

How to Use ULPMark[™]-ML

An overview for ULPMark-ML and tinyMLPerf

Peter Torelli, President, EEMBC

peter.torelli@eembc.org

Objective

What you will learn in this slide deck:

- How to use ULPMark-ML for performance & accuracy measurements
- Energy is covered in "Part 2" slide deck

Prerequisites: A basic understanding of

1. Embedded C programming (UARTs, GPIO),
2. Machine learning frameworks (loading models, read/write tensors, inference)

What is EEMBC?

Formed in 1997, EEMBC is a non-profit consortium of semiconductor manufacturers, integrators, and academics that develops industry-standard performance and energy benchmarks for embedded applications.

This project is a collaboration between EEMBC and MLCommons

<https://www.eembc.org>

What is ULPMark-ML?

A benchmark that measures the energy efficiency of neural net inference on ultra-low power (ULP) embedded devices

It is the 4th benchmark in EEMBC's ULPMark suite (<https://eembc.org/ulpmark>)

ULPMark-ML is also the energy component of the tinyMLPerf benchmark

Both share the same models, training, datasets, rules, and runner software

Create through a collaborative effort between EEMBC and MLCommons since they both share so many members

Terms

Framework - the ecosystem of hardware, software and firmware used to execute benchmarks and take measurements

Host PC - the computer that runs the benchmark software

Runner - host application software used by a developer to run the benchmark

EMON - Energy monitor, provides and/or measures energy consumption

DUT - Device under test, the thing we are measuring

IO Manager - A device used to electrically isolate the DUT from the Host PC

What does the benchmark measure?

1. Throughput, the number of inferences per second per a given model and pre-defined inputs
2. Accuracy, the Top-1 and AUC for a larger set of inputs
3. Energy, the Joules per inference required by the platform

All three metrics are reported for the same firmware image, no mix-and-match of performance and accuracy and energy

How does the benchmark work?

The user takes a pre-trained model (TFLITE/fp32) and converts it to C code using the SDK for their target embedded platform (DUT)

The user imports this C code into boilerplate firmware provided

The user connects the DUT to a PC hosting the benchmark runner

The runner feeds input stimuli to the platform and collects the scores, sometimes using external hardware like an energy monitor to record energy consumption

What platforms does it target?

Small, battery-powered embedded (tinyML) platforms

- Max 256 KB RAM & 1 MB FLASH
- Max active-power requirement of $<0.1\text{W}$

No line-power

No TDP limits

No O/S (unless basic RToS)

Often simply single-task platforms

Important link

<https://github.com/eembc/benchmark-runner-ml>

Contains links to the runner, datasets, videos, and firmware source.

Performance mode

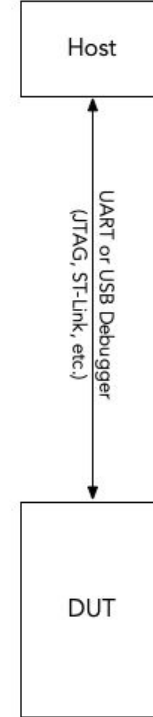
DUT connects directly to the Host PC

Basic text protocol between Host PC and DUT over UART allows downloading data and configuring execution

Baud rate can be set by altering the firmware and the Host Runner initialization file

The basic requirement for tinyMLPerf

Energy measurement is explained in the next set of slides



Setup Steps

1. Obtain the firmware boilerplate, host runner, models, and datasets
2. Review the basic firmware
3. Port the firmware to your SDK
 - a. Configure the UART
 - b. Configure a timer to count at least 1 kHz for the timestamp
4. Connect the DUT to the host system (USB/TTL serial or debugger serial port)
5. Select "ML Performance Mode" in the Host Runner

Overview of the neural nets

| Use Case | Description | Dataset | Model |
|----------------------|---|---------------------------|------------------|
| Keyword Spotting | Small vocabulary keyword spotting | Speech Commands | DS-CNN |
| Visual Wake Words | Binary image classification | Visual Wake Words Dataset | MobileNet |
| Image Classification | Small image classification | Cifar10 | ResNet |
| Anomaly Detection | Detecting anomalies in machine operating sounds | ToyADMOS | Deep AutoEncoder |

Training scripts: <https://github.com/mlcommons/tiny/tree/master/v0.1/training>

Datasets: <https://github.com/eembc/benchmark-runner-ml/tree/main/datasets>

"Golden" models: https://github.com/eembc/benchmark-runner-ml/tree/main/base_models

Rules

- Everyone must start with the same pre-trained fp32 TensorFlow Lite model
 - The model is then converted to your SDK, optimized, and pre-loaded into a firmware image based on some boilerplate C code
 - Each of the four models requires its own firmware image
 - Must be within 3% of target AUC/Accuracy (cannot "over optimize" for perf & energy and break accuracy)
- Everyone must use the same inputs
- The firmware interfaces with the EEMBC Benchmark Runner
 - The runner downloads test images and collects metrics based on a pre-defined measurement test-plan coded into the runner + firmware

Model equivalency rules:

https://github.com/mlcommons/tiny/blob/master/v0.1/TinyMLPerf_Rules.adoc#model-equivalence

Energy rules:

<https://www.eembc.org/ulpmark/ulp-ml/rules.php>

Firmware

<https://github.com/eembc/testharness-ulpmark-ml>

- The th_api folders contain the th_* files & th_* functions that need to be ported
 - Only "th_*" files may be modified, "ee_*" files/functions must remain untouched
- "Monitor" manages the interface to the host PC
 - th_api/th_lib.c contains callback for UART Rx; UART Tx; and timestamp
- "Profile" manages the neural net inference
 - th_api/th_profile.c contains code for writing input tensor, reading output tensor, inferring

Porting: GitHub Firmware Walkthrough

What does the runner actually do?

It communicates with the EMON, IO Manager and DUT to coordinate the execution of the benchmark

It uses a simple text-based protocol via serial port, USB, or VISA to serialize asynchronous events

Every component has a command-set

A benchmark is just a series of commands that trigger a linear progression of events and measurements

It can run GUI mode or "headless" mode for automation (documentation WIP)

Detect - identify compatible hardware connected to the system

Initialize - query the hardware for configuration information (e.g., NN model)

Run - execute the instructions in the main benchmark script (turn on the emon, start tracing, listen for timestamps, download data to the DUT, etc...)

Post-process - read the log and energy files to determine correct execution of the benchmark and extract measurements

Reload - reload and analyze data from previous session (sessions stored in \$HOME/eembc/runner/sessions)

First, copy datasets

Clone the benchmark runner repository

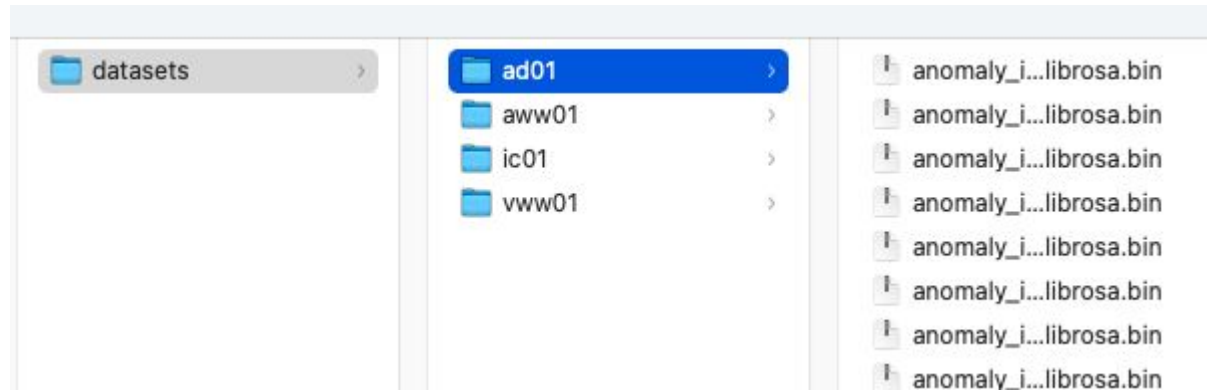
Create a folder called \$HOME/eembc/runner/benchmarks/ulp-mlperf/datasets/

Copy all four models' dataset folders to this area like so:

Note:

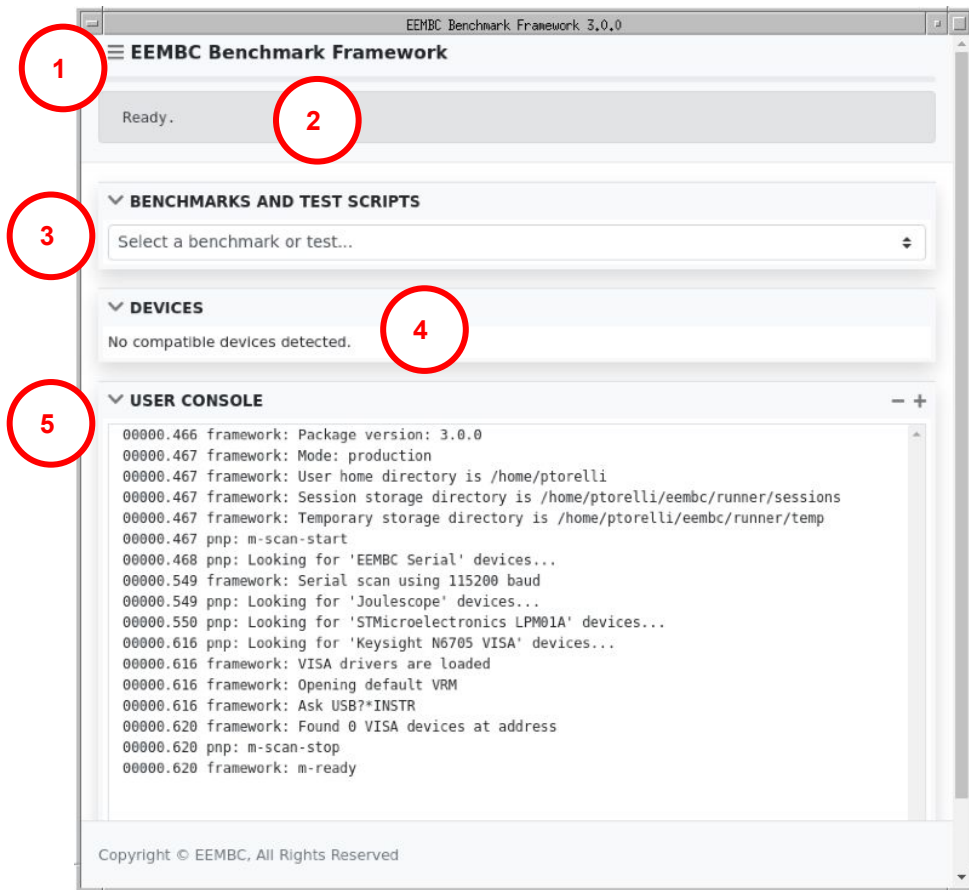
Windows uses

%USERDIR% instead of \$HOME



Runner GUI

1. Load emon data / reload session / exit
2. Status bar
3. List of available benchmarks (varies based on benchmark, others exist; comes with ML installed)
4. Device detection window for mounting / unmounting
5. User console displays messages, manual command entry (detailed docs WIP)



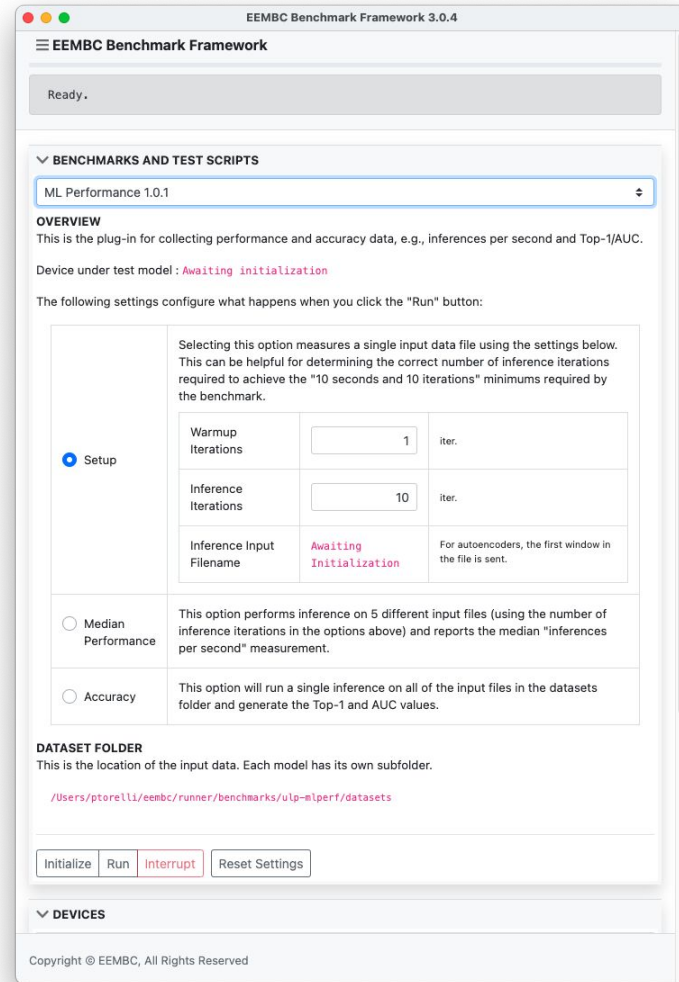
Plug in, start up

Connect the DUT to the host PC and start the runner application.

It will autodetect the DUT (it polls every serial port and asks "name%" at a default 115200 baud)

Select "ML Performance 1.0.0" in the "Benchmarks and Test Scripts" panel.

- Single: adjusting window sizes & debug
- Median...: official benchmark run (5 inputs, median performance)
- Accuracy: generate Top-1/AUC by feeding all input files



Initialize

Clicking "Initialize" performs a handshake

Queries the neural-net model built into the DUT

Needed to determine which datasets to use for input

"User Console" indicates what is being said between Host & DUT

User can also enter commands directly to DUT via its "alias"

What happens during a run?

1. Device is powered on & energy trace started
2. A timestamp is issued (for synchronization purposes)
3. The input file is downloaded from the Host PC to the DUT via repeated "db" commands
 - a. See the firmware GitHub README for explanation of "db" sequence[1]
 - b. Commands NOT printed to console (too many)
4. The "infer" command is sent to DUT
5. N warm-up inferences
6. Timestamp
7. M measured inferences
8. Timestamp
9. Power down
10. Post-process

```
void
ee_infer(size_t n, size_t n_warmup)
{
    th_load_tensor(); /* if necessary */
    th_printf("m-warmup-start-%d\r\n", n_warmup);
    while (n_warmup-- > 0)
    {
        th_infer(); /* call the API inference function */
    }
    th_printf("m-warmup-done\r\n");
    th_printf("m-infer-start-%d\r\n", n);
    th_timestamp();
    th_pre();
    while (n-- > 0)
    {
        th_infer(); /* call the API inference function */
    }
    th_post();
    th_timestamp();
    th_printf("m-infer-done\r\n");
    th_results();
}
```

[1] <https://github.com/eembc/testharness-ulpmark-ml>

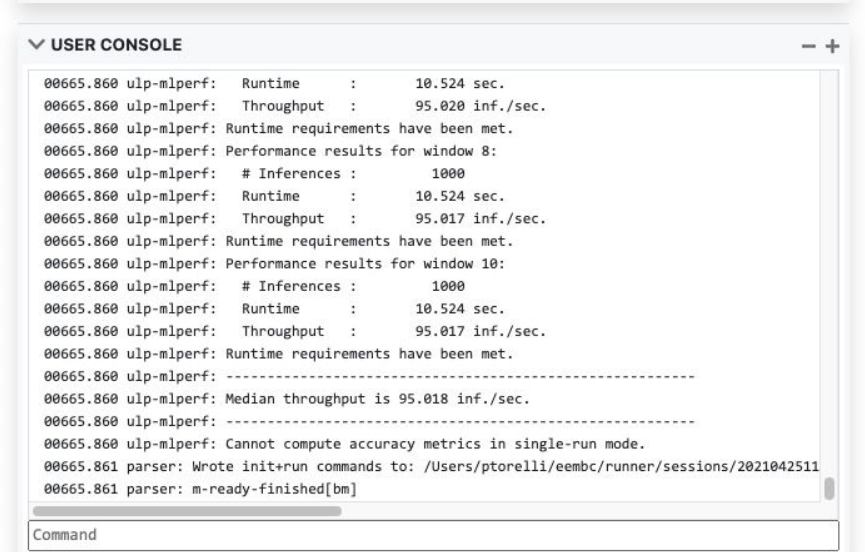
"Median Performance" & "Accuracy" Runs

Default settings run one warmup iteration and 10 measured iterations

A valid run takes at least 10 seconds and 10 iterations minimum to ensure valid measurement; generates an error otherwise

Median run shown: 5 Runs, median inf./ec

Accuracy run can take 1-2 hours, computes Top-1 and AUC (not shown)



```
▼ USER CONSOLE
00665.860 ulp-mlperf: Runtime      :      10.524 sec.
00665.860 ulp-mlperf: Throughput   :      95.020 inf./sec.
00665.860 ulp-mlperf: Runtime requirements have been met.
00665.860 ulp-mlperf: Performance results for window 8:
00665.860 ulp-mlperf:   # Inferences :         1000
00665.860 ulp-mlperf: Runtime      :      10.524 sec.
00665.860 ulp-mlperf: Throughput   :      95.017 inf./sec.
00665.860 ulp-mlperf: Runtime requirements have been met.
00665.860 ulp-mlperf: Performance results for window 10:
00665.860 ulp-mlperf:   # Inferences :         1000
00665.860 ulp-mlperf: Runtime      :      10.524 sec.
00665.860 ulp-mlperf: Throughput   :      95.017 inf./sec.
00665.860 ulp-mlperf: Runtime requirements have been met.
00665.860 ulp-mlperf: -----
00665.860 ulp-mlperf: Median throughput is 95.018 inf./sec.
00665.860 ulp-mlperf: -----
00665.860 ulp-mlperf: Cannot compute accuracy metrics in single-run mode.
00665.861 parser: Wrote init+run commands to: /Users/ptorelli/eembc/runner/sessions/2021042511
00665.861 parser: m-ready-finished[bm]

Command
```

Copyright © EEMBC, All Rights Reserved

Run rules

| Run Rule Type | # | Rule | Justification |
|----------------------------|-----|---|--|
| General | 1.1 | Scores are only valid if collected with the EEMBC Host Runner software. | To ensure the test was run according to specification. |
| General | 1.2 | Energy, performance, and verification (EPV) must use the same firmware. | To ensure the three different scores are consistent. |
| General | 1.3 | Verification score must be within Accuracy % and AUC margins-of-error found here . | To ensure optimizations have not degraded the model's accuracy. |
| General | 1.4 | The DUT must be "typical" power SKU, e.g. a median of a large sample of publicly available parts. | To avoid picking ultra-rare parts for higher scores. |
| General | 1.5 | The DUT hardware must be publicly available. | To ensure that the hardware is available to anyone. |
| Electrical & Environmental | 2.1 | Only one power supply is allowed into the DUT for energy mode. | To prevent escapee current paths that would artificially reduce power. |
| Electrical & Environmental | 2.2 | Any energy used during the benchmark must be drawn from the energy monitor. | To prevent attempts at hiding the amount of energy used (e.g., via supercapacitor). |
| Electrical & Environmental | 2.3 | Minimum 21C ambient temperature. | To prevent thermal-related optimizations (e.g., leakage sensitivity). |
| Electrical & Environmental | 2.4 | The board may be modified by cutting traces, removing jumpers, or desoldering bridges to remove ancillary components that may increase power. | To allow the user to remove unused platform hardware to obtain a more realistic score. |
| Firmware | 3.1 | Only "th_" functions may be modified in the firmware's source code. | To prevent the user from modifying the behavior of the benchmark. |

<https://www.eembc.org/ulpmark/ulp-ml/rules.php>

Accuracy targets

| Model | Source | Dimensions | Min Accuracy | File Format | # Stimuli |
|-------|--------------------|------------|--------------|--|---------------------------|
| vww01 | COCO2014/val2017 | 96x96 | 80% | U8C3, RGB, where [0]=ulc and [9215]=lrc | 500 true, 500 false |
| ic01 | CIFAR-10 | 32x32 | 85% | U8C3, RGB, where [0]=ulc and [1024]=lrc; this is different from original CiFAR-10 array which is 1024R, 1024G, 1024B | 200, 10 classes |
| ad01 | ToyADMOS/car | n/a | AUC: 0.85 | Spectrogram, 5 slices, 128 freq. bins, FP32LE | 108 anomaly, 140 normal |
| kws01 | Speech Commands v2 | n/a | 90% | Spectrogram, 49 frames x 10 MFCCs as INT8 | 1000 features, 12 classes |

<https://github.com/eembc/benchmark-runner-ml/tree/main/datasets#file-specifics>

Submitting your score

Use the score submission form at this page:

<https://www.eembc.org/ulpmark/ulp-ml/submit.php>

You will need to create a generic account first:

<https://www.eembc.org/user/create.php>

And confirm via email.

The submission page is very similar to the JSON submission form for tinyMLPerf with a few notable exceptions.

Energy measurements

See the second set of slides for an explanation of running in Energy Mode.

Q&A