2019 IEEE World Congress on Services (SERVICES)

MLModelScope: Evaluate and Introspect Cognitive Pipelines

Cheng Li, Abdul Dakkak, Wen-mei Hwu
University of Illinois Urbana-Champaign
Urbana, USA
{cli99,dakkak,w-hwu}@illinois.edu

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

***Abstract*—The current landscape of cognitive pipelines exercise many Machine Learning (ML) and Deep Learning (DL) building blocks. These ML and DL building blocks leverage non-uniform frameworks, models, and system stacks. Currently, there is no end-to-end tool that facilitates ML and DL building blocks evaluation and introspection within cognitive pipelines. Due to the absence of such tools, the current practice for evaluating and comparing the benefits of software or hardware innovations on end-to-end cognitive pipelines is both arduous and error-prone — stifling the adoption rate of ML/DL innovations.**

We propose MLModelScope: a hardware and software agnostic platform to facilitate evaluation and introspection of cognitive pipelines within the cloud or on the edge. We describe the design and implementation of MLModelScope and show how it provides a holistic view of the execution of components within cognitive pipelines. MLModelScope aids application developers in experimenting with and discovering cognitive models, data scientists in comparing and evaluating published algorithms, and system architects in optimizing system stacks for cognitive applications.

***Keywords*-Machine Learning; Deep Learning; Performance Profiling; AI Software;**

I. Introduction

Deep Learning (DL) models and Machine Learning (ML) algorithms are being introduced at a rate faster than researchers are able to analyze and study them [4]. As a result, cognitive application builders struggle to experiment with and integrate state-of-the-art models within their application pipelines. Data scientists find it difficult to evaluate and tweak existing models. And, finally, system developers often fail to keep up with current trends, and lag behind in measuring and optimizing frameworks, libraries, and hardware for cognitive workloads.

This outpacing problem is exacerbated by the complexity introduced by abstractions within cognitive pipelines. Figure 1 shows the execution of an cognitive pipeline at different levels of hardware (HW) and software (SW) abstractions. ❶ A cognitive application pipeline (which may span multiple machines) employs a set of models and frameworks. ❷ Each model defines pre-/post-processing operations for input and output handling. ❸ A framework runs a model by executing the network-layer execution graph. ❹ A layer is executed as a sequence of library calls. ❺ A library call in turn invokes a chain of system runtime functions. All the while the ❻ hardware provides counters that capture the execution of instructions, disk and network activity, and other low-level system events. Because of the many levels of abstractions, the HW/SW stacks must work in unison to maintain accuracy, performance, and efficiency.

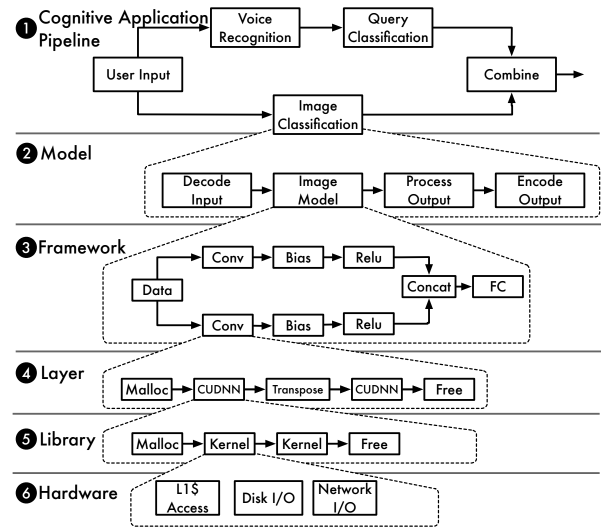

Figure 1. Execution of a cognitive pipeline.

There have been concerted community efforts to evaluate and introspect ML models at different levels of the HW/SW stack [1], [5], [6], [9], [10]. Currently, however, there is no one tool that: ① enables understanding of the proposed models and systems within cognitive pipelines across abstraction levels, ② makes it simple to evaluate, compare, and debug reported results, and ③ allows measuring proposed models and systems within one's own cognitive workflow via a unified API. Having a tool that:

- defines common techniques to specify and provision model evaluation with specified HW/SW stack
- provides a consistent evaluation and reporting methodology
- enables profiling of experiments across abstractions
- performs automated analysis to debug accuracy and performance problems

would accelerate the adoption of state-of-the-art ML research innovations. A tool satisfying these requirements would

978-1-7281-3851-0/19/$31.00 ©2019 IEEE
DOI 10.1109/SERVICES.2019.00093

also allow the steak holders at the different stack levels to communicate with one another, and remove reliance on domain specific tools.

We propose MLModelScope, an open source, extensible, and customizable platform to facilitate evaluation and introspection of ML models within cognitive pipelines. It is designed to aid: ① application developers in experimenting with and discovering existing published models. ② data scientists in comparing algorithms and debugging accuracy and performance problems. ③ system developers in profiling model execution at all levels of the HW/SW stack abstractions to optimize ML frameworks, libraries, and hardware for cognitive workloads.

MLModelScope is framework and hardware agnostic — with current support for Caffe, Caffe2, CNTK, MXNet, PyTorch, TensorFlow, TFLite, and TensorRT, running on ARM, PowerPC, and X86 with CPU, GPU, and FPGA. MLModelScope integrates with both software and hardware profilers, thus allowing users to introspect cognitive pipelines to discern accuracy, performance, and energy information. MLModelScope is extensible and customizable — allowing users to extend it by adding models, frameworks, or library and system profilers. We are actively extending the number of models, frameworks, and profilers that are built-in to MLModelScope. Future work would add debugging and advising capabilities to MLModelScope.

This paper first describes important aspects of MLModelScope's design (Section II), it then details its current capabilities (Section III), before it discusses future work and concludes (Section IV).

II. MLModelScope Design

To mirror the behavior of real-world cognitive pipelines, MLModelScope employs a distributed design that deploys, evaluates, and measures models across systems. The distributed design allows users to perform parallel evaluations on non-local systems. MLModelScope consumes model specifications (referred to as model manifests) and provisions the required environments for running the evaluations. It then automates the collection and comparison of the experimentation results across runs, and writes them to a local file or posts them to a database. Further debugging and advising are performed using the data collected. Figure 2 shows the key components, which can be composed or extended to instantiate customized versions of MLModelScope. We highlight some of the key components in this section and a more detailed description is found in [3].

Model Manifest — MLModelScope specifies a model evaluation procedure via a model manifest and user-provided hardware architecture and configuration parameters. The manifest contains the SW stack along with other information needed for model evaluation, such as inputs and outputs pre/post-processing and model/dataset resources. The hardware details are parameters when performing the evaluation.

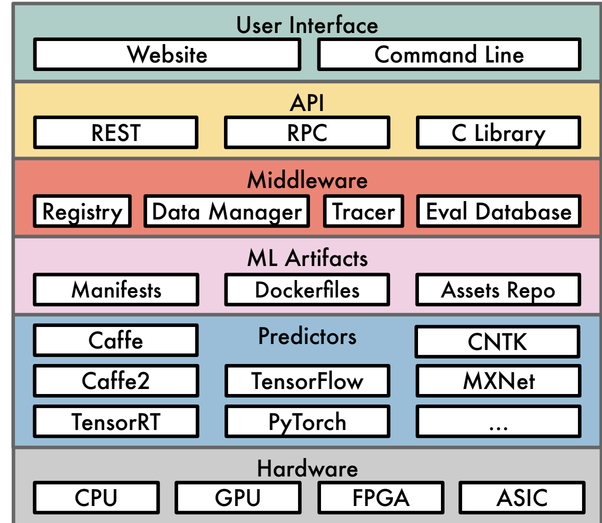

Figure 2. MLModelScope is built from a set of reusable components that allow users to extend and customize.

All artifacts (e.g. model manifest, framework, dataset) are versioned. The manifest is specified in YAML format, and its motivation is described in [8].

Framework Predictors — A MLModelScope worker (called predictor) is a lightweight process that runs on a system and is responsible for evaluating models using the specified manifests and capturing the system's profile information. Predictors provide a thin abstraction layer that exposes frameworks through a common API, and publish their HW/SW stack information to MLModelScope's central server at startup. The predictors constantly listen for jobs. Once a job is accepted the predictor is responsible for setting up the hardware environment, launching the container, running the evaluation, and publishing the results. Multiple instances of a predictor can be run across a cluster of nodes, and all predictors are managed by the central server.

Middleware — MLModelScope's middleware layer is composed of services and utilities for orchestrating, provisioning, aggregating, and monitoring the execution of the predictors. The middleware acts as a conduit between the user-facing API and the internals of the system. Within the middleware layer are the registry which is a distributed key-value database to store the registered model manifests and the running predictors' information. The data manager is responsible for downloading the requested assets (models and datasets) from either web URLs or from MLModelScope's assets repository. The evaluation database stores the experiment results using the constraints of the evaluations as the keys. The results are then summarized and visualized to aid in comparing and debugging accuracy or performance.

The tracer captures the stages of a model evaluation, leverages the predictor's framework profiling capability, and interacts with system and hardware level profiling libraries to capture fine-grained system events. MLModelScope publishes

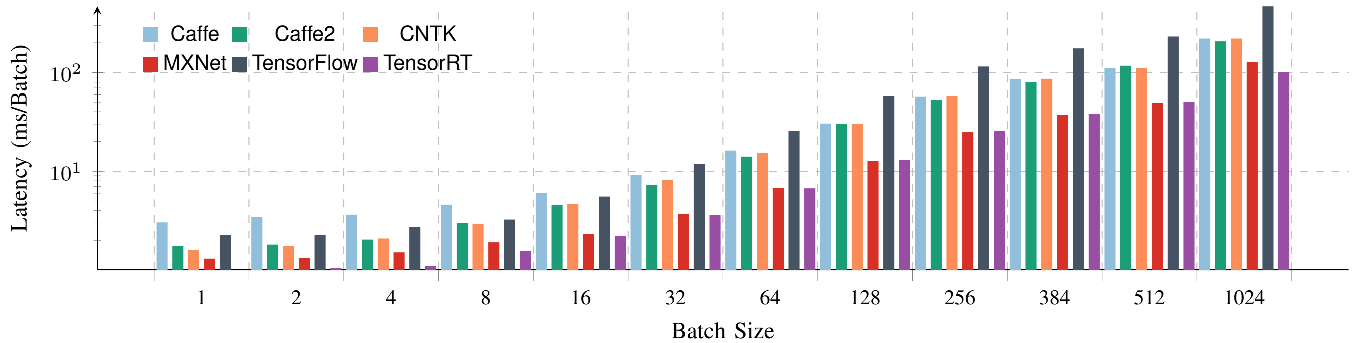

Figure 3. Per-batch latency of AlexNet on the AWS P3 system using different frameworks.

the tracing results asynchronously to a distributed server [12], [13] — allowing users to view a single end-to-end timeline containing all the profiles. Users can view the timeline in the web UI and can "zoom" into specific component (shown in Figure 1) and traverse the profile at different abstraction levels. To reduce tracing overhead, users control the granularity (application, model, framework, library, or hardware) of the profiles captured. Since MLModelScope leverages standard tools to enable whole cognitive application tracing, user can integrate MLModelScope with their exisiting cognitive pipelines.

User Interface and API — MLModelScope can be used as an application or as a library. When used as an application, a user interacts with MLModelScope through its website, command line, or API interface. The website and command line interface allow users to evaluate and profile models without familiarity with the underlying frameworks or profiling tools. Users who wish to integrate MLModelScope within their existing tools or pipelines can use the REST or RPC API. MLModelScope can also be compiled to a standalone library and used within users' C/C++, Python, or Java applications.

III. MLModelScope Current Capabilities

We implemented the MLModelScope design with support for common frameworks and software/hardware stacks, and populated it with over 300 models and the corresponding datasets. To minimize overhead due to scripting languages, MLModelScope is developed using a statically compiled programming language and directly binds to the frameworks' C-level API. It uses Protocol Buffers and gRPC for data exchange to minimize data serializion overhead. This section highlights a few of the current capabilities. For space reasons we omit less important ones, such as model accuracy divergence analysis, complexity analysis, resource utilization prediction, and model/framework/system suggestion.

Public Portal — We maintain a demonstartion version of MLModelScope on a representative set of systems along with the evaluation results of the built-in artifacts. It serves as a portal for the public to evaluate and measure the systems, and to refer to MLModelScope's artifacts. Users can share and publish their evaluation results as well as correlate against historical accuracy and performance information.

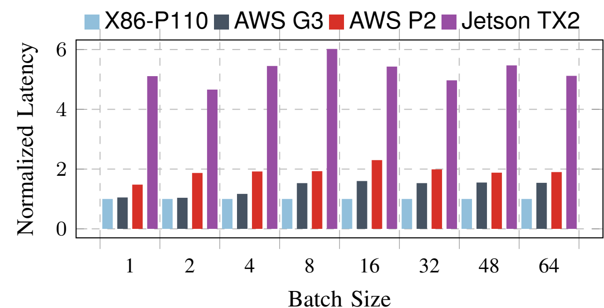

Figure 4. AlexNet using TensorRT evaluated across the systems shown in Table I. Results are normalized to X86-P110's latency.

Table I
THE 6 SYSTEMS WERE CHOSEN TO COVER CURRENT GPU ARCHITECTURES AND CLOUD OFFERINGS.

Name	CPU	GPU
AWS G3	Intel Xeon E5 2686 v4	Tesla M60
AWS P2	Intel Xeon E5 2686 v4	Tesla K80
X86-P110	Intel Core i9-7900X	TITAN Xp P110
AWS P3	Intel Xeon E5-2698 v4	Tesla V100-PCIE
IBM P8	IBM S822LC Power8	Tesla P100-SXM2
Jetson TX2	ARM Cortex-A57	256-core Pascal

Evaluation of Machine Learning Artifacts — MLModelScope can be used to measure the latency of a model across frameworks on a system using a user-defined dataset. MLModelScope can also be used to evaluate frameworks and models across systems. The process is simple in MLModelScope, since multiple predictors can be launched across systems and profiled in parallel. Figure 3 shows an example of framework comparison using AlexNet and ILSVRC2012 ([7]) validation set on an Amazon EC2 P3 machine. The input/output processing time is omitted — since all frameworks share common code — and the model weights are persistent on the GPU. Figure 4 compares the normalized latency of TensorRT across the systems in Table I as we vary

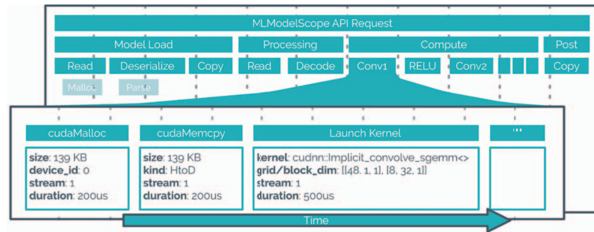

Figure 5. MLModelScope provides fine-grained performance tracing for the end-to-end inference pipeline. Users are presented with a hierarchical profile timeline which they can use to introspect and examine the layers executed, the functions ran within layers, and the hardware resources used within functions.

the batch size. The latency is normalized to the inference latency on the X86-P110 system.

Deep 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. 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 frameworks that MLModelScope supports. MLModelScope integrates with NVIDIA CUDA profiling tools (CUPTI [2] and NVML [11]) to capture all CUDA events that occur during execution. Hardware performance counters can also be captured by MLModelScope using PAPI, Intel's power counters, Linux Perf, and DTrace. MLModelScope incurs the same overhead introduced by the framework and system profiling tools and does not require modifying the framework to perform tracing. Figure 5 shows how we can use MLModelScope as a window by leveraging its sub-model and sub-layer latency analysis — to understand the steps performed per layer, examine the choice of cuDNN functions and algorithms made by the framework, and introspect the evaluation.

IV. Future Work and Conclusion

We are actively adding more frameworks, models, and profiling capabilities to MLModelScope, as well as adding support for specialized ASICs inference hardware. We are currently using the traces captured to suggest the ideal model/framework/system combination for a user-provided dataset, perform intelligent scheduling and hardware selection, and optimize cognitive pipelines. Future work will use the trace information to offer more types of recommendations for cognitive workloads.

Cognitive systems are in their infancy, with algorithm performance and end-to-end cognitive systems design considered an art rather than a science. A big hurdle in cognitive systems is the lack of tools to facilitate understanding of resource usage and identifying bottlenecks of cognitive innovations within end-to-end workflows. MLModelScope is a unified and holistic tool to evaluate and introspect ML models within cognitive pipelines, and gain insights to understand sources of inaccuracy and inefficiency. We therefore believe MLModelScope is an ideal platform for performing cognitive research and help facilitates rapid adoption of ML/DL innovations.

References

[1] T. Ben-Nun, M. Besta, S. Huber, A. N. Ziogas, D. Peter, and T. Hoefler. A Modular Benchmarking Infrastructure for High-Performance and Reproducible Deep Learning. IEEE, May 2019. The 33rd IEEE International Parallel & Distributed Processing Symposium (IPDPS'19).

[2] The CUDA Profiling Tools Interface. https://developer.nvidia.com/cuda-profiling-tools-interface. Accessed: 2019-05-04.

[3] Abdul Dakkak, Cheng Li, Abhishek Srivastava, Jinjun Xiong, and Wen-Mei Hwu. MLModelScope: Evaluate and Measure ML Models within AI Pipelines. *arXiv preprint arXiv:1811.09737*, 2018.

[4] Jeff Dean, David Patterson, and Cliff Young. A new golden age in computer architecture: Empowering the machine-learning revolution. *IEEE Micro*, 38(2):21–29, 2018.

[5] Fursin, Grigori and Lokhmotov, Anton and Savenko, Dmitry and Upton, Eben. A Collective Knowledge workflow for collaborative research into multi-objective autotuning and machine learning techniques. January 2018.

[6] Sindhu Ghanta, Lior Khermosh, Sriram Subramanian, Vinay Sridhar, Swaminathan Sundararaman, Dulcardo Arteaga, Qianmei Luo, Drew Roselli, Dhananjoy Das, and Nisha Talagala. A systems perspective to reproducibility in production machine learning domain. 2018.

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

[8] Cheng Li, Abdul Dakkak, Jinjun Xiong, and Wen mei Hwu. Challenges and Pitfalls of Reproducing Machine Learning Artifacts. *CoRR*, abs/1904.12437, 2019.

[9] MLPerf. https://mlperf.org, 2018. Accessed: 2019-05-04.

[10] Jon Ander Novella, Payam Emami Khoonsari, Stephanie Herman, Daniel Whitenack, Marco Capuccini, Joachim Burman, Kim Kultima, and Ola Spjuth. Container-based bioinformatics with pachyderm. *bioRxiv*, page 299032, 2018.

[11] NVIDIA Management Library. https://developer.nvidia.com/nvidia-management-library-nvml. Accessed: 2019-05-04.

[12] OpenTracing: Cloud native computing foundation. http://opentracing.io, 2018. Accessed: 2019-05-04.

[13] Benjamin H Sigelman, Luiz Andre Barroso, Mike Burrows, Pat Stephenson, Manoj Plakal, Donald Beaver, Saul Jaspan, and Chandan Shanbhag. Dapper, a large-scale distributed systems tracing infrastructure. Technical report, Technical report, Google, Inc, 2010.