_1, const_2$
Initialize: Replay memory D, neural network θ,
capacity of replay memory C, ϵ=1,
target network $\theta^- = \theta$, learning rate α,
first state s
while *in experiment* **do**
  Get next state s';
  /* $\epsilon - greedy$ $algorithm$ */
  Get action a ;
  if *during webpage loading* **then**
    reward $r = \beta_1 \cdot energy + const_1$;
  else
    reward $r = \beta_1 \cdot energy + \beta_2 \cdot PLT + const_2$;
  end
  Append sample (s, a, r, s') into D;
  if $r > 10$ *or* $r < -10$ **then**
    Copy sample (s, a, r, s') and append it into D;
  end
  /* Training with the target network */
  if *number of samples in D exceed* $|C|$ **then**
    Randomly select M batches from D;
    for (s, a, r, s') *in* M **do**
      set $y = r + \gamma \cdot \max_{a' \in \mathcal{A}} Q(s', a'; \theta^-)$;
    end
    /* Perform gradient descent */
    $L(\theta) = 1/2(y - Q(s', a; \theta))^2$;
    $\nabla_\theta L(\theta) = (y - Q(s', a; \theta)) \cdot \nabla_\theta Q(s', a; \theta)$;
    $\theta = \theta - \alpha \cdot \nabla_\theta L(\theta)$;
    ϵ decays;
  end
  s=s';
  Every k steps, $\theta^- = \theta$;
end

TABLE II
EVALUATION PLATFORM.

Device	Google Pixel2 XL
Operating System	Android 10
Soc	Snapdragon835 MSM8998
big CPU	quad-core Kryo 280@ 2.35GHz
LITTLE CPU	quad-core Kryo 280@ 1.9GHz
ISA	ARMv8-A
Memory	4GB
GPU	Adreno 540

1) Evaluation in Fast Network: The `schedtune.boost` value set by KylinTune during the webpages loading process in WiFi network environment is shown in Figure 7. Our model sets the boost value much lower than the default EAS (the `schedtune.boost` value is set to 10 invariantly), which results in different CPU frequency usages as shown in Figure 8 (a, d). We learn that with KylinTune, the highest CPU frequency is less utilized and the lowest CPU frequency is more utilized, both in big and LITTLE CPU clusters. Figure 9 (a) shows energy savings in the WiFi environment and the average savings is 16.04%.



(a) Web page size (b) #Dom nodes

Fig. 6. The CDF of webpage size (a), #DOM nodes (b) .

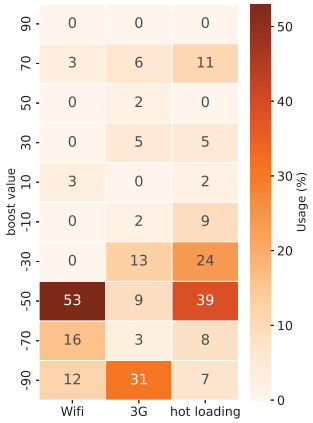
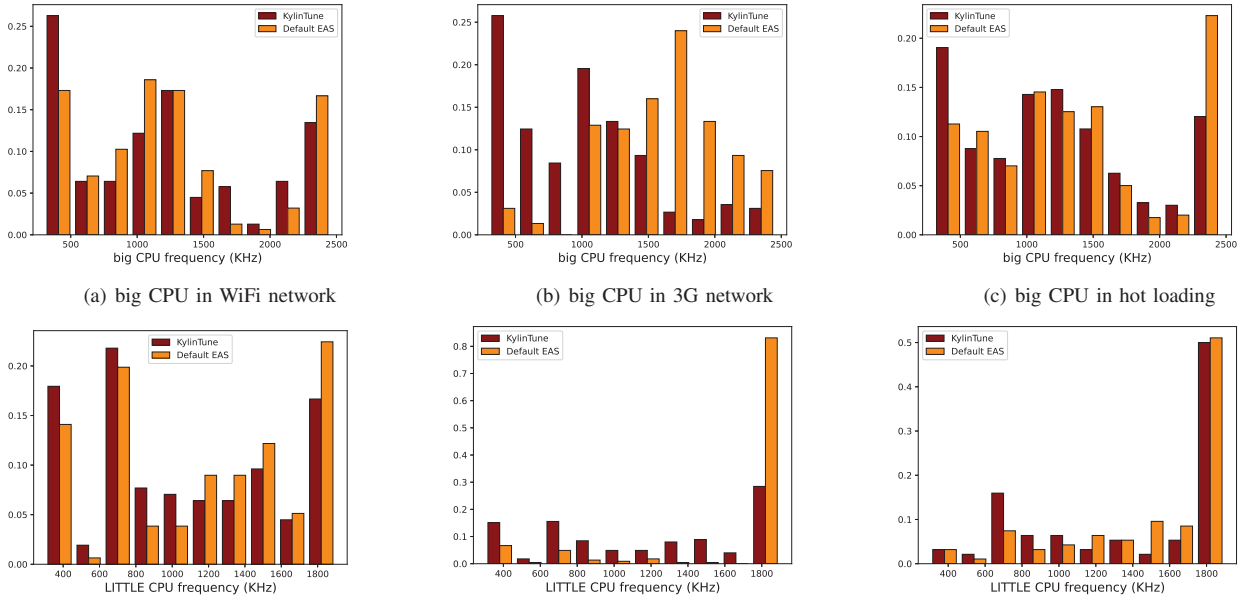


Fig. 7. *schedtune.boost* value usages in different loading environments.

2) Evaluation in 3G Network: As shown in Figure 7 and Figure 8 (b, e), in a 3G network environment, the `schedtune.boost` value is set to a lower level than the WiFi environment, bringing in lower CPU frequency usage than the WiFi environment. The results come from the following reason: when the network speed gets lower, there is no need to load webpage as fast as the WiFi environment, which thereby needs lower CPU frequency usage. The energy savings are shown in Figure 9 (b) and the average energy saving in the 3G network is 24.00%. KylinTune saves more energy in the 3G network than the WiFi network due to the lower boost value settings.

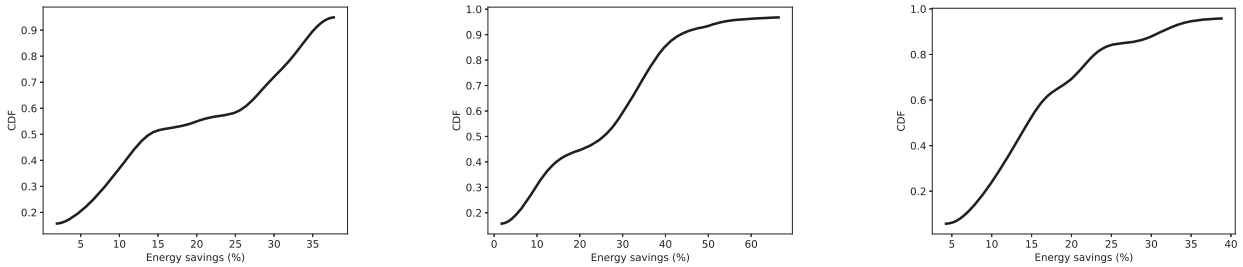
3) Evaluation in Hot Loading: In Figure 7 and Figure 8 (c, f), the boost value settings of hot loading are similar to the WiFi environment and not configured as low as the 3G environment, and the same is true for CPU frequency usage. In hot loading, a webpage needs to load faster, which results in higher boost value settings than the 3G environment and the WiFi environment in cold loading. Energy savings are shown in Figure 9 (c) and the average energy saving is 14.51%. Less energy is saved in hot loading than in cold loading (16.04%) because of the higher boost settings.



(a) big CPU in WiFi network (b) big CPU in 3G network (c) big CPU in hot loading

(d) LILLTE CPU in WiFi network (e) LILLTE CPU in 3G network (f) LILLTE CPU in hot loading

Fig. 8. big/LITTLE CPU frequency usage histogram.



(a) Energy savings in WiFi network (b) Energy savings in 3G network (c) Energy savings in hot loading

Fig. 9. Energy savings in different loading environments.

C. Page Load Time

We evaluate KylinTune with PLT in different network environments and the results are shown in Figure 10. In the 3G network environment, the average page load time is decreased by 2.13% and the average page load time is increased by 3.69% in the WiFi network environment. In a WiFi network environment, the average webpage load time with KylinTune increases, but the degree of QoS violation is acceptable. In a 3G network environment, the average webpage load time slightly decreased. The reason for the results could be that the page load time in the 3G network is much longer than in a fast network, so lower CPU frequency has more effect on page load time in a fast network.

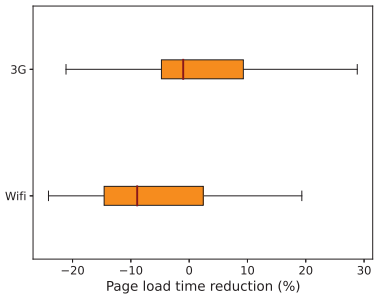


Fig. 10. Page load time in WiFi and 3G network environment.

D. Overhead and Comparison to Q-table

We also evaluate the overhead of KylinTune in Pixel2 XL. The interval between actions of DQN is approximately 100ms, and there is enough time for adjusting the boost value of a webpage loading process (page load time is at least 700ms in our evaluation).

Assuming that we implement the RL algorithm with Q-table and the state number of RL is 8. From our observation, bins about each web feature (first 5 states in RL) are 12; bins about network state are 3 due to 7 classifications; bins about load

state and power state are 1 and 7, respectively. The number of actions is 10, so the total number of rows in the Q-table is $12^5 \times 3 \times 7 \times 10$. In a 64-bit floating-point system, Q-table would take up to 398 MB of memory, which is not practical in our scenario.

However, in DQN, one sample takes up to $12 \times 5 + 3 + 7$ bins and the experience replay memory takes up to 8 KB in a 64-bit system. Given that Q-table takes up large memory, studies usually classify states into discrete features in order to decrease the bins it takes up, but it decreases the accuracy of the model correspondingly.

V. Related Work

A. Energy Optimization for Browser

Prior works for browser energy optimization consist of browser-level optimization and system-level optimization.

A lot of research focuses on optimizing the workflow of the web browser to achieve energy efficiency [14]–[17]. Thiagarajan et al. [18] analyzed the energy consumption of different web components and proposed guidelines for designing an energy-efficient website. Peters et al. [19] divided the browser workflow into different phases to adjust CPU frequency respectively. Hoque et al. [20] utilized parallel downloading for energy savings. The above techniques can be complementary to KylinTune, as KylinTune does not carry out browser-level optimization.

Regarding system-level optimization, Zhu et al. [21] proposed a model to predict webpage load time and energy consumption to select appropriate big/LITTLE core and CPU frequency. Ren et al. [2] collected webpage characteristics and utilized supervised learning to classify webpages into different categories to perform different CPU configurations. Bui et al. [13] improved browser energy efficiency by proposing rendering techniques. Singh et al. [22] focused on task scheduling to achieve energy-efficient in multicore systems. KylinTune adjusts the CPU execution speed for browser processes based on the EAS system, avoiding producing side effects on other processes.

B. RL for Mobile Power Management

Mobile devices do not have enough power like desktop devices. Reinforcement learning has been applied to many fields of power management of mobile devices. There are two RL methods: Q-table and Deep-Q Network.

Chen et al. [23] applied the Q-table approach to DVFS techniques and improved performance under power constraints. Shafik et al. [24] utilized Q-table to adjust voltage/frequency to meet performance requirements in embedded devices. Choi et al. [3] applied Q-table to the mobile web browser energy optimization.

On the other hand, research [25] analyzed two RL methods and find DQN works as well as Q-table, while DQN outperforms Q-table without state discretization and consumes less memory.

Gupta et al. [25] utilized DQN to dynamically adjust the core and frequency in a big/LITTLE architecture processor mobile. Shen et al. [26] proposed a DQN algorithm to achieve autonomous power management without prior information on the workload. Kim et al. [27] proposed ZTT based on DQN to dynamically adjust CPU and GPU frequency to achieve energy efficiency of mobile devices under thermal constraints. To the best of our knowledge, KylinTune is the first to apply DQN to mobile web browser energy optimization.

VI. Discussion and Future Work

Multitasks. KylinTune monitors one webpage (i.e. one tab of the browser) and adjusts the browser process execution speed correspondingly. However, it is common that a user opens several web pages together, putting one page in the foreground and leaving the other in the background. The load time of a new webpage and the total energy consumption could be influenced by the existing web pages in the background. In future work, we shall take multitasks into account and revise the state and reward definition of DQN in KylinTune.

Couple control with GPU. GPU plays a key role when the browser works because the GPU process utilizes GPU cores to handle GPU commands. It is feasible to control the CPU together with GPU when the DQN of KylinTune takes action. Besides, we plan to separate the GPU process from `bprocess` cgroup and put it into a new cgroup to realize the control independently. It's worth studying whether controlling the GPU process independently performs better than controlling the whole three types of processes together.

Control after the webpage having been loaded. KylinTune adjusts the execution speed of the browser in the on-load state. However, users make interaction with webpage in the after-load state. In future work, we plan to collect features of the after-load state and take the interactions with users into account.

VII. Conclusion

Currently, mobile browsers could not achieve energy-efficient given that not only DVFS but also EAS technique set a static configuration on CPU and could not adjust CPU frequency dynamically corresponds to webpage characteristics. In this paper, we propose KylinTune, a model based on DQN, to learn the runtime environment of a specific webpage and adjust the execution speed of the browser process dynamically on the basis of EAS. We compare KylinTune to the default EAS technique in different loading environments and compare DQN to the Q-Table method. Results show that KylinTune outperforms the default EAS in energy savings and has advantages over Q-table in memory usage and model accuracy.

References

[1] J. Ren, L. Gao, H. Wang, and Z. Wang, "Optimise web browsing on heterogeneous mobile platforms: A machine learning based approach," *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, pp. 1–9, 2017.

[2] J. Ren, X. Wang, J. Fang, Y. Feng, D. Zhu, Z. Luo, J. Zheng, and Z. Wang, "Proteus: network-aware web browsing on heterogeneous mobile systems," *Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies*, 2018.

[3] Y. Choi, S. Park, and H. Cha, "Optimizing energy efficiency of browsers in energy-aware scheduling-enabled mobile devices," *The 25th Annual International Conference on Mobile Computing and Networking*, 2019.
[4] "Chrome," https://www.google.com/chrome/ Accessed June 7, 2022.
[5] "Alexa.com," http://www.alexa.com/topsites Accessed May 7, 2022.
[6] "Energy aware scheduling," https://www.kernel.org/doc/html/latest/scheduler/sched-energy.html Accessed May 16, 2022.
[7] M. M. Sabry, M. Gao, G. Hills, C.-S. Lee, G. Pitner, M. M. Shulaker, T. F. Wu, M. Asheghi, J. Bokor, F. Franchetti, K. E. Goodson, C. E. Kozyrakis, I. L. Markov, K. Olukotun, L. T. Pileggi, E. Pop, J. M. Rabaey, C. Ré, H. S. P. Wong, and S. Mitra, "Energy-efficient abundant-data computing: The n3xt 1,000x," *Computer*, vol. 48, pp. 24–33, 2015.
[8] I. O'Connor, A. Poittevin, S. L. Beux, A. Bosio, Z. Stanojevic, O. Baumgartner, C. Mukherjee, C. Maneux, J. Trommer, T. Mikolajick, and G. Larrieu, "Analysis of energy-delay-product of a 3d vertical nanowire fet technology," *2021 Joint International EUROSOI Workshop and International Conference on Ultimate Integration on Silicon (EuroSOI-ULIS)*, pp. 1–4, 2021.
[9] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.
[10] A. Jurgelionis, J.-P. Laulajainen, M. Hirvonen, and A. I. Wang, "An empirical study of netem network emulation functionalities," in *2011 Proceedings of 20th International Conference on Computer Communications and Networks (ICCCN)*, 2011, pp. 1–6.
[11] "Monsoon power monitor," https://www.msoon.com/ Accessed May 29, 2022.
[12] "Sitespeed.io," https://www.sitespeed.io Accessed May 29, 2022.
[13] D. H. Bui, Y. Liu, H. Kim, I. Shin, and F. Zhao, "Rethinking energy-performance trade-off in mobile web page loading," *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, 2015.
[14] F. Qian, K. S. Quah, J. Huang, J. Erman, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck, "Web caching on smartphones: ideal vs. reality," in *MobiSys '12*, 2012.
[15] X. Xie, X. Zhang, and S. Zhu, "Accelerating mobile web loading using cellular link information," *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, 2017.
[16] Z. Wang, F. X. Lin, L. Zhong, and M. Chishtie, "How far can client-only solutions go for mobile browser speed?" *Proceedings of the 21st international conference on World Wide Web*, 2012.
[17] H. Mai, S. Tang, S. T. King, C. Cascaval, and P. Montesinos, "A case for parallelizing web pages," in *HotPar'12*, 2012.
[18] N. Thiagarajan, G. Aggarwal, A. Nicoara, D. Boneh, and J. P. Singh, "Who killed my battery?: analyzing mobile browser energy consumption," *Proceedings of the 21st international conference on World Wide Web*, 2012.
[19] N. Peters, S. Park, D. Clifford, S. Kyostila, R. Mcllroy, B. Meurer, H. Payer, and S. Chakraborty, "Phase-aware web browser power management on hmp platforms," *Proceedings of the 2018 International Conference on Supercomputing*, 2018.
[20] M. A. Hoque, S. Tarkoma, and T. Anttila, "Poster: Extremely parallel resource pre-fetching for energy optimized mobile web browsing," *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, 2015.
[21] Y. Zhu and V. J. Reddi, "High-performance and energy-efficient mobile web browsing on big/little systems," *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, pp. 13–24, 2013.
[22] A. K. Singh, M. A. Shafique, A. Kumar, and J. Henkel, "Mapping on multi/many-core systems: Survey of current and emerging trends," *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–10, 2013.
[23] Z. Chen and D. Marculescu, "Distributed reinforcement learning for power limited many-core system performance optimization," *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1521–1526, 2015.
[24] R. A. Shafik, S. Yang, A. Das, L. A. Maeda-Nunez, G. V. Merrett, and B. M. Al-Hashimi, "Learning transfer-based adaptive energy minimization in embedded systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, pp. 877–890, 2016.
[25] U. D. Gupta, S. K. Mandal, M. Mao, C. Chakrabarti, and U. Y. Ogras, "A deep q-learning approach for dynamic management of heterogeneous processors," *IEEE Computer Architecture Letters*, vol. 18, pp. 14–17, 2019.
[26] H. Shen, Y. Tan, J. Lu, Q. Wu, and Q. Qiu, "Achieving autonomous power management using reinforcement learning," *ACM Trans. Design Autom. Electr. Syst.*, vol. 18, pp. 24:1–24:32, 2013.
[27] S. Kim, K. Bin, S. Ha, K. Lee, and S. Chong, "ztt: learning-based dvfs with zero thermal throttling for mobile devices," *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*, 2021.