ARTIFACT: On-Device Deep Learning for IoT-based Wireless Sensing Applications

Manoj Lenka and Ayon Chakraborty
SENSE Lab, Indian Institute of Technology Madras, Chennai, India
Email: {cs22s008, ayon}@iitm.ac.in

***Abstract*—This artifact covers the dataset, models and code used in our work, *On-Device Deep Learning for IoT-based Wireless Sensing Applications* [1], to appear in the WiSense workshop held in conjunction with IEEE PerCom 2024. The artifact is also available publicly on: https://github.com/senselab-iitm/wisdom. For further details regarding the project, please visit https://cse.iitm.ac.in/%7Esense/wisdom.**

I. INTRODUCTION

The artifact primarily consists of measurement traces obtained from various wireless sensing experiments. As discussed in our paper, we record the Wi-Fi Channel State Information (CSI) corresponding to various human activities. Subsequently, we build neural network models for such Human Activity Recognition (HAR) tasks, which we extensively benchmark.

Overall, our artifact consists of three directories: `data`, `models` and `scripts`. The scripts are available in GitHub, while `data` and `models` are available in public Google Drive folders, link to which is available in GitHub and project website.

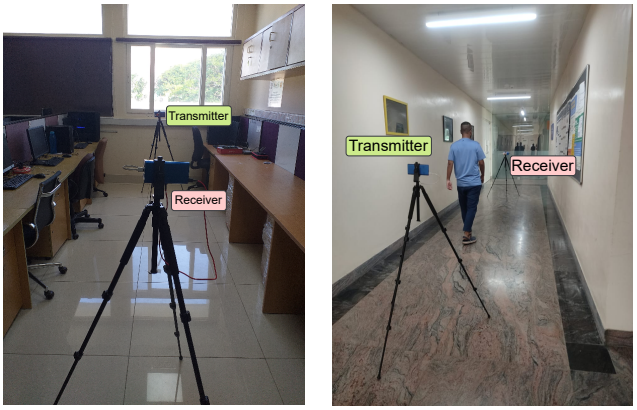
- The `data` directory contains the raw CSI collected for various HAR tasks. Second, it contains current measurement logs obtained by running different inferencing tasks on the ESP32 C3 MINI [2] microcontroller. The current measurements are obtained using the Power Profiler Kit II (PPK2) [3] (more on this in Sec. II).
- Various baseline models that are trained using the CSI HAR data are kept in the `models` directory. It also contains the compressed TF-LITE [4] models (more on this in Sec. III).
- The `scripts` mainly have three parts. First, to preprocess and visualize the HAR CSI data. Second, to generate and train models with different architecture and parameters. Third, to compress and test the models (more on this in Sec. IV).

Further, the above dataset/scripts can be used to create and train any new `Tensorflow` [5] model (beyond the shared ones), as well as compressing them. The current measurement data can be used to look into the energy characteristics of running such deep learning models on ESP32.

II. DATASET

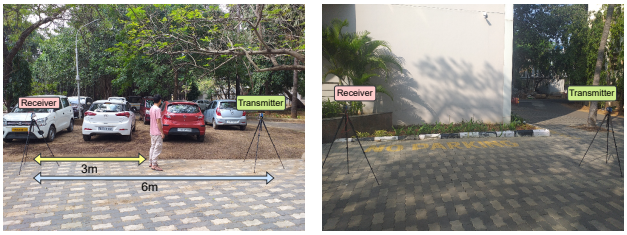
The `dataset` directory consists of two sub-directories, viz., `human_activity_recognition` and `current_measurements` whose contents are described below:

A. CSI Traces for various Human Activities



(a) Lab **(b)** Corridor

Fig. 1: Shows the **indoor** locations where CSI was collected for HAR



(a) Parking **(b)** Yard

Fig. 2: Shows the **outdoor** locations where CSI was collected for HAR

The `human_activity_recognition` directory contains the CSI traces. They are ordered by the location where they were collected in the respective sub-directories (`indoors/lab`, `indoors/corridor`, `outdoors/parking`, `outdoors/yard`, refer to Fig. 1 and 2). Please note the position of the transmitter (Tx) and the receiver (Rx) placed on tripod stands. In the Rx, the ESP32 pushes the collected CSI to a Raspberry-Pi (Model 3B) device via a serial interface. For each location, CSI traces for six different activities are recorded (`empty` – denoting absence of a person; static activities – `standing`, `sitting`; and dynamic activities – `sittingupdown`, `jumping`, `walking`). The naming format of each data file is `<location>_<activity>_<timestamp>.txt`. Each row in the file denotes a recorded sample, which has several entries including the received signal strength (RSS), IQ samples and so on (see `esp_col_desp.md` for details).

B. Current Measurements

The Power Profiler Kit II (PPK II) is used to obtain the live current measurements on ESP32, while the mod-

els are run to make relevant inferences. It is a time-series data with timestamp in milliseconds at one column and current drawn in micro-amperes at the other. Traces are recorded for `non_compressed` as well as for `quantized` models and stored in folders of the same name. In each folder there are further sub-folders denoting the architectures of the models (i.e., `cnn`, `fcn`, `rnn`). Inside each sub-folder, there are nine files, each having current measurements for models with increasing number of parameters. The naming format of current measurement files are `<q/na>_<architecture>_<parameters>.txt`, here `<q>` denotes quantized models, while `<na>` denote non-compressed models.

III. Models

We have two kinds of models in our artifacts as discussed below:

A. Keras *Models [6]*

The regular Keras models cannot be used for inferencing on microcontroller class devices. They are not compatible with the TF-Lite Micro APIs used in the C++ application programs that run on the microcontrollers like Esp32. However, it can run on general purpose computers with enough resources. Such models are saved in the `models/keras` directory and is ordered by architecture type. The saved model has the naming format as, `har_<architecture>_<config>_<compression>`. Here, `<architecture>` can be either `CNN` (Convolutional Neural Network), `FCN` (Fully Connected Network) or `RNN` (Recurrent Neural Network). The `<config>` denotes the configuration of the neural network that determines its size and parameter count.

The `<config>` for `CNN` models consists a series of numbers. Each number indicates the count of residual blocks, following which we double the number of filter, while decreasing the size of input by half. For example, `har_resnet_222_qat` has 8 residual blocks where we double the filter size and decrease the size of input of by half every alternate block.

The `<config>` for `FCN` models consists a series of numbers separated by underscore, denoting the number of layers and their sizes. For example, `har_fc_50_100_100_50_cqat` has 4 layers (excluding the softmax output), where each layer has 50, 100, 100, 50 neurons respectively. RNNs have a similar `<config>` to that of FCN.

B. TF-Lite *Models*

Such models are obtained from the saved Keras models and used for inferencing tasks on the microcontroller. The TF-Lite models are further converted to C++ files using the script in `scripts/compress_models` directory (see Sec. IV). These files are included and refereed to in the C++ application that runs on Esp32 and performs the inferencing. These models are stored in `tflite` directory and is also ordered by the architecture type. The TF-Lite models have same format as Keras models.

There is further a sub-directory called `change_hyperparameters` in `models/tflite`. The models here deal with changing hyperparameters like the number of clusters in weight clustering algorithm and sparsity level for pruning. This folder has further sub-directories, whose names indicates the number of cluster or sparsity percentage for all models.

For more details on the compression techniques used, please refer to our work [1]. To initialize the centroids of the clustering algorithm we have used `K-Means++` algorithm. For pruning, we have used a constant sparsity throughout training.

IV. Codebase

We divide the codebase into three parts as described below:

A. Data Prepossessing and Visualization

Here we have scripts that are used to read the Esp32 log files in the `data` directory (see Sec. II) and compute the CSI amplitude spectrograms that are used for training the models. We have also provided a sample iPython notebook, `data_viz.ipynb`, that can be used to plot and visualize the spectrograms. All code for this is in the `scripts/data_related` folder.

B. Building and Training Models

These contains code that we use to generate models with different architecture and parameter size (see Sec. III). It also trains the model on the data given in `data` folder and tests it, giving us a accuracy measure. The scripts related to it are in folder `scripts/train_models`

C. Compressing Models

These contains code that are used to compress the saved Keras models and also converts them to TF-Lite model. The scripts are stored at `scripts/compress_models`. The main file is `tf_lite_convert.py` which uses TinyML optimization functions defined in file `tinyml_opt.py`, along with some utility functions defined in `tf_lite_convert_utils.py`. It also contains script in file `convert_tflite_to_cc.py` to convert all `tflite` models in a folder to C++ files.

References

[1] M. Lenka and A. Chakraborty, "On-Device Deep Learning for IoT-based Wireless Sensing Applications," in *2024 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*.

[2] "ESP32 C3 MINI," *Expressif Systems*. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32-c3-mini-1_datasheet_en.pdf

[3] "Power Profiler Kit II," *Nordic Semiconductor*. [Online]. Available: https://www.nordicsemi.com/Products/Development-hardware/Power-Profiler-Kit-2

[4] R. David *et al.*, "Tensorflow lite micro: Embedded machine learning on tinyml systems," *arXiv*, 2020.

[5] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015.

[6] F. Chollet *et al.* (2015) Keras. [Online]. Available: https://github.com/fchollet/keras