MLHarness: A Scalable Benchmarking System for MLCommons

Yen-Hsiang Chang¹, Jianhao Pu¹, Wen-mei Hwu¹, Jinjun Xiong¹,²
¹University of Illinois at Urbana-Champaign, ²University at Buffalo
Overview

- Background and Challenge
- Our Solution -- MLHarness
- Experimental Results – Case Studies
Machine learning (ML) and deep learning (DL) innovations are being developed in a rapid pace.

Different people have different needs:

- **Model Builder**
  - Analyze and optimize model, and publish repeatable results.

- **System Developer**
  - Identify and solve bottlenecks on a complex system across hardware and software stacks.

- **User**
  - Choose models and systems under a fair comparison.
Challenge

1. Analyzing ML/DL innovations
2. Sharing ML/DL innovations
3. Reporting fair benchmarking results
Existing Approach

- ML/DL model zoos
  - TorchVision, ONNX Model Zoo, GluonCV
- Collections of reproducible components
  - CloudOps, MLOps
- Plug-and-play shareable containers
  - MLCube
- Benchmarking platforms simulating scenarios
  - MLCommons Inference
- Profilers capturing specific stacks
  - CUDA Profiling Tools Interface
Our Solution -- MLHarness

- MLHarness is built on top of two existing solutions: MLCommons Inference and MLModelScope.

MLCommons Inference
- Properly defined metrics with scenarios
- Fair benchmarking methodologies
- Focus on few models
- Hard to identify bottlenecks

MLModelScope
- Shareable exchange specification
- Across stack profiling
- Limited to computer vision tasks
- No straightforward reports
Our Solution -- MLHarness

- MLHarness extracts advantages from MLCommons Inference and MLModelScope and has the following features:
  - Codifying required benchmarking environments
  - Reporting metrics defined by MLCommons Inference
  - Supporting across stack profiling functionality from MLModelScope
  - A wide range of models
Encapsulating MLModelScope

MLHarness

MLCommons Inference in Python

- Interfaces
  - ConstructSUT
  - DestroySUT
  - LoadQuerySamples
  - UnloadQuerySamples
  - QuerySamplesComplete

 ctypes

Function Wrappers
- Initialize
- Finalize
- IssueQuery
- LoadQuerySamples
- UnloadQuerySamples

Go Function Calls

MLModelScope Runtime
- Tracer
- Profiler
- Database
- Framework
- Model
- Dataset

As C shared Library
Extending Specification

- MLCommons Inference supports few models.
- MLModelScope supports computer vision tasks.
- MLHarness extends MLModelScope's specification to include customized pre and postprocessing so that MLHarness can easily support a wide range of models.
Preprocessing and Postprocessing

- Need to run Python functions from Go.
- Embed Python interpreter into Go, with the help of cgo from Go and ctypes module in Python.

```go
func Processing(tensor interface{}, functionName string) interface{} {
    pyData := MoveDataToPythonInterpreter(tensor)
    pyFunc := FindTheProcessingFunctionByItsName(functionName)
    pyResult := ExecuteProcessingFunction(pyFunc, pyData)
    result := GetResultFromPythonInterpreter(pyResult)
    return result
}
```
Data movement between languages

- Data movement is expensive, especially for serializing and deserializing.
  - Instead, we only send the address and the shape of the tensor from one side, and copy data from memory based on these values on the other side.

- The garbage collector at one end doesn't know that it needs to keep data before really copied or used at the other end, hence memory will be mis-collected.
  - Manually create blocking statements
  - KeepAlive function in Go and reference count in Python
  - Prevent garbage collection until KeepAlive is executed in Go or the reference count is decreased to zero in Python
Contributions of MLHarness

- MLHarness encapsulates MLModelScope as an easy-to-use black box for MLCommons Inference, with the advantage of supporting across stack profiling and reporting properly defined metrics based on scenarios.
- MLHarness extends exchange specification from MLModelScope and supports user-defined processing methods, in order to embrace various models from different tasks.
Case Study 1: V100 v.s. A100

- Compare the performance of ResNet50, provided by MLCommons Inference, between 1x NVIDIA V100 GPU and 1x NVIDIA A100 GPU.

- The following parameters are the same between experiments:
  - CPU: 1x AMD EPYC 7702 64-Core Processor
  - Framework: ONNX Runtime
  - Scenario: Offline and batch size equals to 1

- Results:
  - V100: 202 samples per second
  - A100: 159 samples per second
Case Study 1: V100 v.s. A100

- With the across stack profiling support from MLModelScope, MLHarness provides crucial insight on this abnormal behavior.

![Graph showing performance comparison between V100 and A100](image-url)
Case Study 2: Various Models and Systems

- Compare the performance of AlexNet and models from the ResNet family, provided by TorchVision, between different systems, with variations on frameworks, processors, and accelerators.
- The experiments are done with offline scenario and batch size equals to 1.

<table>
<thead>
<tr>
<th>System Annotations</th>
<th>Framework</th>
<th>Processor</th>
<th>Accelerator</th>
</tr>
</thead>
<tbody>
<tr>
<td>9800-ORT-RTX</td>
<td>ONNX Runtime</td>
<td>1x Intel(R) Core(TM) i7-9800X CPU @ 3.80GHz</td>
<td>1x GeForce RTX 3090</td>
</tr>
<tr>
<td>9800-PT-RTX</td>
<td>PyTorch</td>
<td>1x Intel(R) Core(TM) i7-9800X CPU @ 3.80GHz</td>
<td>1x GeForce RTX 3090</td>
</tr>
<tr>
<td>9800-ORT</td>
<td>ONNX Runtime</td>
<td>1x Intel(R) Core(TM) i7-9800X CPU @ 3.80GHz</td>
<td>None</td>
</tr>
<tr>
<td>9800-PT</td>
<td>PyTorch</td>
<td>1x Intel(R) Core(TM) i7-9800X CPU @ 3.80GHz</td>
<td>None</td>
</tr>
<tr>
<td>7820-ORT-TITAN</td>
<td>ONNX Runtime</td>
<td>1x Intel(R) Core(TM) i7-7820X CPU @ 3.60GHz</td>
<td>1x TITAN V</td>
</tr>
<tr>
<td>7820-PT-TITAN</td>
<td>PyTorch</td>
<td>1x Intel(R) Core(TM) i7-7820X CPU @ 3.60GHz</td>
<td>1x TITAN V</td>
</tr>
</tbody>
</table>
Case Study 2: Various Models and Systems
Case Study 2: Various Models and Systems
Case Study 2: Various Models and Systems
Case Study 2: Various Models and Systems

- With the across stack profiling support from MLModelScope, MLHarness is able to get profiling results from each layer, identify potential bottlenecks, and compare results between models.

The top-3 most time-consuming layers of AlexNet and ResNet18 on system 9800-PT

<table>
<thead>
<tr>
<th>Layer Name</th>
<th>Latency (ms)</th>
<th>Layer Name</th>
<th>Latency (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>aten::mm</td>
<td>47.99</td>
<td>aten::maxpool2d</td>
<td>6.31</td>
</tr>
<tr>
<td>aten::mm</td>
<td>17.99</td>
<td>aten::convolution</td>
<td>2.46</td>
</tr>
<tr>
<td>aten::mm</td>
<td>4.13</td>
<td>aten::convolution</td>
<td>2.24</td>
</tr>
</tbody>
</table>
MLHarness solves the challenges of analyzing, benchmarking and sharing machine learning and deep learning innovations, by extracting advantages from MLCommons Inference and MLModelScope.

The next step for MLHarness is to extend its usage from inference to training, such as making it compatible with MLCommons Training.
MLHarness is Open Source

- You can try MLHarness following the instructions in the QR code or in the url https://github.com/c3sr/mlharness.
- Contributions are welcome.
Thanks ...