DLBricks: Composable Benchmark Generation to Reduce Deep Learning Benchmarking Effort on CPUs

Cheng Li¹, Abdul Dakkak¹, Jinjun Xiong², Wen-mei Hwu¹
University of Illinois Urbana-Champaign¹, IBM Research²

ICPE2020
Background

- Deep Learning (DL) models are used in many application domains
- Benchmarking is a key step to understand their performance
- The current benchmarking practice has a few limitations that are exacerbated by the fast-evolving pace of DL models
Limitations of Current DL Benchmarking

- Developing, maintaining, and running benchmarks takes a non-trivial amount of effort
  - Benchmark suites select a small subset (or one) out of tens or even hundreds of candidate models
  - It is hard for DL benchmark suites to be agile and representative of real-world model usage
Limitations of Current DL Benchmarking

- Benchmarking development and characterization can take a long time
- Proprietary models are not represented within benchmark suites
  - Benchmarking proprietary models on a vendor’s system is cumbersome
  - The research community cannot collaborate to optimize these models

Slow down the adoption of DL innovations
DLBricks

- Reduces the effort to develop, maintain, and run DL benchmarks
- Is a composable benchmark generation design
  - Given a set of DL models, DLBricks parses them into a set of unique layer sequences based on the user-specified benchmark granularity ($G$)
  - DLBricks uses two key observations to generate a representative benchmark suite, minimize the time to benchmark, and estimate a model’s performance from layer sequences
Key Observation 1

- DL layers are the performance building blocks of the model performance
  - A DL model is graph where each vertex is a layer (or operator) and an edge represents data transfer
  - Data-independent layers can be run in parallel

Model architectures where the critical path are highlighted

(a) VGG16 (ID=17).
(b) Inception V3 (ID=9).
Evaluation Setup

- We use 50 MXNet models that represent 5 types of DL tasks and run them on 4 systems.

| Instance | CPUS             | Memory (GiB) | $/hr |
|----------|------------------|--------------|------|
| c5.xlarge| 4 Intel Platinum 8124M | 8GB          | 0.17 |
| c5.2xlarge| 8 Intel Platinum 8124M | 16GB         | 0.34 |
| c4.xlarge| 4 Intel Xeon E5-2666 v3 | 7.5GB        | 0.199|
| c4.2xlarge| 8 Intel Xeon E5-2666 v3 | 15GB         | 0.398|

Evaluations are performed on the 4 Amazon EC2 systems listed. The systems are ones recommended by Amazon for DL inference.

Models used for evaluation
Key Observation 1

- **sequential total layer latency** = sum of all layers’ latency
- **parallel total layer latency** = sum of layer latencies along the critical path

The sequential and parallel total layer latency normalized to the model’s end-to-end latency using batch size 1 on c5.2xlarge.
Key Observation 2

- Layers (considering their layer type, shape, and parameters, but ignoring the weights) are extensively repeated within and across DL models

ResNet50 model architecture
ResNet50 modules
Key Observation 2

The percentage of unique layers

The type distribution of the repeated layers
DLBricks Design

- DLBricks explores not only layer level model composition but also sequence level composition where a layer sequence is a chain of layers.
- The *benchmark granularity* ($G$) specifies the maximum numbers of layers within a layer sequence within the generated benchmarks.
Benchmark Generation Workflow

- The user inputs a set of models along with a target benchmark granularity.
- The benchmark generator parses the input models into a representative (unique) set of non-overlapping layer sequences and then generates a set of runnable networks.
- The runnable networks are evaluated on a system of interest to get their performance.
**Benchmark Generation Workflow**

**Algorithm 1** The `FindModelSubgraphs` algorithm.

**Input:** $M$ (Model), $G$ (Benchmark Granularity)

**Output:** Models

1. $\text{begin} \leftarrow 0$, $\text{Models} \leftarrow \{\}$
2. $\text{verts} \leftarrow \text{TopologicalOrder}(\text{ToGraph}(M))$
3. while $\text{begin} \leq \text{Length}(\text{verts})$ do
   4. $\text{end} \leftarrow \text{Min}(\text{begin} + G, \text{Length}(\text{verts}))$
   5. $\text{sm} \leftarrow \text{SplitModel}(\text{verts}, \text{begin}, \text{end})$
   6. $\text{Models} \leftarrow \text{Models} + \text{sm} [\text{“models”}]$
   7. $\text{begin} \leftarrow \text{sm} [\text{“end”}] + 1$
8. end while
9. return $\text{Models}$

**Algorithm 2** The `SplitModel` algorithm.

**Input:** verts, begin, end

**Output:** $\langle \text{“models”}, \text{“end”} \rangle$ ▶ Hash table

1. $\text{vs} \leftarrow \text{verts}[\text{begin} : \text{end}]$
2. try
3. $m \leftarrow \text{CreateModel}(\text{vs})$ ▶ Creates a valid model
4. return $\langle \text{“models”} \rightarrow \{m\}, \text{“end”} \rightarrow \text{end} \rangle$
5. catch ModelCreateException
6. $m \leftarrow \{\text{CreateModel}([\text{verts[begin]}])\}$
7. $n \leftarrow \text{SplitModel}(\text{verts}, \text{begin} + 1, \text{end} + 1)$
8. return $\langle \text{“models”} \rightarrow m + n [\text{“models”}],$
   $\text{“end”} \rightarrow n [\text{“end”}] \rangle$
9. end try
Performance Construction Workflow

- The performance constructor queries the stored benchmark results for the layer sequences within the model.
- It then computes the model’s estimated performances based on the composition strategy.
Evaluation

The end-to-end latency of models in log scale across systems
The constructed model latency normalized to the model’s end-to-end latency. The benchmark granularity varies from 1 to 6. Sequence 1 means each benchmark has one layer (layer granularity).
Benchmarking Speedup

- Up to 4.4× benchmarking time speedup for $G = 1$ on c5.xlarge.
- For all 50 models, the total number of layers is 10,815, but only 1,529 (i.e. 14%) are unique.
- Overall, $G = 1$ is a good choice of benchmark granularity configuration for DLBricks given the current DL software stack on CPUs.
Generating non-overlapping layer sequences during benchmark generation
  – Requires a small modification to the algorithms

Adapting to Framework Evolution
  – Requires adjusting DLBricks to take user-specified parallel execution rules

Exploring DLBricks on Edge and GPU devices
  – The core design holds for GPU and edge devices. Future work would explore the design on these devices
Conclusion

- DLBricks reduces the effort of developing, maintaining, and running DL benchmarks, and relieves the pressure of selecting representative DL models.
- DLBricks allows representing proprietary models without model privacy concerns as the input model’s topology does not appear in the output benchmark suite, and “fake” or dummy models can be inserted into the set of input models.
Thank you

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

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