IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 14, 2019, accepted June 9, 2019, date of publication June 17, 2019, date of current version July 2, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2923334

Energy-Efficient Control of Mobile Processors Based on Long Short-Term Memory

JAEHWAN LEE, SANGHYUCK NAM, AND SANGOH PARK
School of Computer Science and Engineering, Chung-Ang University, Seoul 06974, South Korea

Corresponding author: Sangoh Park (sopark@cau.ac.kr)

This work was supported in part by the National Research Foundation of Korea (NRF) funded by the Korea government (MSIT) under Grant NRF-2017R1C1B5075856 and in part by the "Cooperative Research Program for Agriculture Science Technology Development (Project No. PJ01395912019)" from the Rural Development Administration, Republic of Korea.

ABSTRACT Smartphones that are equipped with high-clock frequency and multi-core processors are being commercially released to provide various services. As the number of cores and the clock speed of a mobile processor increases, its power consumption also increases, and several software approaches to reducing power consumption have been studied. Existing techniques estimate processor usage by measuring the processor usage at a previous time. However, these techniques often waste energy because they assign frequencies above the usage required to prevent degraded user responsiveness. Therefore, this paper proposes a machine learning method to predict the usage that the processor currently requires to prevent performance degradation while reducing power consumption. The proposed method is implemented through a processor power management system based on Long Short-Term Memory (LSTM). This system learns processor usage patterns in a variety of situations and predicts the processor usage required for the current situation. The number of computations required by the LSTM-based technique is analyzed according to the number of neurons and layers, and the computational load is then compared to an existing technique. Furthermore, a benchmarking tool that reflects the characteristics of mobile applications is used to test the performance of the proposed system, which is shown to reduce the power consumption of mobile processors by a maximum of 19% compared to the existing Android processor power management system.

INDEX TERMS Energy management, dynamic voltage and frequency scaling, recurrent neural networks, mobile device.

I. INTRODUCTION

As smartphones have evolved and become more widespread, users have spent increasing amounts of time using these devices. Therefore, smartphone manufacturers are strengthening the competitiveness of their products in a variety of respects, including improved performance, extended usage time, and improved user convenience. These enhanced services have necessitated multi-core processors running at high-clock frequencies. However, as the processor's performance increases, its power consumption also increases. To address this problem, manufacturers have adopted low-power hardware technologies and increased battery capacities, but these approaches have had limited success. Therefore, a variety of software-based approaches have been applied to operating systems on smartphones control processors, such as Dynamic Voltage and Frequency Scaling (DVFS) [1], which adjusts the frequency and voltage according to the processor usage, and hotplugging [2], which adjusts the activation states of the processor's individual cores.

Unlike the operating systems used in computers, smartphone operating systems must consider user responsiveness when managing processor power. Computer-based operating systems assign all processor resources as they are needed by applications, controlling the operating frequency and the number of activated cores accordingly. In smartphone operating systems, the responsiveness perceived by the user is affected more by the foreground application with which the user is interacting than the applications that are running in the background. Therefore, several studies [3]–[7] have been conducted on limiting processor power consumption by assigning maximum processor resources to the foreground

The associate editor coordinating the review of this manuscript and approving it for publication was Anup Kumar Goswami.

application and limiting resources to background applications. However, the previously proposed approaches have wasted energy by allocating more resources than the processor required to prevent performance degradation as perceived by the user. Existing DVFS-based techniques assume that accurate predictions of how much processor usage will be needed in the future are difficult to achieve by analyzing the current processor usage. Therefore, they increase the processor's frequency to the maximum value, preventing a decrease in user-perceived performance when the processor's usage exceeds a threshold where the current processor usage is considered to be high.

Recently, studies [8]–[10] focused on maximizing energy efficiency, or performance per watt (PPW), by finding mobile processors' performance-energy configuration. The existing methods to find optimal PPW could not prevent degradation of application performance with respect to the interactive governor[11]. They always choose the maximum PPW configuration which leads to the reduced execution performance of processors. The amount of how much processor performance is needed was not considered.

This paper proposes a method based on Long Short-Term Memory (LSTM) to track and predict processor usage more accurately than the existing methods. LSTM is an extension of Recurrent Neural Networks (RNN) [12], [13]. Processor usage information constitutes time-series data that change at each fixed time interval. Therefore, a network based on LSTM is designed to analyze and predict the time series processor usage data. If processor usage can be more accurately predicted, processor resources can be assigned as they are needed to minimize the processor performance degradation with further reduced power consumption.

The remainder of this article is organized as follows. In Section 2, we summarize the existing studies on smartphone power management techniques and studies on LSTM. In Section 3, we introduce the proposed processor power management architecture. In section 4, we report the power consumption and performance evaluation results of our proposed system and compare it with existing system. Finally, in Section 5 we present conclusions and describe future work.

II. RELATED WORKS

A. PROCESSOR POWER MANAGEMENT

Processor power consumption can be reduced by dynamically adjusting the processor's operating frequency or inactivating its cores. The Android processor power management system, which is typically used in smartphones, is shown in Fig. 1. The core controller measures the processor core loads and adjusts the number of active cores. The governor measures the processor's load and adjusts the processor's operating frequency. The processor's power consumption is proportional to the operating frequency and the square of the input voltage [14]. Thus, the governor uses the DVFS technique to reduce power consumption.

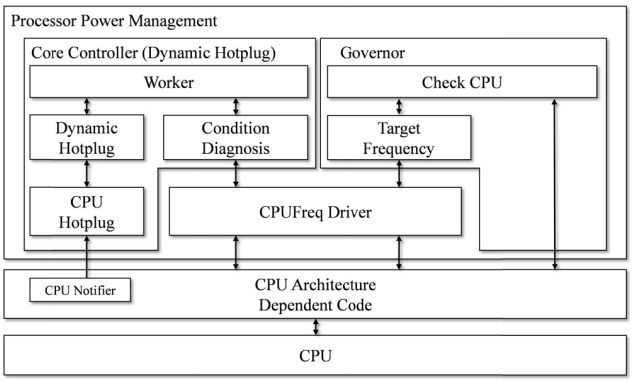

FIGURE 1. Android processor's power management architecture.

If the processor usage that was sampled from the previous time period exceeds the experimentally established threshold value for efficient operation of the target device, Android's on-demand governor [15] and interactive governor [11] determine that the current processor usage is high and set the processor's operating frequency to the maximum value. The frequency is then lowered incrementally according to the processor usage to prevent performance degradation. In such systems, unnecessary power consumption can occur. Computer-based operating systems allocate all processor resources required by all running applications and control the processor frequency and number of active cores. However, in smartphone-based operating systems, power consumption can be reduced without user-perceived performance degradation by considering the response time of the foreground applications. In previously proposed approaches [3]–[7], the power management system examines the processor usage of foreground applications and assigns processor resources as needed, and processor resources for background applications are limited to reduce processor power consumption.

In studies [4], [6] on reducing power consumption when running a video playback application in a smartphone-based operating system, the resources required during decoding were predicted and processor resources were allocated accordingly. An integrated CPU-GPU governor based on machine learning[16] was introduced to improve energy efficiency with minimizing performance degradation. The integrated governor infers whether the current state is performance demanding or powersaving from current frames per second of a game, processor frequency, and processor usage. In a study [3] on increasing power efficiency by examining the characteristics of applications that used smartphone sensors, the current situational information was analyzed to determine whether the processor, network equipment, and sensor equipment were to be turned on or off. Another study [7] increased energy efficiency by controlling the governor and hotplug depending on whether smartphone applications were running in the foreground or background. In this study, the sensitivity was divided into three levels—high, medium, and low—considering perceived

performance. The technique estimated the required processor usage according to the applications' sensitivities and set the processor frequency accordingly. When the sensitivity was high, maximum processor usage was allocated to prevent perceived performance degradation. When the sensitivity was medium or low, the processor usage was limited to reduce power consumption.

The existing techniques are limited in that they propose power management strategies that are optimized for applications with specific purposes, and thus these techniques are difficult to use in a general application usage environment. Furthermore, even when the techniques can be used universally, they allocate more processor resources than are needed to prevent performance degradation. On the other hand, studies[8]–[10] were conducted to optimize energy efficiency of mobile processors by finding configurations that maximize performance per watt (PPW) or instructions per second per watt (IPS/W) in a given state. These methods inevitably result in performance degradation of application performance since higher frequency requires more supply voltage and current, and the processor power consumption is proportional to the square of the current. As an example, a machine learning based scheme to find Pareto-optimal processor configuration of energy and performance in a given execution phase[8], showed approximately 2 times higher PPW but also showed approximately 2 times longer execution time than the interactive governor in performance evaluation. The methods rather choose a processor configuration with much lower performance than the ondemand and interactive governor to maximize PPW. The amount of how much processor performance is needed or wasted in a given state was not considered.

Currently, AMD and Samsung are launching processors [17], [18] with branch prediction technology [19], which uses neural networks implemented at the processor's silicon die level. These processors require less usage than in existing technologies in a given work environment, and therefore offer reduced power consumption. However, these technologies are limited in that their complex high-level design is difficult to adjust structurally.

Therefore, this paper proposes a method whereby processor usage data is collected and learned to predict current processor usage more accurately than existing methods and to set power levels accordingly. The proposed method in this paper also reduces power consumption effectively while preventing degradation in application performance. For this purpose, an LSTM-based processor usage prediction model and a neuro-governor architecture are proposed. This method offers the advantages of easier design and implementation compared with hardware-based approaches, and the proposed method also operates independently of the processor architecture by reducing power consumption via software.

B. LONG SHORT-TERM MEMORY

LSTM [13], [20] is a model that was proposed to overcome a limitation of existing RNNs [12], whereby they could learn correlations only for short-term data because of the gradient vanishing [21] problem. Fig. 2 shows the LSTM architecture, and the parameters associated with this structure are defined in Table I. LSTM adds the forget gate f_t, input gates g_t and i_t, memory cell c_t, and output gate o_t to the existing RNN neuron [12], in which only *tanh* exists. The forget gate first adjusts the disposal of data saved in the memory cell; it is defined as the value obtained by applying sigmoid to the input x_t at time t and the value h_{t-1} output by the LSTM at time $t-1$, as shown in Eq. (1).

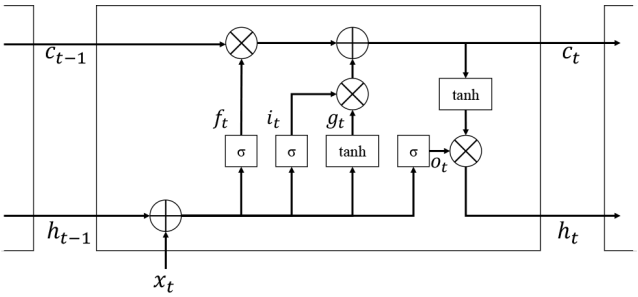

FIGURE 2. An architecture of long short-term memory.

TABLE 1. Long short-term memory parameter definitions.

Symbol	Definition
t	Time at which an input vector is fed to an LSTM
x_t	Input vector to an LSTM at time t
h_t	Output vector of LSTM at time t
σ	Sigmoid activation function
$tanh$	Tangent hyperbolic activation function
f_t	Forget gate function at time t to control the amount of discarded stored data
c_t	Memory cell to store data in LSTM at time t
g_t	Input gate function at time t to generate data to be stored in c_t
i_t	Input gate function at time t to select data to be stored in c_t
o_t	Output gate function at time t to generate data as an output of LSTM
$w_{x_f}, w_{x_g}, w_{x_i}, w_{x\_o}$	Vector weighted to x_t in functions f_t, g_t, i_t, o_t, respectively
$w_{h_f}, w_{h_g}, w_{h_i}, w_{h\_o}$	Vector weighted to h_{t-1} in functions f_t, g_t, i_t, o_t, respectively
b_f, b_g, b_i, b_o	Bias constant for functions f_t, g_t, i_t, o_t, respectively

$$f_t = \sigma(x_t w_{x\_f} + h_{t-1} w_{h\_f} + b_f) \tag{1}$$

The input gate g_t creates the value saved in the memory cell from x_t because of h_{t-1}, as expressed in Eq. (2). The input gate i_t selects the value saved in the memory cell at x_t from h_{t-1}, as expressed in Eq. (3).

$$g_t = tanh(x_t w_{x\_g} + h_{t-1} w_{h\_g} + b_g) \tag{2}$$

$$i_t = \sigma(x_t w_{x\_i} + h_{t-1} w_{h\_i} + b_i) \tag{3}$$

The data left in the memory cell because of the forget gate and the new data to be saved from the input gate are determined, and the memory cell at time t is updated, as expressed in Eq. (4).

$$c_t = f_t \circ c_{t-1} + i_t \circ g_t \tag{4}$$

The output gate determines the value to be output by LSTM. A sigmoid is applied to h_{t-1} and x_t, as expressed in Eq. (5).

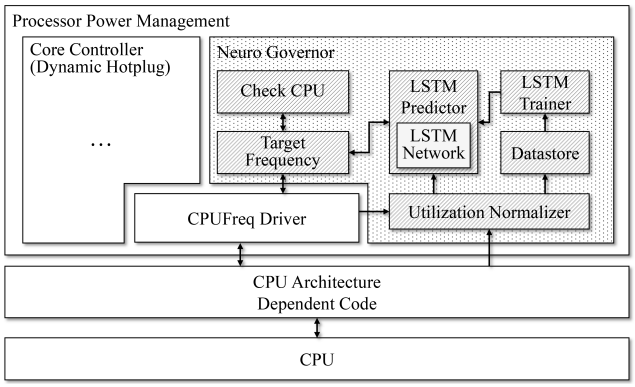

FIGURE 3. Neuro-governor architecture based on LSTM.

Subsequently, h_t, which is the LSTM's output at time t, is the output value determined at the output gate multiplied by the data saved in the memory cell at time t, as expressed in Eq. (6).

$$o_t = \sigma(x_t w_{x\_o} + h_{t-1} w_{h\_o} + b_o) \tag{5}$$

$$h_t = o_t \circ tanh(c_t) \tag{6}$$

LSTM learning is performed through the Back-Propagation Through Time (BPTT) [22], [23] algorithm. The BPTT algorithm unfolds a recursive structure unit, in which the output at time t is influenced by that at time $t-1$, as many times as the number of data items simultaneously learned, such as in the RNN neuron or LSTM. The BPTT algorithm then creates a sequential structure and applies the backpropagation algorithm [24]. After data input processing, the difference between the actual output and the target output is calculated through an error function, and the calculated error value is again backpropagated to update the weights.

LSTM shows excellent prediction performance when applied to time-series data that are temporally correlated, and it is used in a variety of fields such as natural language processing [25], stock price or market index prediction [26], [27], human activity classification [28], and object movement path prediction [29] , [30]. In this paper, the processor usage information that is to be predicted to reduce processor power consumption is time-series data that are sampled at discrete time intervals. Therefore, we design an LSTM network that can analyze and predict processor usage.

III. CONTROLLING MOBILE PROCESSORS WITH NEURO-GOVERNOR

This paper proposes a neuro-governor that predicts processor usage and controls processor power based on processor usage patterns. The overall architecture of the processor power management system, which includes the neuro-governor, is shown in Fig. 3. Additional parameters related to the neuro-governor are defined in Table II.

Processor performance varies not only according to architecture, but also according to the current operating frequency. Therefore, processor usage must be normalized to determine

TABLE 2. Neuro-governor parameter definitions.

Symbol	Definition
n	Number of cores in a processor
m	Number of entries in a processor frequency table
p_j	j-th value of a processor frequency table
$r_\forall$	Maximum number of instructions that all cores can execute in one second
ν_{l_t}	Frequency of the l-th core at time t
ν_{l_min}	Operable minimum frequency of the l-th core
ν_{l_max}	Operable maximum frequency of the l-th core
$er\nu_{l\_t}$	Estimated maximum number of instructions that the l-th core can execute in one second at ν_{l_t}
$r\nu_{l\_min}$	Number of instructions that the l-th core can execute in one second at ν_{l_min}
$r\nu_{l\_max}$	Number of instructions that the l-th core can execute in one second at ν_{l_max}
u_{l_t}	Utilization rate of the l-th core at time t
$\hat{u}_t$	Normalized utilization rate of all cores at time t
x_{t_k}	k-th element of an input vector x_t, where $x_{t\_k} \in \{0,1\}$
z_t	Most energy-efficient processor frequency index that does not incur performance degradation for $\hat{u}_t$

the amount of processor resources required by applications based on the maximum performance that can be achieved by the processor. Here, processor usage is normalized with a utilization normalizer that defines the system's maximum amount of processing power $r_\forall$ as the sum of $r\nu_{l\_max}$ for all n cores, as shown in Eq. (7).

$$r_\forall = \sum_{l=1}^{n} r\nu_{l\_max} \tag{7}$$

The unit of $r\nu_{l\_max}$, $r\nu_{l\_min}$ in this paper is DMIPS, with 1 DMIPS representing the speed of a reference machine. Most of the industries adopted the VAX 11/780 as the reference 1 DMIPS machine on Dhrystone benchmark [31], [32]. For example, 80 DMIPS means that a processor running Dhrystone benchmark is 80 times faster than the reference machine. The $r\nu_{l\_max}$, $r\nu_{l\_min}$ values are then obtained by running Dhrystone benchmark. In addition, if it is assumed that the performance of each core changes linearly according to the frequency[33], then $er\nu_{l\_t}$ can be defined as in Eq. (8), using linear interpolation when the frequency of the l-th processor core at time t is ν_{l_t}.

$$er\nu_{l\_t} = r\nu_{l\_min} + (r\nu_{l\_max} - r\nu_{l\_min}) \frac{\nu_{l\_t} - \nu_{l\_min}}{\nu_{l\_max} - \nu_{l\_min}} \tag{8}$$

Lastly, $\hat{u}_t$ is the normalized processor usage at time t, as defined in Eq. (9).

$$\hat{u}_t = \frac{\sum_{i=1}^{n} er\nu_{l\_t} u_{l\_t}}{r_\forall} \tag{9}$$

The utilization normalizer refers to the processor frequency table of the CPU frequency driver for the normalized processor usage $\hat{u}_t$ and finds the index z_t of the most energy-efficient processor frequency that does not incur performance degradation. Subsequently, z_t, defined as in Eq. (10), is output

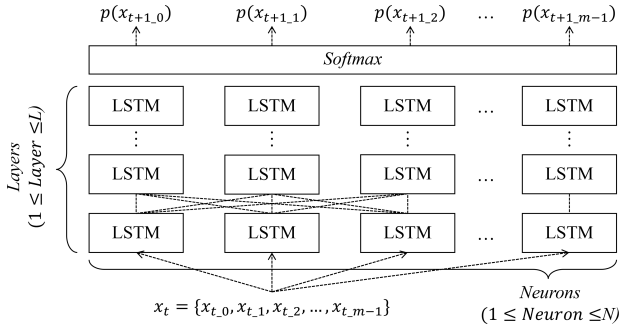

FIGURE 4. Fully connected LSTM network architecture to predict processor utilization.

to the LSTM predictor and Datastore.

$z_t = j$, *where*

$$p_{j-1} < \hat{u}_t \times v_{l\_max} \leq p_j (0 \leq j \leq m-1) \quad (10)$$

The LSTM predictor converts z_t to the input vector x_t, as shown in Eq. (11), and inputs it to the trained LSTM network. The LSTM network then performs forward propagation to predict the future processor frequency index.

$$x_t = \{x_{t_0}, x_{t_1}, x_{t_2}, \ldots, x_{t_m-1}\}, \textit{where}$$
$$x_{t\_k} = \begin{cases} 1, & \textit{if } z_t = k \\ 0, & \textit{otherwise} \end{cases} \textit{for } (0 \leq k \leq m-1) \quad (11)$$

Fig. 4 shows the architecture of the LSTM network for predicting processor usage. This architecture is a fully connected structure with L layers and N LSTM neurons. The output of the final layer in the LSTM network is normalized to a probability value via softmax [34]. The LSTM network outputs $p(x_{t+1\_k})$, which is the probability from softmax that the k-th index of the processor frequency table is selected at time $t+1$. The LSTM predictor determines the k with the largest probability to be z_{t+1}, defined as shown in Eq. (12), and outputs that k to the target frequency, which in turn sets the processor frequency by referring to the frequency table.

$$z_{t+1} = \underset{k}{argmax}\, p(x_{t+1\_k}) \textit{ for } (0 \leq k \leq m-1) \quad (12)$$

The LSTM trainer loads the stored z_t records from Datastore and trains the LSTM network. The LSTM is unfolded to the length of the data which are to be learned at once, forming a feed-forward network. The LSTM trainer then uses a BPTT algorithm to update the network's weight parameters through backpropagation, using the mean squared error (MSE) [35] as the backpropagation loss function and the ADAM [36] optimizer to make the training process efficient. When training is complete, the LSTM trainer updates the LSTM network. If the LSTM network has not been trained yet, Check CPU operates in interactive governor mode.

IV. EXPERIMENTAL EVALUATION

In this section, we describe the test environment and methods for measuring the performance of the proposed neuro-governor and its performance results. A Google Nexus 6P smartphone with a Qualcomm Snapdragon 810 processor, detailed in Table III, was used as the target device for the neuro-governor performance assessment. None of the processor cores were manually deactivated in the performance benchmark. In Section 4.1, we analyze the number of computation cycles required for the LSTM network architecture in the target device. In Section 4.2, we compare the performance of Android's existing power management technique to the neuro-governor proposed in this paper.

TABLE 3. Target device specifications.

Parameter	Specification
Model	Google Nexus 6P
Processor	Qualcomm Snapdragon 810 MSM8994 SoC ARM Cortex-A57 2GHz + ARM Cortex-A53 1.55GHz
Memory	3GB LPDDR4 SDRAM
Display	WQHD(2560x1440) AMOLED Touchscreen Display
Storage	32GB Flash Storage
OS	Android 7.1.2 (Nougat) with Linux Kernel 3.10.73

TABLE 4. Processor usage characteristics of applications.

Type	Definition
H/C	Average, maximum, minimum processor utilization is medium or high
H/B	Average processor utilization is low, and maximum processor utilization is medium or high
L/C	Average and maximum processor utilization is low

A. DESIGN OF LSTM NETWORK

We designed an LSTM network architecture to identify processor usage patterns and predicted the most efficient processor frequencies based on processor usage. To establish an efficient LSTM network architecture, prediction accuracy for processor usage was collected by changing the number of layers of the LSTM network, denoted by L, and the number of LSTM neurons per layer, denoted by N.

A tool running on Android was developed to collect training data with the characteristics of mobile applications in a simulated user environment. As described in Table IV, depending on the processor usage characteristics, mobile applications can be categorized as H/C a type that continuously uses the processor, H/B a type that uses the processor in the near-idle state and then uses the processor more intensively for a short period of time, and L/C a type that keeps the processor in a near-idle state. Note that the processor utilization is equally divided into three intervals and denoted as low, medium, high. Because applications that use the processor in the near-idle state for a short period of time are included in either H/C or H/B, development of a tool according to the type H/C, H/B, and L/C can reflect all application characteristics.

Previous research has described a reproducible method of collecting processor usage [4], [6], [7] to evaluate performance. Therefore, with MediaCodec and WebView from Android's open source API, we tested the LSTM network architecture with processor usage data samples collected

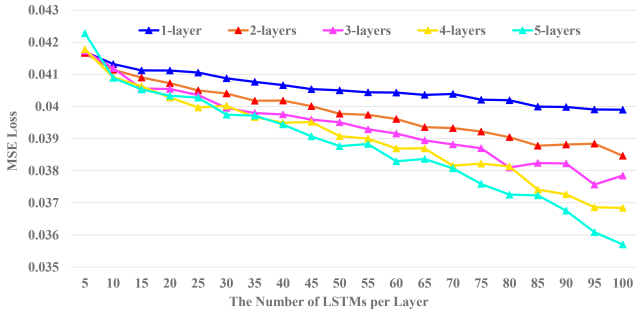

FIGURE 5. MSE loss for each LSTM architecture.

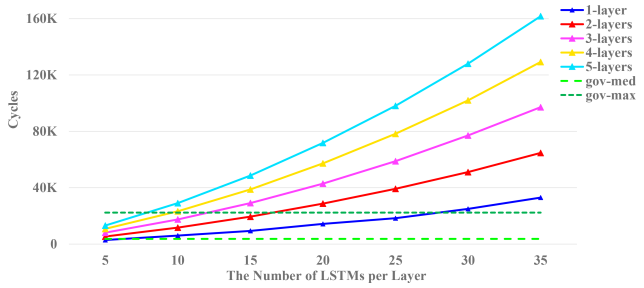

FIGURE 6. Measured total cycles needed to execute.

during CPU decoding of 2160p H.264 video to reflect type H/C and Vorbis music to reflect type L/C, as well as during web browsing using WebView to reflect type H/B.

The MSE loss of the LSTM network is shown in Fig. 5. The maximum difference of MSEs was approximately 0.0053 when the number of LSTM neurons N was 10. The MSE decreased as the number of LSTM neurons and the number of layers increased.

Because we aim to deploy the LSTM network in a smartphone to reduce power, the computational overhead of the LSTM network's forward propagation task must be considered. Therefore, the computational overhead of the LSTM predictor's code was analyzed with different numbers of layers and LSTM neurons per layer in the LSTM model. The number of execution cycles required by the LSTM network can vary in different implementations; however, with a fully connected architecture, the number of cycles increases exponentially with N and increases linearly with L and m. The computational load required for the forward propagation of the LSTM network implemented in this paper was calculated as follows. A layer consisting of N LSTM neurons with each neuron performs the arithmetic operation described by Eq. (1), where the input x_t has m elements and the network is a fully connected architecture so that each neuron performs multiplication of h_{t-1} with size N. Because a neuron takes a sigmoid of the multiplication result, the number of computations required to obtain f_t for every neuron in a layer is $N(m + N + 1)$. This approach also applies equally to Eqs. (2), (3), and (5). In Eq. (4), the number of computations is $2N$ because two multiplication operations are performed for each neuron. Each neuron uses h_t to generate data with size m to be passed as an input to the neuron in the next layer, and the number of computations for this step is Nm.

When all neurons in the LSTM network finish their operations, a softmax is performed on the output data with m elements. By definition, the softmax operation applies a natural exponential function and division operation to each of the m elements; therefore, a total of $2m$ operations are performed. The cumulative number of computations p is defined as in Eq. (13).

$$p = L\left(4N^2 + 5Nm + 8N\right) + 2m \tag{13}$$

In addition to the analysis of the LSTM network architecture, the PMU register [37] of the ARM processor was used to collect the processor cycles required for the target device to execute the code. Fig. 6 shows the number of cycles obtained by Eq. (13) with varying numbers of layers and LSTM neurons per layer. The number of cycles of the existing interactive governor were analyzed and added to compare the LSTM network to the existing method. Because of a branch in the interactive governor's code, the LSTM network does not always execute the same amount of code. We defined the number of cycles required by the interactive governor to perform all the codes as gov-max and the number of cycles reduced by the branch as gov-med.

In this paper, the LSTM architectures used required fewer cycles to execute their functions than gov-max. The selected 10 LSTM network architectures consisted of 5, 10, 15, 20, and 25 LSTM neurons for one layer; 5, 10, and 15 neurons for two layers; and 5 and 10 neurons for three layers. Given that the execution overhead exponentially increases with N, an architecture with a number of computations larger than gov-max will have a power consumption disadvantage that exceeds the advantages gained from the increased accuracy.

The MSE loss of 10 selected LSTM network architectures is shown in Fig. 7. The losses start to converge around 10k training samples, and the losses are oscillating in converging process.

B. EVALUATION OF NEURO-GOVERNOR

The smartphone's battery was removed and replaced with Hardkernel's SmartPower2 monitoring device to measure the smartphone's current power consumption. The hardware specifications of the power measuring device are described in Table V. The supply voltage was set to 4.33V during the benchmark. The smartphone's average power consumption in Ampere was calculated from the accumulated power consumption data. The smartphone was switched to airplane mode to minimize impacts not related to the processor power consumption, such as wireless and peripheral devices. The display brightness is also set to its minimum.

Each LSTM network architecture is denoted according to the number of layers and the number of LSTM neurons per layer in benchmark results. For example, an LSTM network architecture consisting of one layer and 5 neurons is denoted

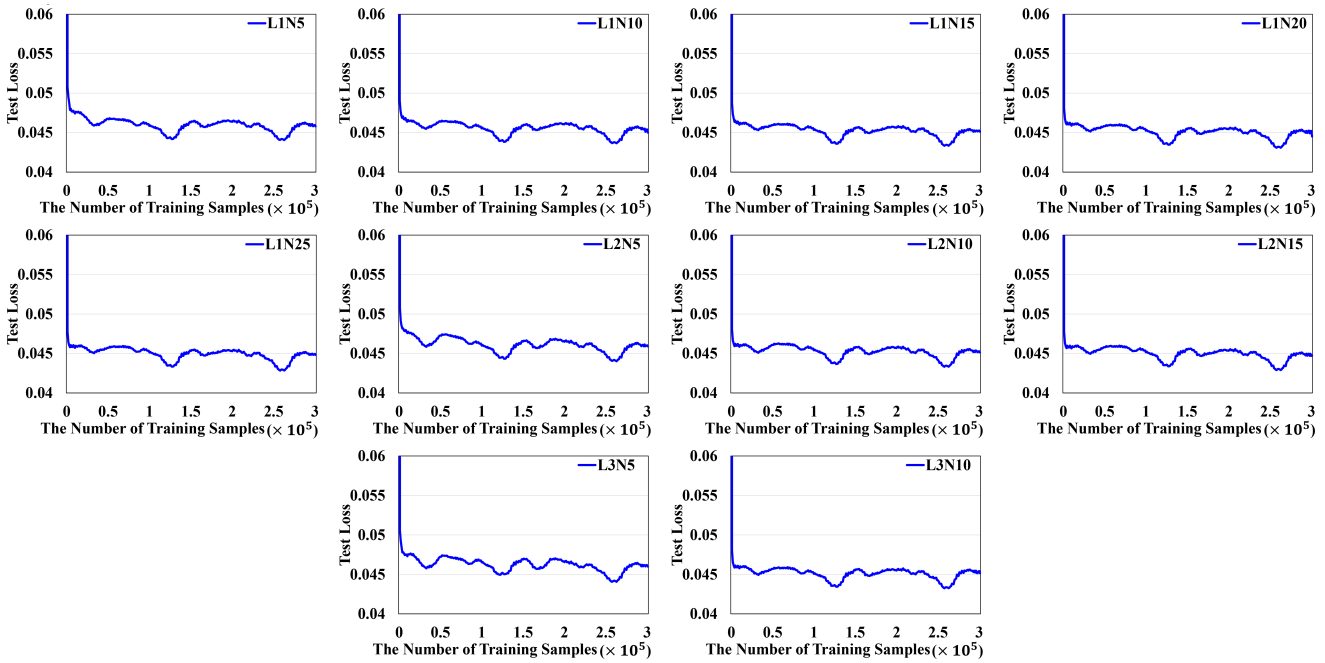

FIGURE 7. MSE losses for the number of training samples.

TABLE 5. Measuring device specifications.

Parameter	Specification
Model	SmartPower2
Output	USB Host (DC 4 ~ 5.3V, Maximum 1A) Terminal Block (DC 4 ~ 5.3V, Maximum 5A)
Measure	Voltage, Current, Watt
Protocol	HTTP, Telnet, Serial

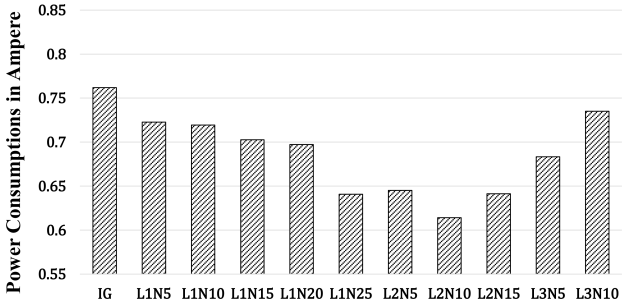

FIGURE 8. Average power consumption on PCMark for Android Work 2.0 run.

by L1N5. The interactive governor that operates by default in the Android kernel was selected as the comparison target and denoted by IG.

We evaluated the performance with PCMark for Android Work 2.0 benchmark[38], a smartphone benchmark to determine whether the proposed neuro-governor would degrade the smartphone's performance. The performance score of PCMark for Android Work 2.0 is a geometric mean of its sub-benchmark scores[39], which are web browsing, photo and video editing, data compression. Then computation time, memory access time, and rendering time is measured during each sub-benchmark task. With all other smartphone components except governor being controlled constantly, the higher scores mean better processor performance. The precision of the benchmark's performance measure is reported to better than 3%[39], meaning that the performance variation is a range of less than 3% with a device in a well-controlled environment.

Fig. 8 shows the average power consumption results during PCMark for Android Work 2.0 run where the proposed LSTM architecture consumes less power than IG. In addition, LSTM architecture saved more power when moving from L1N5 to L1N25, which was due to the increase in prediction accuracy. There was no significant difference in power consumption between L1N25 and L2N5 due to L2N5's lower prediction accuracy. The L2N10 architecture was the most efficient LSTM architecture with regard to prediction accuracy and computational overhead, reducing the average power consumption by approximately 19.4% compared to IG. Then the power consumption increased sharply from L3N10. The higher the number of layers, the faster the computational overhead increased, which led the processor to consume more power than could be recovered by prediction accuracy.

The execution time of each benchmark run was measured to find the execution performance of the processor and to find the system's actual energy consumption. The results in Fig. 9 show that the execution performance measures of neuro-governor with 10 network architectures are fall within a 3% precision range of IG. Additionally, the PCMark for Android Work 2.0 score was also recorded and shown in Fig. 10, supporting that our proposed neuro-governor performs almost equivalently with IG.

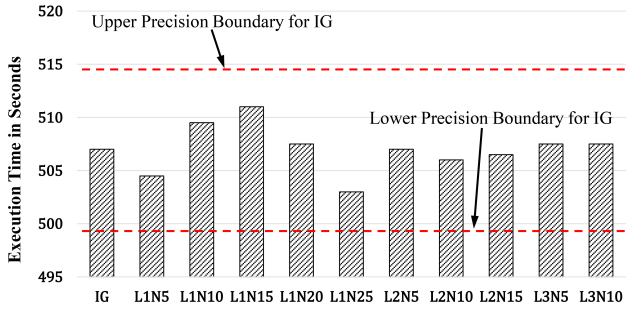

FIGURE 9. Average execution time on PCMark for Android Work 2.0 run.

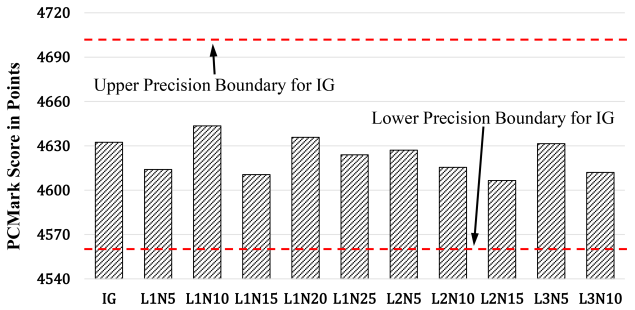

FIGURE 10. Average performance score on PCMark for Android Work 2.0 run.

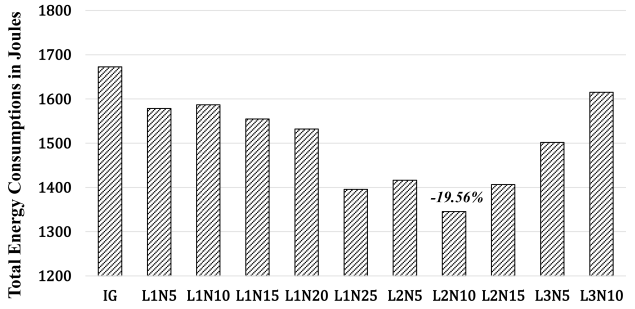

FIGURE 11. Calculated total energy consumption on PCMark for Android Work 2.0 run.

Finally, the actual energy consumption in Joules was calculated by multiplying the average power consumption, 4.33 volts of constant supply voltage, and the average execution time of each benchmark run. The energy consumption results are shown in Fig. 11. The trends of the power consumption results and the actual energy consumption results were similar since the execution time measure showed the almost equivalent performance between the governors. The L2N10 architecture was the most energy efficient LSTM architecture while minimizing the performance degradation. It consumed approximately 19% less energy compared to IG.

V. CONCLUSION

Existing software approaches for reducing the power consumption of mobile processors waste power because they allocate more resources than necessary to prevent deterioration of the processor performance. When a simple model is used to reduce the overhead of the computation used to estimate the processor usage, this can result in errors in relation to actual usage, resulting in wasted power. Therefore, this paper proposed a power management technique that can predict processor usage based on an LSTM network and can reduce the wasted power due to processor usage estimation error. In addition, we proposed the methodology on how to identify the most efficient LSTM architecture by comparing the computational overhead according to the number of LSTM neurons and layers of the new prediction model and the computational overhead of the existing technique. We constructed a power measurement environment for smartphones and measured power consumption to evaluate performance based on a benchmarking tool that reflects the characteristics of mobile applications. The comparison between the proposed method and the existing method showed that the proposed method reduced energy consumption by approximately 19% without degrading the processor performance.

We expect that smartphone usage time can be extended without degrading the usability using our proposed method. The limitation of our proposed method is that it uses only the processor usage to learn and make predictions with the LSTM network, and more detailed user context information that could change processor usage patterns was not considered. In the future, we plan to study a processor usage prediction model that considers the characteristics of each application and the overall situation of smartphone users.

REFERENCES

[1] J. M. Kim, Y. G. Kim, and S. W. Chung, "Stabilizing CPU frequency and voltage for temperature-aware DVFS in mobile devices," *IEEE Trans. Comput.*, vol. 64, no. 1, pp. 286–292, Jan. 2015.

[2] T. Gleixner, P. E. McKenney, and V. Guittot, "Cleaning up Linux's CPU hotplug for real time and energy management," *ACM SIGBED Rev.*, vol. 9, no. 4, pp. 49–52, Nov. 2012.

[3] R. Herrmann, Z. Piero, and T. S. Rosing, "Context aware power management of mobile systems for sensing applications," in *Proc. 11th ACM/IEEE Int. Conf. Inf. Process. Sensor Netw.*, Apr. 2012, pp. 16–20.

[4] W. Hu and G. Cao, "Energy-aware CPU frequency scaling for mobile video streaming," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Atlanta, GA, USA, Jun. 2017, pp. 2314–2321.

[5] K. Kwon, S. Chae, and K.-G. Woo, "An application-level energy-efficient scheduling for dynamic voltage and frequency scaling," in *Proc. IEEE Int. Conf. Consum. Electron. (ICCE)*, Las Vegas, NV, USA, Jan. 2013, pp. 3–6.

[6] Y. G. Kim, M. Kim, and S. W. Chung, "Enhancing energy efficiency of multimedia applications in heterogeneous mobile multi-core processors," *IEEE Trans. Comput.*, vol. 66, no. 11, pp. 1878–1889, Nov. 2017.

[7] P. C. Hsiu, P. H. Tseng, W. M. Chen, C. C. Pan, and T. W. Kuo, "User-centric scheduling and governing on mobile devices with big. LITTLE processors," *ACM Trans. Embedded Comput. Syst.*, vol. 15, no. 1, p. 17, Feb. 2013.

[8] U. Gupta, C. A. Patil, G. Bhat, P. Mishra, and U. Y. Ogras, "DyPO: Dynamic pareto-optimal configuration selection for heterogeneous MpSoCs," *ACM Trans. Embedded Comput. Syst.*, vol. 16, no. 5s, Oct. 2017, Art. no. 123.

[9] A. Aalsaud, R. Shafik, A. Rafiev, F. Xia, S. Yang, and A. Yakovlev, "Power–aware performance adaptation of concurrent applications in heterogeneous many-core systems," in *Proc. Int. Symp. Low Power Electron. Design*, Aug. 2016, pp. 368–373.

[10] U. Gupta, S. K. Mandal, M. Mao, C. Chakrabarti, and U. Y. Ogras, "A deep Q-learning approach for dynamic management of heterogeneous processors," *IEEE Comput. Archit. Lett.*, vol. 18, no. 1, pp. 14–17, Jan./Jun. 2019.

[11] S. Kim, H. Kim, J. Hwang, J. Lee, and E. Seo, "An event-driven power management scheme for mobile consumer electronics," *IEEE Trans. Consum. Electron.*, vol. 59, no. 1, pp. 259–266, Feb. 2013.

[12] Q. Yang, Z. He, F. Ge, and Y. Zhang, "Sequence-to-sequence prediction of personal computer software by recurrent neural network," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Anchorage, AK, USA, May 2017, pp. 934–940.
[13] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.
[14] E. Le Sueur and G. Heiser, "Dynamic voltage and frequency scaling: The laws of diminishing returns," in *Proc. Int. Conf. Power Aware Comput. Syst.*, Oct. 2010, pp. 1–8.
[15] V. Pallipadi and A. Starikovskiy, "The ondemand governor," in *Proc. Linux Symp.*, vol. 2, Jul. 2006, pp. 215–230.
[16] J. G. Park, N. Dutt, and S. S. Lim, "ML-Gov: A machine learning enhanced integrated CPU-GPU DVFS governor for mobile gaming," in *Proc. 15th ACM/IEEE Int. Symp. Embedded Syst. Real-Time Multimedia*, Oct. 2017, pp. 12–21.
[17] B. Burgess, "Samsung exynos M1 processor," in *Proc. IEEE Hot Chips 28 Symp. (HCS)*, Cupertino, CA, USA, Aug. 2016, pp. 1–18.
[18] A. Fog. (2018). The Microarchitecture of Intel. AMD and VIA CPUs. An Optimization Guide for Assembly Programmers and Compiler Makers. Tech. Univ. Denmark. Accessed: Dec. 12, 2018. [Online]. Available: https://www.agner.org/optimize/microarchitecture.pdf
[19] A. Smith, "Branch prediction with neural networks: Hidden layers and recurrent connections," Dept. Comput. Sci., Univ. California, San Diego La Jolla, CA, USA, Tech. Rep. 92307, 2004.
[20] Y. Zhang, R. Xiong, H. He, and M. G. Pecht, "Long short-term memory recurrent neural network for remaining useful life prediction of lithium-ion batteries," *IEEE Trans. Veh. Technol.*, vol. 67, no. 7, pp. 5695–5705, Jul. 2018.
[21] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *Int. J. Uncertainty, Fuzziness Knowl.-Based*, vol. 6, no. 2, pp. 102–116, 1998.
[22] P. J. Werbos, "Generalization of backpropagation with application to a recurrent gas market model," *Neural Netw.*, vol. 1, no. 4, pp. 339–356, Oct. 1988.
[23] A. M. Ahmad, S. Ismail, and D. F. Samaon, "Recurrent neural network with backpropagation through time for speech recognition," in *Proc. IEEE Int. Symp. Commun. Inf. Technol., (ISCIT)*, Sapporo, Japan, Oct. 2004, pp. 98–102.
[24] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, Oct. 1986.
[25] H. Palangi, L. Deng, Y. Shen, J. Gao, X. He, J. Chen, X. Song, and R. Ward, "Deep sentence embedding using long short-term memory networks: Analysis and application to information retrieval," *IEEE/ACM Trans. Audio, Speech, Language Process.*, vol. 24, no. 4, pp. 694–707, Apr. 2016.
[26] R. Akita, A. Yoshihara, T. Matsubara, and K. Uehara, "Deep learning for stock prediction using numerical and textual information," in *Proc. IEEE/ACIS 15th Int. Conf. Comput. Inf. Sci.*, Okayama, Japan, Jun. 2016, pp. 1–6.
[27] H. D. Huynh, L. M. Dang, and D. Duong, "A new model for stock price movements prediction using deep neural network," in *Proc. 8th Int. Symp. Inf. Commun. Technol.*, Dec. 2017, pp. 57–62.
[28] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese, "Social LSTM: Human trajectory prediction in crowded spaces," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Las Vegas, NV, USA, Jun. 2016, pp. 961–971.
[29] R. Villegas, J. Yang, S. Hong, X. Lin, and H. Lee, "Decomposing motion and content for natural video sequence prediction," in *Proc. Int. Conf. Learn. Represent.*, 2017, pp. 1–22.
[30] Z. Duan, Y. Yang, K. Zhang, Y. Ni, and S. Bajgain, "Improved deep hybrid networks for urban traffic flow prediction using trajectory data," *IEEE Access*, vol. 6, pp. 31820–31827, 2018.
[31] ARM. (2015). *ARMv8-A Processor Properties*. Accessed: Jan. 28, 2019. [Online]. Available: http://infocenter.arm.com/help/topic/com.arm.doc.den0024a/DEN0024A_v8_architecture_PG.pdf
[32] ARM. (2007). *The Dhrystone Benchmark*. Accessed: Jan. 28, 2019. [Online]. Available: http://infocenter.arm.com/help/topic/com.arm.doc.dai0093a/DAI0093A_benchmarking_appsnote.pdf
[33] J. Lee, J. Ko, and Y.-J. Choi, "Dhrystone million instructions per second–based task offloading from smartwatch to smartphone," *Int. J. Distrib. Sensor Netw.*, vol. 13, no. 11, Nov. 2017, Art. no. 1550147717740073.
[34] S. Lim and D. Lee, "Stable improved softmax using constant normalisation," *Electron. Lett.*, vol. 53, no. 23, pp. 1504–1506, Nov. 2017.
[35] K. Janocha and W. Czarnecki, "On loss functions for deep neural networks in classification," in *Proc. Conf. Theor. Found. Mach. Learn.*, 2017, pp. 1–10.
[36] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learn. Represent.*, 2015, pp. 1–15.
[37] ARM. (2014). Performance Monitor Unit. ARM Cortex-A53 MPCore Processor Techinical Reference Manual, ARM Inc. Accessed: Dec. 12, 2018. [Online]. Available: http://infocenter.arm.com/help/topic/com.arm.doc.ddi0500d/DDI0500D_cortex_a53_r0p2_trm.pdf
[38] Futuremark. (2017). PCMark for Android Benchmark (2.0.3716). Futuremark Oy. Accessed: Oct. 28, 2018. [Online]. Available: https://play.google.com/store/apps/details?id=com.futuremark.pcmark.android.benchmark&hl=en_US
[39] Futuremark. (2018). PCMark for Android. PCMark for Android Tech. Guide, Futuremark Corp. Accessed: Dec. 12, 2018. [Online]. Available: http://akamai-dl.futuremark.com.akamaized.net/pcmark-android-technical-guide.pdf