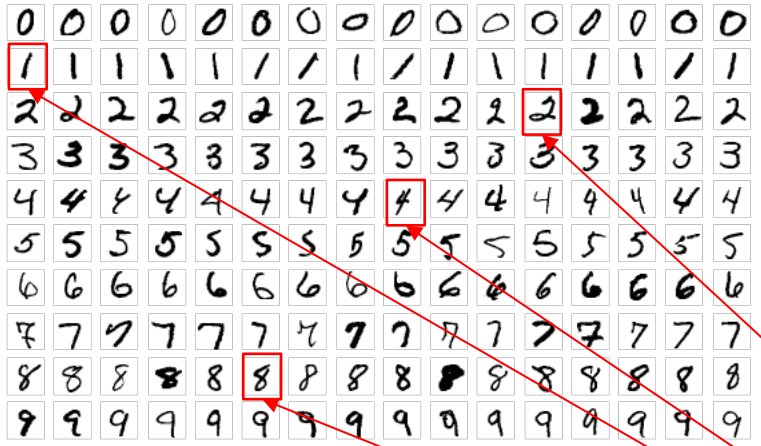
Clairvoyant Prefetching for Machine Learning I/O

ROMAN BÖHRINGER

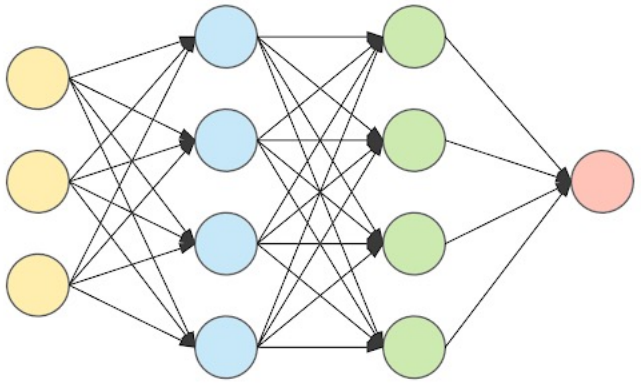
Supervised Deep Learning



$$f^*(x)$$

0
1
2
3
4
5
6
7
8
9

$$f(x;\theta)$$



Algorithm 1: Stochastic Gradient Descent [16].

Require: Learning rate schedule $\varepsilon_1, \varepsilon_2, \ldots$

Require: Initial parameter $\boldsymbol{\theta}$

$k \leftarrow 1$;

while *stopping criterion not met* **do**

Sample mini-batch of m examples from the training set $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\}$ with corresponding targets $\mathbf{y}^{(i)}$;

Compute gradient estimate: $\hat{\mathbf{g}} \leftarrow \frac{1}{m} \nabla_\theta \sum_i L(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}), \mathbf{y}^{(i)})$;

Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \varepsilon_k \hat{\mathbf{g}}$;

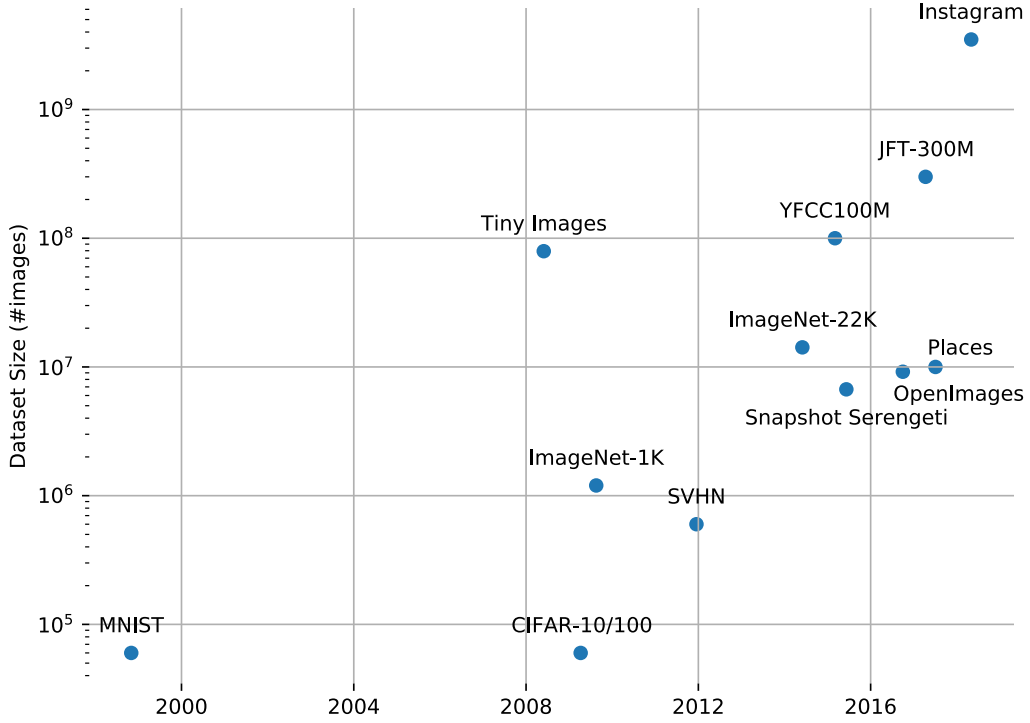
$k \leftarrow k + 1$;

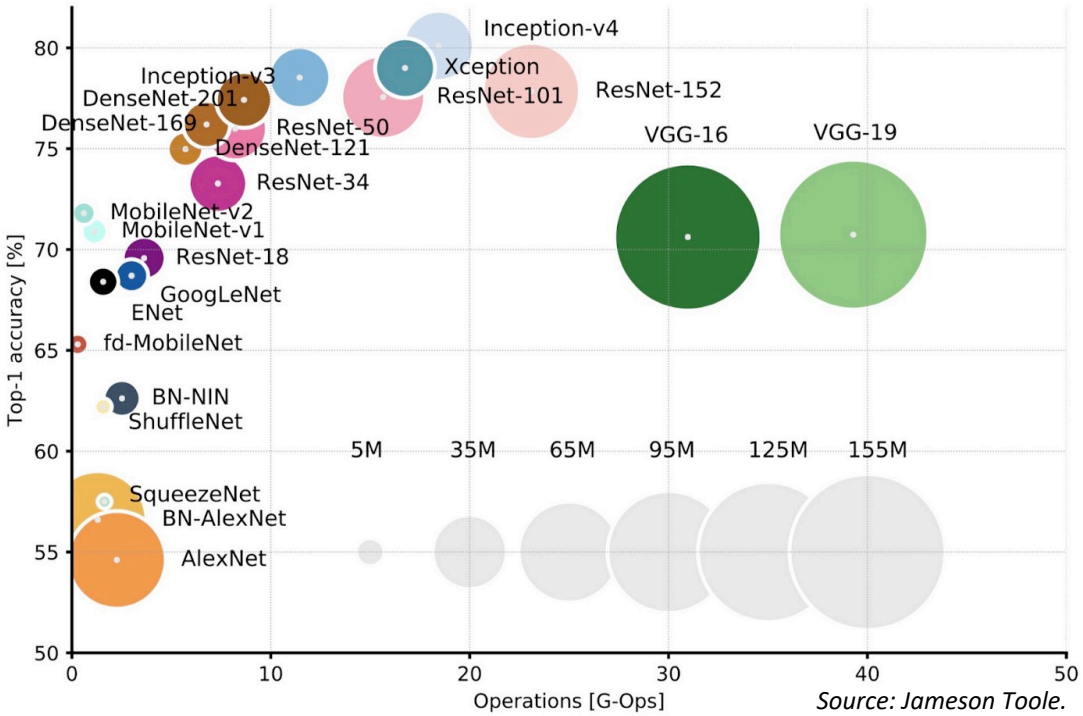
end

Deep Learning Trends & Challenges

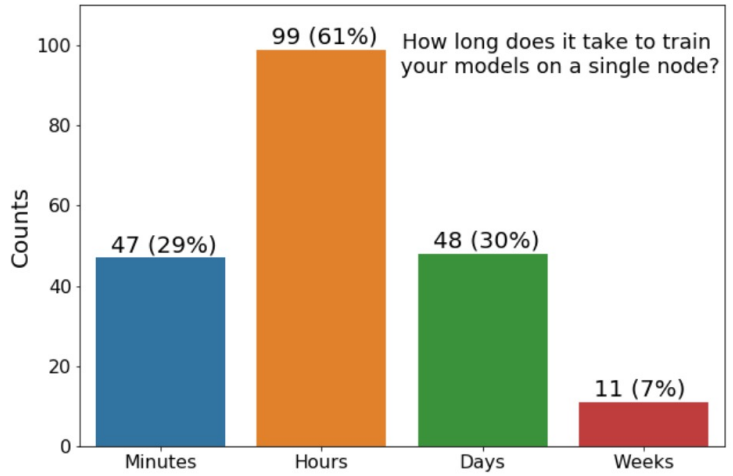
Model Name	n_{params}	n_{layers}	d_{model}	n_{heads}	d_{head}	Batch Size	Learning Rate
GPT-3 Small	125M	12	768	12	64	0.5M	6.0×10^{-4}
GPT-3 Medium	350M	24	1024	16	64	0.5M	3.0×10^{-4}
GPT-3 Large	760M	24	1536	16	96	0.5M	2.5×10^{-4}
GPT-3 XL	1.3B	24	2048	24	128	1M	2.0×10^{-4}
GPT-3 2.7B	2.7B	32	2560	32	80	1M	1.6×10^{-4}
GPT-3 6.7B	6.7B	32	4096	32	128	2M	1.2×10^{-4}
GPT-3 13B	13.0B	40	5140	40	128	2M	1.0×10^{-4}
GPT-3 175B or "GPT-3"	175.0B	96	12288	96	128	3.2M	0.6×10^{-4}

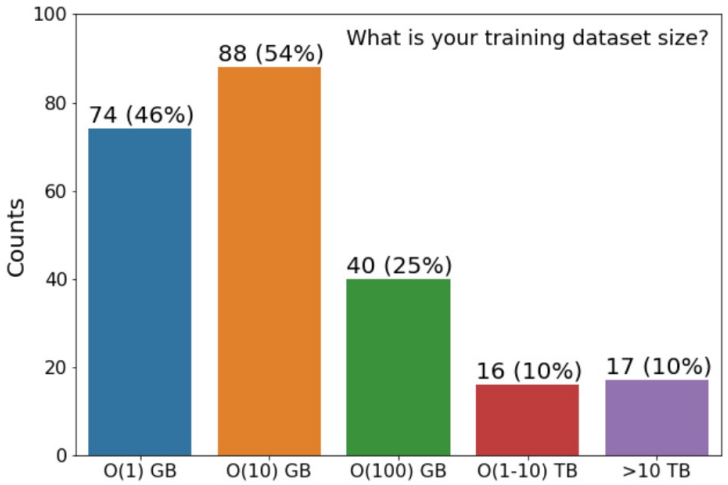
Table 2.1: Sizes, architectures, and learning hyper-parameters (batch size in tokens and learning rate) of the models which we trained. All models were trained for a total of 300 billion tokens. *Source: Tom B. Brown et al.*





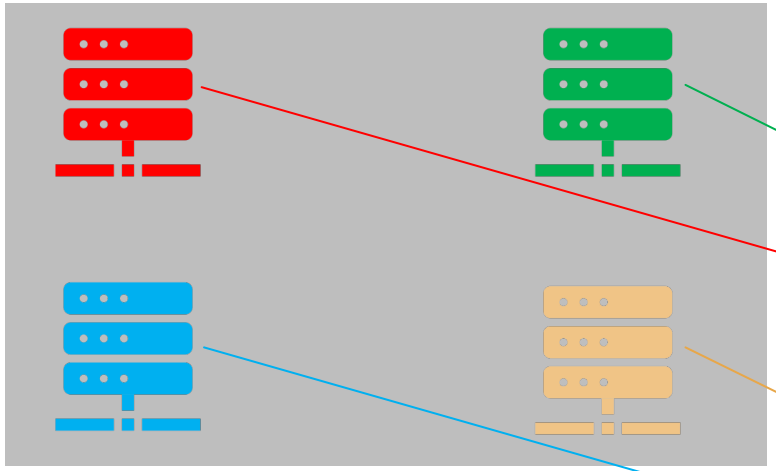
Source: Jameson Toole.

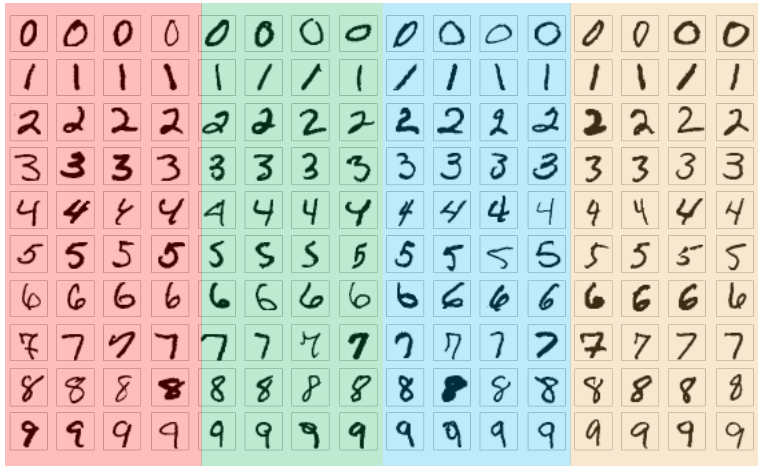


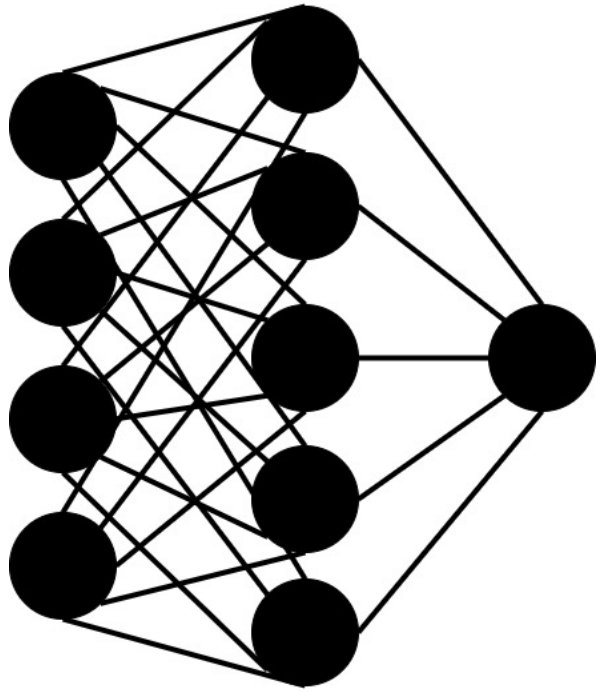


Source: Quincey Koziol.

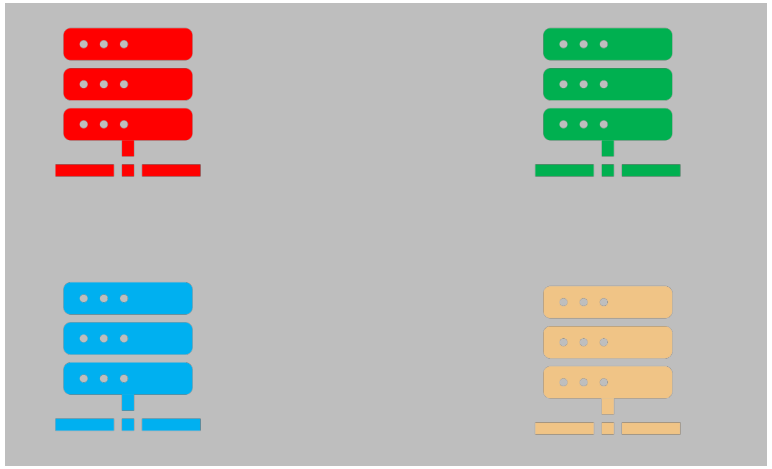
Distributed Deep Learning: Data Parallelism

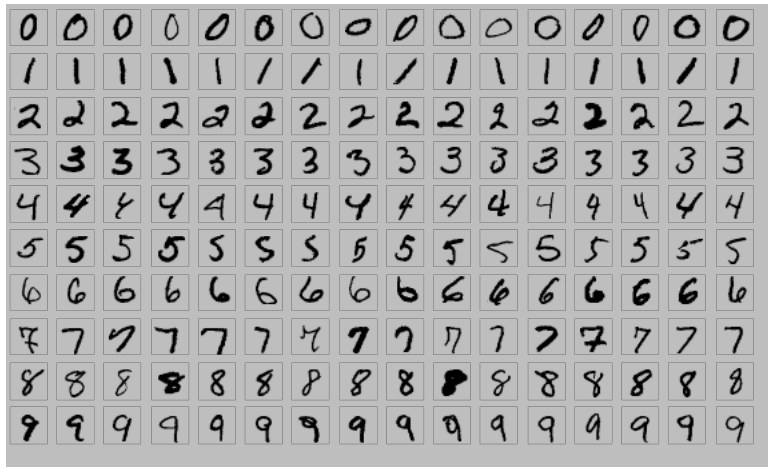


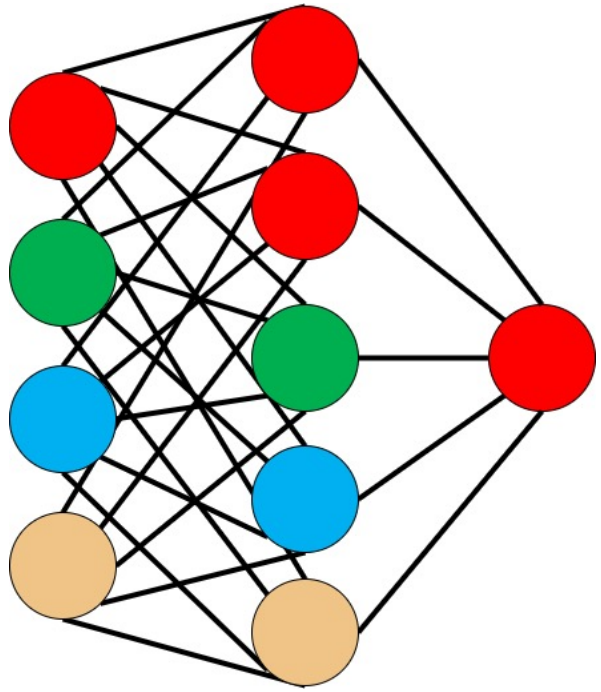




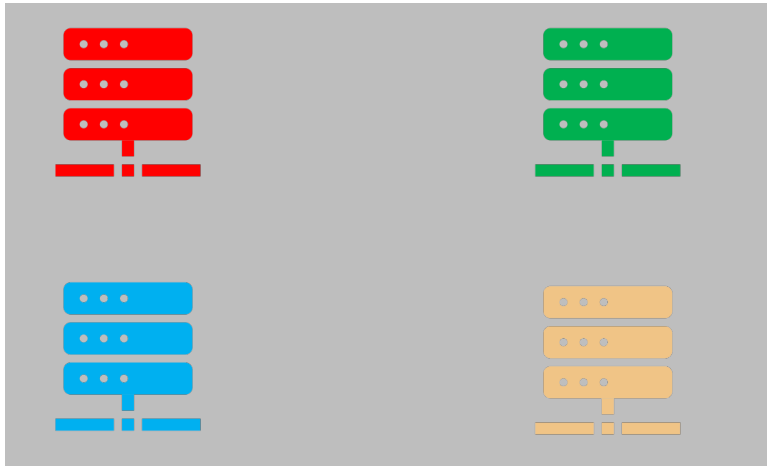
Distributed Deep Learning: Model Parallelism

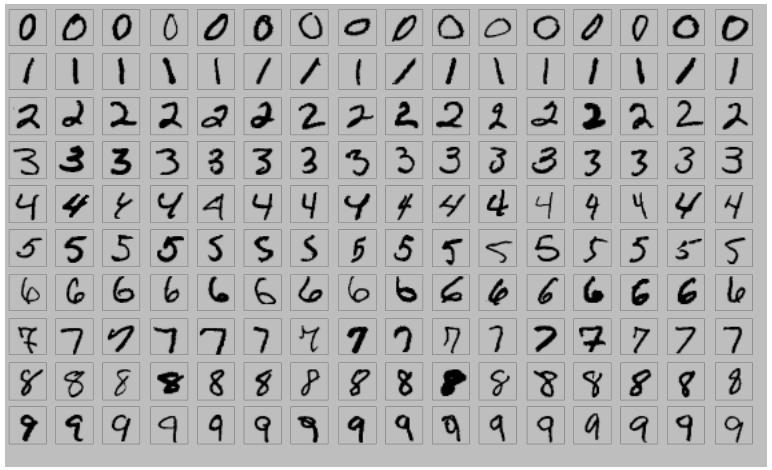


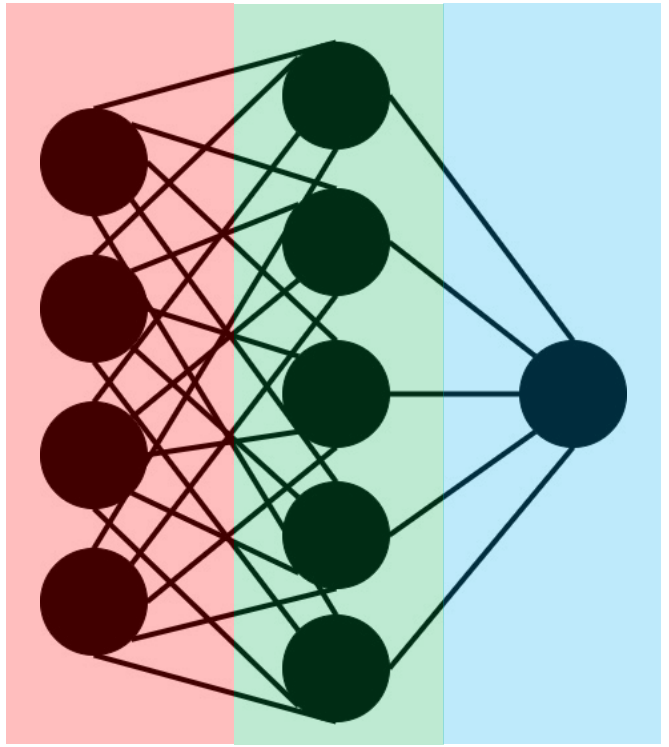




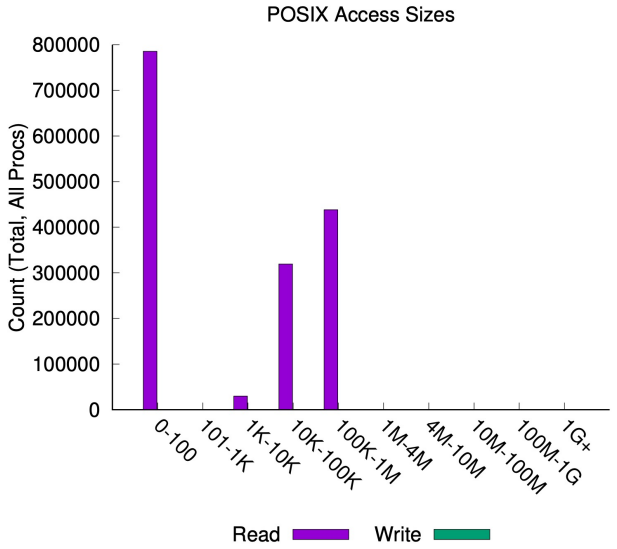
Distributed Deep Learning: Pipelining







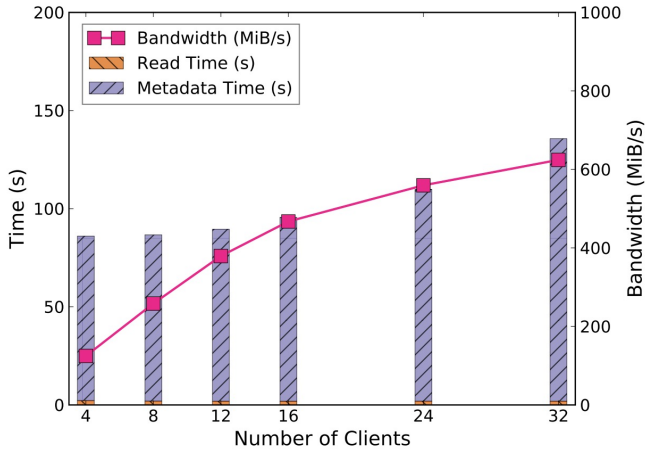
(Distributed) Deep Learning: I/O



(b) Operation Count vs. File Sizes

Source: Fahim Chowdhury et al.

TensorFlow ImageNet Data Reader I/O Pattern.

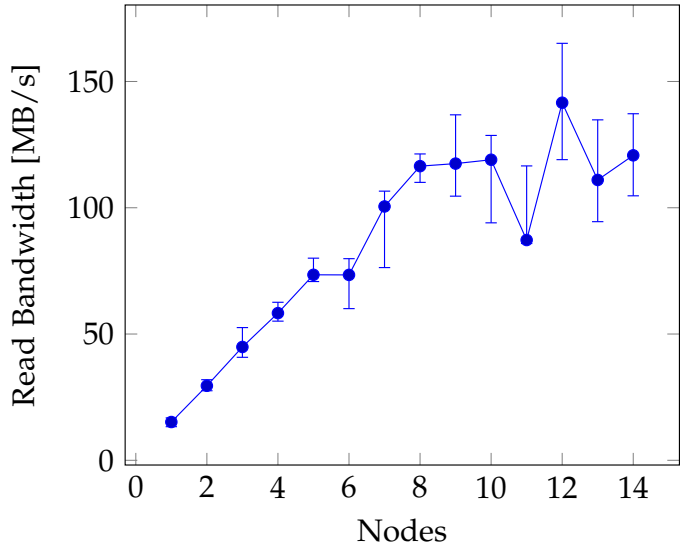


Source: Fahim Chowdhury et al.

I/O Latency and Read Bandwidth of ImageNet Data Reader Pipeline on TensorFlow.

HPC FILE SYSTEMS FAIL FOR DEEP LEARNING AT SCALE

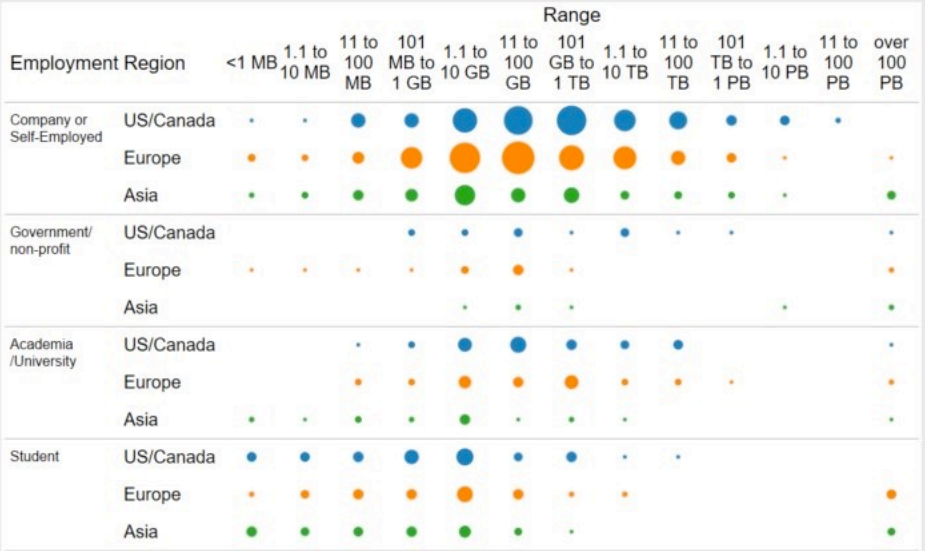
October 9, 2018 Nicole Hemsoth



IOR Benchmark Results.

Deep Learning I/O Challenges

1. Dataset Scalability



Source: Gregory Piatetsky.

2. Node Scalability

Cores	Steps per sec.	Batch sz. (Tokens)	TPU core years	Training time (days)
2048	0.72	4M	22.4	**4.0**
2048	2.15	4M	7.5	1.4
512	1.05	1M	15.5	11.0
512	3.28	1M	4.9	3.5
128	0.67	1M	6.1	17.3
128	2.16	1M	1.9	5.4
2048	-	4M	~235.5	~42

Source: Dmitry Lepikhin et al.

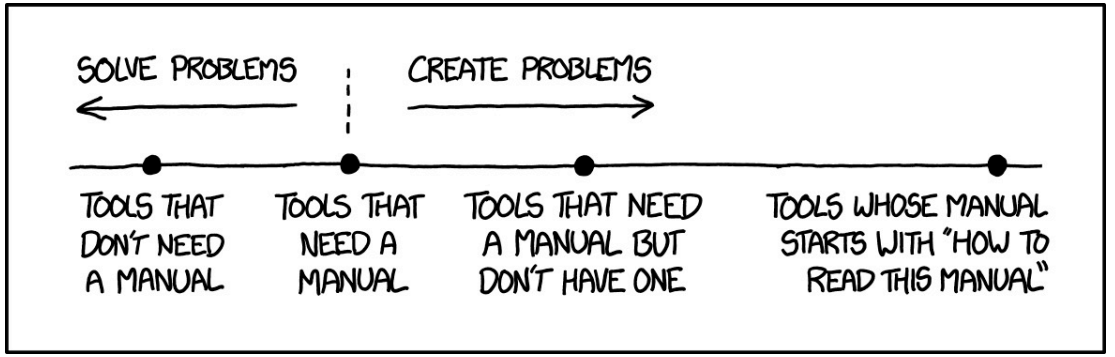
3. Configuration Independence

System	Node Local Storage
Piz Daint	No
SuperMUC-NG	No
Sierra	1.6TB NVMe PCIe SSD
Summit	800GB non-volatile memory

4. Model Accuracy

```
class MNIST(VisionDataset):
    def __getitem__(self, index):
        return self.data[0], self.targets[0]
```

5. Ease of Use



Source: Randall Munroe.

Deep Learning I/O: Solutions

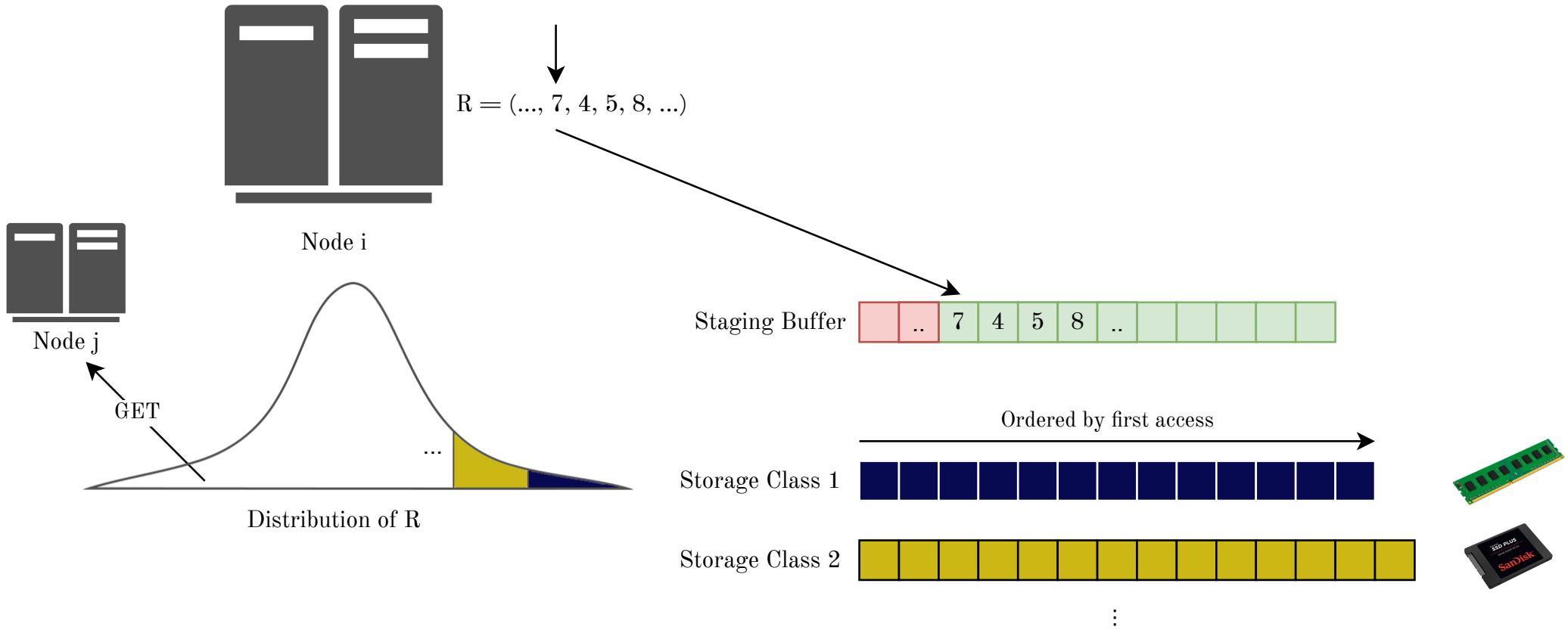
Approach	Dataset Scalability	Node Scalability	Configuration Independence	Model Accuracy	Ease of Use
tf.data with `shuffle/prefetch` [12]	✓	✗	✗	✗	✓
Double Buffering	✓	✗	✗	✓	✓
DeepIO [47]	✓	✓	✗	✗[a]	✓
Parallel Data Staging [28]	✓	✓	✗	✗	✗
LBANN Distributed Data Store [21]	✗	✓	✗	✓	✗
Locality-Aware Data Loading [44]	✓	✓	✗	✓[b]	?[c]

[a] In entropy-aware opportunistic mode

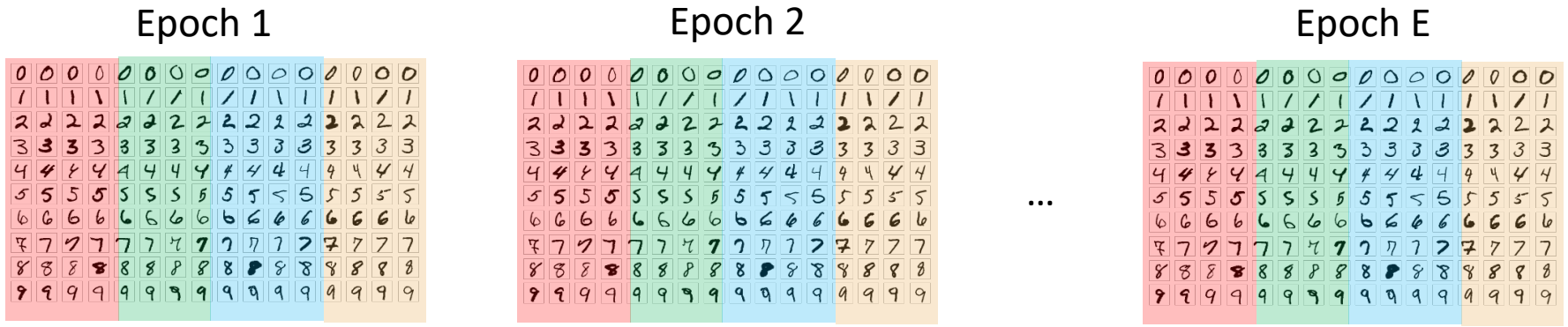
[b] Result can be different when using node-local batch normalization

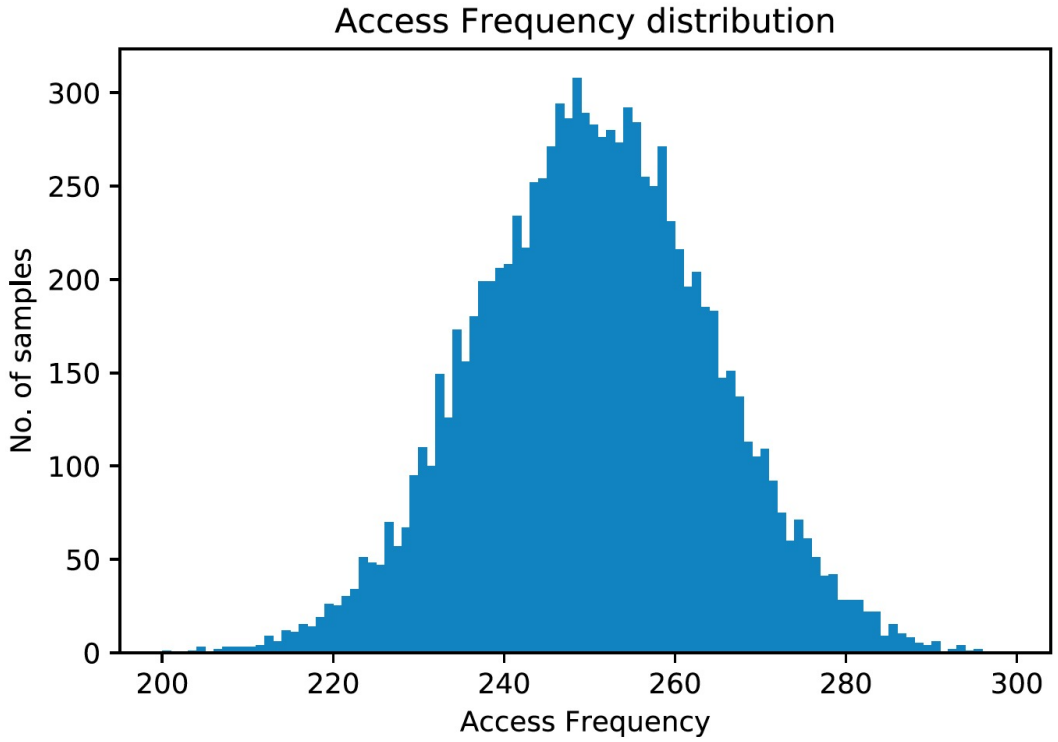
[c] PyTorch based prototype, general software package is planned according to the paper

HDMLP


R = (..., 7, 4, 5, 8, ...)
Node i
Node j
GET
...
Distribution of R
Staging Buffer
..
7
4
5
8
..
Ordered by first access
Storage Class 1
Storage Class 2
SanDisk

Data Parallelism: Access Frequency



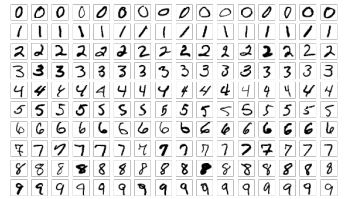


Challenges 1 – 4: Performance Model

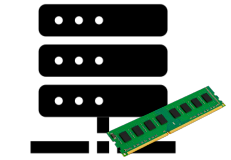
Variable	Unit	Definition
N		Number of nodes
c	MB/s	Compute throughput
b_c	MB/s	Network bandwidth for inter-node communication
b_{fs}	MB/s	Network bandwidth for communication with the PFS
p_j		Number of threads for prefetching to storage class j
d_j	MB	Capacity of storage class j
D	MB	Total local storage of a node
$r_j(p)$	MB/s	Random aggregate read throughput of storage class j (with p reader threads)
$w_j(p)$	MB/s	Random aggregate write throughput of storage class j (with p writer threads)
$t(\gamma)$	MB/s	Random aggregate read throughput (with γ clients) of the PFS
β	MB/s	Preprocessing rate
F		Number of files
s_k	MB	Size of sample k
S	MB	Size of the whole dataset
B		Batch size
E		Number of epochs
T		Number of iterations
B^h		Subset of samples that are processed at iteration h
$B^{h,i}$		Subset of samples that are processed by node i at iteration h
R		Access pattern of a node

Performance Model: Scenarios

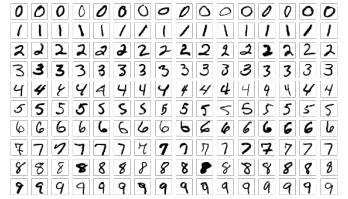
1. Whole dataset fits into memory of a node



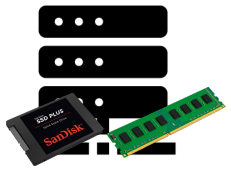
<



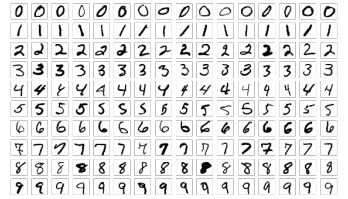
2. Dataset fits into aggregated storage of a node



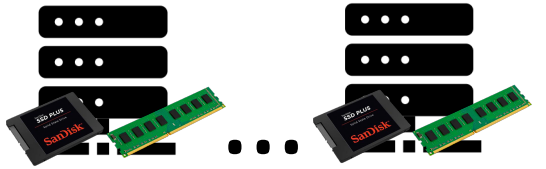
<



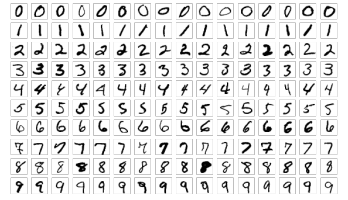
3. Dataset is smaller than aggregated storage of all nodes



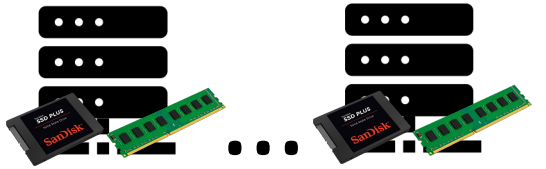
<



4. Larger datasets



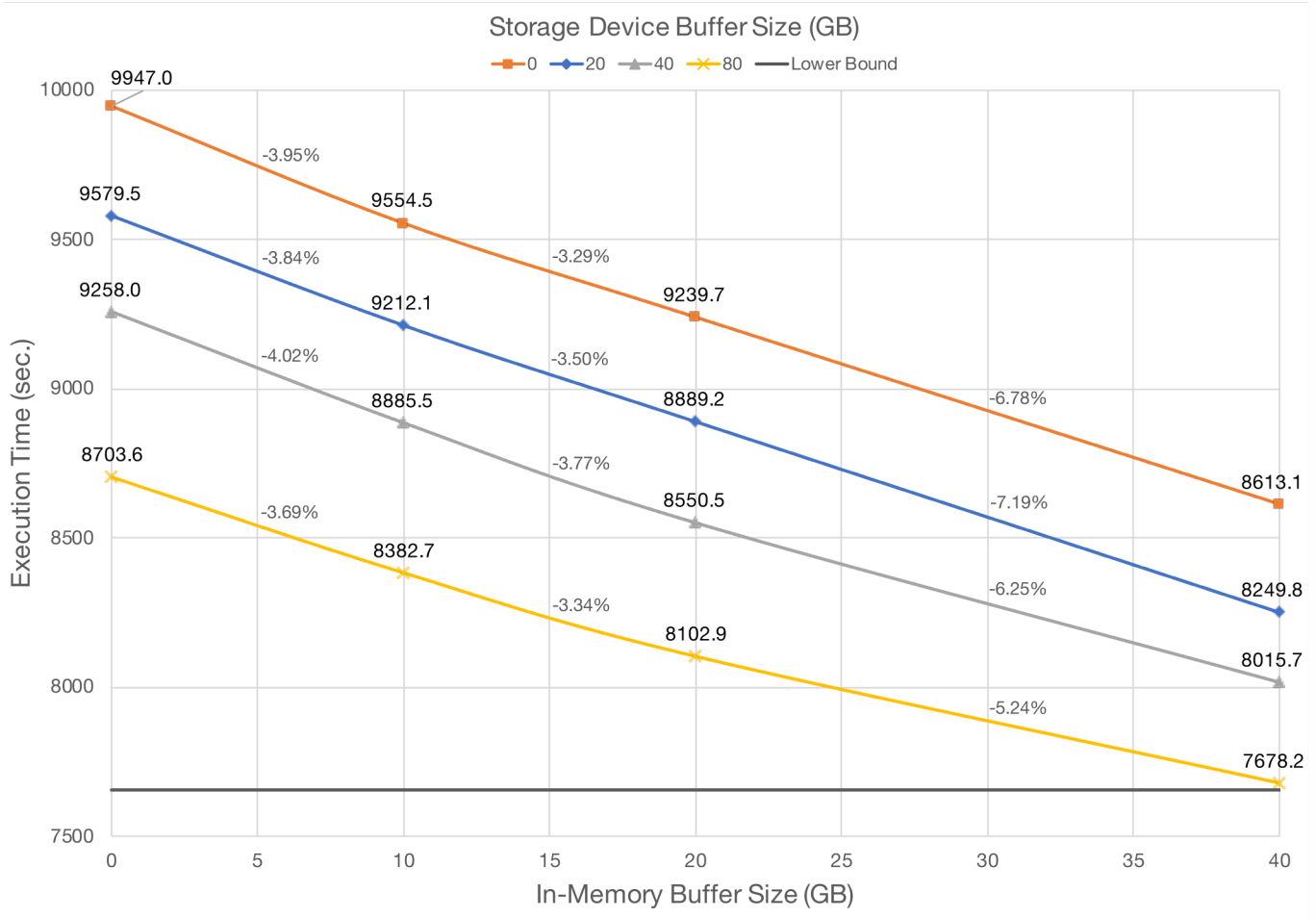
>



Performance Simulation: Design Validation & Environment Evaluation

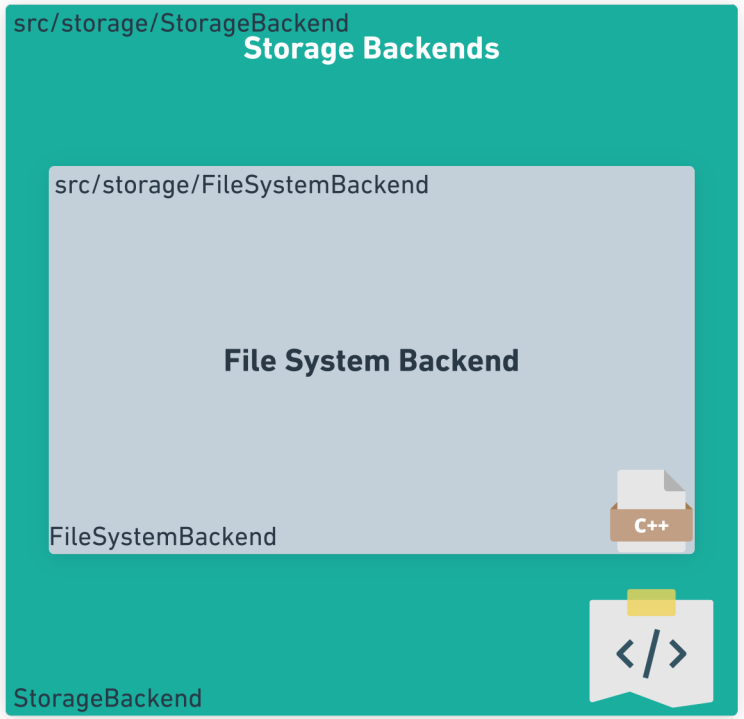
Policy	Runtime (s)
Lower Bound	3,825.66
HDMLP	**3,836.12**
LBANN In Memory Store (Dynamic)	4,051.12
DeepIO (Opportunistic)	4,432.90
DeepIO (Ordered)	4,433.27
LBANN In Memory Store (Preloading)	4,433.27
Locality-Aware Data Loading	4,433.27
Parallel Data Staging	4,972.65
StagingPoolPrefetcher	4,982.27
Naive	15,130.54

Scenario 2 Results.



System Implementation: Challenge 1 (Dataset Scalability)

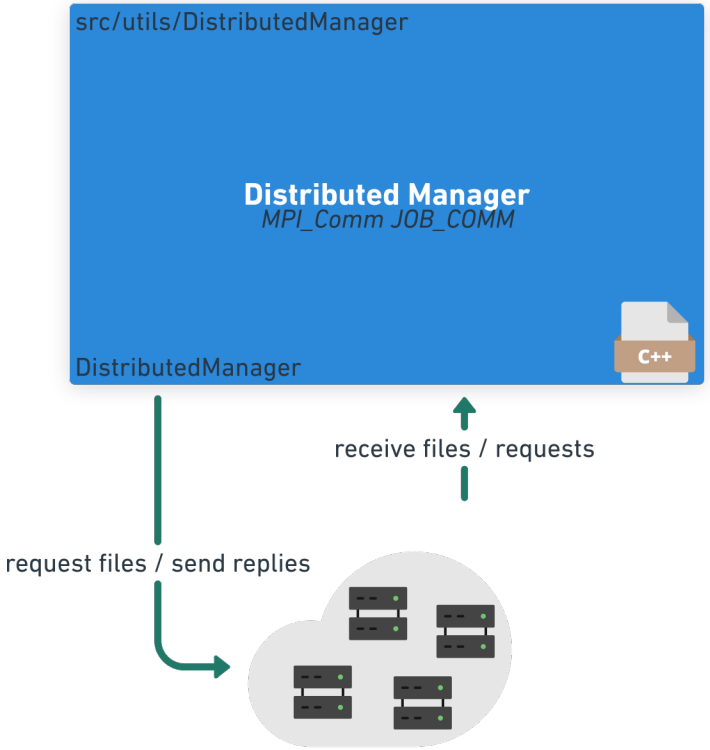
- **Arbitrary storage backends with minimal assumptions**

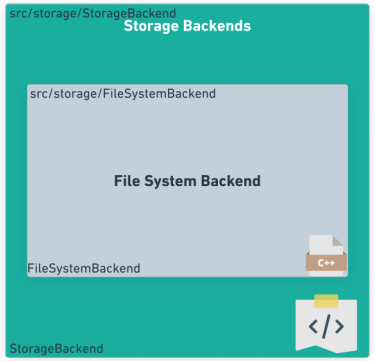


https://github.com/spcl/hdmlp

System Implementation: Challenge 2 (Node Scalability)

- **Distributed caching without metadata messages**

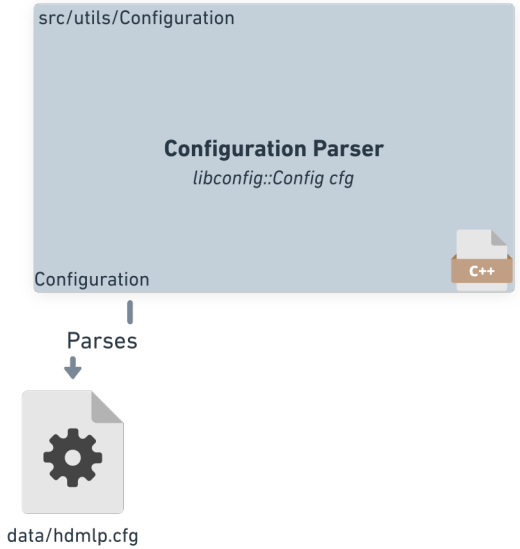


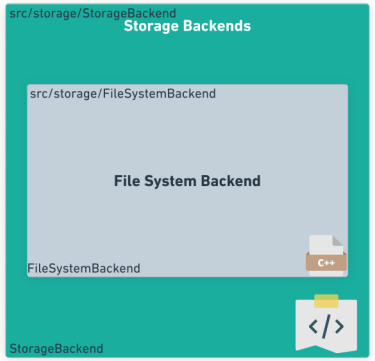


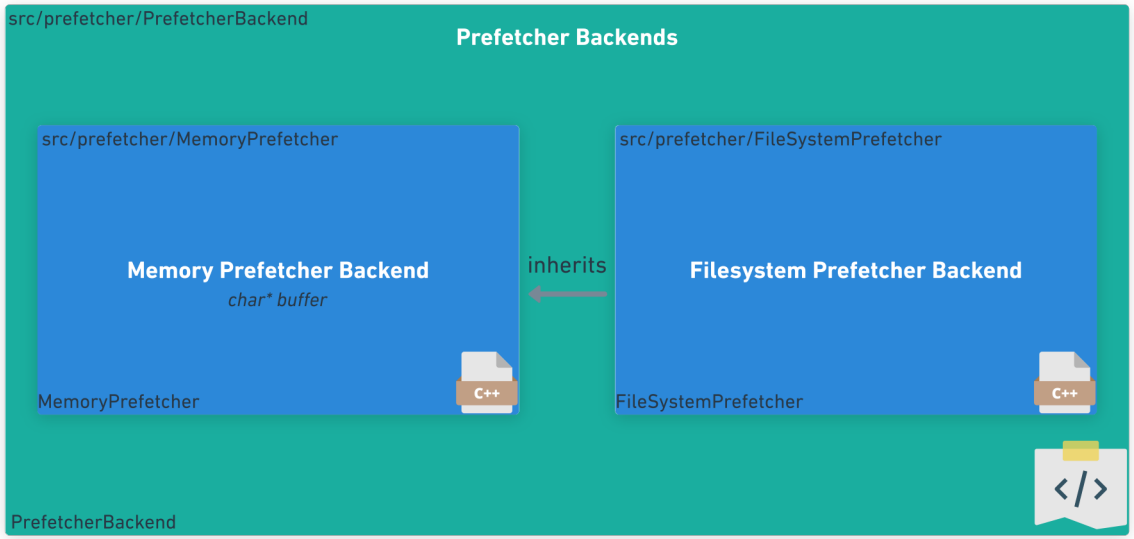
System Implementation: Challenge 3 (Configuration Independence)

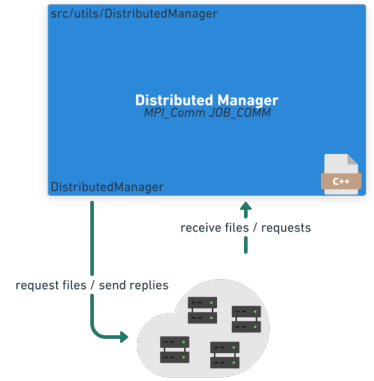
- Configurable and extensible prefetcher backends

```
b_c = 10000 # Network bandwidth for node communication [MB/s]
b_fs = 2000 # Network bandwidth for FS communication [MB/s]
distributed_threads = 2 # Number of threads for file distribution
checkpoint = true # Store label / size information in checkpoint file
checkpoint_path = "/tmp"
storage_classes: ( {
        capacity: 1024 # [MB]
        threads: 4 # Number of threads for prefetching
        bandwidth: (
            { threads = 1; bw = 16550; },
            { threads = 2; bw = 21164; },
            { threads = 3; bw = 21186; },
            { threads = 4; bw = 21415; }
        );
},
{
        capacity: 2048 # [MB]
        backend: "memory"
        threads: 2 # Number of threads for prefetching
        bandwidth: (
            { threads = 1; bw = 16550; },
            { threads = 2; bw = 21164; },
            { threads = 3; bw = 21186; },
            { threads = 4; bw = 21415; }
        );
},
{
        capacity: 10240 # [MB]
        backend: "filesystem"
        backend_options: {
            path = "/tmp/hdmlp";
        }
        threads: 2 # Number of threads for prefetching
        bandwidth: (
            { threads = 1; bw = 16550; },
            { threads = 2; bw = 21164; },
            { threads = 3; bw = 21186; },
            { threads = 4; bw = 21415; }
        );
});
```





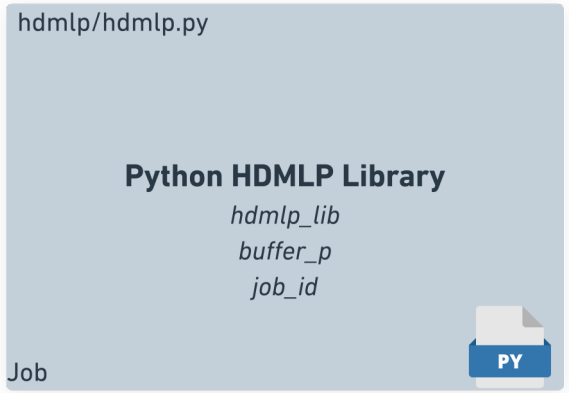




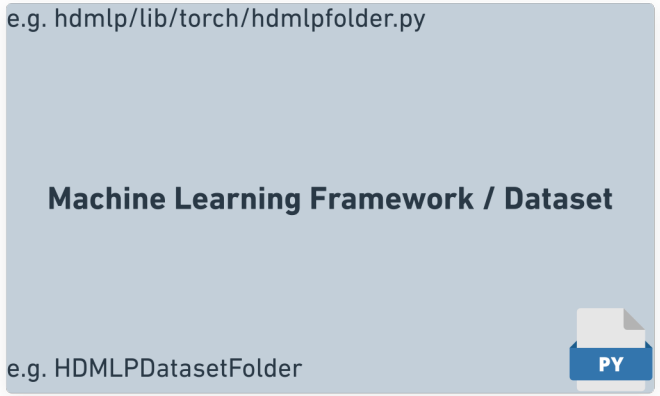
HDMLP https://github.com/spcl/hdmlp

System Implementation: Challenge 5 (Ease of Use)

- **Easy-to-use Python library**



- **PyTorch datasets / dataloader**




```
dataset = ImageFolder(data_dir, data_transforms)
dsampler = DistributedSampler(dataset, num_replicas=n, rank=node_id)
dataloader = DataLoader(dataset, batch_size, sampler=dsampler)
```

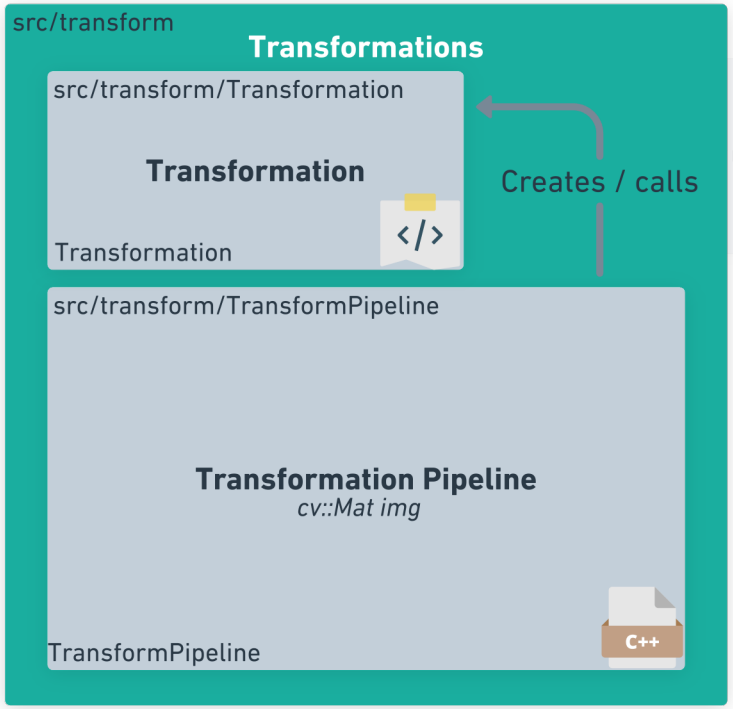
PyTorch

```
job = Job(data_dir, batch_size, num_epochs, 'uniform', drop_last_batch)
dataset = HDMLPImageFolder(data_dir, job, data_transforms)
dataloader = HDMLPDataLoader(dataset)
```

HDMLP

HDMLP https://github.com/spcl/hdmlp

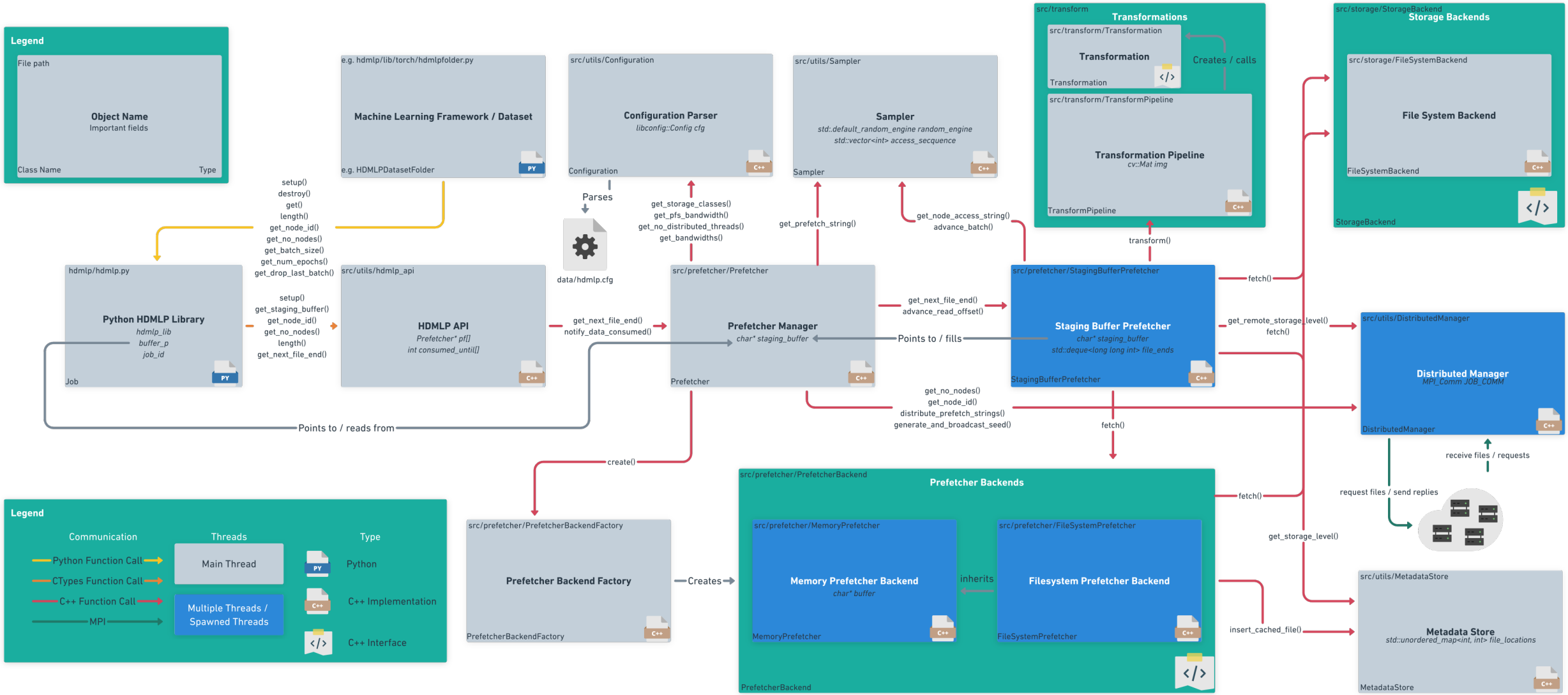
System Implementation: Zero-Copy Transformation Pipeline




```
 9  void Resize::transform(TransformPipeline* pipeline) {
10      cv::resize(pipeline->img, pipeline->img, dsize: cv::Size(width, height));
11  }
```

HDMLP https://github.com/spcl/hdmlp

System Implementation

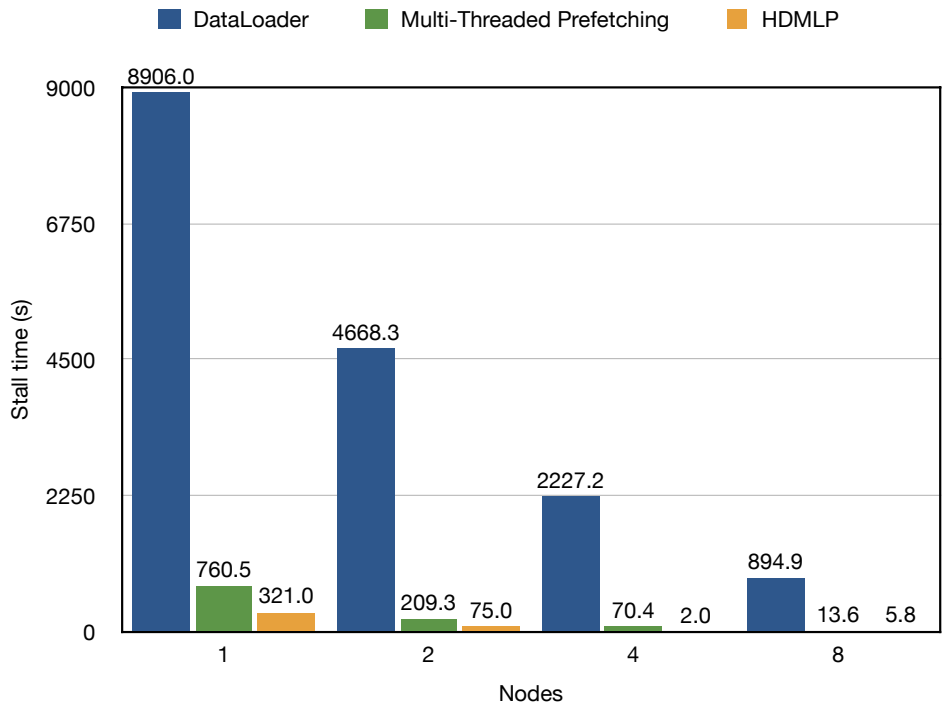


https://github.com/spcl/hdmlp

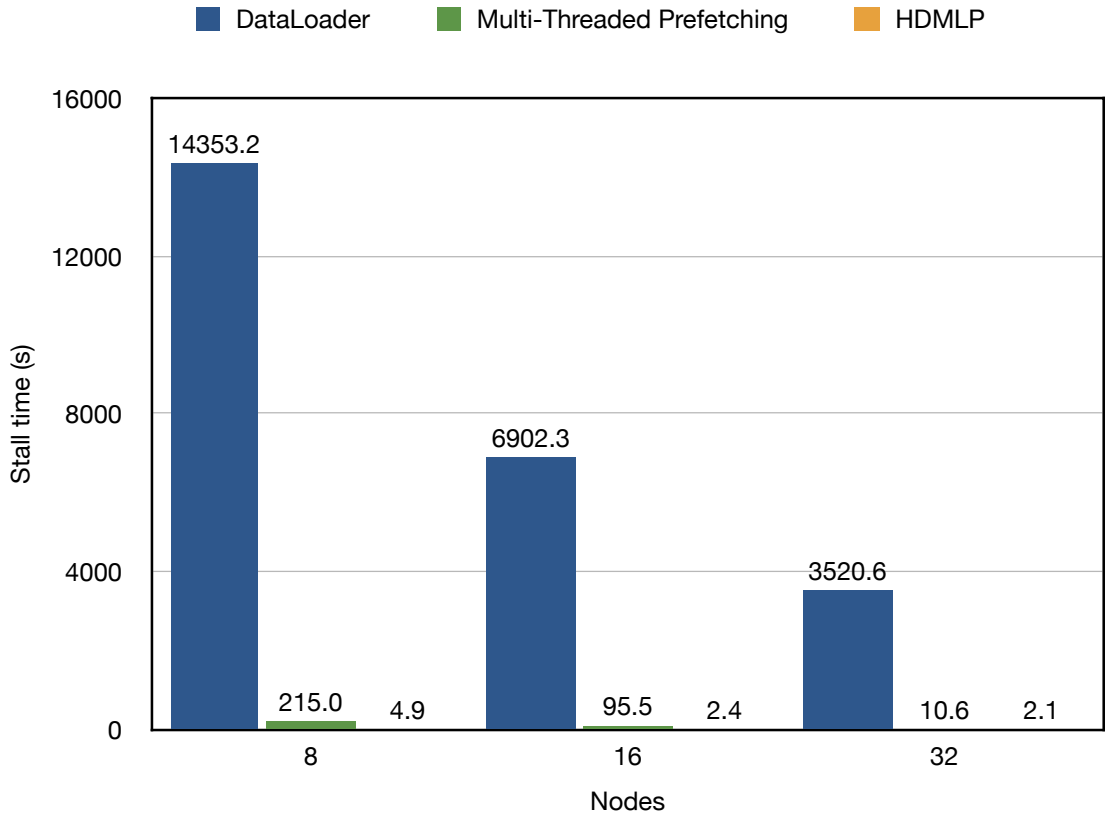
Evaluation

Nodes	Speedup	Strong Scaling Speedup	Strong Scaling Efficiency
1	2.433	1	1
2	2.408	2.005	1.002
4	4.057	7.192	1.798
8	2.147	7.640	0.955
16	2.298	17.987	1.124
32	2.433	40.146	1.255
64	2.476	85.591	1.337

Speedup, Strong Scaling Speedup, and Efficiency of HDMLP.



ResNet-50 ImageNet (10%) Classification Benchmark Results.



ResNet-50 ImageNet Classification Benchmark Results.

Discussion, Next Steps

https://github.com/spcl/hdmlp

Approach	Dataset Scalability	Node Scalability	Configuration Independence	Model Accuracy	Ease of Use
tf.data with `shuffle/prefetch` [12]	✓	✗	✗	✗	✓
Double Buffering	✓	✗	✗	✓	✓
DeepIO [47]	✓	✓	✗	✗[a]	✓
Parallel Data Staging [28]	✓	✓	✗	✗	✗
LBANN Distributed Data Store [21]	✗	✓	✗	✓	✗
Locality-Aware Data Loading [44]	✓	✓	✗	✓[b]	?[c]
HDMLP	✓	✓	✓	✓	✓

- **Provide loaders for other ML frameworks**
- **Extend HDMLP (storage backends, prefetcher backends, transformations, …)**