A. Artifact Appendix

A.1 Abstract

We provide the source codes of ElasticTrainer, which aims to speed up the Neural Network (NN) training on weak embedded devices with limited computing power. ElasticTrainer adaptively selects trainable NN tensors on the fly during the training process, to achieve the desired training speedup while maximizing the training accuracy with best efforts. Its tensor selection is optimized by dynamic programming, based on the profiled information about the timing cost and importance of NN tensors in training. ElasticTrainer can gain 2x-3.5x training speedup without noticeable accuracy drops, compared to baseline schemes.

ElasticTrainer is generically applicable to training any NN models on different types of embedded devices with different computing hardware and software configurations. In this paper, we used several popular NN models (ResNet50, VGG16, and MobileNetV2) and two embedded platforms: Nvidia Jetson TX2 (Jetson) and Raspberry Pi 4B (Pi). To facilitate performance comparisons, our source codes also include implementations of multiple baseline schemes.

A.2 Artifact check-list (meta-information)

- **Program:** Python programs for NN training.
- **Binary:** Python scripts for source codes; OS images for Jetson TX2 and Raspberry Pi; NVIDIA Jetson Linux development tools.
- **Model:** Widely used public NN models, including ResNet50, VGG16, and MobileNetV2.
- **Data set:** Publicly available image datasets downloaded in advance in OS images, including CUB-200, Oxford-IIIT Pet, and Stanford Dogs.
- **Run-time environment:** Ubuntu 18.04 LTS on PC, Jetson Linux (Linux4Tegra) 32.7.1 on Jetson TX2, and Raspberry Pi OS 11 on Raspberry Pi 4B.
- **Hardware:** Nvidia Jetson TX2, Raspberry Pi 4B, a generic x86-64 PC and an external SATA SSD.
- **Execution:** NN training programs executed on the hardware above.
- **Metrics:** Quantitative results evaluation.
- **Output:** NN training results are plotted and saved as images on devices.
- **Experiments:** Install OS and set up hardware platform, download our source codes onto devices, and run shell scripts for automated result reproduction with NN training.
- **How much disk space required (approximately)?:** 300GB on PC, 64GB SD card on Raspberry Pi, 32GB internal storage for Jetson TX2, and at least 64GB for external SSD on Jetson TX2.
- **How much time is needed to prepare workflow (approximately)?:** 5 hours.
- **How much time is needed to complete experiments (approximately)?:** 10 hours to 790 hours, depending on whether to exclude running baseline schemes and choice of figures to reproduce.
- **Publicly available?:** Yes.
- **Code licenses (if publicly available)?:** MIT License.
- **Workflow framework used?:** No.
- **Archived (provide DOI)?:**
  Jetson image archive: `10.5281/zenodo.7812219`.
  Pi image and source code archive: `10.5281/zenodo.7812234`.

A.3 Description

A.3.1 How to access

Our source codes can be accessed from `https://github.com/HelloKevin07/ElasticTrainer`. The archived source codes and run-time system configurations in the form of OS images can be retrieved from Zenodo archive using the Archived DOIs above.

A.3.2 Hardware dependencies

The preparation of run-time environment requires a generic x86-64 PC with USB ports available. The execution of artifacts requires a standard NVIDIA Jetson TX2 Developer Kit and a Raspberry Pi 4B computer with at least 4GB RAM. Other required peripheral hardware includes:

- 64GB micro-SD card for Raspberry Pi.
- Micro-SD to USB type-A adapter for SD card flashing.
- SATA SSD with at least 64GB space (such as 1TB Samsung 870 EVO) for Jetson TX2.

A.3.3 Software dependencies

A clean Ubuntu 18.04 LTS operating system needs to be installed on PC in advance. Other auxiliary software on PC includes Etcher v1.18.4 (Linux AppImage, provided together with our artifact) and unrar tool.

The NVIDIA Jetson TX2 Developer Kit needs Jetson Linux (Linux4Tegra) version 32.7.1 flashed on the board. The Jetson Linux OS and required run-time software dependencies are provided as OS image in the artifact.

The Raspberry Pi 4B hardware needs a Raspberry Pi OS 11 flashed in the 64GB micro-SD card. The Raspberry Pi OS and required run-time software dependencies are provides as OS image in the artifact.

A.3.4 Data sets

We use publicly available datasets, including CUB-200, Oxford-IIIT Pet, and Stanford Dogs, which have been already included in our provided OS images. Our scripts will automatically download any missing datasets if they are accidentally removed.

A.3.5 Models

We use publicly available NN models, including ResNet50, VGG16, and MobileNetV2, which have been already included in our provided OS images. Our scripts will automatically download missing model architectures if they are accidentally removed.

A.4 Installation

This section describes the steps of setting up the run-time environment on Jetson TX2 platform and Raspberry Pi platform.

Before the setup, all provided artifacts need to be retrieved on the generic PC ("host PC") with Ubuntu 18.04 LTS installed as operating system. Install unrar tool on PC by executing `sudo apt install unrar`.

A.4.1 Jetson TX2 Platform Setup

1. Correctly assemble the Jetson TX2 board and connect external peripherals (keyboard, mouse, SSD and monitor). Configure it into Force USB Recovery Mode following Quick Start Guide provided with the Developer Kit. Connect the Jetson board with host PC using the provided USB cable following the instruction of Quick Start Guide.
2. On the host PC, place all files from Jetson TX2 artifact repository in the same directory on PC. Execute the script with command `bash ./prepare-jetson.sh` to extract the OS image.
3. On the host PC, execute the following commands to flash the Jetson Linux OS image onto the board:

```
cd ~/nvidia/ ; bash ./burn-img.sh
```

4. After the flashing finishes, the Jetson board shall boot into text-only interface with Ubuntu 18.04 LTS prompt.
5. On the Jetson board, log in using `nvidia` for both username and password, and execute the following command to finish SSD configuration:

```
sudo su - ; ./create-ssd-swap.sh
```

A.4.2 Raspberry Pi Platform Setup

1. Insert the micro-SD card onto host PC using the USB adapter.
2. Place all files from Pi image artifact repository in the same directory on host PC. Execute `bash ./prepare-rpi.sh` to extract the Pi OS image.
3. Launch Etcher software using `./balenaEtcher.AppImage`, and follow GUI instructions to flash the extracted image file (`rpi-system.img`) onto the SD card.
4. Insert the flashed micro-SD card onto Raspberry Pi board. Connect the Raspberry Pi board with peripherals (keyboard and mouse). Power up the board and confirm that the system has booted into the graphical interface.

A.5 Experiment workflow

With runtime environment being properly configured, we provide scripts to reproduce Figure 15, 16, 17(a)(c), and 19, which are the main experiment results reported in the paper.

Our source codes for ElasticTrainer are embedded in the OS images and can be found in `~/src/ElasticTrainer` directory of the system. It is also provided separately as `ElasticTrainer.tar.xz` in the source code artifact archive.

To access the code directory on Jetson, login with both username and password being `nvidia`, and execute the following commands after login:

```
sudo su -
cd ~/src/ElasticTrainer
chmod +x *.sh
```

To access code directory on Pi, execute the following commands in a terminal:

```
cd ~/src/ElasticTrainer
. ../kai_stl_code/venv/bin/activate
chmod +x *.sh
```

Due to the limited amount of time that may be available for artifact evaluation, reviewers may opt to validate partial experiments where results are the most representative (e.g., only on Jetson or excluding baselines). We annotate the device and estimated execution time for each experiment, allowing reviewers to customize their experiment workflow.

We recommend starting with our minimal working example that reproduces Figure 15 (a) and (d) using the following command:

```
./run_figure15ad.sh # 10 hrs on Jetson
```

To further reproduce the whole set of our figures, run the following command lines one by one:
(1) `./run_figure15.sh` (33 hrs on Jetson).
(2) `./run_figure16.sh` (221 hrs on Pi).
(3) `./run_figure17ac.sh` (15 hrs on Jetson, 190 hrs on Pi).
(4) `./run_figure19.sh` (20 hrs on Jetson, 310 hrs on Pi).

Alternatively, much time in (1)-(4) above can be waived by not running baseline schemes. To do so, modify the previous command lines by suffixing the shell script name with "_ego" suffix:
(1) `./run_figure15_ego.sh` (6.5 hrs on Jetson).
(2) `./run_figure16_ego.sh` (52 hrs on Pi).
(3) `./run_figure17ac_ego.sh` (9 hrs on Jetson, 85 hrs on Pi).
(4) `./run_figure19_ego.sh` (3.5 hrs on Jetson, 50 hrs on Pi).

A.6 Evaluation and expected results

After each experiment finishes execution, the corresponding figures should be generated as pdf files and saved to `figures/` directory. On Pi, navigate to `figures/` and directly click to view figures. On Jetson, figures can be viewed after switching to graphic mode:
(1) `sudo systemctl start graphical.target`
(2) In graphic mode, open a terminal, gain root privilege with `sudo su -`, and navigate to `figures/` under our code directory
(3) Use `ls` to check names of figure files. Use `evince` to view figures, e.g., `evince xxx.pdf`
(4) Reboot to switch back to text-only interface

The produced results should generally match the figures in our paper but could have small variations due to stochasticity of NN training itself. Compared to the results in our paper, the reproduced final accuracy of ElasticTrainer should not be worse by 5% and its wall-clock time should not be extended by 10%.

A.7 Experiment customization

We provide a list of configurable parameters (e.g., batch size and number of training epochs) for experiment customization. These parameters can be directly passed to `main.py` in command lines. For example:

```
python3 main.py \
        --model_name resnet50 \
        --dataset_name caltech_birds2011 \
        --train_type elastic_training \
        --num_epochs 12 \
        --batch_size 4 \
        --rho 0.6
```

More details are described in "General Usage" section in `README.md` on our source code repository.

A.8 Methodology

Submission, reviewing and badging methodology:

- `https://www.acm.org/publications/policies/artifact-review-badging`
- `http://cTuning.org/ae/submission-20201122.html`
- `http://cTuning.org/ae/reviewing-20201122.html`