Technical Disclosure Commons

Defensive Publications Series

November 20, 2018

User-Behavior-Guided Dynamic Loaders in Embedded Operating Systems

Hanumant Prasad Singh

User-Behavior-Guided Dynamic Loaders in Embedded Operating Systems

Abstract:

This publication describes a new user-behavior-guided dynamic loader in embedded operating systems (OS). It is well-understood that embedded devices, such as smartphones, are constrained in their random-access memory (RAM) resource. To better utilize the RAM resource, developers use shared libraries when building their application software (often referred to as apps or applications). Shared libraries reduce the memory footprint of individual application software. However, the utilization of shared libraries, even though beneficial in minimizing the volatile memory footprint, comes at a performance cost. The performance is improved, however, when shared libraries are pre-loaded before the application software is accessed by the end user. Current OS platforms lack the heuristic guided approach to predict which shared libraries are going to be needed at the time the end user accesses the application software. To this end, a new dynamic loader is developed to help predict the pre-loading of the needed shared libraries. To enable the OS to predict and pre-load shared libraries tailored to the end user, the new user-behavior-guided dynamic loader employs three components: user embedding, current time, and current location. To improve the performance of the dynamic loader, federated learning is utilized to democratize the computational power needed and benefit from each end user's input data. By so doing, the described techniques optimize the prediction of the relevant shared libraries to be pre-loaded, while protecting the end user's privacy. Consequently, user-behavior-guided dynamic loaders reduce the memory pressure of the embedded devices, while optimizing the performance of these devices.

Keywords: user embedding, embedded operating systems, loader, dynamic loader, shared libraries, neural network, dense neural network, deep learning, federated learning, dynamic linking

Background:

RAM resources constrain embedded devices, such as smartphones. To mitigate this, developers use shared libraries when building mobile application software for these devices. The use of shared libraries reduces the volatile memory footprint of individual application software, as it allows the logic represented by a shared library to be resident in memory in only one location but mapped to multiple process address spaces that use it. This removes the need to replicate a specific logic for different application software, allowing them to reduce their own individual memory footprint, which reduces memory pressure on the device's OS. When no application is using a memory resident shared library, the library is removed from RAM, freeing up the associated memory cells. However, the utilization of shared libraries, even though beneficial in minimizing the memory footprint, comes at a performance cost. Whenever an application accesses symbols from a shared library, there is a chance that the library is not loaded in memory, or even if it is loaded in memory, a symbol lookup within that library may still need to be performed. This penalizes the startup and run time of the application. This time penalty is a combination of the following operations at application startup and run time:

1) Load the requisite shared library from the non-volatile memory (often NAND flash) to the volatile memory (RAM).

2) Run format parser to setup the library in RAM.

3) Perform the dynamic linking involving symbol look-up. The executable files (or binaries) and libraries reference each-other through linking, which is achieved via a linker. Depending on the operating system (OS), the symbol look-up procedure varies, but ordinarily a look-up table is utilized.

Currently, operating systems lack the heuristic guided approach to predict which shared libraries are going to be needed at the time the end user accesses the application software. It is advantageous for the device's OS to predict the end user's device usage pattern, so the OS can pre-load the requisite shared library from the non-volatile memory to RAM (above-mentioned step 1) and run the format parser to setup the library in RAM (above-mentioned step 2) before the end user accesses the application software.

Description:

In embedded operating systems, the set of shared libraries is fixed by the platform provider. The application software that use the shared libraries are installed by end users from various online sources. It is the end user's specific mix of application software that determine the resident shared libraries in RAM. In addition, the type of application software an end user is likely to use has a correlation with which shared libraries are likely to be present in memory at any given time, because the end user does not use all downloaded applications at once or on a regular basis. For instance, many end users might have downloaded a specific application software, but only a few end users might use this application on a regular basis. Furthermore, the end users who use this specific application on a regular basis, might only do so during a given time of the day (e.g., after 7:00PM).

Each end user uses their mobile device in a unique pattern. Therefore, it is advantageous to employ differently each end user's pattern of starting and running applications on embedded devices. Such patterns help predict the likely shared libraries to be pre-loaded for a given end user at a given time. To this end, a user embedding is employed to create a representation of the end user's time-dependent behavior.

User embeddings are condensed vector representations of individual end users collected from the usage pattern of application software installed on the embedded devices. A time-stamped sequence when the end user visits applications during the day, is fed to a neural network to generate a dense vector representing the end user's behavior—a user embedding.

Fig.1 demonstrates an end user's pattern of visiting the various application software, which is fed to a neural network to produce a statistically determined user embedding. A periodic task time-stamps and records the end user's application software usage.

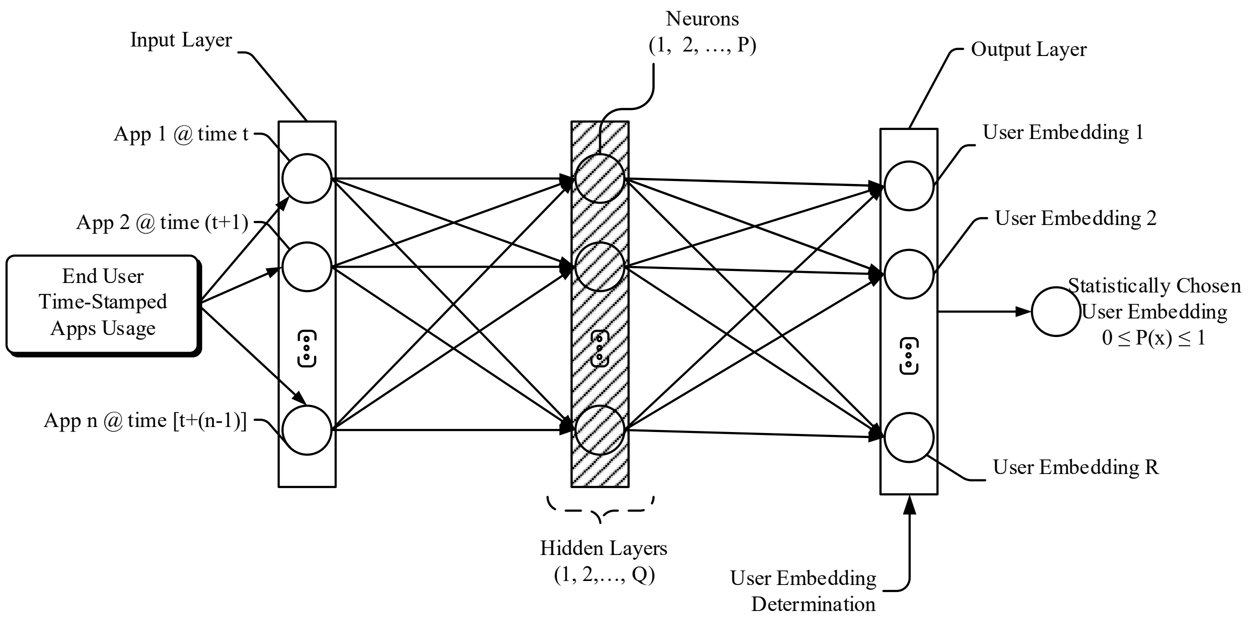

Fig. 1

The neural network in Fig. 1 illustrates an input layer, several hidden layers, and an output layer. The input layer includes "n" number of applications, which are time-stamped. There are "Q" number of hidden layers, with up to "P" number of neurons in each layer. There can be a different quantity of neurons in each hidden layer. The output layer includes "R" number of bins with different probabilities as the correct outcome. The bin with the closest probability to one (1) is interpreted as the correct output.

The statistically determined user embedding is re-evaluated periodically based on a pre-set time frame. The re-evaluation of user embedding is important because the end user might change a usage pattern of the mobile device, and the user embedding is improved with each learning exercise. The aim is to determine user embeddings that will aid the OS to only pre-load the anticipated shared libraries expected to be used. There is a give-and-take between the minimization of an application's RAM footprint and the performance cost of not pre-loading a shared library used by that application software. The re-iteration of this learning exercise benefits the end user's overall experience with the mobile device.

As shown in Fig. 1, the training of the model to determine user embeddings is accomplished by utilizing machine learning via a neural network. Furthermore, the following considerations are employed to achieve optimal success without comprising the end user's privacy:

1) A copy of the neural network model for predicting a user embedding resides on the embedded device as well as on a cloud-computing server.

2) The cloud-computing server negotiates an appropriate time with the embedded device so that any related activity between the server and the device is executed during a time when the embedded device is not in use (e.g., during the end user's sleep, morning meeting, family dinner, religious activity, etc.).

3) During the pre-negotiated time, the embedded device performs the training of the model using the collected data throughout the pre-determined time frame (e.g., the data collected during the last 24 hours).

4) After the embedded device has completed the training of the model, also during the pre-negotiated time, the updated model state is conveyed back to the cloud-computing server. To protect the end user's privacy, the collected data themselves need not be shared.

5) The model stored at the cloud-computing server benefits from such training exercises that are being performed by multiple end users. Each end user contributes to create a better

model without sharing any individual input data. This is sometimes referred to as federated learning.

Fig. 2 illustrates the basic idea of federated learning. Federated learning is not a fully de-centralized learning, but rather each end user contributes in accomplishing most of the training, while the cloud-computing server is mainly responsible for coordinating this learning exercise.

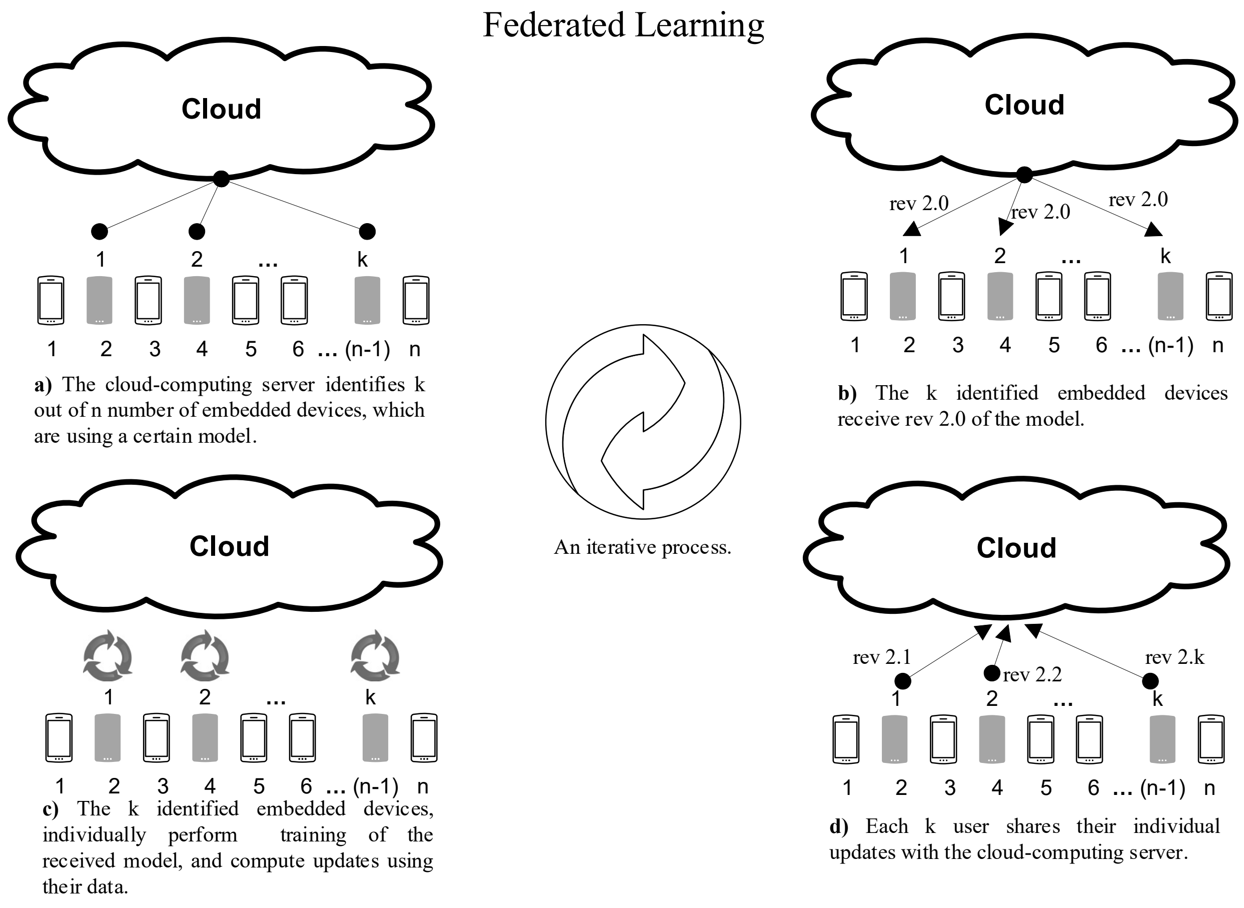
Federated Learning
Cloud
1
2
...
k
1 2 3 4 5 6 ... (n-1) n
a) The cloud-computing server identifies k out of n number of embedded devices, which are using a certain model.
Cloud
rev 2.0
rev 2.0
rev 2.0
1
2
...
k
1 2 3 4 5 6 ... (n-1) n
b) The k identified embedded devices receive rev 2.0 of the model.
An iterative process.
Cloud
1
2
...
k
1 2 3 4 5 6 ... (n-1) n
c) The k identified embedded devices, individually perform training of the received model, and compute updates using their data.
Cloud
rev 2.1
rev 2.2
rev 2.k
1
2
...
k
1 2 3 4 5 6 ... (n-1) n
d) Each k user shares their individual updates with the cloud-computing server.

Fig. 2

Consider a case when many end users who have downloaded a specific application software. The cloud-computing server notices that only a certain number of end users use this application on a regular basis; thus, it identifies those end users as good candidates for model training (Fig. 2a). The identified end users receive the most-current model of the user embedding model (Fig. 2b). Each end user performs model training during a pre-set time frame, which can differ for each end user (Fig. 2c). Then, each end user shares their own updates without sharing the input data that were used to perform the model training (Fig. 2d). The cloud-computing server collects and analyzes the individual updates, modifies the model, and sends it back to the end users (back to Fig. 2b). Fig. 2 shows how the cloud-computing server and the individual end users are working in concert to achieve the best-performing model through an iterative learning exercise.

User embedding (via time-stamping) is one piece of the complex puzzle to determine which shared libraries need to be pre-loaded. The applications an end user chooses are also influenced by current location (e.g., home, work, downtown, outdoors, etc.). In addition, current time gives granularity on the day of the week (e.g., weekday, weekend, Friday night, Sunday morning, etc.), or day of the year (e.g., Christmas, Easter, 4th of July in America, month of August in Europe, etc.).

To achieve a truly user-behavior-guided dynamic loader, user embedding, current location, and current time, are fed into yet another neural network, called a library-determination dense neural network. This dense neural network determines which shared libraries need to be pre-loaded. Fig. 3 illustrates the inputs and outputs to the library-determination dense neural network.

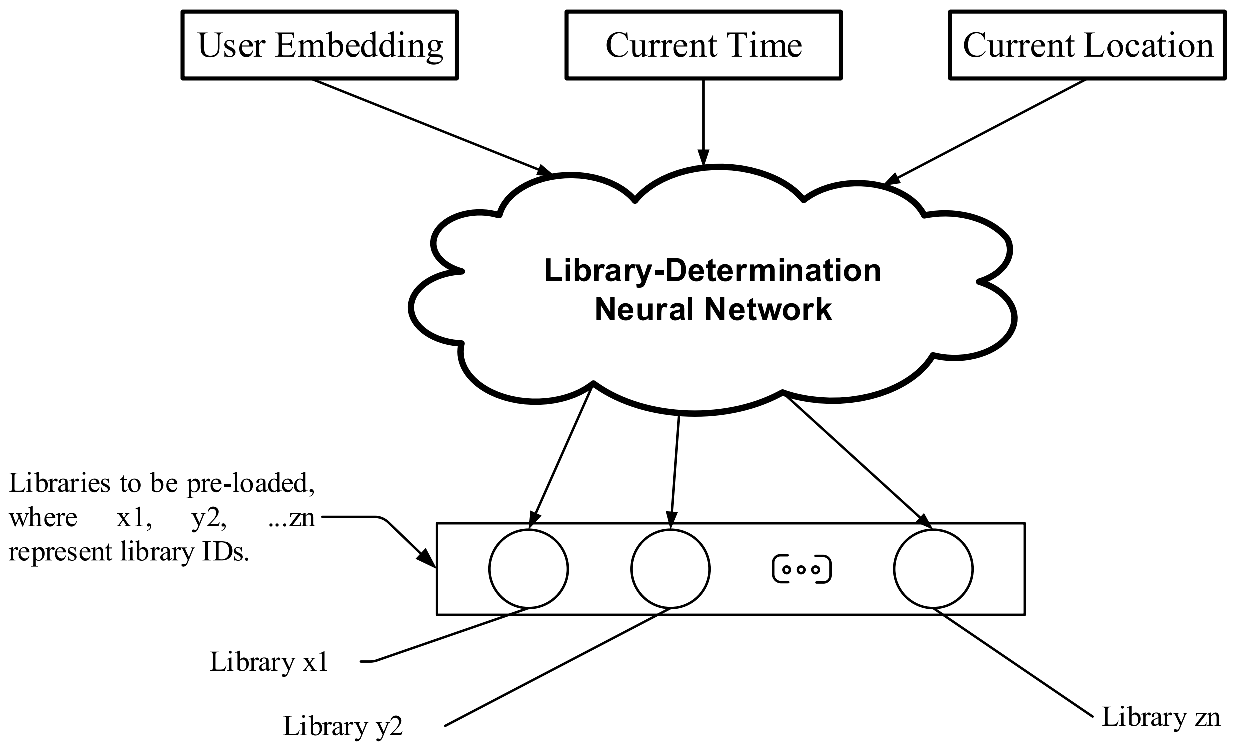
User Embedding
Current Time
Current Location
Library-Determination Neural Network
Libraries to be pre-loaded, where x1, y2, ...zn represent library IDs.
Library x1
Library y2
Library zn

Fig. 3

The dense neural network shown in Fig. 3 has three types of inputs: user embedding (already illustrated in Fig. 1), current time, and current location. There are multiple hidden layers that are not shown in Fig. 3. The output layer is a list of libraries scheduled for pre-loading. As in the case of user embedding, the training of the loader model is done using federated learning.

To evaluate the performance of the dynamic loader, the following data points are tracked:

1) Predicted shared libraries referenced by applications during tracking window—True positives.

2) Predicted shared libraries evicted by the OS without reference—False positives.

3) Libraries not predicted but referenced—False negatives.

A system constrained by memory would choose to maximize true positives and minimize false positives. The exclusion of a misidentified shared library is relatively inexpensive in terms of input/output (I/O). Thus, the performance can be evaluated under a receiver operating characteristic (ROC) curve, which is created by plotting a true positive rate (TPR) vs. a false positive rate (FPR) and the settings are adjusted to optimize the prediction of the shared libraries to be pre-loaded.

In conclusion, user embedding, coupled with current time and location, enable the OS to predict and pre-load shared libraries tailored to the end user. Federated learning democratizes the computational needed power and utilizes each the end user's input data to optimize the prediction of the relevant shared libraries to be pre-loaded, while protecting the end user's privacy. The user-behavior-guided dynamic loaders in the new OS reduce the memory pressure in the embedded devices, while optimizing the performance of these devices.