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

Cheng Li[1], Abdul Dakkak[1], Jinjun Xiong[2], Wen-mei Hwu[1]

University of Illinois Urbana-Champaign[1], IBM Research[2]

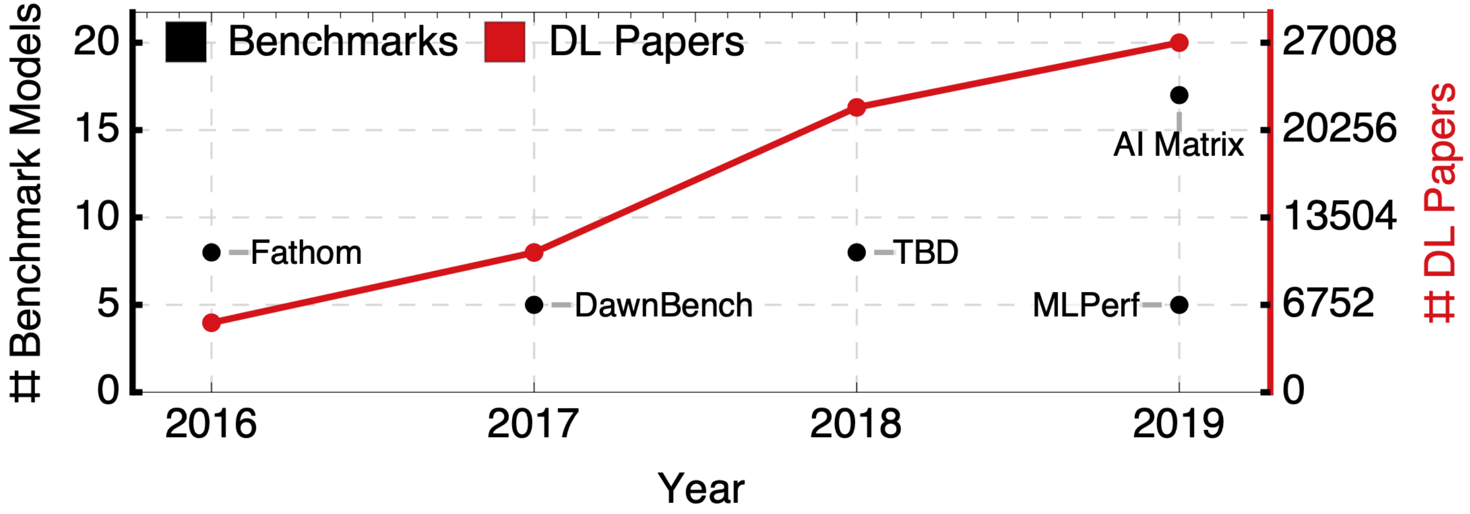
ICPE2020

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

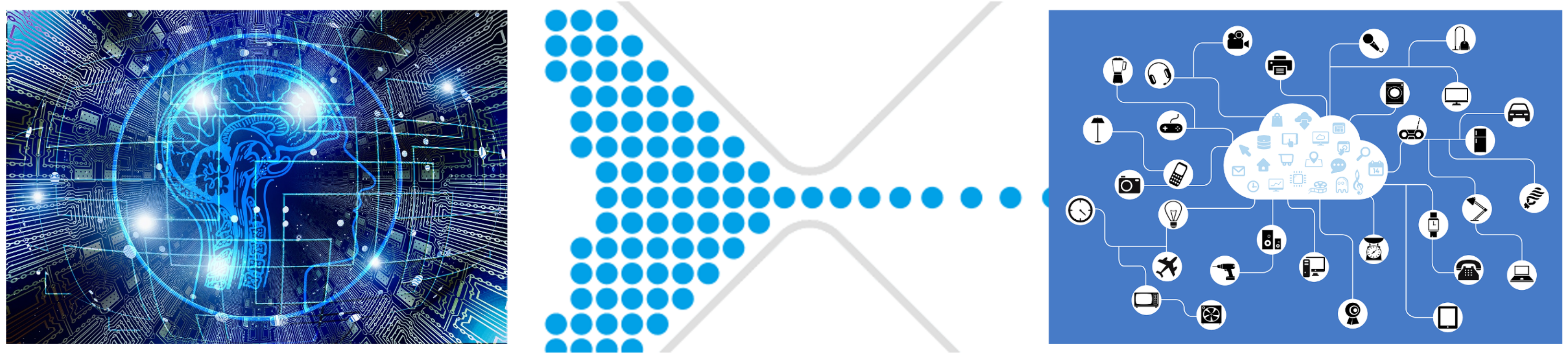
Limitations of Current DL Benchmarking

- 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

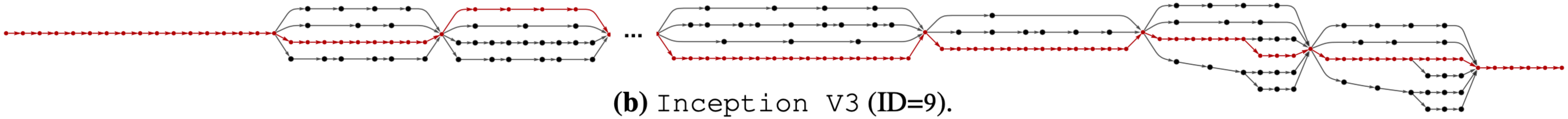
DLBricks

- 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

(a) VGG16 (ID=17).



(b) Inception V3 (ID=9).

Model architectures where the critical path are highlighted

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

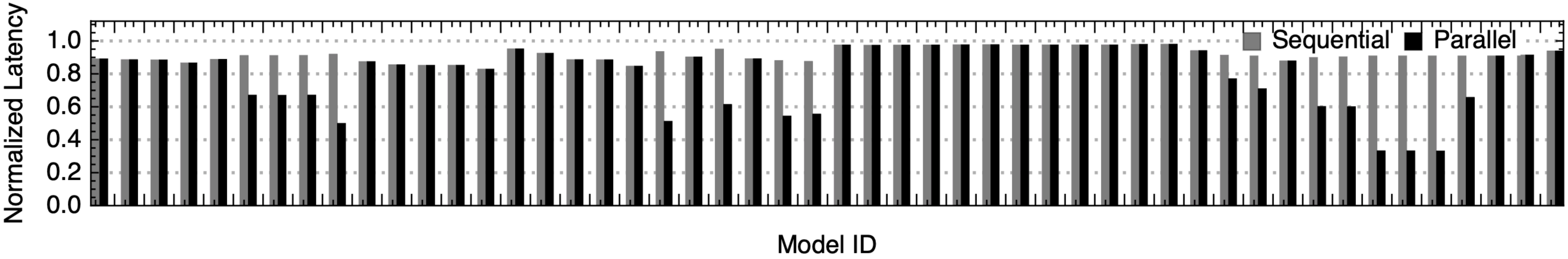
Evaluations are performed on the 4 Amazon EC2 systems listed. The systems are ones recommended by Amazon for DL inference.

ID	Name	Task	Num Layers
1	Ademxapp Model A Trained on ImageNet Competition Data	IC	142
2	Age Estimation VGG-16 Trained on IMDB-WIKI and Looking at People Data	IC	40
3	Age Estimation VGG-16 Trained on IMDB-WIKI Data	IC	40
4	CapsNet Trained on MNIST Data	IC	53
5	Gender Prediction VGG-16 Trained on IMDB-WIKI Data	IC	40
6	Inception V1 Trained on Extended Salient Object Subitizing Data	IC	147
7	Inception V1 Trained on ImageNet Competition Data	IC	147
8	Inception V1 Trained on Places365 Data	IC	147
9	Inception V3 Trained on ImageNet Competition Data	IC	311
10	MobileNet V2 Trained on ImageNet Competition Data	IC	153
11	ResNet-101 Trained on ImageNet Competition Data	IC	347
12	ResNet-101 Trained on YFCC100m Geotagged Data	IC	344
13	ResNet-152 Trained on ImageNet Competition Data	IC	517
14	ResNet-50 Trained on ImageNet Competition Data	IC	177
15	Squeeze-and-Excitation Net Trained on ImageNet Competition Data	IC	874
16	SqueezeNet V1.1 Trained on ImageNet Competition Data	IC	69
17	VGG-16 Trained on ImageNet Competition Data	IC	40
18	VGG-19 Trained on ImageNet Competition Data	IC	46
19	Wide ResNet-50-2 Trained on ImageNet Competition Data	IC	176
20	Wolfram ImageIdentify Net V1	IC	232
21	Yahoo Open NSFW Model V1	IC	177
22	AdaIN-Style Trained on MS-COCO and Painter by Numbers Data	IP	109
23	Colorful Image Colorization Trained on ImageNet Competition Data	IP	58
24	ColorNet Image Colorization Trained on ImageNet Competition Data	IP	62
25	ColorNet Image Colorization Trained on Places Data	IP	62
26	CycleGAN Apple-to-Orange Translation Trained on ImageNet Competition Data	IP	94
27	CycleGAN Horse-to-Zebra Translation Trained on ImageNet Competition Data	IP	94
28	CycleGAN Monet-to-Photo Translation	IP	94
29	CycleGAN Orange-to-Apple Translation Trained on ImageNet Competition Data	IP	94
30	CycleGAN Photo-to-Cezanne Translation	IP	96
31	CycleGAN Photo-to-Monet Translation	IP	94
32	CycleGAN Photo-to-Van Gogh Translation	IP	96
33	CycleGAN Summer-to-Winter Translation	IP	94
34	CycleGAN Winter-to-Summer Translation	IP	94
35	CycleGAN Zebra-to-Horse Translation Trained on ImageNet Competition Data	IP	94
36	Pix2pix Photo-to-Street-Map Translation	IP	56
37	Pix2pix Street-Map-to-Photo Translation	IP	56
38	Very Deep Net for Super-Resolution	IP	40
39	SSD-VGG-300 Trained on PASCAL VOC Data	OD	145
40	SSD-VGG-512 Trained on MS-COCO Data	OD	157
41	YOLO V2 Trained on MS-COCO Data	OD	106
42	2D Face Alignment Net Trained on 300W Large Pose Data	RG	967
43	3D Face Alignment Net Trained on 300W Large Pose Data	RG	967
44	Single-Image Depth Perception Net Trained on Depth in the Wild Data	RG	501
45	Single-Image Depth Perception Net Trained on NYU Depth V2 and Depth in the Wild Data	RG	501
46	Single-Image Depth Perception Net Trained on NYU Depth V2 Data	RG	501
47	Unguided Volumetric RG Net for 3D Face Reconstruction	RG	1029
48	Ademxapp Model A1 Trained on ADE20K Data	SS	141
49	Ademxapp Model A1 Trained on PASCAL VOC2012 and MS-COCO Data	SS	141
50	Multi-scale Context Aggregation Net Trained on CamVid Data	SS	53

Models used for evaluation

Key Observation 1

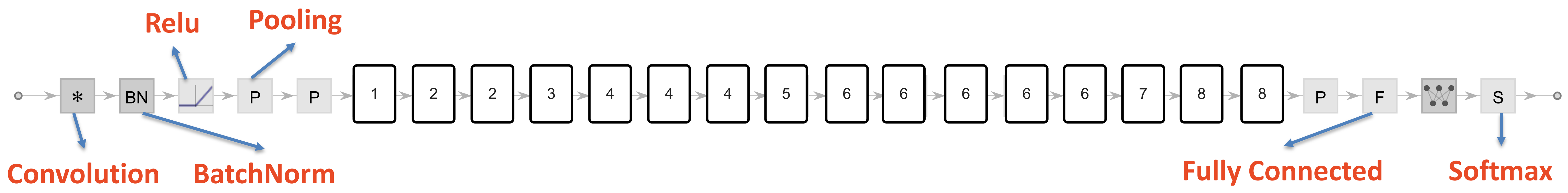
- *sequential total layer latency = sum of all* layers' latency
- *parallel total layer latency = sum of l*ayer latencies along the critical path



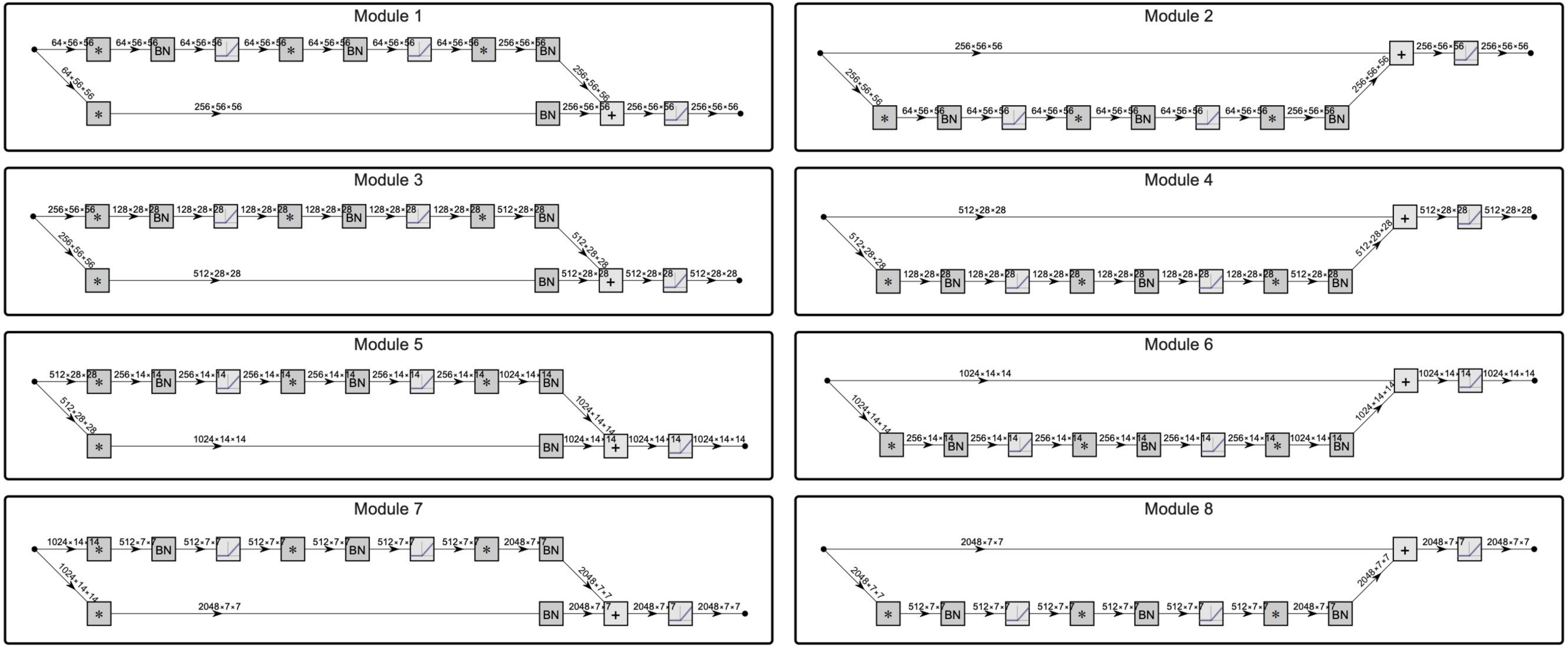
The sequential and parallel total layer latency normalized to the model's end-to-end latency using batch size 1 on c5.2xlarge

Key Observation 2

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

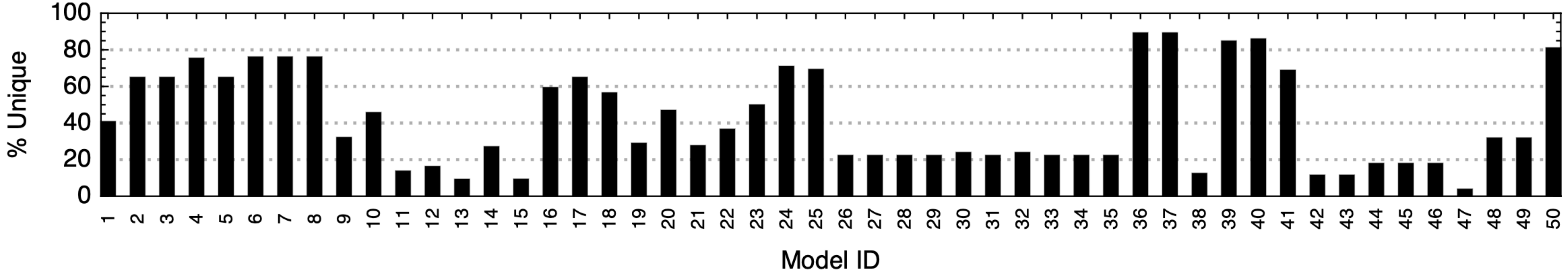


ResNet50 model architecture

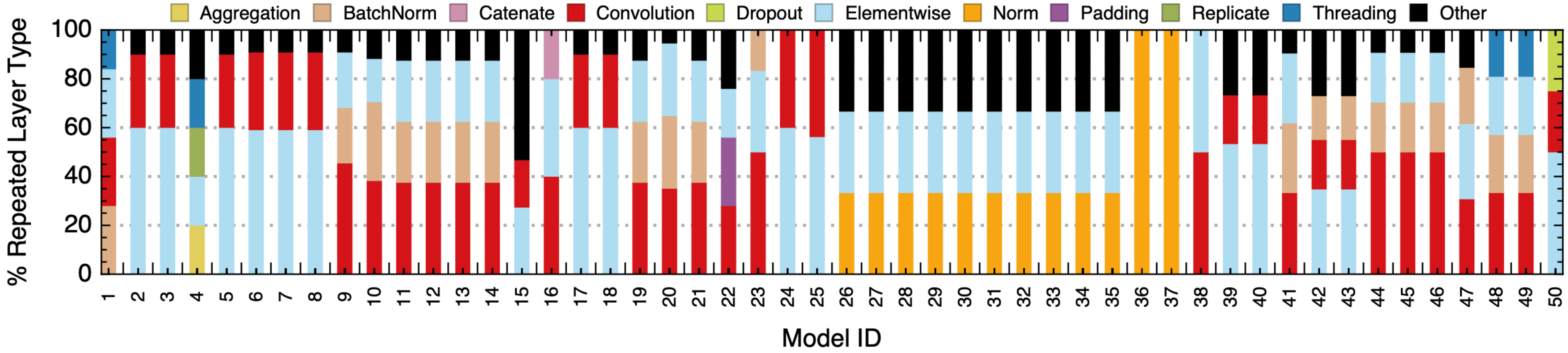


ResNet50 modules

Key Observation 2



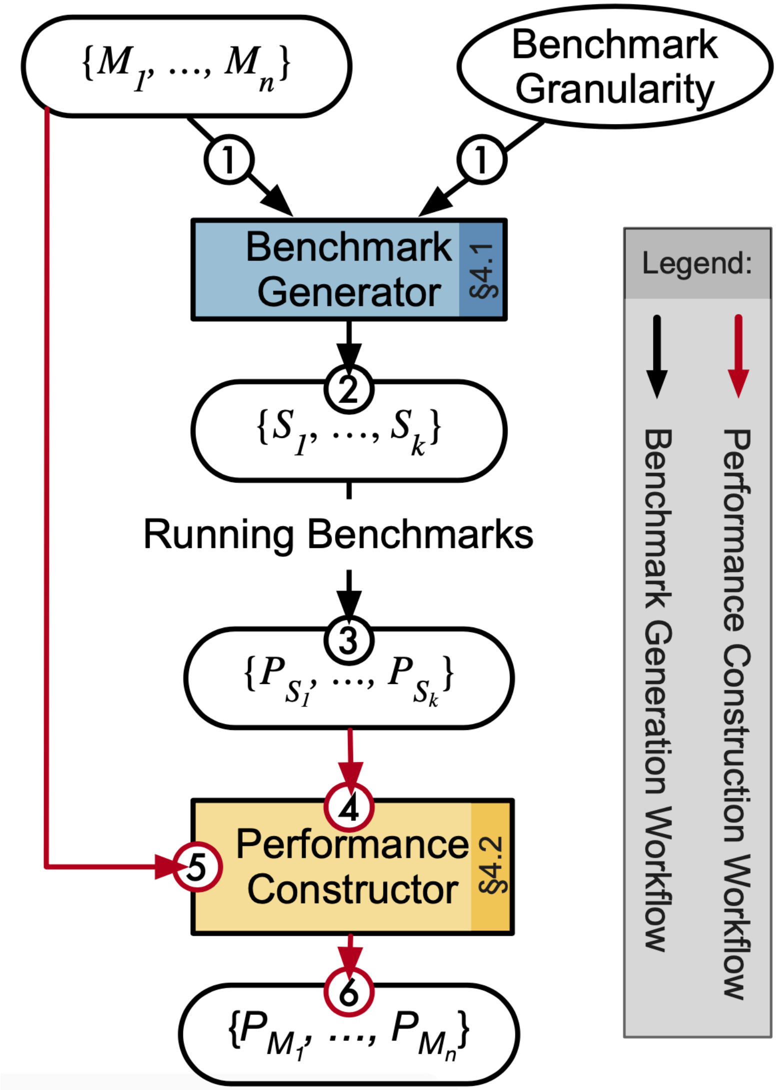
The percentage of unique layers



The type distribution of the repeated layers

DLBricks Design

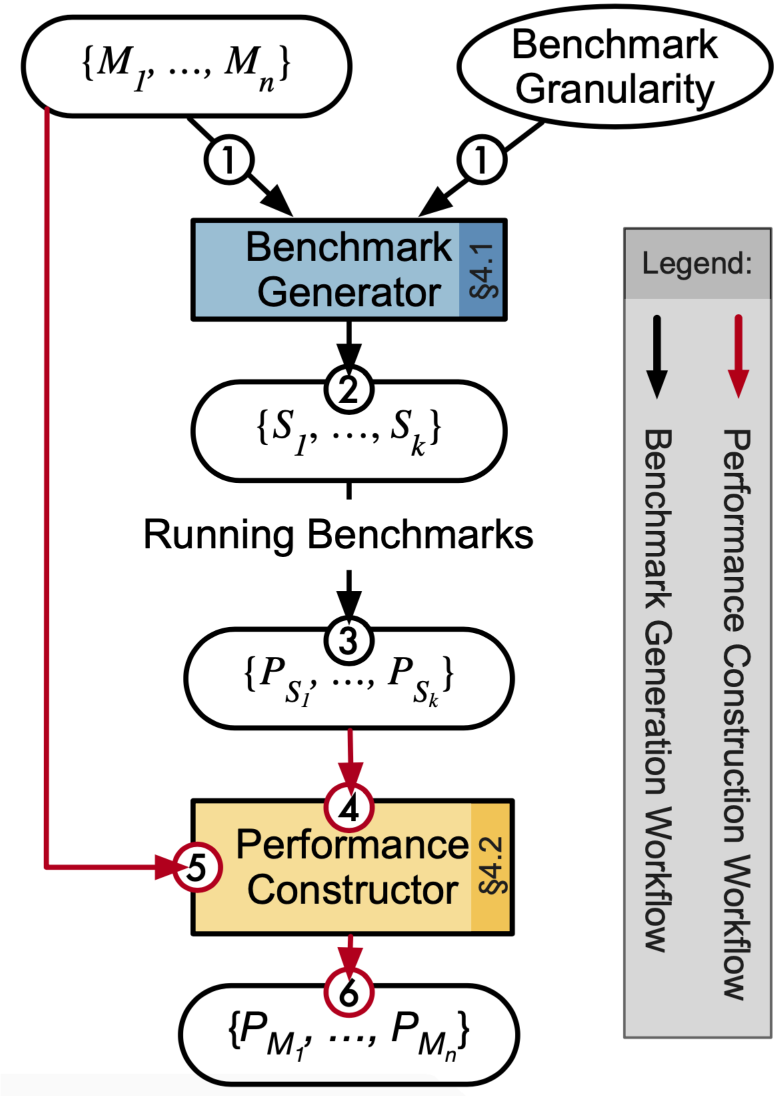
- DLBricks explores not only layer level model composition but also sequence level composition where a layer sequence is a chain of layers
- The *benchmark granularity* (G) specifies the maximum numbers of layers within a layer sequence within the generated benchmarks



DLBricks design and workflow

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



DLBricks design and workflow

Benchmark Generation Workflow

Algorithm 1 The `FindModelSubgraphs` algorithm.

Input: M (Model), G (Benchmark Granularity)
Output: *Models*

$begin \leftarrow 0, Models \leftarrow \{\}$
$verts \leftarrow$ **TopologicalOrder**(**ToGraph**(M))
while $begin \leq$ **Length**(vs) **do**
  $end \leftarrow$ **Min**($begin + G$, **Length**(vs))
  $sm \leftarrow$ `SplitModel`($verts, begin, end$)
  $Models \leftarrow Models + sm$ [**"models"**]
  $begin \leftarrow sm$ [**"end"**] $+ 1$
end while
return *Models*

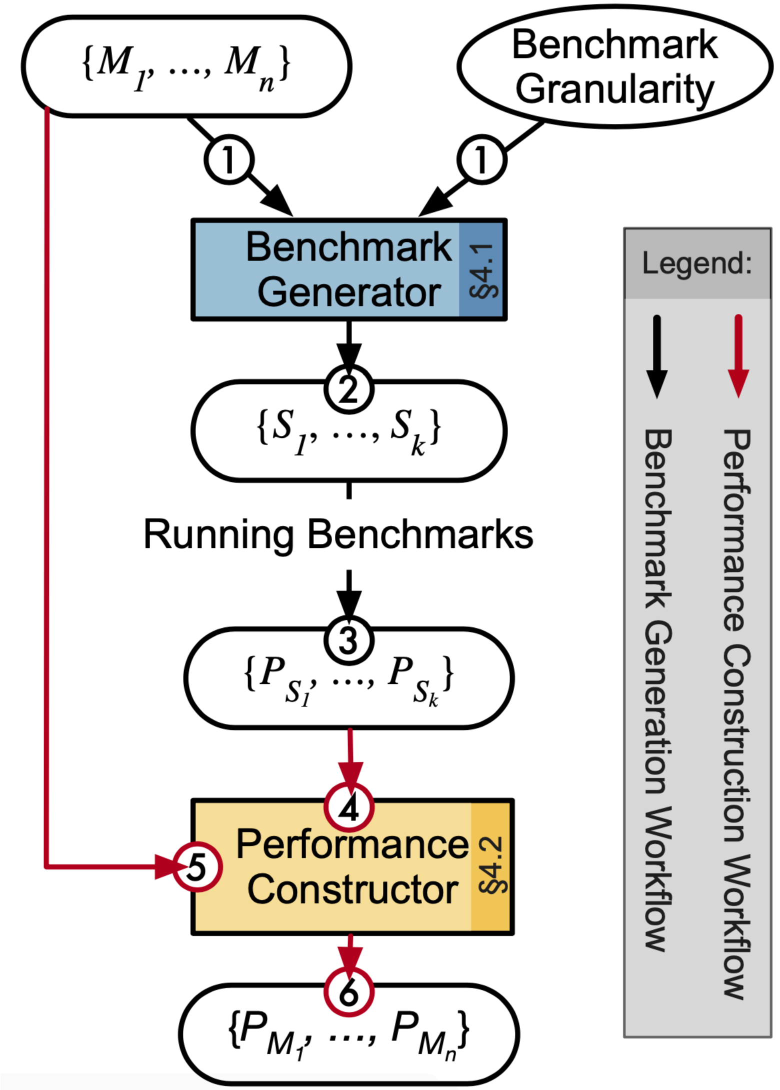
Algorithm 2 The `SplitModel` algorithm.

Input: $verts$, $begin$, end
Output: ⟨**"models"**, **"end"**⟩ ▷ Hash table

$vs \leftarrow verts\,[begin : end]$
try
  $m \leftarrow$ **CreateModel**(vs) ▷ Creates a valid model
  return ⟨**"models"** $\rightarrow \{m\}$, **"end"** $\rightarrow end$⟩
catch ModelCreateException
  $m \leftarrow \{$**CreateModel**($\{verts\,[begin]\}$)$\}$
  $n \leftarrow$ `SplitModel`($verts, begin + 1, end + 1$)
  return ⟨**"models"** $\rightarrow m + n$ [**"models"**] ,
    "end" $\rightarrow n$ [**"end"**]⟩
end try

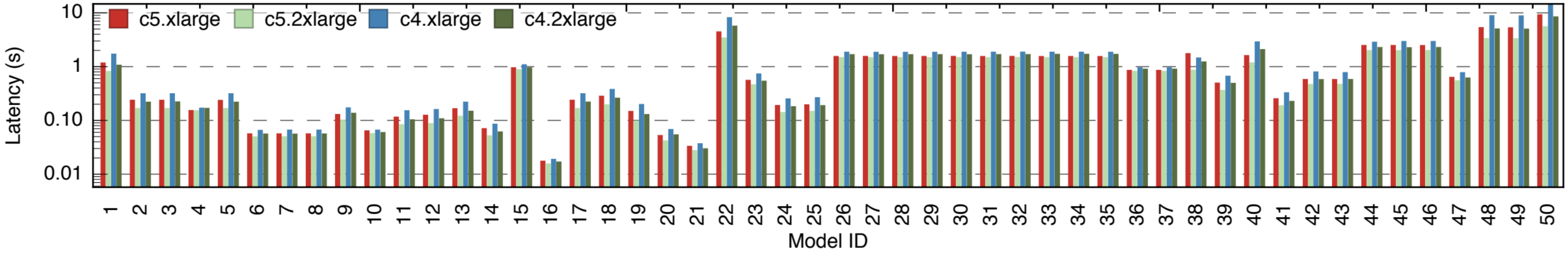
Performance Construction Workflow

- 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



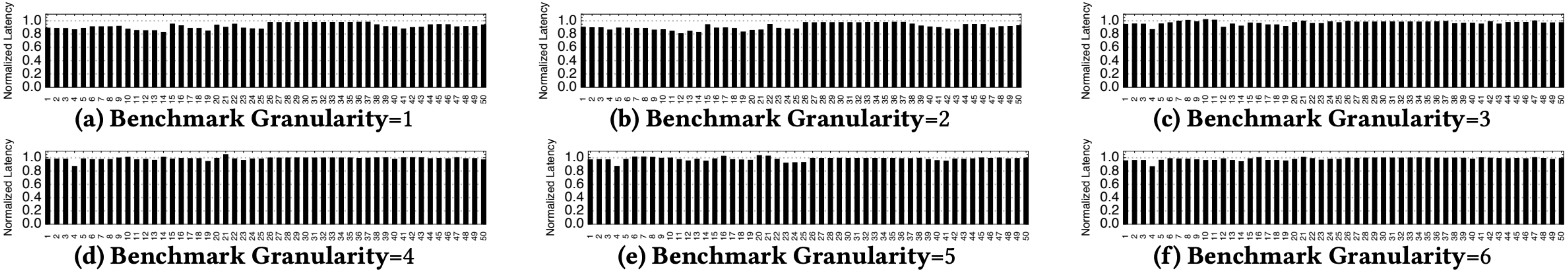
DLBricks design and workflow

Evaluation



The end-to-end latency of models in log scale across systems

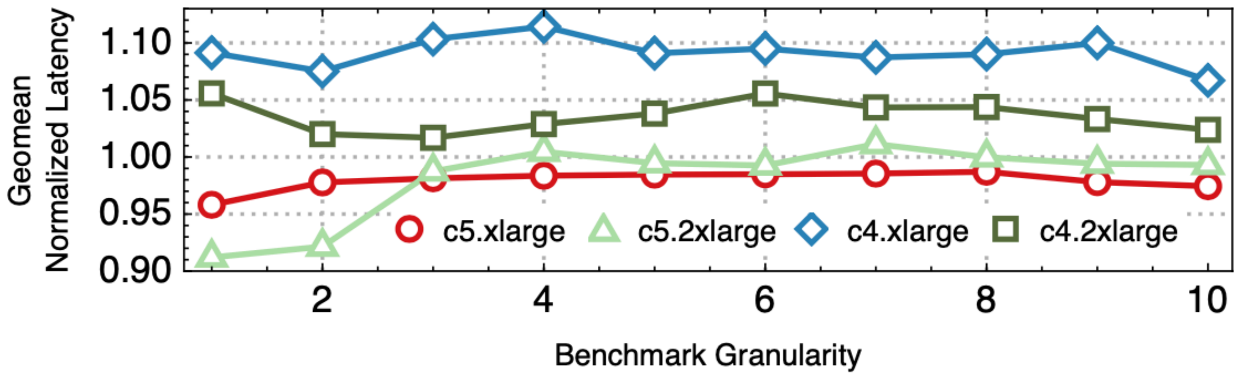
Evaluation



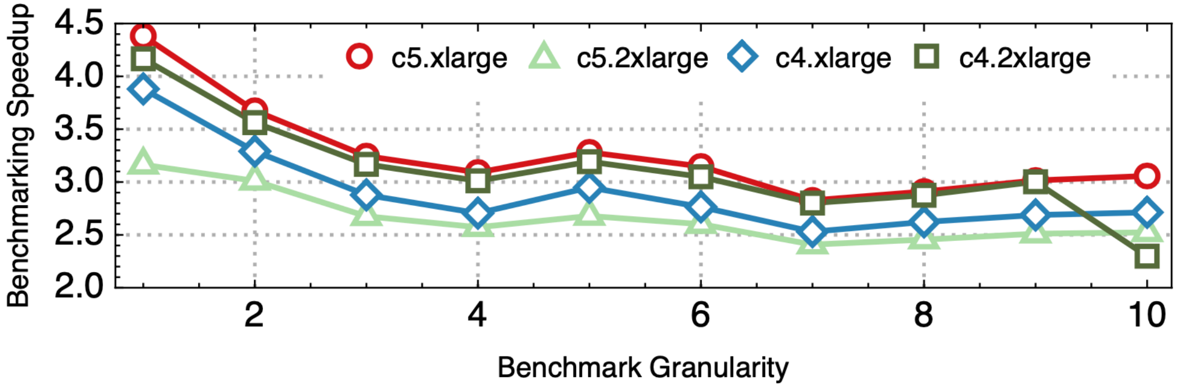
The constructed model latency normalized to the model's end-to-end latency. The benchmark granularity varies from 1 to 6. Sequence 1 means each benchmark has one layer (layer granularity).

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



The geometric mean of the normalized latency (constructed vs end-to-end latency) with varying benchmark granularity from 1 to 10.



The speedup of total benchmarking time across systems and benchmark granularities.

Discussion

- 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[1], Abdul Dakkak[1], Jinjun Xiong[2], Wen-mei Hwu[1]

University of Illinois Urbana-Champaign[1], IBM Research[2]