MLPerf User Guide

Version 0.5
May 2nd, 2018

1 Overview

This document describes how to implement the MLPerf Suite using an ML framework and how to use that implementation to measure the performance of an ML software framework or hardware.

The MLPerf name and logo are trademarks. In order to refer to a result using the MLPerf name, the result must conform to the letter and spirit of the rules specified in this document. The MLPerf organization reserves the right to solely determine if a use of its name or logo is acceptable.

1.1 Definitions (read this section carefully)

The following definitions are used throughout this document:

Performance always refers to execution speed.

Quality always refers to a model's ability to produce "correct" outputs.

A *system* consists of a defined set of hardware resources such as processors, memories, disks, and interconnect. It also includes specific versions of all software such as operating system, compilers, libraries, and drivers that significantly influences the running time of a benchmark, excluding the ML framework.

A *framework* is a specific version of a software library or set of related libraries, possibly with associated offline compiler, for training ML models using a system. Examples include specific versions of Caffe2, MXNet, PaddlePaddle, pyTorch, or TensorFlow.

A *benchmark* is an abstract problem that can be solved using ML by training a model based on a specific dataset or simulation environment to a target quality level.

A *suite* is a specific set of benchmarks.

A *division* is a set of rules for implementing a suite to produce a class of comparable results.

A reference implementation is a specific implementation of a benchmark provided by the MLPerf organization.

A *benchmark implementation* is an implementation of a benchmark in a particular framework by a user under the rules of a specific division.

A *suite implementation* is a set of benchmark implementations for the entire suite using the same framework under the rules of a specific division.

A *run* is a complete execution of an implementation on a system, training a model from initialization to the quality target.

A *run result* is the wallclock time required for a run.

A *reference result* is the median of five run results provided by the MLPerf organization for each reference implementation

A *benchmark result* is the median of five run results normalized to the reference result for that benchmark. Normalization is of the form (reference result / benchmark result) such that a better benchmark result produces a higher number.

A *suite result* is the geometric mean of the benchmark results for a suite implementation under the rules of a division.

2 General rules

The following rules apply to all benchmark implementations.

2.1 Strive to be fair

Benchmarking should be conducted to measure the framework and system performance as fairly as possible. Ethics and reputation matter.

2.2 System and framework must be consistent

The same system and framework must be used for a suite result or group of benchmark results reported in a single context.

Note that the reference implementations use different frameworks and hence cannot be used collectively for a valid suite result.

2.3 System and framework must be available

If you are measuring the performance of a publicly available and widely-used system or framework, you must use publicly available and widely-used used versions of the system or framework.

If you are measuring the performance of an experimental framework or system, you must make the system and framework you use available upon demand for replication.

2.4 Benchmark implementations must be shared

Source code used for the benchmark implementations must be open-sourced under a license that permits a commercial entity to freely use the implementation for benchmarking. The code must be available as long as the results are actively used.

2.5 Non-determinism is restricted

The only forms of acceptable non-determinism are:

- Floating point operation order
- Random initialization of the weights and/or biases
- Random traversal of the inputs
- Reinforcement learning exploration decisions

All random numbers must be drawn from the framework's stock random number generator. The random number generator seed must entirely determine its output sequence. Random numbers must be utilized in a logical and consistent order across runs.

Additional rules may apply as described in later sections.

2.6 Benchmark detection is not allowed

The framework and system should not detect and behave differently for benchmarks.

2.7 Pre-training is not allowed

The implementation should not encode any information about the content of the dataset or a successful model's state in any form.

3 Benchmarks

The benchmark suite consists of the benchmarks shown in the following table.

Area	Problem	Dataset	Quality Target
Vision	Image classification	ImageNet	74.90% classification
	Object detection	COCO	0.377 Box min AP and 0.339 Mask min AP
Language	Translation	WMT English-German	25.00 BLEU
	Speech recognition	Librispeech	23.00 WER
Commerce	Recommendation	MovieLens-20M	0.9562 HR@10
	Sentiment analysis	IMDB	90.60% accuracy
General	Reinforcement learning	Go	40.00% pro move prediction

The MLPerf organization provides a reference implementation of each benchmark, which includes the following elements:

Code that implements the model in a framework.
A plain text "README.md" file that describes:

- Problem
- Dataset/Environment
    - Publication/Attribution
    - Data preprocessing
    - Training and test data separation
    - Training data order
    - Test data order
    - Simulation environment (RL models only)
- Model

    - Publication/Attribution
    - List of layers
    - Weight and bias initialization
    - Loss function
    - Optimizer
- Quality
    - Quality metric
    - Quality target
    - Evaluation frequency (training items between quality evaluations)
    - Evaluation thoroughness (test items per quality evaluation)
- Directions
    - Steps to configure machine
    - Steps to download and verify data
    - Steps to run and time

A “download_dataset” script that downloads the dataset.
A “verify_dataset” script that verifies the dataset against the checksum.
A “run_and_time” script that executes the benchmark and reports the wall-clock time.

4 Divisions

There are two divisions of the benchmark suite, the Closed division and the Open division.

4.1 Closed Division

The Closed division requires using the same preprocessing, model, and training method as the reference implementation.

The closed division models are:

Area	Problem	Model
Vision	Image classification	Resnet-50 v1
	Object detection	Mask R-CNN
Language	Translation	Transformer
	Speech recognition	Deep Speech 2
Commerce	Recommendation	Neural Collaborative Filtering
	Sentiment analysis	Seq2-BowN-CNN
General	Reinforcement learning	Mini Go (based on Alpha Go paper)

The unqualified name “MLPerf” must be used when referring to a Closed Division suite result, e.g. “a MLPerf result of 4.5.”

4.2 Open Division

The Open division allows using arbitrary preprocessing, model, and/or training method. However, the Open division still requires using supervised or reinforcement machine learning in which a model is iteratively improved based on training data, simulation, or self-play.

The qualified name “MLPerf Open” must be used when referring to an Open Division suite result, e.g. “a MLPerf Open result of 7.2.”

5 Data Set

5.1 Data State at Start of Run

Each reference implementation includes a script to download the input dataset and script to verify the dataset using a checksum. The dataset must be unchanged at the start of each run.

5.2 Preprocessing

All preprocessing time is included in the wall-clock time for a run result.

CLOSED: The same preprocessing steps as the reference implementation must be used.

OPEN: Any preprocessing steps are allowed. However, each datum must be preprocessed individually in a manner that is not influenced by any other data.

5.3 Training and Test Sets

If applicable, the dataset must be separated into training and test sets in the same manner as the reference implementation.

5.4 Training Data Order

CLOSED: the training and test data must be traversed in the same conceptual order as the reference implementation. For instance, the data might be traversed sequentially or randomly with uniform distribution. Batch size and the random number generator will affect order.

Future versions of the benchmark suite may specify the Closed traversal order.

OPEN: the training data may be traversed in any order. The test data must be traversed in the same order as the reference implementation.

6 RL Environment

CLOSED: The implementation must use the same RL algorithm and simulator or game as the reference implementation, with the same parameters.

OPEN: The implementation may use a different RL algorithm but must use the same simulator or game with the same parameters. If the reference implementation generates all data online, the Open division implementation must also generate all data online.

It is allowed and encouraged to parallelize and otherwise optimize (e.g. by implementing in a compiled language) the RL environment provided that the semantics are preserved.

7 Model

CLOSED: The benchmark implementation must use the same model as the reference implementation, as defined by the remainder of this section.

OPEN: The benchmark implementation may use a different model.

7.1 Graph Definition

CLOSED: Each of the current frameworks has a graph that describes the operations performed during the forward propagation of training. The frameworks automatically infer and execute the corresponding back-propagation computations from this graph. Benchmark implementations must use the same graph as the reference implementation.

7.2 Weight and Bias Initialization

CLOSED: Weights and biases must be initialized using the same constant or random value distribution as the reference implementation.

OPEN: Weights and biases must be initialized using a consistent constant or random value distribution.

7.3 Graph Execution

CLOSED: Frameworks are free to optimize the “non-stateful” parts of the computation graph provided that the semantics are unchanged. So optimizations and graph / code transformations of the flavor of dead code elimination, common subexpression elimination, and loop-invariant code motion are entirely allowed.

OPEN: Frameworks are free to alter the graph.

8 Training Loop

8.1 Hyperparameters

Hyperparameters (e.g. batch size, learning rate) may be selected to best utilize the framework and system being tested.

8.2 Loss function

CLOSED: The same loss function used in the reference implementation must be used.

OPEN: Any loss function may be used. Do not confuse the loss function with target quality measure.

8.3 Optimizer

CLOSED: The same optimizer used in the reference implementation must be used.

OPEN: Any optimizer must be used, provided that it is used consistently and is deterministic.

8.4 Quality measure

Each run must reach a target quality level on the reference implementation quality measure. The time to measure quality is included in the wallclock time.

The same quality measure as the reference implementation must be used. The quality measure must be evaluated *at least* as frequently (in terms of number of training items between test sets) and *at least* as thoroughly (in terms of number of tests per set) as in the reference implementation. Typically, a test consists of comparing the output of one forward pass through the network with the desired output from the test set.

9 Run Results

A run result consists of a wall-clock timing measurement for an entire run, including model construction, data preprocessing, and quality testing.

10 Benchmark Results

Each benchmark result is the median of five run results produced using the integer random number generator seeds 1 through 5. The seeds will not produce the same random number sequences on different frameworks/systems but make cherry-picking runs harder. All five run results must also be reported.

Each benchmark result should be normalized by dividing the reference result for the corresponding reference implementation by the benchmark result. This normalization produces higher numbers for better results, which better aligns with human intuition.

11 Division Results

In order to report a division result, a benchmark score must be reported for each benchmark in the suite. The division score is the geometric mean of the benchmark results.

12 Framework Reporting

Report the framework used, including version.

13 System Reporting

If the system is available in the cloud it should be benchmarked in the cloud. On premise benchmarking is allowed when the required system is not available in the cloud.

13.1 With Cloud Hardware

13.1.1 Replication recipe

Report a recipe that starts from a vanilla VM image or Docker container and a sequence of steps that creates the system that performs the benchmark measurement.

13.1.2 Price

Include the total cost of obtaining the median run result using fixed prices for the general public at the time the result is collected. Do not use spot pricing.

13.2 With On-premise Hardware

13.2.1 Replication recipe

Report everything that will eventually be required by a third-party user to replicate the result when the hardware and software becomes widely available.

13.2.2 Power

Include the total power consumed to produce the median run result.

14 Submissions

The MLPerf organization will create a database that collects submission data; one feature of the database is producing a leaderboard.

14.1 Submission Form

Submissions to the database must use the provided submission form to report all required information.

14.2 Submission Process

Submit the completed form and supporting code to the MLPerf organization Github mlperf/results repo as a PR.