MLModelScope: Evaluate and Measure Machine Learning Models within AI Pipelines

Extended Abstract

Abdul Dakkak
Department of Computer Science
University of Illinois, Urbana-Champaign
dakkak@illinois.edu

Cheng Li
Department of Computer Science
University of Illinois, Urbana-Champaign
cli99@illinois.edu

Jinjun Xiong
IBM Thomas J. Watson Research Center
Yorktown Heights, NY
jinjun@us.ibm.com

Wen-mei Hwu
Department of Electrical and Computer Engineering
University of Illinois, Urbana-Champaign
w-hwu@illinois.edu

ABSTRACT

The current landscape of Machine Learning (ML) and Deep Learning (DL) is rife with non-uniform frameworks, models, and system stacks but lacks standard tools to facilitate the evaluation and measurement of model. Due to the absence of such tools, the current practice for evaluating and comparing the benefits of proposed AI innovations (be it hardware or software) on end-to-end AI pipelines is both arduous and error prone — stifling the adoption of the innovations.

We propose MLModelScope— a hardware/software agnostic platform to facilitate the evaluation, measurement, and introspection of ML models within AI pipelines. MLModelScope aids application developers in discovering and experimenting with models, data scientists developers in replicating and evaluating for publishing models, and system architects in understanding the performance of AI workloads.

1 INTRODUCTION

Machine Learning (ML) and Deep Learning (DL) models are being introduced at a faster pace than researchers are able to analyze and study them. Application builders — who may have limited ML knowledge — struggle to discover and experiment with state-of-the-art models within their application pipelines. Data scientists find it difficult to reproduce, reuse, or gather unbiased comparison between published models. And, finally, system developers often fail to keep up with current trends, and lag behind in measuring and optimizing frameworks, libraries, and hardware.

We propose MLModelScope, an open source[1], extensible, and customizable platform to facilitate evaluation and measurements of ML models within AI pipelines. It is a batteries-included platform for evaluating and profiling ML models across datasets, frameworks, and systems. These evaluations can be used to assess model accuracy and performance across different stacks. We provide an online hub of continuously updated assets, evaluation results, and access to hardware resources — allowing users to test and evaluate models without installing or configuring systems. It is framework and hardware agnostic — with current support for Caffe [6], Caffe2 [5], CNTK [9], MXNet [4], Tensorflow [2], and TensorRT [11] running on ARM, PowerPC, and X86 with CPU, GPU, and FPGA. MLModelScope can be used as an application with a web, command line, or API interface or can be compiled into a standalone library.

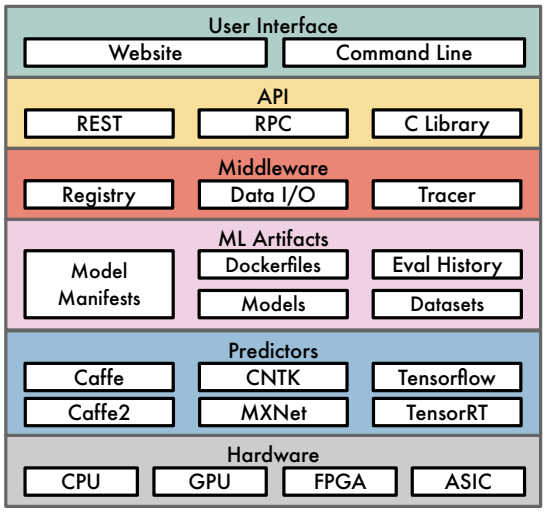

Figure 1: MLModelScope is built from a set of reusable components that allow users to extend and customize.

More specifically, MLModelScope:

- Requires no familiarity with the framework APIs, instead provides a common abstractions that allows programmers to use models.
- No coding is needed to publish models, and enables testing of custom software and hardware stacks.
- Lowers the cost and effort for performing model analysis and evaluation, making it easier for others to reproduce, evaluate, and analyze the model author's claims.
- Makes it simple for system designers to profile and introspect the model and its interaction with the software and hardware stack.

2 MLMODELSCOPE DESIGN

MLModelScope employs a distributed system design (shown in Figure 1) that deploys, evaluates, and measures models across systems to mirror the behavior of real-world AI pipelines. This design decision has the byproduct of allowing users to perform parallel evaluation and measurement of models across systems. The components

[1]The repository is hosted on Github at https://github.com/rai-project/mlmodelscope

can be extended or composed to instantiate customized versions of the platform. Some key components include:

(1) **User Interface and API:** MLModelScope can be used as an application or as a library. Users interact with MLModelScope application through its website, command line, or its API interface. They can also compile MLModelScope as a standalone shared library and use it within their C/C++, Python, or Java projects.
(2) **Tracer:** Tracer captures the stages of the model evaluation, leverages the predictor's framework profiling capability, and interacts with hardware and system level profiling libraries to capture fine grained metrics.
(3) **ML Artifacts:** A model manifest contains all the information needed to reproduce a model's evaluation: the HW/SW stack to instantiate, code to run, model and dataset sources. Models and datasets are downloaded on demand. All artifacts are versioned using a semantic versioning [8] scheme.
(4) **Predictors:** A predictor is a thin abstraction layer that exposes a framework through a common API. The wrapper code interacts directly with the framework's C interface and runs within a container.
(5) **Manifest and Predictor Registry:** MLModelScope uses distributed key-value database to store the registered model manifests and running predictors. MLModelScope leverages the registry to facilitate discovery of models, load balancing request across predictors, and to solve user constraint for selecting the predictor.

3 PERFORMANCE PROFILING OF END-TO-END MODEL INFERENCE

MLModelScope captures different granularities of the end-to-end profile — web API calls, per-layer timing, hardware performance counter and CUDA execution profile. Since we leverage the framework and system profiling libraries, we incur the same overheads introduced by the framework and system profiling tools. Web API calls within an end-to-end model inference include loading the data, deserializing the model, performing the inference, and presenting the results. Per-layer timing is obtained through injecting observers into the framework that MLModelScope supports. When a layer in the framework is launched, the duration a layer takes and other meta data is captured by MLModelScope, shown in Figure 2. MLModelScope integrates with NVIDIA CUDA Profiling Tools Interface (CUPTI) to capture all CUDA events that occur during execution, shown in Figure 3. Hardware performance counters can also be captured by MLModelScope using PAPI, Intel's power counters, Linux Perf, and DTrace. Unlike other tools, MLModelScope does not require a custom framework to perform tracing.

MLModelScope can be used to measure the latency of a model on a system using a user defined dataset. Figure 4 shows an example of framework comparison using AlexNet and ILSVRC2012 validation set on an Amazon P3 machine [7]. The input/output processing time is omitted and the model weights are persistent on the GPU. Other existing capabilities in MLModelScope include model accuracy (Top1, Top5, or the full probabilities), divergence analysis, static analysis and etc.

4 CONCLUSION AND FUTURE WORK

ML systems are in their infancy, with algorithm performance and end-to-end system design considered an art rather than a science.

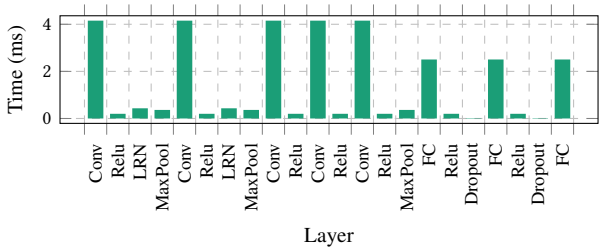

Figure 2: Layer information of Caffe AlexNet on AWS P2.

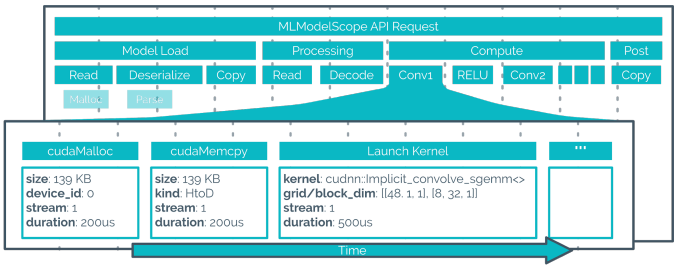

Figure 3: MLModelScope provides fine grained performance tracing for the end-to-end inference pipeline. The traces include events from the GPU, hardware counters, and disk I/O.

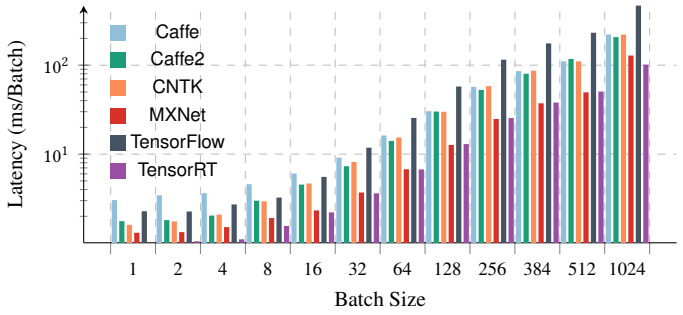

Figure 4: Per-batch inference latency (log scale) of AlexNet across different frameworks and batch sizes on the AWS P3 machine.

A big hurdle in developing systems to optimize ML workflows is understanding the current usage and bottlenecks. MLModelScope offers a unified and holistic way to evaluate and measure ML models within AI pipelines, and gain insights to understand sources of inaccuracy and inefficiency.

We are currently working on using the data captured from MLModelScope to give users suggestions on the model to use for a dataset, perform intelligent scheduling and hardware selection, and manage the models to optimize the inference pipeline. We are currently adding predictors that work within cycle accurate simulators [1, 3, 10]. We believe this would lower the barrier for hardware architects when designing their own custom inference architectures. We are also adding support for specialized ASICs inference hardware, and expanding the number of models supported through our FPGA interface.

REFERENCES

[1] Tor M Aamodt, Wilson WL Fung, I Singh, A El-Shafiey, J Kwa, T Hetherington, A Gubran, A Boktor, T Rogers, A Bakhoda, et al. 2012. GPGPU-Sim 3. x manual. (2012).

[2] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: a system for large-scale machine learning.. In *OSDI*, Vol. 16. 265–283.

[3] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al. 2011. The gem5 simulator. *ACM SIGARCH Computer Architecture News* 39, 2 (2011), 1–7.

[4] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. 2015. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274* (2015).

[5] Yangqing Jia. 2017. Caffe2. https://www.caffe2.ai. (2017).

[6] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*. ACM, 675–678.

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[8] Tom Preston-Werner. 2017. Semantic Versioning 2.0.0. https://www.semver.org. (2017).

[9] Frank Seide and Amit Agarwal. 2016. CNTK: Microsoft's Open-Source Deep-Learning Toolkit. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2135–2135.

[10] Yakun Sophia Shao, Brandon Reagen, Gu-Yeon Wei, and David Brooks. 2014. Aladdin: A pre-rtl, power-performance accelerator simulator enabling large design space exploration of customized architectures. In *Computer Architecture (ISCA), 2014 ACM/IEEE 41st International Symposium on*. IEEE, 97–108.

[11] TensorRT 2018. NVIDIA TensorRT. https://developer.nvidia.com/tensorrt. (2018). Accessed: 2018-09-04.