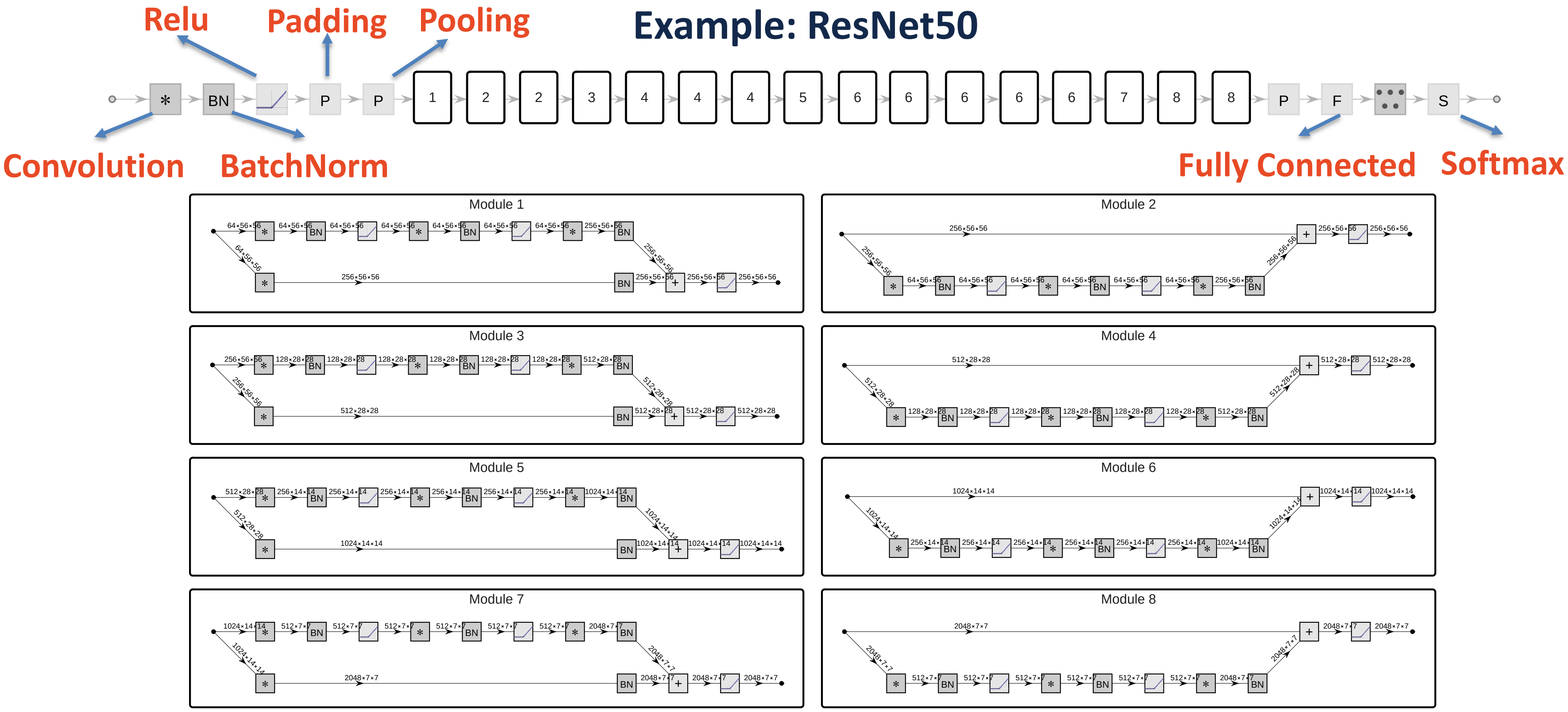
The Design and Implementation of a Scalable DL Benchmarking Platform

Cheng Li[1*], Abdul Dakkak[1*], Jinjun Xiong[2], Wen-mei Hwu[1]
University of Illinois Urbana-Champaign[1], IBM Research[2]

10/14/2020

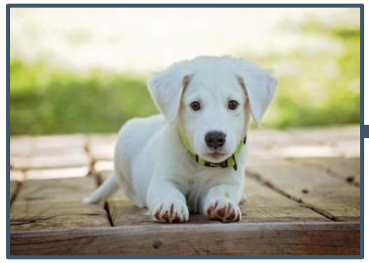
Deep Learning (DL) Model

A graph where each vertex is a layer (or operator) and an edge represents data transfer



DL Inference Pipeline

Input Image



Pre-processing

- Image decoding
- Resizing
- Normalization
- Type conversion

Input Tensor

Prediction

Model prediction using framework API

Output Tensor

Post-processing

Unpacking into pairs (label, probabilities) and sorting

Top1

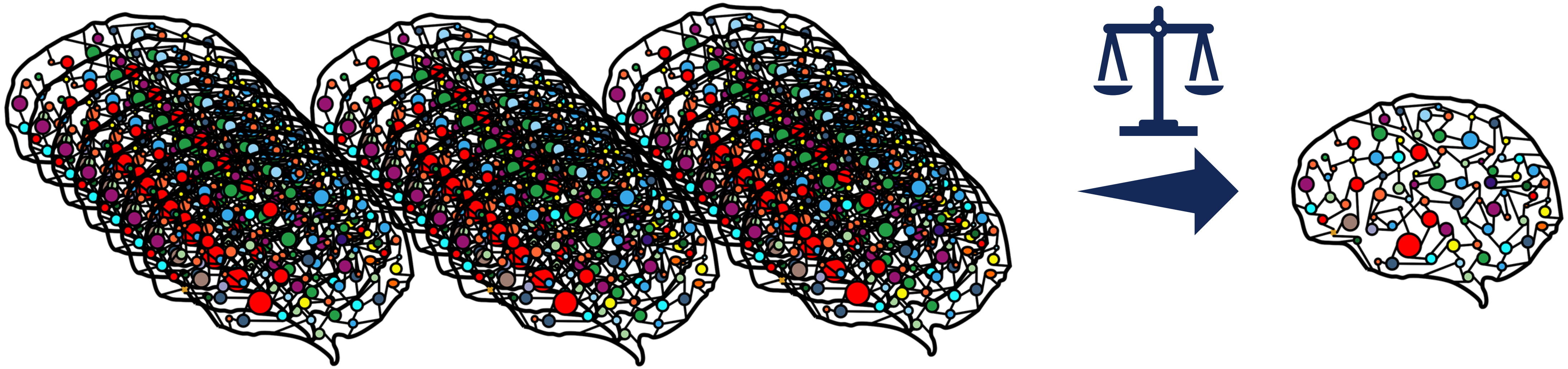
(dog, 0.99)

Motivation

- DL models are used in many application domains
- Diverse DL models, as well as hardware/software (HW/SW) solutions, are increasingly being proposed
- However, evaluating and comparing DL innovations is arduous and error-prone due to lack of standard
- There is an urging need for a DL benchmarking platform that consistently evaluates and compares different DL models across HW/SW stacks, while coping with the fast-paced and diverse DL landscape

MLModelScope

- A DL benchmarking platform aiming to facilitate evaluation and comparison of DL innovations



- 10 objectives inform the design

Desired Features for a DL benchmarking platform

1. Reproducible Evaluation
2. Consistent Evaluation
3. Framework & Hardware Agnostic
4. Scalable Evaluation
5. Artifact Versioning
6. Efficient Evaluation Workflow
7. Different Benchmarking Scenarios
8. Benchmarking Analysis and Reporting
9. Model Execution Inspection
10. UIs for different use cases

1. Reproducible Evaluation

- Model, dataset, evaluation method, and HW/SW stack must work in unison to maintain the accuracy and performance claims
- Reproducibility is currently a "pain-point" within the DL community
  - Lack of standard specification
- All aspects of a model evaluation must be specified and provisioned by the design

2. Consistent Evaluation

- Models are published in an ad-hoc manner
  - A tight coupling between model execution and the underlying HW/SW
  - Difficult to quantify or isolate the benefits of an individual component
- Fair comparisons require a consistent evaluation methodology rather than running ad-hoc scripts

3. Framework & Hardware Agnostic

- Many choices of frameworks and hardware for DL models
- Each framework or hardware has its own use scenarios, features, and performance characteristics
- The design must support different frameworks and hardware, and does not require modifications to the frameworks

4. Scalable Evaluation

- DL innovations are introduced at a rapid pace
- Performing DL evaluations with different model/HW/SW setups in parallel
- A centralized management of the benchmarking results
- E.g., choosing the best hardware out of N candidates for a model is ideally performed in parallel and the results should be automatically gathered for comparison

5. Artifact Versioning

- DL frameworks are continuously updated by the DL community
- Many unofficial variants of models, frameworks, and datasets as researchers might update or modify them to suite their needs
- To enable management and comparison of model evaluations , evaluation artifacts (models, frameworks, and datasets) should be versioned

6. Efficient Evaluation Workflow

- The data loading and pre-/post-processing can take a non-negligible amount of time, and become a limiting factor for quick evaluations
- The evaluation workflow should handle and process data efficiently

7. Different Benchmarking Scenarios

- DL benchmarking is performed under specific scenario
  - Online, offline, or interactive applications on mobile, edge, or cloud systems
- The design should support common inference scenarios and be flexible to support custom or emerging workloads as well

8. Benchmarking Analysis and Reporting

- Benchmarking produces raw data which needs to be correlated and analyzed to produce human-readable results
- An automated mechanism to summarize and visualize these results within a benchmarking platform can help users quickly understand and compare the results

9. Model Execution Inspection

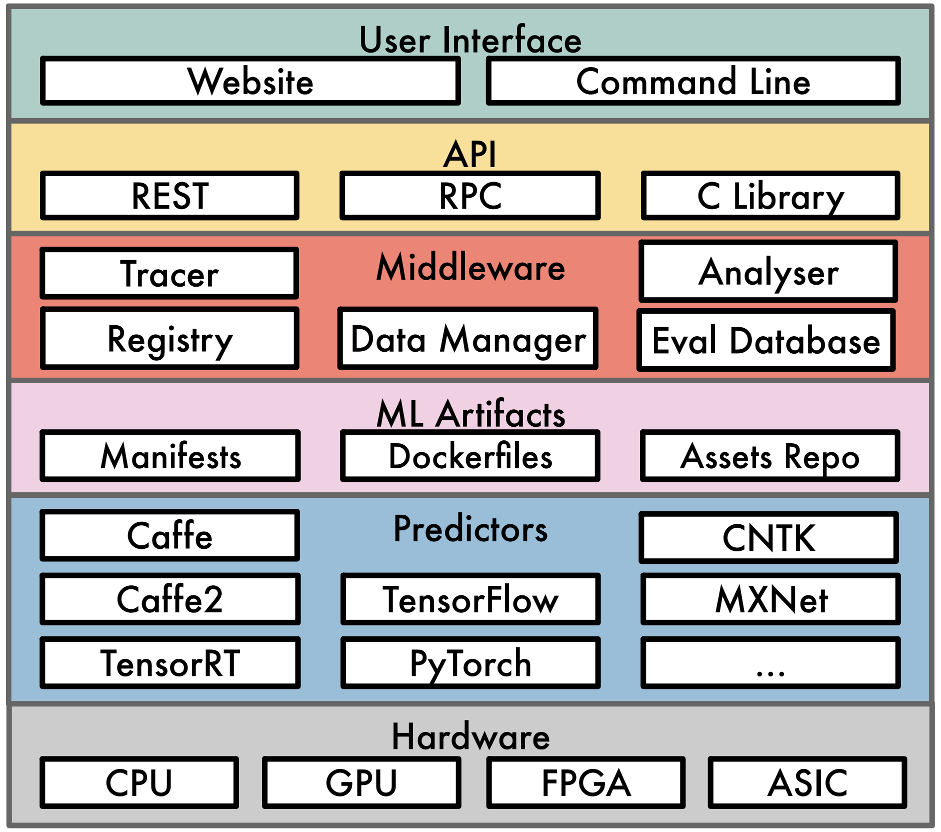
- The complexity of DL model evaluation makes performance debugging challenging
  - each level within the HW/SW abstraction hierarchy can be a suspect when things go awry
- To ease inspecting model execution bottlenecks, the design should provide tracing capability at all levels of HW/SW stack
  - Integration with XSP

10. Different User Interfaces

- Command-line interface is often used in scripts to quickly perform combinational evaluations across models, frameworks, and systems
- Web UI serves as a “push- button” solution to benchmarking and provides an intuitive flow for specifying, managing evaluations, and visualizing benchmarking results

MLModelScope Design

- A DL artifact exchange specification to describe DL inference from model, data, software and hardware aspects
- A distributed runtime that consumes the DL specification
  – Web and command line UI
  – Middleware, e.g. registry, database, tracer
  – Framework agents
  – Other modular components



MLModelScope Manifest

- Specifies the HW/SW stack to instantiate and how to evaluate the model
  - Container Images
  - Inputs and Outputs and Pre-/Post-Processing
  - Model Sources
  - Asset Versioning

```yaml
name: MLPerf_ResNet50_v1.5 # model name
version: 1.0.0 # semantic version of the model
description: ...
framework: # framework information
  name: TensorFlow
  version: '>=1.12.0 <2.0' # framework ver constraint
inputs: # model inputs
  - type: image  # first input modality
    layer_name: 'input_tensor'
    element_type: float32
    steps: # pre-processing steps
      - decode:
          data_layout: NHWC
          color_mode: RGB
      - resize:
          dimensions: [3, 224, 224]
          method: bilinear
          keep_aspect_ratio: true
      - normalize:
          mean: [123.68, 116.78, 103.94]
          rescale: 1.0
outputs: # model outputs
  - type: probability # first output modality
    layer_name: prob
    element_type: float32
    steps: # post-processing steps
      - argsort:
          labels_url: https://.../synset.txt
preprocess: [[code]]
postprocess: [[code]]
model: # model sources
  base_url: https://zenodo.org/record/2535873/files/
  graph_path: resnet50_v1.pb
  checksum: 7b94a2da05d...23a46bc08886
attributes: # extra model attributes
  training_dataset:  # dataset used for training
    - name: ImageNet
    - version: 1.0.0
```

Example model manifest

MLModelScope Runtime

User Inputs – the required inputs for model evaluation

Client - the web UI or command-line interface that sends REST requests to the Sever

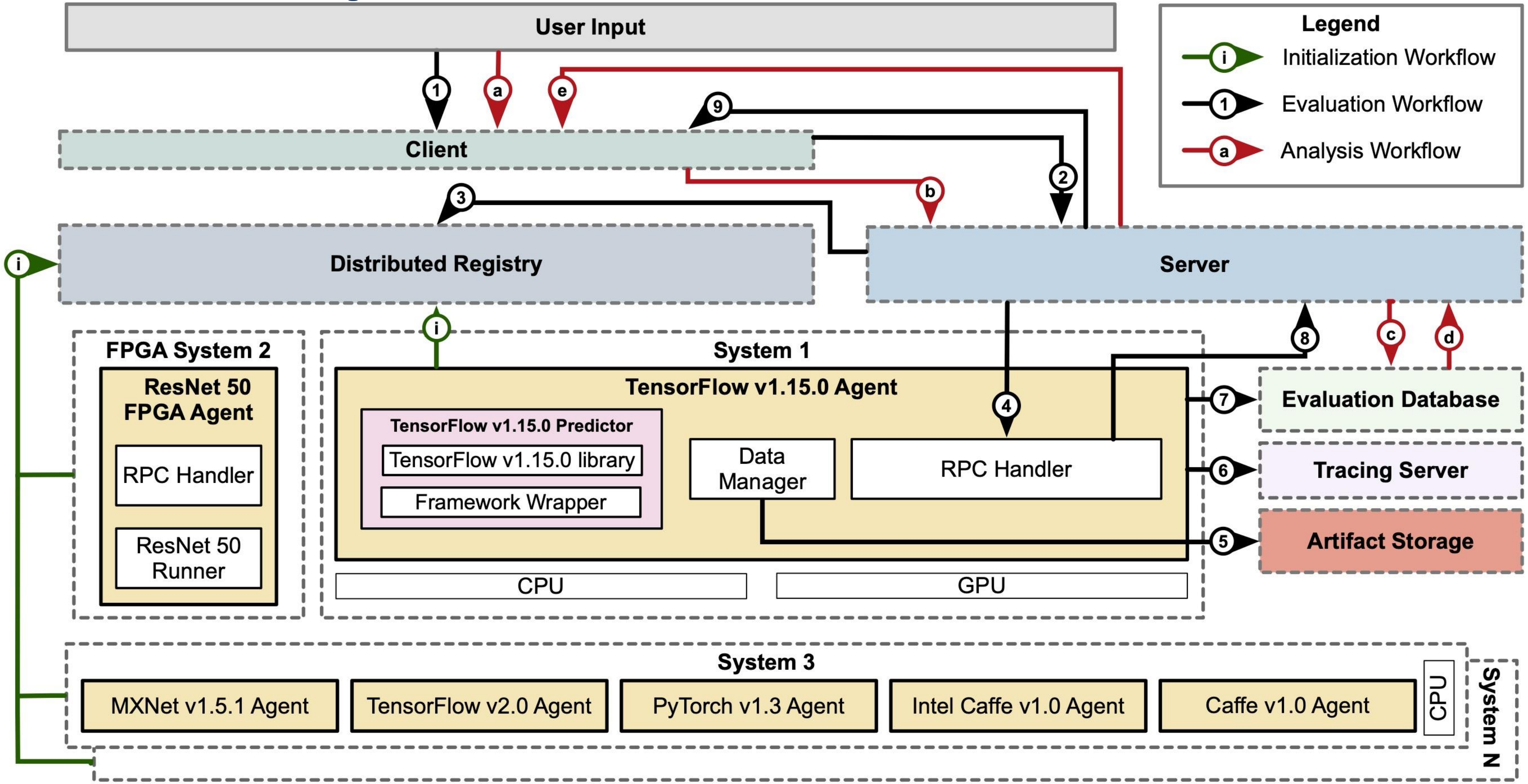
Server - acts on the client requests and performs REST API handling, dispatching the model evaluation tasks to the Agents

Agents - runs on different systems of interest and perform model evaluation based on requests sent by the server

Framework Predictor – resides in an Agent and wraps around a framework into a consistent interface across different DL frameworks

Middleware - a set of support services

MLModelScope Runtime and Workflows



```
service Predict {
message PredictOptions {
  enum TraceLevel {
    NONE      = 0;
    MODEL     = 1;  // steps in the evaluation pipeline
    FRAMEWORK = 2;  // layers within the framework and above
    SYSTEM    = 3;  // the system profilers and above
    FULL      = 4;  // includes all of the above
  }
  TraceLevel trace_level = 1;
  Options    options     = 2;
}
message OpenRequest {
  string model_name                          = 1;
  string model_version                       = 2;
  string framework_name                      = 3;
  string framework_version                   = 4;
  string model_manifest                      = 5;
  BenchmarkScenario benchmark_scenario = 6;
  PredictOptions    predict_options    = 7;
}
// Opens a predictor and returns a PredictorHandle.
rpc Open(OpenRequest) returns (PredictorHandle){}
// Close a predictor and clear its memory.
rpc Close(PredictorHandle) returns (CloseResponse) {}
// Predict receives a stream of user data and runs
// the predictor on each element of the data according
// to the provided benchmark scenario.
rpc Predict(PredictorHandlePredictorHandle, UserInput) ↩
     returns (FeaturesResponse) {}
}
```

Listing 4. MLModelScope's minimal gRPC interface in protocol buffer format.

Current Support

- Different framework backends
  - TensorFlow, PyTorch, Caffe2, MXNet, Caffe, CNTK, and TensorRT
- Different hardware support
  - ARM, PowerPC, and X86 with CPU, GPU, and FPGA
- Common ML models (>300) and datasets
- Integration with XSP
  - Built-in framework, library, and hardware profilers
- Allows users to add models, frameworks, or profilers

Evaluation

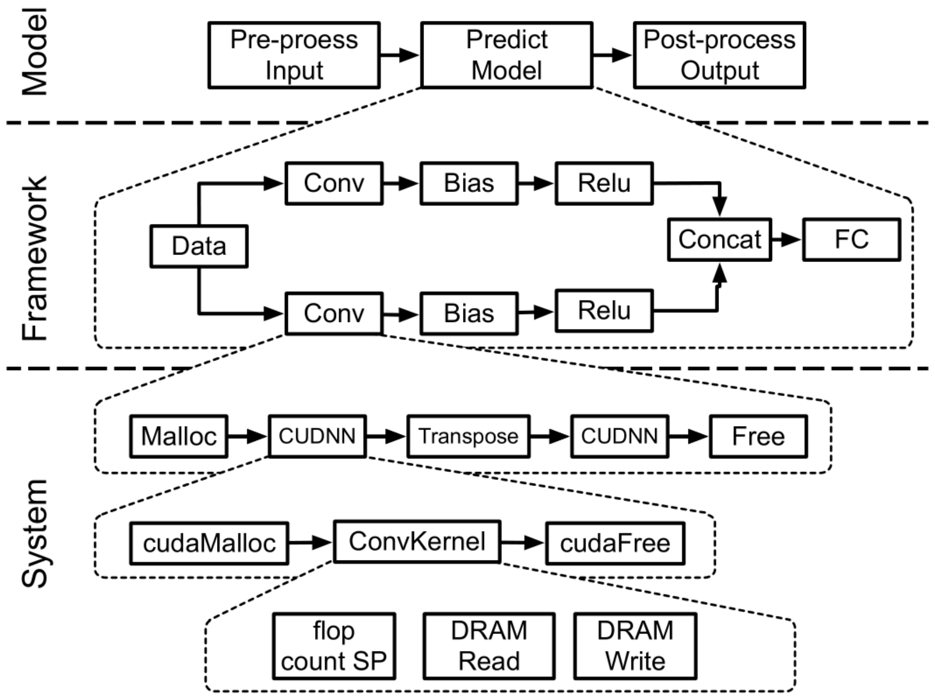
- We demonstrated MLModelScope by using it to evaluate a set of models on 4 representative systems and show how model, hardware, and framework selection affects model accuracy and performance under different bench marking scenarios

Name	CPU	GPU	GPU Architecture	GPU Theoretical Flops (TFlops)	GPU Memory Bandwidth (GB/s)	Cost ($/hr)
AWS P3 (2XLarge)	Intel Xeon E5-2686 v4 @ 2.30GHz	Tesla V100-SXM2-16GB	Volta	15.7	900	3.06
AWS G3 (XLarge)	Intel Xeon E5-2686 v4 @ 2.30GHz	Tesla M60	Maxwell	9.6	320	0.90
AWS P2 (XLarge)	Intel Xeon E5-2686 v4 @ 2.30GHz	Tesla K80	Kepler	5.6	480	0.75
IBM P8	IBM S822LC Power8 @ 3.5GHz	Tesla P100-SXM2	Pascal	10.6	732	-

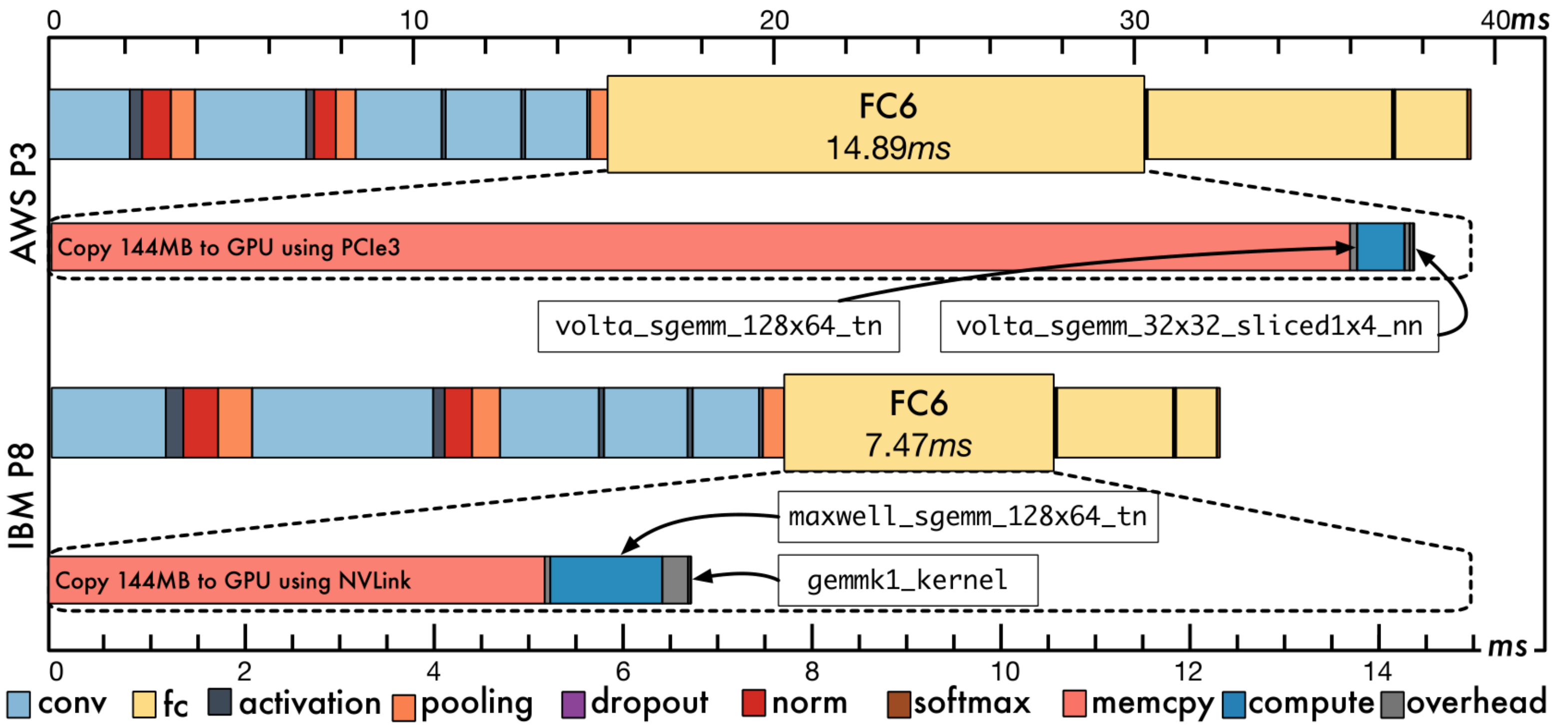
Table 1. Four systems with Volta, Pascal, Maxwell, and Kepler GPUs are selected for evaluation.

Model Execution Introspection

- The inspection capability helps users understand the model execution and identify performance bottlenecks



A hierarchical view of model execution



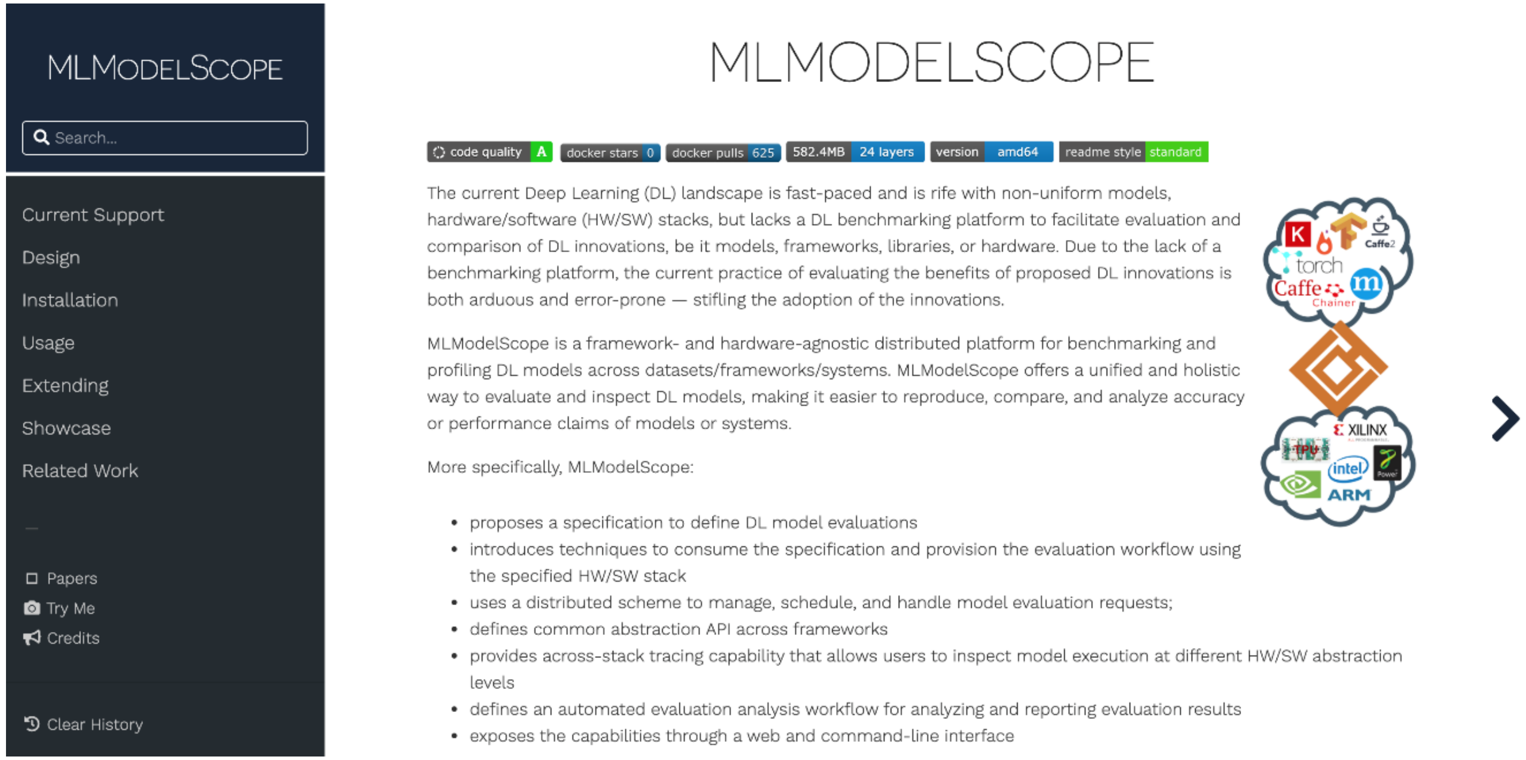
Example: AlexNet "cold-start" inference

Conclusion

- A big hurdle in adopting DL innovations is to evaluate, analyze, and compare their performance
- We identified 10 desired features of a DL benchmarking platform and described MLModelScope that achieves these design objectives
- MLModelScope offers a unified and holistic way to evaluate and inspect DL models, and provides an automated analysis and reporting workflow to summarize the results

Resources

- docs.mlmodelscope.org
- github.com/rai-project



Thank you

Cheng Li[1*], Abdul Dakkak[1*], Jinjun Xiong[2], Wen-mei Hwu[1]
University of Illinois Urbana-Champaign[1], IBM Research[2]