Computer Systems
Science and Engineering

Tech Science Press

DOI: 10.32604/csse.2023.038139
Article

An Efficient Memory Management for Mobile Operating Systems Based on Prediction of Relaunch Distance

Jaehwan Lee[1] **and Sangoh Park**[2,*]

[1]Department of Computer Science and Engineering, Kongju National University, Cheonan, 31080, Korea
[2]School of Computer Science and Engineering, Chung-Ang University, Seoul, 06974, Korea
*Corresponding Author: Sangoh Park. Email: sopark@cau.ac.kr
Received: 28 November 2022; Accepted: 24 February 2023; Published: 26 May 2023

Abstract: Recently, various mobile apps have included more features to improve user convenience. Mobile operating systems load as many apps into memory for faster app launching and execution. The least recently used (LRU)-based termination of cached apps is a widely adopted approach when free space of the main memory is running low. However, the LRU-based cached app termination does not distinguish between frequently or infrequently used apps. The app launch performance degrades if LRU terminates frequently used apps. Recent studies have suggested the potential of using users' app usage patterns to predict the next app launch and address the limitations of the current least recently used (LRU) approach. However, existing methods only focus on predicting the probability of the next launch and do not consider how soon the app will launch again. In this paper, we present a new approach for predicting future app launches by utilizing the relaunch distance. We define the relaunch distance as the interval between two consecutive launches of an app and propose a memory management based on app relaunch prediction (M2ARP). M2ARP utilizes past app usage patterns to predict the relaunch distance. It uses the predicted relaunch distance to determine which apps are least likely to be launched soon and terminate them to improve the efficiency of the main memory.

Keywords: Mobile operating systems; memory management; background app caching; relaunch distance; neural networks

1 Introduction

With the widespread use of smartphones, various mobile apps are being developed. The number of smartphone users has exceeded 6 billion as of 2022, continuously increasing. Following this trend, the number of mobile apps registered in the Android App Store is steadily increasing, and currently, there are about 3 million [1]. Smartphone users usually install and use dozens to hundreds of apps, and more features have been loaded into smartphone apps to improve user convenience. Therefore, requirements such as main memory or processing power are increasing. The increasing main memory demand can be addressed by hardware or software approaches. The hardware approach is to increase the main

memory of the smartphone, whereas the software approach is to use the main memory through a software management policy efficiently. In addition to the memory management policy, the software approach includes the enlargement of the main memory capacity using page swap techniques.

Android or iOS adopts app life cycle management [2] to release some of the app's memory according to the running state of the app in order to keep as many apps as possible in the main memory with its limited capacity. These apps cached in the main memory can be quickly brought to the front of the screen when switching or relaunching the apps. Mobile operating systems terminate the cached apps they determined not frequently used when the available main memory is insufficient to prevent the user from perceiving system performance degradation. Furthermore, to efficiently utilize the main memory of the smartphone, a swapping technique [3] that utilizes secondary storage as the main memory space was considered. However, in smartphones, the NAND flash-based secondary storage is hard to be used as a swap space due to wear-out problems. Therefore, zram [4], which compresses a fixed portion of main memory and uses it as swap space, or zswap [5], which uses it as a swap cache, is mainly deployed to smartphone operating systems.

The existing smartphone operating systems widely adopt the LRU (Least Recently Used) [6]-based app termination and main memory reclaim technique. They terminate the least recently used apps to reclaim their memory in a low-memory situation. The LRU-based approach considers that the most recently launched app has a higher probability of being relaunched compared to other apps. However, the LRU-based approach should be aware of frequently and infrequently launched apps. When infrequently used apps launch at some point, the LRU-based approach can firstly terminate frequently launched apps even if they are likely to be relaunched soon as the frequent apps are in the least recently used position than the infrequent ones. They need to be loaded back into memory upon relaunch incurring overall degradation of user-perceived performance. Recently, studies have been conducted to analyze the user's app usage pattern and predict the app to be launched next [7–15]. These studies suggested the possibility of addressing the limitations of the LRU-based approach by analyzing the user's app usage pattern. Even though existing methods focus primarily on predicting the probability of the next launch, they do not consider the timing of when the app will launch again. Therefore, it is important to note that simply adopting the state-of-the-art approach of the next app prediction for selecting the termination app is not sufficient.

In this paper, we address this limitation by proposing memory management based on app relaunch prediction (M2ARP), which uses relaunch distance to manage main memory by terminating apps according to the predicted relaunch distance. We define the relaunch distance of an app, which is a metric for how much later the app will launch. Further, we propose a machine learning technique based on the user's usage patterns to predict the relaunch distance. This method utilizes past app usage patterns to predict the relaunch distance, which it uses to determine which apps are least likely to be launched soon. It terminates the least likely one to improve the efficiency of the main memory. In addition, we consider a way to adopt both app usage prediction and the existing LRU-based approach for the actual usage case scenario in which apps with rich historical usage data and scarce data are combined. To this end, we propose to adopt the existing LRU-based app termination as a fallback method using relaunch distance when we meet such apps that are difficult to predict usage patterns. The main contributions of this paper are as follows:

- We define app relaunch distance, a measure of when an app launch again.
- We designed a mechanism to compare the relaunch distance between apps in a low-memory situation to terminate the least likely to be launched in the future.

- We exploited app usage patterns and relationships for time-series prediction of app relaunch distance with long short-term memory.
- We designed an LRU fallback mechanism for difficult-to-predict apps.

The rest of the paper is structured as follows. The existing studies in predicting the user's app usage patterns and managing the memory of mobile devices by predicting user usage patterns are introduced in Section 2. The M2ARP, a memory management system that can adopt both app usage prediction and existing LRU, is proposed in Section 3. In Section 4, we compare and analyze the existing LRU-based method and the proposed M2ARP by performing a benchmark with a real-world usage history on a smartphone device. Finally, we make a conclusion and discuss future research directions in Section 5.

2 Related Work

A PC-based operating system allocates system resources fairly to all running apps and services, whereas a mobile operating system prioritizes allocating system resources to foreground apps and related services. When the user launches an app and launches another app thereafter, Android caches the app that is not in the foreground [16]. This is to provide maximum responsiveness to the user with limited system resources. Since Android keeps apps cached in memory, the user relaunch of the cached app improves the launch speed and user responsiveness. The cached apps are maintained as an LRU list [17], and the apps in the LRU position are terminated in a low-memory situation. However, LRU-based memory management is less efficient as the number of apps used increases because frequently used apps are more likely to be terminated by various kinds of apps. If apps likely to be launched near future can be determined accurately, memory management efficiency will increase accordingly.

Mobile devices such as smartphones and tablets are a more convenient platform for analyzing usage patterns due to the characteristics that users alone carry and use most of the time. Therefore, studies such as *next app prediction* [9–12], next app prediction and prelaunch [7,8], next app prediction and kill [14, 15] that predict the next app to be launched next by analyzing the user's app usage history have been conducted.

Next app prediction is a study that predicts the next app to be launched. In a study that predicted users' app usage based on k-nearest neighbor (kNN), information about apps switching from one to another and each app usage time were modeled as graphs [9]. In a study to model and predict app usage, the probability of the next app being launched was estimated by combining a string prediction and a Markov-based model [10]. However, the Markov-based probabilistic model assumes that the app to be launched next is related only to the app just used, which is a limit to analyzing various usage patterns. In the actual app usage pattern, the app that the user launches is related to several previously launched apps [11]. In order to analyze the correlation between a series of app launches, long short-term memory (LSTM)-based app launch prediction was studied. However, hyperparameters of the LSTM model were not considered, but only fixed time windows of usage data were considered. Accordingly, the proposed LSTM model is not able to learn how long the model should analyze the usage history of inputs. WhatsNextApp [18] proposed a bidirectional LSTM model that analyzes variable time windows at the minute, hour, and day levels in order to learn long-term app launch records. AppUsage2Vec [12] analyzes and predicts app usage history by applying Doc2Vec [19], which is originally proposed for natural language processing. AppUsage2Vec viewed each app as a single

word in a document and the entire app usage history of a user as a single document. However, the AppUsage2Vec model only takes the fixed-size recent usage history and shows the highest prediction performance for four historical usage records given, which is a limitation for long-term usage data analysis. CoSEM [20] predicts the probability of the app that will be launched next using app launch records and semantic information. Like Appusage2Vec, it performs embedding of app launch records, but it also performs embedding of semantic information, such as search records, location, and time, to improve prediction performance.

Next app prediction and prelaunch is a study that predicts the app to be launched next and load into the main memory before the user launches it to improve the perceived launch performance of the user. FALCON [7] tried to predict apps that are likely to be launched soon by analyzing app usage and smartphone sensor information. It was shown that the app launch time could be improved by prelaunching with the app usage prediction, but the correlation between launched apps was not considered. In the prelaunch study based on the LSTM model, the apps with the highest estimated launch probability are loaded into the main memory [8]. The study showed high prediction accuracy of the next app to be launched; however, there is a limitation in long-term usage analysis owing to the model's fixed size historical usage data input. Furthermore, if the predicted app is launched during the foreground app execution, the currently running app may be degraded on account of the use of the system resource consumption for launching the app.

Next App Prediction and Kill is a study that predicts the apps' probability of launching next and terminating the apps with the lowest launch probability in low-memory situations. In a study to apply reinforcement learning to improve Android's app termination behavior in low-memory situations, an app termination method considering the launch probability and launch time of an app was proposed [14]. The agent of reinforcement learning terminates apps that have not been used for a long time, with its memory footprint being large and relaunch probability being low. However, the simple statistical probability analysis based on unused duration limits the accuracy of predicting the next app. Unlike previous neural network-based app launch prediction studies, AMMS [15] is a memory management system that adopts the LSTM model for analyzing users' long and short-term app usage relationships. AMMS predicts the least likely to be launched apps next, then terminates them in low-memory situations.

Table 1 is the result of a comparison and analysis of existing studies. Long-term analysis refers to whether or not the study was conducted on the relationship between two or more app launch records when predicting a single app launch. Next app prediction is whether or not the probability of what app will be launched next is predicted when an app is launched. Long-term prediction indicates whether or not the likelihood of an app to be launched in the near future, beyond the next app to be launched, is predicted. As shown in Table 1, there are no existing studies that consider both long-term analysis and long-term prediction. However, as explained in the LRU case, in order to effectively utilize the main memory space in a mobile operating system, it is necessary to accurately predict which apps will be launched in the near future and which will not. Most existing studies are only able to predict the probability that the next app will be launched. They are unable to predict the likelihood that an app will be launched in the near future. If an app has a high probability of launching in the near future, keeping it in the main memory can lower the overall launch time of apps, even if it is unlikely to be launched right next time.

Table 1: Comparison of existing app prediction approaches

Category	Citation	Long-term analysis	Next app prediction	Long-term prediction
NAP	[9]	No	Yes	No
	[10]	No	Yes	No
	[12]	Yes	Yes	No
	[18]	Yes	Yes	No
	[20]	Yes	Yes	No
NAPP	[7]	No	Yes	Yes
	[8]	Yes	Yes	No
NAPK	[14]	No	Yes	No
	[15]	Yes	Yes	No

3 M2ARP: Memory Management Based on App Relaunch Prediction

3.1 Proposed Method

In order to address the problems stated in Section 2, we propose memory management based on app relaunch prediction (M2ARP), a method that addresses the problem of selecting apps to terminate in low-memory situations by designing a relaunch distance metric for an app, which is a measure of how far in the future an app will be launched again. We further propose to predict the relaunch distance by analyzing users' previous app usage records in order to improve the overall app launch speed.

The overall prediction and utilization of relaunch distances are shown in Fig. 1. In the example, the apps $a2$, $a1$, $a3$, and $a1$ are launched at time $i = 1$, $i = 2$, $i = 3$, $i = 4$, respectively. The relaunch distance of an app is defined as the number of different types of apps launched between the current launch and the relaunch of an app in the future. In the figure, the relaunch distance of app $a1$ at time $i = 4$ is 3, and the relaunch distance of app $a2$ at time $i = 1$ is 2. As different apps launched at time $i = 2$ and $i = 3$ after $a2$ at time $i = 1$, the relaunch distance of $a2$ decreases to 0 at the time $i = 3$. Similarly, the relaunch distance of app $a3$ decreases to 1 as app $a1$ is launched at $i = 4$. Therefore, app a3 is the first to be killed by M2ARP in the low-memory situation. M2ARP employs time series prediction to accurately estimate the relaunch distance of an app. The prediction model predicts the relaunch distance of an app that has launched each time. By combining the use of relaunch distance and prediction, M2ARP can terminate the least likely to-be-launched app in the near future.

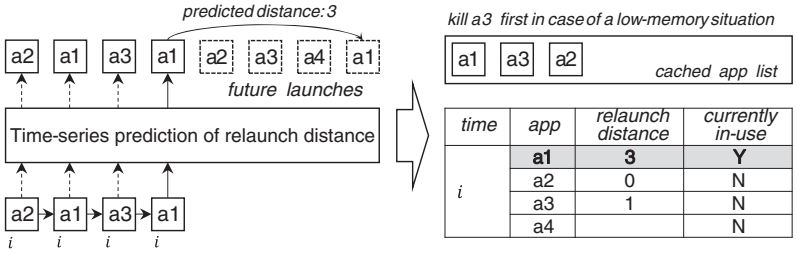

Figure 1: Proposed method to predict and utilize relaunch distance

3.2 Memory Management Based on App Relaunch Distance

Existing app prediction studies estimate the probability of the app being launched next; thus, it is not suitable as a metric for selecting an app to be terminated in a low-memory situation. This is because the app that is least likely to be launched next may be an app the user will launch again soon. Hence, it may be more beneficial to keep these apps cached in the main memory to enhance the overall launch performance of the system. We address this problem by predicting the relaunch distance of apps by analyzing users' previous app usage records. M2ARP predicts the relaunch distance of an app and terminates the apps with the longest relaunch distance in a low-memory situation. In addition to prediction-based memory management, we propose a method that can provide the same behavior of LRU for apps with few training samples or low prediction accuracy. The symbols for the prediction of app relaunch distance are defined as Table 2.

Table 2: Definition of parameters for relaunch distance

Symbol	Definition
A	Set of apps that the user can launch on the system
B	Set of apps cached in the background by the system
a	Apps in set A
a_i	i-th app of set A
L	History of all app launches by user
l_t	App corresponding to the t-th record in the set L
$I(l_t)$	Set of relaunch interval in l_t
$d(l_t)$	Relaunch interval of l_t
$d'(l_t)$	The predicted value of $d(l_t)$
A'	Set of $l_t \in A$ from which $d'(l_t)$ can be obtained

The relaunch distance of an app indicates how many different types of apps are launched between the current launch and the relaunch in the future. The app relaunch interval and relaunch distance can be defined based on the study [21] that modeled the page cache reuse behavior, given that the behavior of an app being cached in the background is similar to the behavior of pages in memory being cached. First, the relaunch interval set $I(l_t)$ is the set of all app launch records in the relaunch interval of the user's app launch record l_t and is defined as Eq. (1).

$$I(l_t) = \left\{(l_j, j) \middle| \, t < j < k \wedge l_t = l_k \wedge l_t \neq l_j\right\} \tag{1}$$

$I(l_t)$ is the set of all app launch records between the launch and the relaunch of the app l_t. Therefore, the relaunch interval includes duplicate launch records of the same app. However, it is difficult to reflect the behavior of running out of free space with only $I(l_t)$ when caching more apps into the main memory. This is because the size of the set $|I(l_t)|$ can become large even if a small number of apps are run alternately. In order to model the behavior correctly, the types of all launched apps belonging to $I(l_t)$ are defined as the relaunch distance $d(l_t)$. $d(l_t)$ is defined as Eq. (2).

$$d(l_t) = \begin{cases} \left|\{l_j \middle| (l_j, j) \in I(l_t)\}\right|, & if \; \exists k(t < k \wedge l_t = l_k) \\ -1, & otherwise \end{cases} \tag{2}$$

For example, assuming the set $L = \{a3, a1, a4, a1, a2, a4, a3, a1, a2\}$. The set of relaunch interval $I(l_1)$ of the app launch record $l_1 = a3$ is {(a1,2), (a4,3), (a1,4), (a2,5), (a4,6)}. The $|I(l_1)|$ equals 5. On the other hand, the relaunch distance $d(l_1)$ of $l_1 = a3$ is $|\{a1, a4, a2\}|$, which equals to 3.

By accurately estimating the app's relaunch distance, the least likely to-be-launched app near future can be terminated in a low-memory situation. To this end, each launched app's relaunch distance is reset to the predicted distance. The predicted relaunch distance means that the app is likely to be relaunched after that distance. The relaunch distance of the app is subsequently decreased by the launch of other apps. Therefore, terminating the app with the largest relaunch distance is equivalent to terminating an app that is not likely to be launched in the future. The overall app launch speed can be improved by keeping the apps that are likely to be launched in the near future. However, it is difficult to make accurate predictions when the app is launched for the first time, or the app has insufficient launch records for training. This can lead to improperly terminating the app, which negatively affects the app launch speed.

Algorithm 1: Executed when an app is launched

```
1:  procedure onAppLaunch(l_t)
2:    if l_t ∈ A' then
3:      Compute d (l_t) with launch context of l_t
4:    else
5:      Set d (l_t) as 0
6:    end if
7:    Relaunch distance of l_t ← d (l_t)
8:    for each a ∈ {a|a ∈ B, a ≠ l_t} do
9:      Decrement the relaunch distance of a by 1
10:   end for
11: end procedure
```

M2ARP manages the relaunch distance for apps that are difficult to predict so that the management behavior is the same as LRU. The initial relaunch distance of those apps is set to 0. Subsequently, the relaunch distance is decreased by one each time a different type of app is launched. This is equivalent to updating the relaunch distance according to Eq. (2). If the predicted relaunch distance of all apps in the system is set to 0, then it works the same as LRU.

Algorithm 1 describes how M2ARP uses the relaunch distance to terminate apps with a low likelihood of being launched in the near future and perform main memory management. Additionally, Algorithm 1 describes a method that can provide prediction performance similar to the LRU method in cases where the currently launched app has insufficient launch history to achieve a high prediction accuracy. The algorithm is executed on each app launch. The set A' of Algorithm 1 is defined as Eq. (3).

$$A' = \{a | a = l_j \wedge d\left(l_j\right) \geq 0\} \tag{3}$$

As in line 2, it is checked whether l_t is included in A'. If it is included, $d'(l_t)$ is calculated through the prediction method, as in line 3. Otherwise, $d'(l_t)$ is set to 0 by the LRU fallback mechanism, as in line 5. As in lines 8–10, each relaunch distance is decreased by one for apps cached in the background. In such a manner, the apps with the largest absolute relaunch distance are terminated in the case of a low-memory situation. M2ARP can efficiently perform memory management by considering both predictable and non-predictable apps altogether.

3.3 Prediction of App Relaunch Distance

M2ARP adopts the long short-term memory (LSTM) [22] model to analyze the user's app usage pattern and predict the app's relaunch distance. Our proposed M2ARP's LSTM model differs from the LSTM model proposed in the next app prediction studies in that our proposed LSTM is a nonlinear regression analysis that predicts the relaunch distance of the currently launched app, while previous studies are multiclass classification analyses that predict the probability of the next app to be launched. The parameters for the LSTM model are defined in Table 3. Some of the parameters are from the previous design [15]. LSTM was originally proposed to address the vanishing gradient problem of recurrent neural networks. Through its memory or forget gate structures, the LSTM can learn the long short-term patterns of data better than the existing neural networks.

Table 3: Definition of parameters related to long short-term memory

Symbol	Definition
t	Time at which an input vector is fed to an LSTM
x_t	Input vector to an LSTM at time t
$v_{app}\,(l_t)$	Vector to represent an app
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 generate data to be stored in c_t
o_t	Output gate function at time t to generate data to be stored in c_t
$w_{x_f}, w_{x_g}, w_{x_i}, w_{x_o}$	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 in functions f_t, g_t, i_t, o_t, respectively
w_{h_l}	Vector weighted to h_t in function $lstm(x_t)$
b_f, b_g, b_i, b_o	Bias constant for functions f_t, g_t, i_t, o_t, respectively
b_h	Bias constant for function $lstm(x_t)$

The goal of the LSTM model adopted in M2ARP is to minimize the difference between $d(l_t)$ and $d'(l_t)$ for L, the actual relaunch distance, and the predicted relaunch distance by the model, respectively. $d'\,(l_t)$ is determined as the output of the LSTM model. However, if the relaunch distance cannot be determined by the model, e.g., no previous usage data for an app, it should be prevented from being reflected in the training phase of the model. Therefore, we propose a loss function called selective mean squared error (SMSE) as in Eq. (4).

$$SMSE = \frac{1}{|L|}\sum_{t=1}^{|L|}\left(d'\,(l_t) - d\,(l_t)\right)^2, where\, d'\,(l_t) = \begin{cases} lstm\,(x_t)\,, & if\ l_t \in A' \\ d\,(l_t)\,, & otherwise \end{cases} \tag{4}$$

The input vector x_t is composed of $v_{app}(l_t)$, which is the one-hot encoded vector for each launched app l_t. The vector for the launched app is defined as Eq. (5).

$$v_{app}(l_t) = \left\{v_{t_0}, v_{t_1}, \ldots, v_{t_j}, \ldots, v_{t_{m-1}}\right\}, where\ v_{t_j} = \begin{cases} 1, & if\ l_t \in a_j \\ 0, & otherwise \end{cases} \tag{5}$$

The output layer of the LSTM model consists of a dense layer for generating $d'(l_t)$ from the outputs of LSTM layers. The dense layer generates a weighted sum of the output vectors h_t from all LSTM neurons in the previous layer, which is represented as Eq. (6). The output value h_t of the i-th LSTM neuron is h_{t_i}.

$$lstm(x_t) = \sum_{i=0}^{n-1} w_{h_i} h_{t_i} + b_h \tag{6}$$

The input layer and hidden layer of the LSTM model have a fully connected structure of n LSTM neurons. As in (7)–(12), each LSTM neuron consists of forget gate f_t, input gate g_t and i_t, memory cell c_t, output gate o_t, and each neuron's output h_t.

$$f_t = \sigma\left(x_t w_{x_f} + h_{t-1} w_{h_f} + b_f\right) \tag{7}$$

$$g_t = tanh\left(x_t w_{x_g} + h_{t-1} w_{h_g} + b_g\right) \tag{8}$$

$$i_t = \sigma\left(x_t w_{x_i} + h_{t-1} w_{h_i} + b_i\right) \tag{9}$$

$$c_t = f_t \circ c_{t-1} + i_t \circ g_t \tag{10}$$

$$o_t = \sigma\left(x_t w_{x_o} + h_{t-1} w_{h_o} + b_o\right) \tag{11}$$

$$h_t = o_t \circ tanh(c_t) \tag{12}$$

When training the LSTM model, the BPTT algorithm [23] is used to create a feed-forward network by unfolding the LSTM model as much as the length of the data to be learned at one time, and then the algorithm updates the network parameters using the backpropagation algorithm. The LSTM model is trained to minimize Eq. (4). Therefore, the model can predict the relaunch distance, enabling efficient app termination in a low-memory situation.

4 Performance Evaluation

4.1 Dataset Preparation

In order to evaluate the performance of the M2ARP proposed in this paper, we used the Tsinghua app usage dataset [24] from real-world app usage tracking records. The dataset is collected for about seven days of app usage records from 1000 users. The user ID, app launch timestamp, and app ID were extracted for performance evaluation. The relaunch distance is calculated through Eq. (2) in preparation for training the LSTM model of M2ARP. As a data preprocessing, algorithm 2 describes the process of calculating the relaunch distance from the dataset. The relaunch distance can be calculated from the app launch history dataset. This is done by searching through the history after each app launch record. If there are no further relaunches after a given launch record, the record is marked separately to prevent the prediction model from learning.

The list of relaunch distances to add to the dataset is initialized, as shown in line 2. The relaunch distance for each user in the dataset is calculated as shown in line 3. The app launch records for each

user are extracted as in line 4, and the relaunch distance for each app launch record is calculated thereafter, as in lines 5–12. The list of future launched apps is generated as in line 6. If the app is relaunched in the future, as in lines 7–8, the distance is added to the relaunch distance list; otherwise, −1 is added to the list, as in line 10. Finally, the list of relaunch distances in the dataset is updated, as shown in line 14. The dataset is used for smartphone-based benchmarks.

Algorithm 2: Relaunch distance generation

```
1:  procedure compute Relaunch Distance (dataset)
2:      relaunch_dist_list ← ∅
3:      for each user in users from dataset do
4:          L ←app usage sequence of user from dataset
5:          for t ← 0 to |L| − 1 do
6:              future_usage ← {l_k|k > t}
7:              if found l_t in future_usagethen
8:                  append relaunch distance to relaunch_dist_list
9:              else
10:                 append −1 to relaunch_dist_list
11:             end if
12:         end for
13:     end for
14:     insert relaunch_dist_list to dataset
15:     return dataset
16: end procedure
```

4.2 Model Preparation

The M2ARP's model is implemented to predict the relaunch distance of an app that just launched. The LSTM model operates by first generating a one-hot encoded vector of an app when it launches. This vector is fed to the model's first layer, which serves as the input for the model. The input vector then goes through one or more LSTM layers. These LSTM layers analyze the sequence of app launches and extract relevant information. The last LSTM layer passes the output to a fully connected layer. It generates abstract information from the previous layers by reducing the number of neurons by half of the previous layer. We place the last layer composed of a single neuron which takes the abstracted information from the previous layer. Additionally, we place a dropout layer with a ratio of 0.2 between each layer to prevent overfitting. Finally, we use the SMSE loss function in Eq. (4) to train the model and the Adam optimizer as an optimizer.

To decide which parameters for the model to use, we measure the prediction performance by varying the number of neurons in the LSTM neural network. We implement the LSTM model using Tensorflow [25] and Keras [26]. We trained the model with the number of neurons in the first layer varying from 32, 64, 96, and 128 and the number of LSTM layers varying from 2 to 3. We set the maximum training epoch to 600 and the batch size to 1. Every LSTM layer for the model is set to stateful, with input timestep to 1. After evaluating the performance of different configurations, we set to use 2 layers and 128 neurons as the first layer.

The performance impact of the LSTM model is also considered when selecting the model parameters in addition to the model losses. We measured the execution time of the LSTM model on the Android OS-based mobile device shown in Table 4 [27]. The model was converted to a Tensorflow Lite

[28] model and deployed into the Android open-source project (AOSP) [29] framework. The number of LSTM neurons varied in the range 32, 64, 128, 256, and 512, and the time taken to perform the prediction was measured. The LSTM model's average prediction time is shown in Table 5.

Table 4: Target device specification

Symbol	Definition
Model	Google Pixel 3
Processor	Qualcomm snapdragon 845
Memory	4GB LPDDR4X SDRAM
OS	Android open-source project 9 Pie

Table 5: LSTM model execution time

Neurons	32	64	128	256	512
Time	1 ms	1.5 ms	2 ms	4 ms	6 ms

According to the execution results, it tends to increase as the number of neurons in the network increases. However, considering that the app launch time varies from hundreds of milliseconds to thousands of milliseconds, the relaunch prediction overhead of the model is negligible, with the time difference between the lowest and highest number of neurons being only 5 ms.

4.3 Smartphone-Based Benchmark

In order to compare the performance of the proposed M2ARP and the existing method in real devices. We used the LRU method as a baseline, and we used the device in Table 4 and the Tsinghua app dataset as the performance evaluation environment. Due to the excessive evaluation time on a real device, we chose a dataset from a randomly selected user. The user id 8, which we used for the benchmark, contains 2601 usage records among 93 different apps. Since the dataset does not provide any information on which application the user actually used, the application ID and its app category data were used to match the Google play store apps. The app usage count and the Google play store's app popularity by category were investigated to match the apps in order. We compared the performance of M2ARP with LRU and an optimal method. The optimal method uses the actual relaunch distance that was computed from the dataset. We evaluated the average launch time and the hit ratio for each method. The hit ratio is computed as the hit count divided by the total number of launches. The hit count increases when the launched app is in the cached app list.

The average launch time of the apps according to the real-world usage history is shown in Fig. 2. The existing LRU method and the proposed M2ARP method, and the optimal method was measured. The performance measurement results show that the LRU method has an average app launch time of 957.6 ms, and the M2ARP method shows an average launch time of 910.9 ms, resulting in a reduction of 4.9% in average app launch time. The optimal method achieves the best performance with an average launch time of 893.1 ms, resulting in a reduction of 6.7% compared to the LRU method.

As the next step, we measured the hit ratio of apps that were launched from the cached app list of the Android frameworks. The existing LRU method and the proposed M2ARP method, and the optimal method were measured. The hit ratio of each method is shown in Fig. 3. The result shows

that the hit ratio of the cached app launches is 59.5%, 60.6%, and 61.6% for the LRU, M2ARP, and Optimal methods, respectively. In other words, the M2ARP method shows an improvement on an average hit ratio of 1.9% and the Optimal method of 3.6%. The results in Figs. 2 and 3 support the hypothesis that the use of relaunch distance can contribute to the increased hit ratio of the cached app launches, thus resulting in faster app launches.

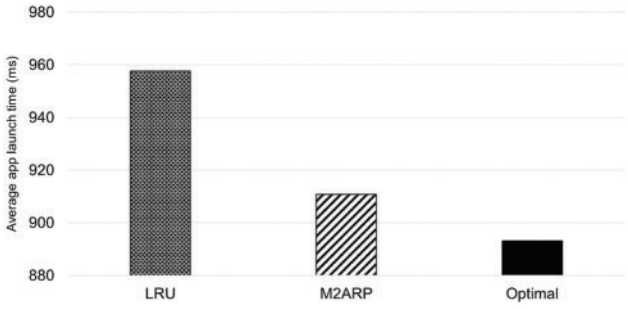

Figure 2: Comparison of average launch time

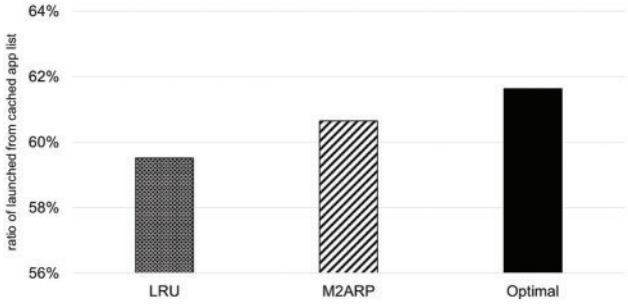

Figure 3: Comparison of hit ratio of the cached app list

4.4 Simulation-Based Benchmark

To further investigate the impact of the relaunch distance, we conducted a simulation-based benchmark. We designed a simulation that mimics the behavior of apps cached in the main memory under a smartphone-based environment. The purpose of this benchmark is to analyze how the relaunch distance contributes to the improvement in the hit ratio that apps launch from the cached app list. Unlike the smartphone-based benchmark as in Section 4.3, the simulation assumes that all apps have the same main memory footprint as it is difficult to extract the exact footprint from the runtime environment. We conducted the benchmark for the three methods which are used in the smartphone-based benchmark. We measured the hit ratio of apps launched from the cached list for each method by varying the number of apps m that can reside in the main memory. This allowed us to isolate the impact of relaunch distance on the launch hit ratio from the various system factors that can potentially affect the hit ratio of app launches.

As shown in Table 6, the simulation-based benchmark shows that the use of the relaunch distance (M2ARP and Optimal) exhibits superiority over the LRU method of all cases for m. The hit ratio improves as the available main memory space increases. The optimal method shows the highest hit ratio for all cases of m, with the hit ratio reaching up to 0.88 when the main memory can cache up to 15 apps. This indicates that we can improve the app launch hit ratio if we approximate the actual

relaunch distance. The hit ratio of the LRU and M2ARP methods also supports this observation. The M2ARP approximates the relaunch distance of an app with LSTM model by analyzing past app launch patterns. As we mentioned in Section 3.2, the M2ARP method with all the relaunch distances of an app set to 0 is equivalent to the LRU method. Therefore, we can safely say that the LRU method is the farthest approximation of the relaunch distance among the three we compared. Fig. 4 illustrates the comparison of app launch hit ratio in varying m. According to the figure, the performance difference between the three gets significant when the value of m gets smaller. These results show that the relaunch distance can be more effective in environments with more significant physical main memory constraints.

Table 6: App launch hit ratio@***m*** for all methods

m	LRU	M2ARP	Optimal
5	0.50	0.60	0.67
6	0.58	0.66	0.73
7	0.66	0.72	0.77
8	0.71	0.75	0.80
9	0.74	0.77	0.81
10	0.76	0.79	0.83
11	0.78	0.81	0.84
12	0.80	0.82	0.85
13	0.81	0.83	0.86
14	0.82	0.84	0.87
15	0.83	0.85	0.88

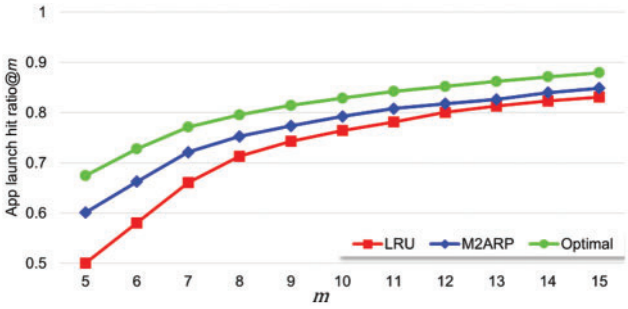

Figure 4: Comparison of app launch hit ratio@***m***

In conclusion, the simulation-based benchmark conducted demonstrates that the relaunch distance can have a significant impact on the improvement in hit ratio for app launches from the cached app list. By combining the results in Sections 4.3 and 4.4, the M2ARP contributed to the enhancement of the overall launch performance of apps in smartphones.

5 Conclusion

The increasing demand for main memory space on smartphones due to more resource-intensive apps calls for efficient software-based memory management techniques. The commonly used LRU-based app termination to free up memory space does not take into account the frequency of app usage and can negatively impact the user experience by terminating frequently used apps. There are studies that aim to analyze the smartphone app usage patterns of users in order to predict the next app to be launched. However, most existing studies only consider the probability of the next app being launched. They are unable to predict the likelihood of an app being launched in the near future or to analyze the correlation between app launch records.

In this paper, we proposed M2ARP to predict applications' relaunch distances and utilize them for app termination of memory management. Relaunch distance is a measure of the likelihood that an app will be launched in the near future, so if it is accurately predicted, then OS can keep apps with a high likelihood of being launched in main memory to improve overall app launch speed. M2ARP analyzes the user's app usage history and predicts the app's relaunch distance. In addition, unlike the existing studies that had to make predictions despite the insufficient data for specific apps, M2ARP combines both prediction and LRU behavior such that it includes an LRU fallback mechanism on apps with insufficient training samples or inaccurate launch prediction. As a result of performance evaluation on the smartphone-based benchmark, the proposed system showed that the overall app launch time was reduced compared to the existing LRU method.

M2ARP showed promising results that the user's perceived app launch performance can be improved by the use of relaunch distance. However, it was assumed that the user patterns in the past usage records and the future usage are the same. In real-world scenarios, user patterns may vary, and this could impact the model's prediction performance. Additionally, this study differs from previous studies in that it collects learning data from a single user, which limits the data that can be learned. There are several general models for the next app prediction works in the literature that can learn general patterns from various user data. In the future, we plan to improve the performance of predicting relaunch distance by studying a model that extracts universal patterns from the usage data of multiple users. This will allow for a more generalizable model that can better train and predict relaunch distances from a much larger scale user dataset.

Funding Statement: This work was supported in part by the National Research Foundation of Korea (NRF) Grant funded by the Korea Government (MSIT) under Grant 2020R1A2C100526513, and in part by the R&D Program for Forest Science Technology (Project No. 2021338C10-2323-CD02) provided by Korea Forest Service (Korea Forestry Promotion Institute).

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] L. Ceci, "*Number of smartphone users worldwide from 2016 to 2026*," Statista, 2021. [Online]. Available: https://www.statista.com/statistics/266210/numberof-available-applications-in-the-google-play-store/

[2] Google, "*The activity lifecycle*," 2020. [Online]. Available: https://android.com/guide/components/activities/activity-lifecycle.html

[3] J. Belzer, A. Holzman and A. Kent, "Operating systems," in *Encyclopedia of Computer Science and Technology*. Vol. 11. Boca Raton, FL, USA: CRC Press, pp. 442–443, 1981.

[4] N. Gupta, "*zram: Generic ram based compressed r/w block devices*," LWN, 2011. [Online]. Available: https://lwn.net/Articles/537422

[5] S. Jennings, "*The zswap compressed swap cache*," LWN, 2013. [Online]. Available: https://lwn.net/Articles/537422

[6] P. R. Jelenkovic and A. Radovanovic, "Least-recently-used caching with dependent requests," *Theoretical Computer Science*, vol. 326, no. 1–3, pp. 293–327, 2004.

[7] T. Yan, D. Chu, D. Ganesan, A. Kansal and J. Liu, "Fast app launching for mobile devices using predictive user context," in *Proc. of the 10th Int. Conf. on Mobile Systems, Applications, and Services*, Low Wood Bay, Lake District, UK, pp. 113–126, 2012.

[8] A. Martins, C. Duarte and J. Jeong, "Improving application launch performance in smartphones using recurrent neural network," in *Proc. the 2018 Int. Conf. on Machine Learning Technologies*, Jinan, China, pp. 58–62, 2018.

[9] Z. -X. Liao, S. -C. Li, W. -C. Peng, S. Y. Philip and T. -C. Liu, "On the feature discovery for app usage prediction in smartphones," in *Proc. of the IEEE 13th Int. Conf. on Data Mining*, Dallas, TX, USA, pp. 1127–1132, 2013.

[10] A. Parate, M. Bohmer, D. Chu, D. Ganesan and B. M. Marlin, "Practical prediction and prefetch for faster access to applications on mobile phones," in *Proc. of the ACM Int. Joint Conf. on Pervasive and Ubiquitous Computing*, Zurich, Switzerland, pp. 275–284, 2013.

[11] G. S. Moreira, H. Jo and J. Jeong, "NAP: Natural app processing for predictive user contexts in mobile smartphones," *Applied Sciences*, vol. 10, no. 19, pp. 6657, 2020.

[12] S. Zhao, Z. Luo, Z. Jiang, H. Wang, F. Xu *et al.,* "Appusage2vec: Modeling smartphone app usage for prediction," in *Proc. of the IEEE 35th Int. Conf. on Data Engineering*, Macao, China, pp. 1322–1333, 2019.

[13] S. Xu, W. Li, X. Zhang, S. Gao, T. Zhan *et al.,* "Predicting smartphone app usage with recurrent neural networks," in *Proc. of the Int. Conf. on Wireless Algorithms, Systems, and Applications*, Tianjin, China, pp. 532–544, 2018.

[14] C. Li, J. Bao and H. Wang, "Optimizing low memory killers for mobile devices using reinforcement learning," in *Proc. of the 13th Int. Wireless Communications and Mobile Computing Conf.*, Valencia, Spain, pp. 2169–2174, 2017.

[15] J. Lee and S. Park, "Mobile memory management system based on user's application usage patterns," *CMC-Computers, Materials & Continua*, vol. 68, no. 3, pp. 4031–4050, 2021.

[16] Google, "*Application lifecycle*," 2019. [Online]. Available: https://android.com/guide/components/activities/process-lifecycle

[17] Google, "*Low Memory Killer*," 2019. [Online]. Available: https://source.android.com/devices/tech/perf/lmkd

[18] Y. Khaokaew, M. Rahaman, R. White and F. Salim, "CoSEM: Contextual and semantic embedding for app usage prediction," in *Proc. of the 30th ACM Int. Conf. on Information & Knowledge Management*, Queensland, Australia, Virtual Event, pp. 3137–3141, 2021.

[19] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *Proc. of the 31st Int. Conf. on Machine Learning*, Beijing, China, vol. 32, pp. 1188–1196, 2014.

[20] K. Katsarou, G. Yu and F. Beierle, "WhatsNextApp: LSTM-based next-app prediction with app usage sequences," *IEEE Access*, vol. 10, pp. 18233–18247, 2022.

[21] R. Sen and D. A. Wood, "Reuse-based online models for caches," in *Proc. of the ACM SIGMETRICS/Int. Conf. on Measurement and Modeling of Computer Systems*, Pittsburgh, PA, USA, pp. 279–292, 2013.

[22] K. Greff, R. K. Srivastava, J. Koutn´ık, B. R. Steunebrink and J. Schmidhuber, "Lstm: A search space odyssey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 10, pp. 2222–2232, 2016.

[23] P. J. Werbos, "Backpropagation through time: What it does and how to do it," *Proc. of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[24] D. Yu, Y. Li, F. Xu, P. Zhang and V. Kostakos, Smartphone app usage prediction using points of interest. In: *Proc. Of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*. Vol. 1. Association for Computing Machinery, 2018.

[25] TensorFlow, "Release 2.6.0," 2021. [Online]. Available: https://github.com/tensorflow/tensorflow/releases/

[26] TensorFlow, "Keras release 2.6.0," 2021. [Online]. Available: https://github.com/keras-team/keras/releases/

[27] GSMArena, "Google pixel 3 specifications," 2019. [Online]. Available: https://www.gsmarena.com/googlepixel3-9256.php

[28] TensorFlow, "Tensorflow lite," 2021. [Online]. Available: https://www.tensorflow.org/lite

[29] Google, "Android open-source project," 2019. [Online]. Available: https://source.android.com/