**D****-ESCA 2 Project**

**Demo Guidance on Think Edge SE70**

SPARC Laboratory

*Technical Report*

| **Created by** | **Ho Trong Thanh, Nguyen Quang Thinh, Nguyen Minh Quan** |
| --- | --- |
| **Date of Report:** | **10 Feb-2023** |
| **Reporting Period:** |  |
| **Version** | **1.1** |

Foreword

This document guides on how to use the D-ESCA2 program and implement transfer learning on Think Edge SE70-12A7S00E.

# I. Instruction for using the D-ESCA2 program

## 1. Directory structure

In the D-ESCA2 folder, there are 7 sub-folders, each with its own use:

* **Docs**: includes all related documents.
* **dataset**: this folder is for storing the audio data created by the microphone.
* **config**: includes files that define configuration parameters for all programs.
* **core:** There are 5 sub-folders:
  + **DataLoader:** Includes essential functions for preparing data.
  + **Models**: includes all the definition of different models.
  + **Postprocessing**: contains a file defining a helper class that helps with drawing graphs and storing metrics. This is normally used with training and testing process.
  + **Preprocessing**: contains all helper functions for data preprocessing.
  + **Trainer**:
    - **Trainer.py:** defines a helper function for base training.
    - **TL\_Trainer.py:** defines a helper function for tl training.
* **helper:** 
  + **Resource\_monitoring.py:** monitors the GPU, CPU, RAM usage.
  + **audio\_cleanup.py**: remove all the 2 second recorded audio files from ./**D-ESCA\_v2/Results/rt\_test\_results/temp** and save them as a concatenated audio file in ./**D-ESCA\_v2/Results/rt\_test\_results/history** with the date\_time as file name.
  + **usbmictest.py:** records the audio from microphone using PyAudio library.
  + **micUSB\_cmd.py**: records the audio from microphone using cmd Linux
  + **check\_inputdevice.py:** list all existing audio input devices.
  + **trunking\_audio.py**: cuts down the long audio file into smaller, desirable trunks.
  + **utils.py**: includes some small functions that might be useful.
* **tools:**
  + **create\_dataset.py:** Generate a new dataset using the microphone.
  + **prepare\_data.py:** Extract the data from the WAV file and convert it to the tfrecord format for training purposes.
  + **b\_training.py**: defines the base training process (setting up => loading data => training model => saving the results).
  + **tl\_training.py**: just like base training process but this time with transfer learning.
  + **rt\_test.py**: runs a real-time test
* **Results**: contains all the results of the processes: base training, tl training and real-time test.
  + base\_training\_results:
    - logs: this directory contains TensorBoard log files that keep track of the coefficient variables, graphs of the model and loss function, and more during the base training process.
    - monitor: contains the resource monitoring results of the base training process.
    - saved\_model: Containing the model saved after the base training process.
    - saved\_parameter: Containing log files to save precision, recall, and threshold.
  + tl\_training\_results:
    - logs: this directory contains TensorBoard log files that keep track of the coefficient variables, graphs of the model and loss function, and more during the transfer learning training process.
    - monitor: contains the resource monitoring results of the transfer learning training process.
    - saved\_model: Containing the model saved after the transfer learning process.
    - saved\_parameter: Containing log files to save precision, recall, and threshold.
  + rt\_test\_log:
    - monitor: contains the resource monitoring results of the real-time test process.

## 2. Running instructions

### 2.1. Create dataset

***Text

Description automatically generated***

With create new dataset (./tool/create\_dataset.py), the config will be setup config file and add argument as follows:

Text

Description automatically generated with medium confidence

* RECORD.DEVICE\_INDEX\_INPUT: specify the index of mircodevice. (Refer to the instructions on how to retrieve the index in the real-time testing **section I.2.5**)
* RECORD.ANOMALY: Set to True if you want to create an anomaly dataset, set to False if you want to create a normal dataset.
* RECORD.SECOND: Length of each wav file in the dataset.
* RECORD.DATASET\_PATH: Specified path to save the dataset.
* **--config** (shorthand: -cfg): **Set Specify by path to config file (default to ./config/params.yaml)**

**Run: python ./tools/** **crearte\_dataset.py -cfg ./config/params.yaml**

***Note:***  When running the program on a PC, the DEVICE.JETSON parameter in the configuration file needs to be set to False and vice versa to True when running on a Jetson device (Jetson Nano, ThinkEdge SE70).

### 2.2. Prepare data

***Text, timeline

Description automatically generated***

With prepare data (./tool/prepare\_data.py), the config will be setup config file and add argument as follows:

Text

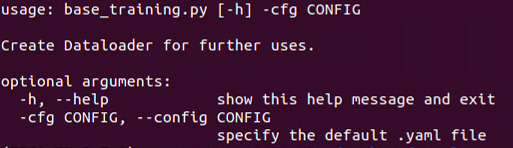
Description automatically generated

* DATASET.PATH.TFRECORDS: The path of dataset in tfrecord format.
* DATASET.PATH.NORMAL: The path of normal dataset in WAV format.
* DATASET.PATH.ANOMALY: The path of anormaly dataset in WAV format.
* **--config** (shorthand: -cfg): **Set Specify by path to config file (default to ./config/params.yaml)**

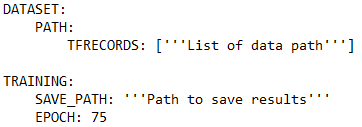
**Run: python ./tools/** **prepare\_data.py -cfg ./config/params.yaml**

***Note:***  When running the program on a PC, the DEVICE.JETSON parameter in the configuration file needs to be set to False and vice versa to True when running on a Jetson device (Jetson Nano, ThinkEdge SE70).

### 2.3. For base training



With base training (./tool/base\_training.py), the config will be setup config file and add argument as follows:

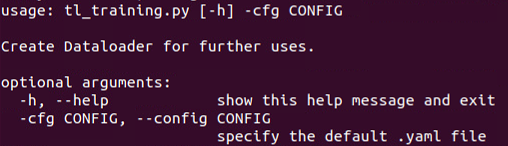


* DATASET.PATH.TFRECORDS: List of paths to data folder in tfrecord format (For the base training, there is only one element in list of paths, path to the data source folder).
* TRAINING.EPOCH: Specify the number of epochs for training (default to 75).
* TRAINING.SAVE\_PATH: choose the path to save results of base-learning, include: the log, model, parameter.
* **--config** (shorthand: -cfg): **Set Specify by path to config file (default to ./config/params.yaml)**

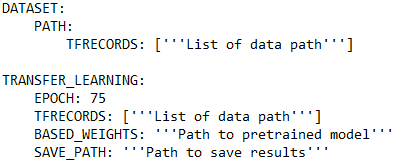
**Run: python ./tools/base\_training.py -cfg ./config/params.yaml**

***Note:***  When running the program on a PC, the DEVICE.JETSON parameter in the configuration file needs to be set to False and vice versa to True when running on a Jetson device (Jetson Nano, ThinkEdge SE70).

### 2.4. For transfer learning



With transfer learning training (./tool/tl\_training.py), the config will be setup config file and add argument as follows:



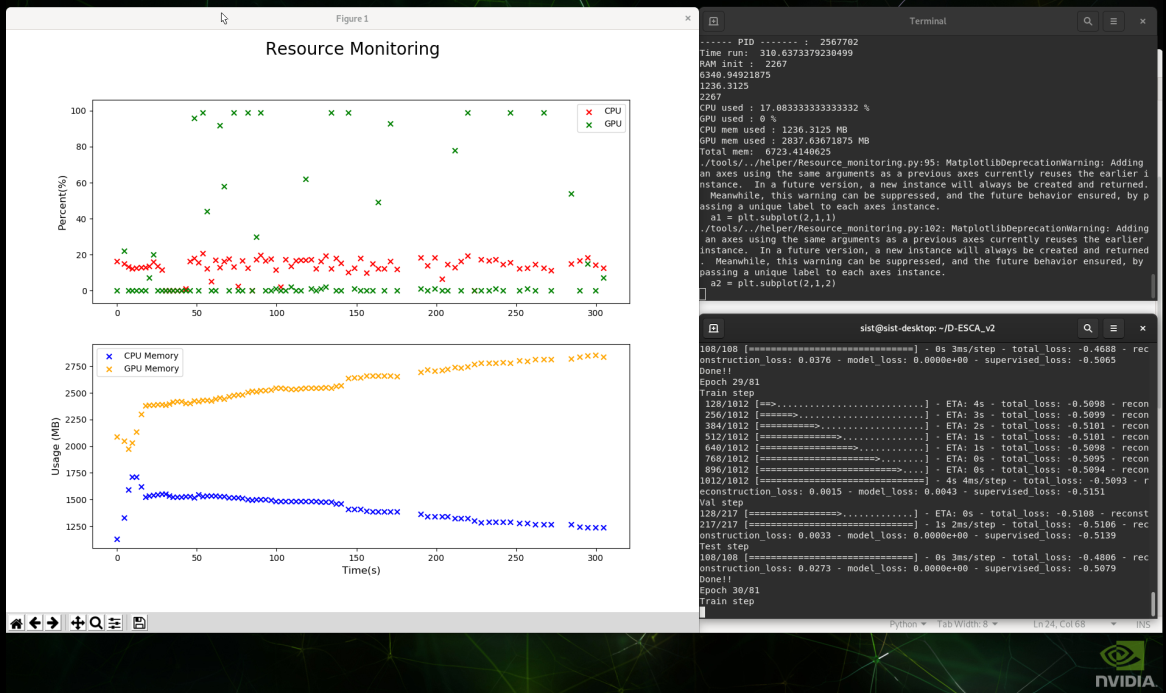
* DATASET.PATH.TFRECORDS: List of paths to data folder in tfrecord format (For the transfer training, the list of data path folder is specified according to the table below).
* TRANSFER\_LEARNING.TFRECORDS: List of paths to data folder (There is only one element in list of paths, path to the transfer-learning data folder).
* TRANSFER\_LEARNING.EPOCH: Specify the number of epochs for transfer-learning (default to 75).
* TRANSFER\_LEARNING.BASED\_WEIGHTS: Path to pretrained model
* TRANSFER\_LEARNING.SAVE\_PATH: Choose the path to save results of transfer-learning, include: the log, model, parameter.
* **--config** (shorthand: -cfg): **Set Specify by path to config file (default to ./config/params.yaml)**

**Run: python ./tools/tl\_training.py -cfg ./config/params.yaml**

**Table 1. Explanation on how to set the dataset path for transfer learning.**

| **Domain** | **Time** | **TRANSFER\_LEARNING** | **DATASET.PATH** |
| --- | --- | --- | --- |
| **1** | **1** | **Target1** | **Source** |
| **2** | **2** | **Target2** | **Source + Target1** |
| **3** | **3** | **Target3** | **Source + Target1 +Target2** |
| … | … | … | … |
| **N** | **N** | **Dataset N** | **Source + Target1 + Target2 +… + Target (N-1)** |

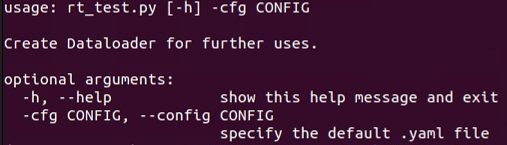
**Example:**



**Figure 1. Illustration of running transfer learning on Think Edge SE70.**

***Note:***  When running the program on a PC, the DEVICE.JETSON parameter in the configuration file needs to be set to False and vice versa to True when running on a Jetson device (Jetson Nano, ThinkEdge SE70).

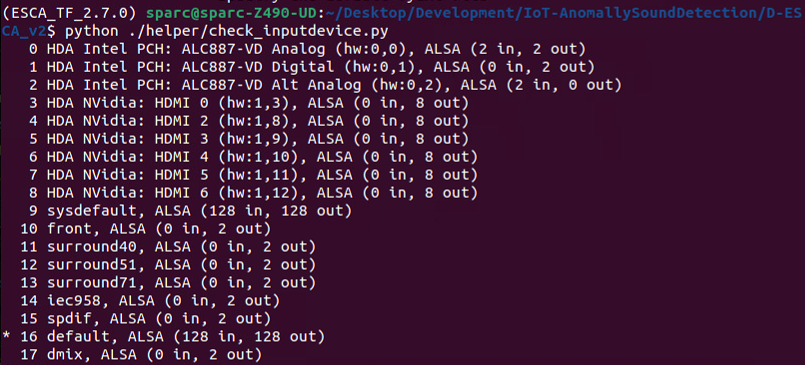
### 2.5. For real-time test



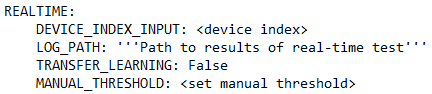
Before running the real-time test, please check the input micro device and the corresponding index of the device in the system, run command:

**python ./helper/check\_inputdevice.py**

The index is the order corresponding to the device to be printed, then select the index corresponding to your microdevice. Example my microdevice name is XXX, the index corresponding to that device is xx



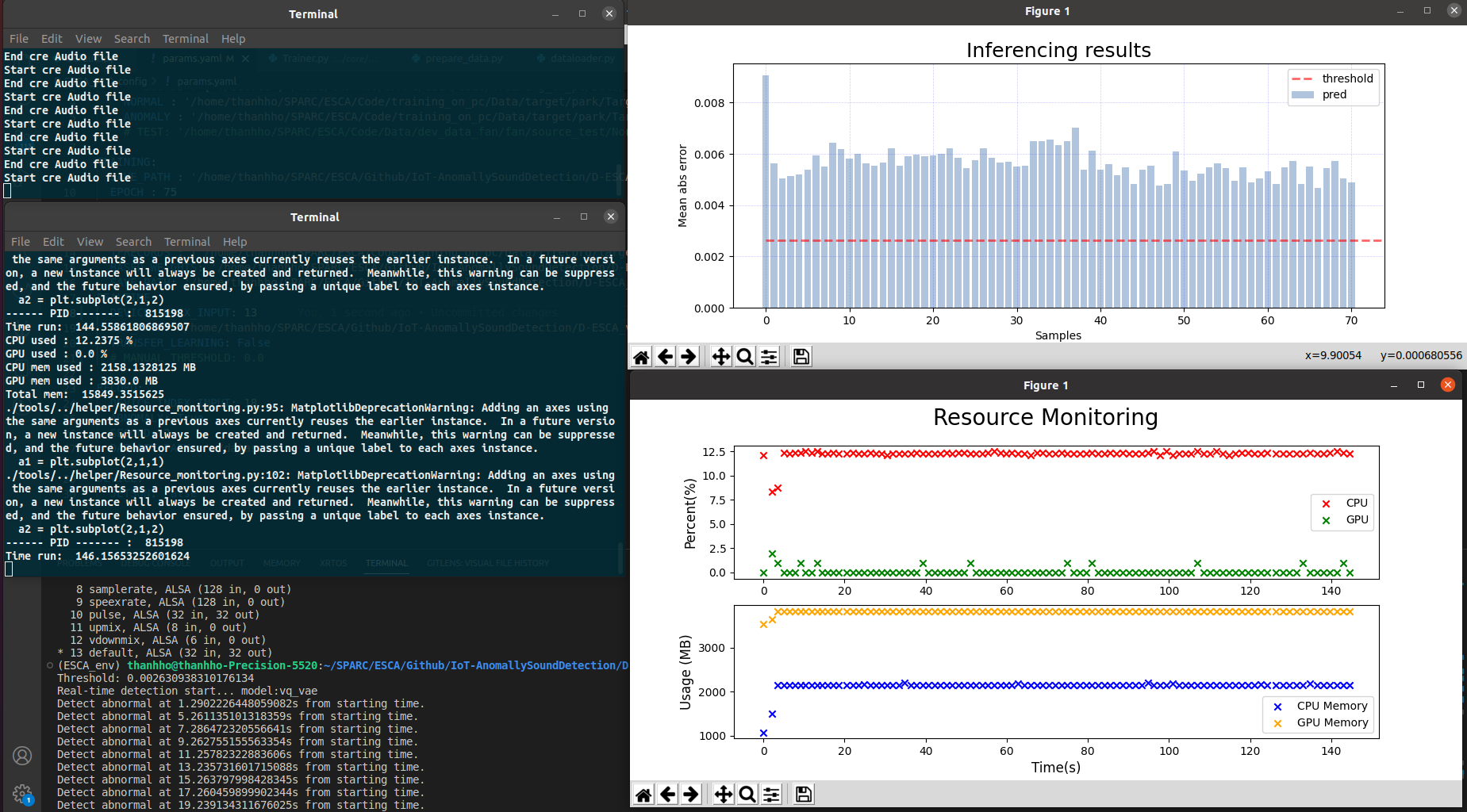
With real-time test (./tool/rt\_test.py), the config will be setup config file and add argument as follows:



* REALTIME.DEVICE\_INDEX\_INPUT: specify the index of mircodevice (get the index of device by instruction above).
* REALTIME.LOG\_PATH: Specify path to save results of real-time test.
* REALTIME.TRANSFER\_LEARNING: By default, it is set to True, the real-time inference process will load the transfer learning model saved in TRANSFER\_LEARNING.SAVE\_PATH, if set to False the real-time inference process will load the base model saved in TRAINING.SAVE\_PATH.
* REALTIME.MANUAL\_THRESHOLD: specify the hand-picked threshold value for a certain inferencing process (default to the threshold with highest combination of precision and recall score when evaluating with clean-up set).
* **--config** (shorthand: -cfg): **Specify config by path to config file (default to ./config/params.yaml)**

**Run: python ./tools/rt\_test.py -cfg ./config/params.yaml**

**Example:**



**Figure 2. Illustration of running real-time testing.**

***Note:***  When running the program on a PC, the DEVICE.JETSON parameter in the configuration file needs to be set to False and vice versa to True when running on a Jetson device (Jetson Nano, ThinkEdge SE70).

# II. Implementation of evaluation and testing.

## Data Prepared for Evaluation and Testing

### Dataset

The dataset used for training is comprised of 2s audio files that are described in detail in Table 2 below.

**Table 2. Description of the training dataset**

| **Categories** | | **Normal** | **Abnormal** |
| --- | --- | --- | --- |
| Park | Source | 780 | 19 |
| Target 1 | 947 | x |
| Target 2 | 711 | x |
| Target 3 | 700 | 38 |

This dataset has undergone a data prepare process and has been converted to the tfrecord format in the following path.

**Link:** <https://drive.google.com/drive/folders/1gSblQxDlIMN6HyzKIrO_bE9G8HssDw1t?usp=share_link>

### Audio demo

The dataset used for real-time testing is comprised of 2s audio files that are described in detail in Table 3 below.

**Table 3. Description of the real-time testing dataset.**

| **Categories** | | **Normal** | **Abnormal** |
| --- | --- | --- | --- |
| Park | Source | 35 | 17 |
| Target 3 | 19 | 50 |

This dataset is located in the following path

**Link:** <https://drive.google.com/drive/folders/15ogWIfKvGeBfZln598L3lpGO_yBn2Gem?usp=share_link>

## 2. Evaluation of the performance of devices when implementing transfer learning.

There are 3 devices used to evaluate the performance of transfer learning, including: PC, ThinkEdge SE70 and Jetson Nano. The specifications of the 3 devices are compared as shown in Table 4 below.

**Table 4. Comparing the specification of devices**

|  | **PC** | **Jetson Nano** | **Thinkedge SE70** |
| --- | --- | --- | --- |
|
| **CPU** | Intel® Core™ i7-7820HQ CPU @ 2.90GHz × 8 | Quad-core ARM A57 @ 1.43 GHz | 6-core NVIDIA® Carmel ARM v8.2 64-bit CPU- 1.9GHz, 6MB L2 + 4MB L3 |
| **GPU** | NVIDIA Corporation GM107GLM [Quadro M1200 Mobile] | 128-core Maxwell | 384-core NVIDIA Volta GPU with 48 Tensor Cores |
| **RAM** | 16G | 4 GB 64-bit LPDDR4 25.6 GB/s | 8 GB 128-bit LPDDR4x, 59.7GB/s |
| **Power**  **Consumption** | 130W | 5W | 10W | 10W | 15W | 20W |
| **Operating System (version Linux/ Jetpack)** | Ubuntu 20.04.5 LTS-64 bit | Ubuntu Desktop 18.04, kernel 5.4, JetPack 4.5.1 SDK | Ubuntu Desktop 20.04, kernel 5.10, Jetpack v5.0.2 SDK |

The survey will be conducted by performing transfer learning training with the park Target 3 data described in section II.1.1 while changing the GPU memory configuration. The results of the survey are presented in Table 5.

**Table 5. Results of Performance Survey of Devices after Transfer Learning with 81 Epochs**

| **Configuring GPU memory usage (using Tensorflow)** | **Transfer Learning Runtime** | | |
| --- | --- | --- | --- |
| **ThinkEdge SE70** | **Jetson Nano** | **PC** |
| **1GB** | 764.4 s | 3886.8 s | 218.2 s |
| **1.5 GB** | 753.6 s | 3889.9 s | 226.2 s |
| **2 GB** | 752.5 s | x | 224.2 s |
| **2.5 GB** | 768.7 s | x | 219.8 s |
| **3 GB** | 803.2 s | x | 223.1 s |
| **3.5 GB** | 814.7 s | x | 215.2 s |
| **4 GB** | 1005.9 s | x | x |

## 3. Demo real-time inferencing

### 3.1. Training phase

Based on the theoretical basis of the transfer-learning pipeline we mentioned **(Figure 3)**. We will do 4 domain training and 1 domain implementation **(Figure 4)**.

Diagram

Description automatically generated

**Figure 3. Transfer-learning pipeline**

With the ultimate goal of transferring learning on a ThinkEdge SE70 device and running real-time tests. We set up running pretrained on the computer to get the pretrained model. Use that model transfer learning on ThinkEdge devices to get new models for real-time testing on ThinkEdge devices.

Diagram

Description automatically generated

**Figure 4. Propose implementation for transfer-learning and real-time test**

With the basic settings and how to run we mentioned in the above items. In this section we describe the data sets used for the proposed implementation (imp) **(Figure 4)** and the last model (Model 4) will be used for real-time testing:

**Table 6.**

|  | Dataset | Epochs | Output model |
| --- | --- | --- | --- |
| Base-training (1) | Source | 75 | Model 1 |
| Transfer-learning (2) | Target1 | 75 | Model 2 |
| Transfer-learning (3) | Target2 | 75 | Model 3 |
| Transfer-learning (4) | Target3 | 81 | Model 4 |

Each Domain will be forwarded with the new data set and retrained with the old one, so we will set up the config based on the instructions in ***section I.2.4*** as follows:

**Table 7.**

| Domain | TRANSFER\_LEARNING.TFRECORDS | DATASET.PATH.TFRECORDS |
| --- | --- | --- |
| 1 | --- | Source |
| 2 | Target1 | Source |
| 3 | Target2 | Source + Target1 |
| 4 | Target3 | Source + Target1 + Target2 |
| Imp | --- | --- |

At Implementation (Imp) follow the instructions in **section II.3.2**

### 3.2. Inferencing phase (on Think Edge using microphone)

#### 3.1.1. Real-time inferencing using base model

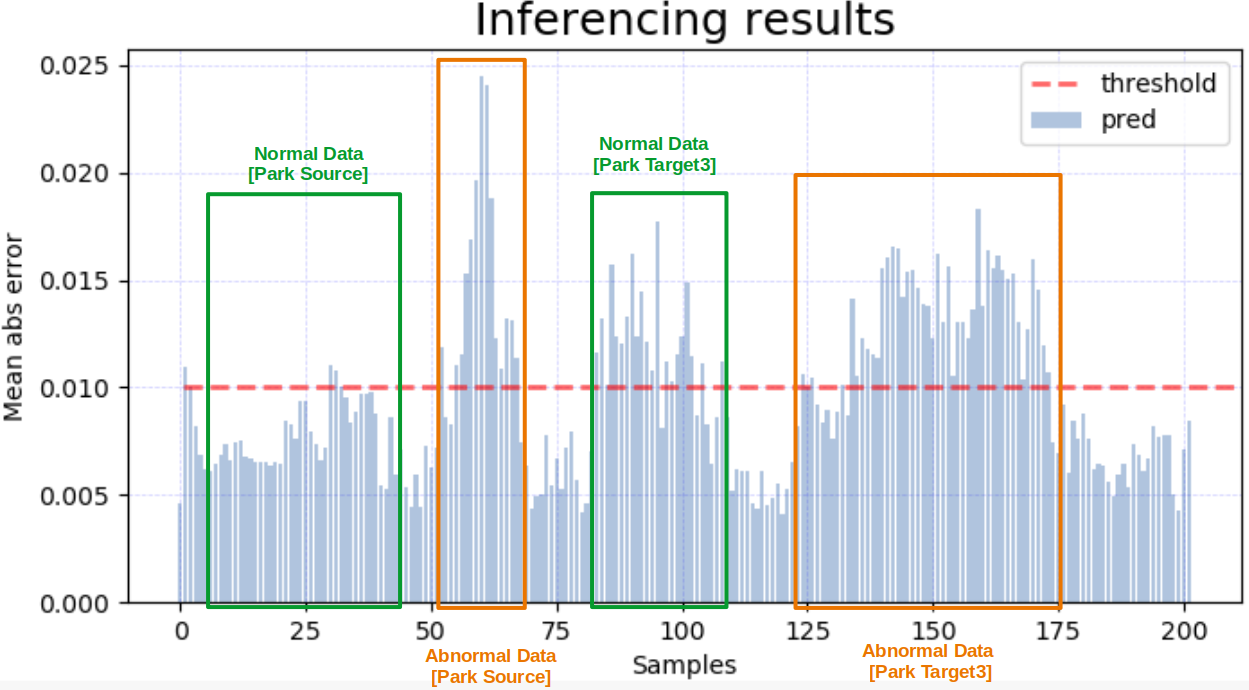
Run the ./tools/rt\_test.py program with the base model as instructed in section I.2.5 and play back the audio data described in section II.1.2.

A screenshot of a computer

Description automatically generated with low confidence

**Figure 5****. Description of how to perform real-time testing with the base model.**

Below are the results after performing real-time inference with the base model.



**Figure 6. Actual inferencing results when using the base model**

**Note:** MAE refers to the values in the blue column for each sample. The threshold here is the red horizontal line, typically the threshold will be automatically loaded from saved\_parameters/ saved from the training process. However, in some cases where the audio captured contains noise or other objective conditions, the threshold may be customized through the REALTIME.MANUAL\_THRESHOLD parameter in the configuration file.

#### 3.1.2. Real-time inferencing using transfer learning models

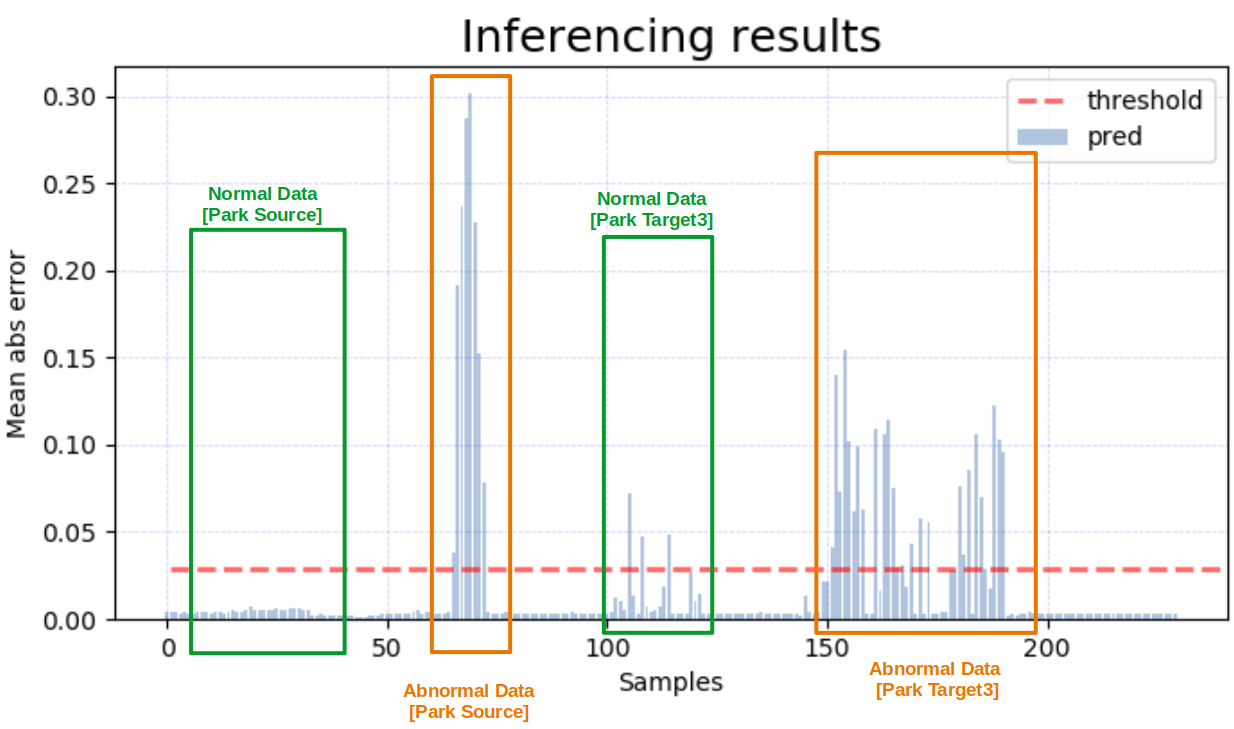
Run the ./tools/rt\_test.py program with the transfer learning model as instructed in section I.2.5 and play back the audio data described in section II.1.2.

Diagram

Description automatically generated

**Figure 7. Description of how to perform real-time testing with the transfer learning model.**

Below are the results after performing real-time inference with the transfer learning model.



**Figure 8. Actual inferencing results when using the transfer learning model.**

**Note:** MAE refers to the values in the blue column for each sample. The threshold here is the red horizontal line, typically the threshold will be automatically loaded from saved\_parameters/ saved from the training process. However, in some cases where the audio captured contains noise or other objective conditions, the threshold may be customized through the REALTIME.MANUAL\_THRESHOLD parameter in the configuration file.