DLSpec: A Deep Learning Task Exchange Specification

Abdul Dakkak\textsuperscript{1*}, Cheng Li\textsuperscript{1*}, Jinjun Xiong\textsuperscript{2}, Wen-mei Hwu\textsuperscript{1}
University of Illinois Urbana-Champaign\textsuperscript{1}, IBM Research\textsuperscript{2}

OpML’2020
Background

- Deep Learning (DL) innovations are introduced at a fast pace
- Current lack of standard specification of DL tasks makes sharing, running, reproducing, and comparing DL innovations difficult
Current Practice of Publishing DL Artifacts

- Ad-hoc scripts and textual documentation to describe the execution process of *DL tasks*
  - Curation of DL tasks in framework model zoo
  - Model catalogs that can be used through a cloud provider’s API
- Hard to reproduce the reported accuracy or performance results and have a consistent comparison across DL artifacts
DLSpec Objectives

- A DL artifact exchange specification with clearly defined model, data, software, and hardware aspects
  - Model-, dataset-, software-, and hardware agnostic
  - Works with runtimes built using existing MLOp tools
- We developed a DLSpec runtime for DL inference tasks in the context of benchmarking
DLSpec is Based on a Few Key Principles

- Reproducible
- Minimal
  - Only contains essential information to increase the transparency and ease the creation
- Program-/human-readable
  - Executed by a runtime/easy to introspect and repurpose
- Maximum expressiveness
  - Describes both training and inference
DLSpec is Based on a Few Key Principles

- Decoupling DL task description
  - Increases the reuse/portability and enables easy of comparison

- Splitting the DL task pipeline stages
  - Enables consistent comparison and simplifies accuracy and performance debugging

- Avoiding serializing intermediate data into files
  - Avoids high serializing/deserializing overhead
  - Supports DL tasks that use streaming data
DLSpec Design

**Hardware**
id: uuid
cpu:
- arch: x86-64
- min_ncpu: 4
- max_ncpu: 16
- 
gpu:
- arch: nvidia/sm70
- min_memory: 2gb
- 
- cude_version: 10.2+
- 
- interconnect: nvlink2
memory:
- min: 16
- 
- setup: |

```
echo 1 > /sys/devices/system/cpu/intel_pstate/no_turbo
```

**Software**
id: uuid
name: Tensorflow framework name
version: 1.0.0 semantic version
container: dlspec/tf:2-1-0_amd64-gpu
env:
- TF_ENABLE_WINOGRAD_NONFUSED: 0

**Dataset**
id: uuid
name: ILSVRC 2012
version: 1.0.0 semantic version
license: ...

```
sources:
- source: s3://.../test_set.zip
  name: test_set
- source: ...
```

**Model**
job_type: inference or training
run: |
def run(ctx, data):
  ...
  # tf.Session.run(ctx["model"], data)
  return run_output
model: # model for retraining or inference

```
graph_path: https://.../inception_v3.pb
checksum: XXXX
```
post-process: |
def post_processing(ctx, data):
  ...
  # e.g. import numpy as np
  return post_processed_data
outputs: # model outputs

```
type: probability # 1st output modality
layer_name: prob_element_type: float32
```

```
type: image # 1st input modality
layer_name: data element_type: float32
```

DLSpec Runtime
Hardware Manifest

- Defines the hardware requirements for a DL task
- Some hardware settings cannot be specified within a container (E.g. the runtime set Intel’s turbo-boosting outside the container)

```
Hardware

id: uuid
cpu:
  - arch: x86-64
  - ncpu: 4
  - ...
gpu:
  - arch: nvidia/sm70
  - memory: 16gb
  - driver_version: XXX
  - ...
interconnect: nvlink2
memory: 32gb
...
setup: |
  echo 1 > /sys/devices/system/cpu/intel_pstate/no_turbo
```
Software Manifest

- Defines the software environment for a DL task
- All executions occur within the specified container
- Specified environment variables are setup after running the container

```yaml
id: uuid
name: Tensorflow # framework name
version: 1.0.0 # semantic version
container: dlspec/tf:2-1-0_amd64-gpu
env:
  - TF_ENABLE_WINOGRAD_NONFUSED: 0
```
Dataset Manifest

- Defines the training, validation, or test dataset
- The source location defines where to download the dataset from

```yaml
Dataset
id: uuid
name: ILSVRC 2012
version: 1.0.0 # semantic version
license: ... # dataset license

sources:
- source: s3://.../test_set.zip
  name: test_set
- source: ...
```
Model Manifest

- Defines the logic to run a DL task and the required artifact sources

```python
id: uuid # model unique id
name: Inception-v3 # model name
version: 1.0.0 # semantic version
license: MIT # model license
author: Jane Doe # model author
task: image classification
description: ...

pre-process:
    def pre_processing(ctx, data):
        from PIL import Image
        img = Image.open(data["test_set"][0])
        img = np.asarray(img)
        img = np.transpose(img, (2,0,1))
        ...
        return pre_processed_data

inputs: # model inputs
- type: image # 1st input modality
  layer_name: data
  element_type: float32

Model

job_type: inference # or training
run:
    def run(ctx, data):
        ...
        # tf.Session.run(ctx["model"], data)
        return run_output

model: # model for retraining or inference
graph_path: https://.../inception_v3.pb
checksum: XXXX...XXX

post-process:
    def post_processing(ctx, data):
        ...
        # e.g. import numpy as np
        return post_processed_data

outputs: # model outputs
- type: probability # 1st output modality
  layer_name: prob
  element_type: float32
  system_requirements: [gpu]
```

Python functions, executed by the runtime through the Python sub-interpreter.
Model Manifest

- Defines the logic to run a DL task and the required artifact sources

```
id: uuid # model unique id
name: Inception-v3 # model name
version: 1.0.0 # semantic version
license: MIT # model license
author: Jane Doe # model author
task: image classification
description: ...
pre-process: |
  def pre-processing(ctx, data):
    from PIL import Image
    img = Image.open(data[“test_set”][0])
    img = np.array(img)
    img = np.transpose(img, (2, 0, 1))
    return pre_processed_data

inputs: # model inputs
  - type: image # 1st input modality
    layer_name: data
    element_type: float32

outputs: # model outputs
  - type: probability # 1st output modality
    layer_name: prob
    element_type: float32

job_type: inference # or training
run: |
  def run(ctx, data):
    ... # tf.Session.run(ctx[“model”], data)
    return run_output
model: # model for retraining or inference
  graph_path: https://.../inception_v3.pb
  checksum: XXXX...XXXX
post-process: |
  def post_processing(ctx, data):
    ... # e.g. import numpy as np
    return post_processed_data

system_requirements: [gpu]```
Model Manifest

- Defines the logic to run a DL task and the required artifact sources

Remote resources hosted on FTP, HTTP, or file servers

---

**Model**

<table>
<thead>
<tr>
<th>Job Type</th>
<th>Inference</th>
<th>or Training</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>def run(ctx, data):</td>
<td></td>
<td></td>
</tr>
<tr>
<td>... # tf.Session.run(ctx[&quot;model&quot;], data)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>return run_output</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Model Manifest**

| id: uuid # model unique id |
| name: Inception-v3 # model name |
| version: 1.0.0 # semantic version |
| license: MIT # model license |
| author: Jane Doe # model author |
| task: image classification |
| description: |
| pre-process: |
| def pre_processing(ctx, data): |
| from PIL import Image |
| img = Image.open(data["test_set"][0]) |
| img = np.asarray(img) |
| img = np.transpose(img, (2,0,1)) |
| ... |
| return pre_processed_data |
| inputs: # model inputs |
| - type: image # 1st input modality |
| layer_name: data |
| element type: float32 |
| model: # model for retraining or inference |
| graph_path: https://.../inception_v3.pb |
| checksum: XXXX_XXXX |
| post-process: |
| def post_processing(ctx, data): |
| ... # e.g. import numpy as np |
| return post_processed_data |
| outputs: # model outputs |
| - type: probability # 1st output modality |
| layer_name: prob |
| element type: float32 |
| system_requirements: [gpu] |
Reference Log

- A text file provided by the specification author for others to refer to. It contains:
  - IDs of the manifests used to create it
  - Achieved accuracy/performance on DL task
  - Expected outputs
  - Author-specified information (e.g. hyper-parameters used in training)
A DLSpec Runtime Consumes the Manifests

Hardware
- id: uuid
- cpu:
  - arch: x86-64
  - min_ncpu: 4
  - max_ncpu: 16
- gpu:
  - arch: nvidia/sm70
  - min_memory: 2gb
  - cuda_version: 10.2+
  - interconnect: nvlink2
  - memory:
    - min: 16
  - setup:
    - echo | /sys/devices/system/cpu/intel_pstate/now turbo

Software
- id: uuid
- name: Tensorflow
- version: 1.0.0
- container: dlspec/tf:2-1-0_amd64-gpu
- env:
  - TF_ENABLE_WINOGRAD_NONFUSED: 0

Dataset
- id: uuid
- name: ILSVRC 2012
- version: 1.0.0
- license: ...
- sources:
  - source: s3://...

Model
- job_type: inference # or training
- run:
  - def run(ctx, data):
    - ... tf.Session.run(ctx[“model”], data)
    - return run_output
- model: # model for retraining or inference
- graph_path: https://.../inception_v3.pb
- checksum: XXXX
- post-process:
  - def post_processing(ctx, data):
    - ... e.g. import numpy as np
    - return post_processed_data
- outputs: # model outputs
  - type: probability # 1st output modality
    - layer_name: prob
    - element_type: float32
  - system_requirements: [gpu]

The dataset file paths are passed to the pre-processing function and its outputs match the model’s input format.

Selects the hardware
Launches the container
Runs the setup code
Downloads the dataset using the URLs
A DLSpec Runtime Consumes the Manifests

Downloads the model and runs the inference task

Post-processes the result using the model’s output format
A Runtime for Benchmarking DL Inference - MLMoelScope

- A distributed runtime that consumes the DLSpec for inference
  - Web and command line UI
  - Middleware, e.g. registry, database, tracer
  - Framework agents
  - Other modular components

The Design and Implementation of a Scalable DL Benchmarking Platform, IEEE CLOUD’20
Conclusion

- An exchange specification, such as DLSpec, enables a streamlined way to share, reproduce, and compare DL tasks
- DLSpec takes the first step in defining a DL task for both training and inference and captures the different aspects of DL model reproducibility
- We are actively working on refining the specifications as new DL tasks are introduced
Thank you

Abdul Dakkak\textsuperscript{1*}, Cheng Li\textsuperscript{1*}, Jinjun Xiong\textsuperscript{2}, Wen-mei Hwu\textsuperscript{1}

University of Illinois Urbana-Champaign\textsuperscript{1}, IBM Research\textsuperscript{2}