OAK RIDGE National Laboratory

CCGRID 2026

Tula: Optimizing Time, Cost, and Generalization in Distributed Large-Batch Training

Sahil Tyagi and Feiyi Wang

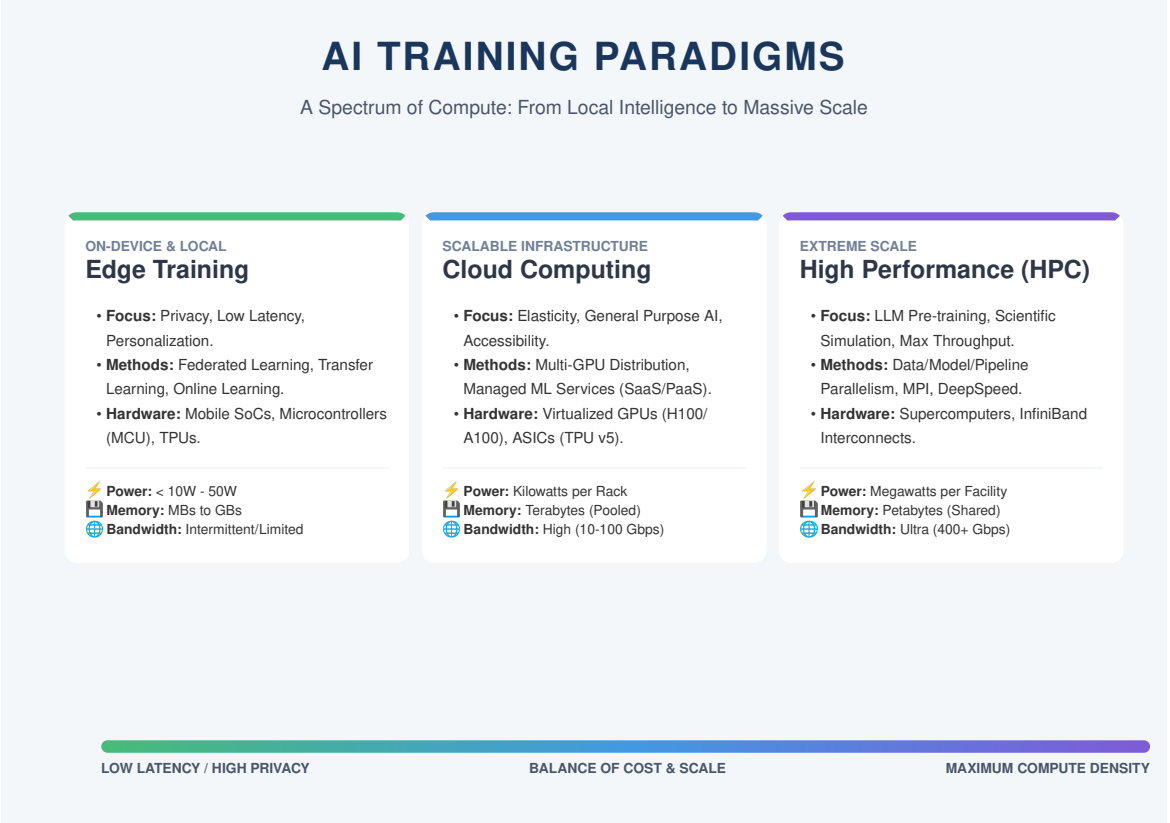
National Center for Computational Sciences (NCCS)

Oak Ridge National Laboratory (ORNL), USA

Introduction

Current Practices

- Large models trained/ fine-tuned over vast datasets
- Leverage different parallelism techniques as needed
- Span the entire compute continuum



* Image generated via AI

Background and Challenges

Performance Trade-offs in Distributed Training

- Despite its advantages, there are conflicting trade-offs
  - Parallel performance efficiency
  - Statistical performance efficiency

- Data-parallelism follows a pareto-relationship

* Image generated via AI

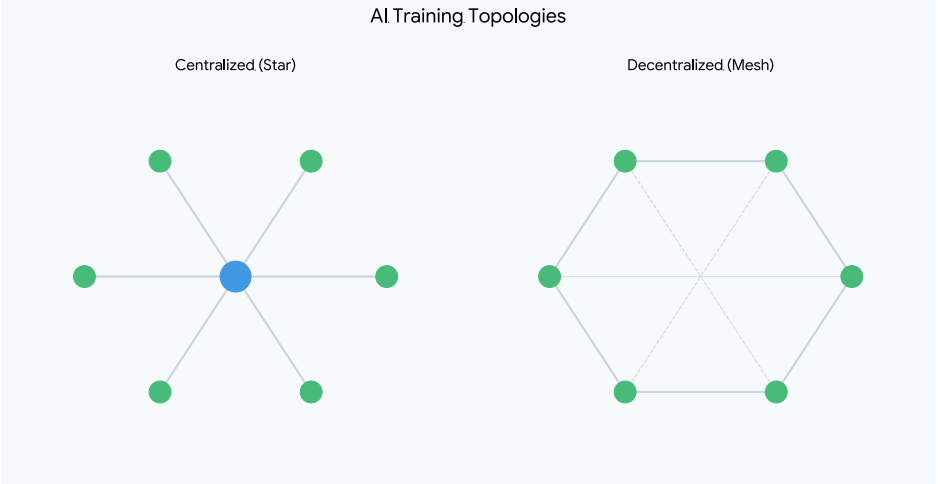
Parallel Performance Efficiency

- SGD-based optimization is iterative

$$w_{(n)}^{(i+1)} = w_{(n)}^{(i)} - \eta \cdot \frac{1}{N} \sum_{n=1}^{N} \left(\frac{1}{|b|} \sum_{b_{(i)} \in \mathcal{B}_{(i)}^{(n)}} \nabla f(w_{(n)}^{(i)}, b_{(i)}) \right)$$

- Aggregation mechanism can be centralized or decentralized

- Training scaled **vertically** or **horizontally**
  - *Scale-up:* With larger batches
  - *Scale-out:* With larger clusters



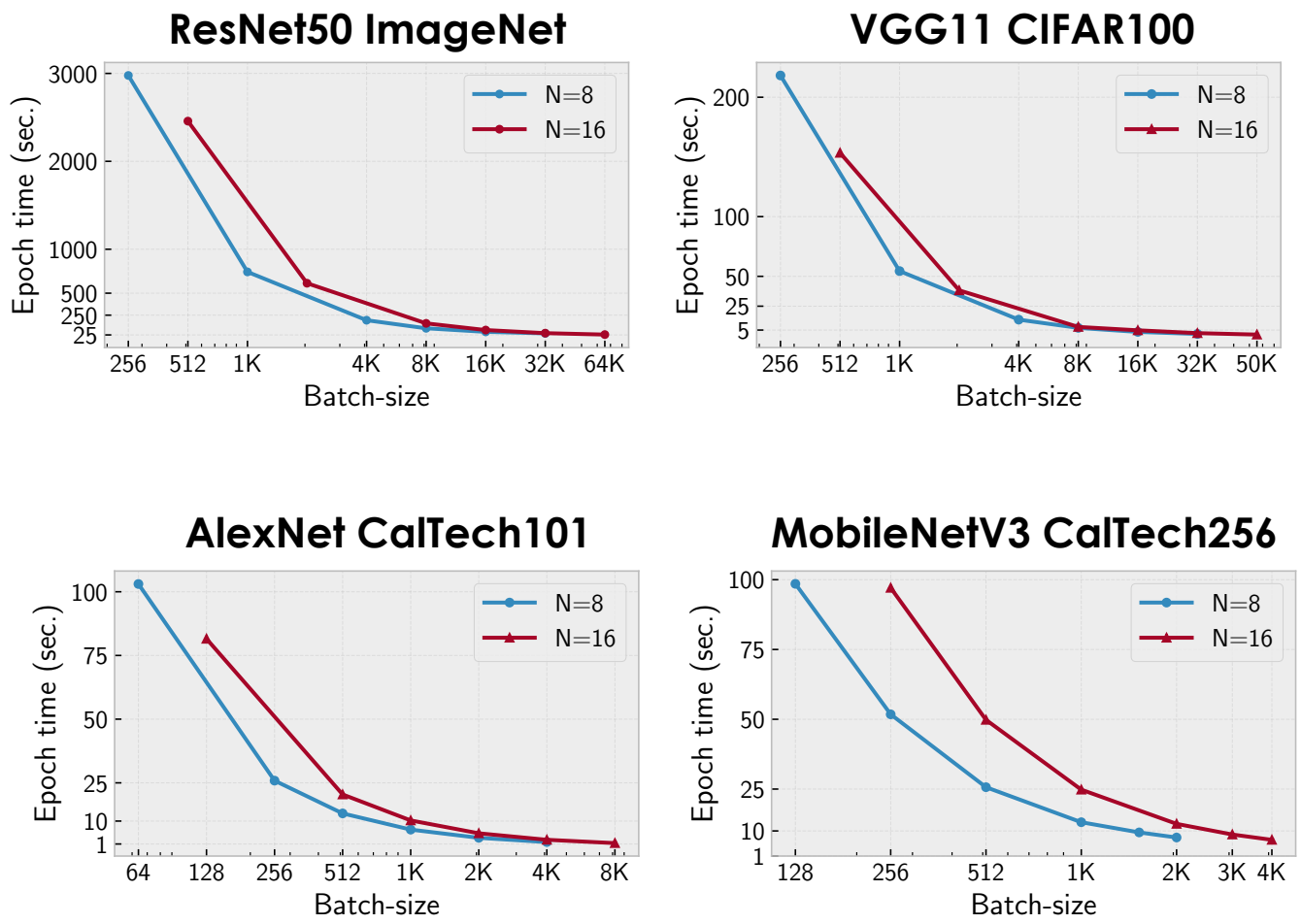
* Image generated via AI

Parallel Performance Efficiency

- Vertical scaling increases SGD computation and memory overhead
- Horizontal scaling may result in additional communication overhead

$$1 \text{ Epoch} = {}^{D}/_{B} \text{ iterations}$$

Distributed training exhibits non-linear scalability and diminishing returns!

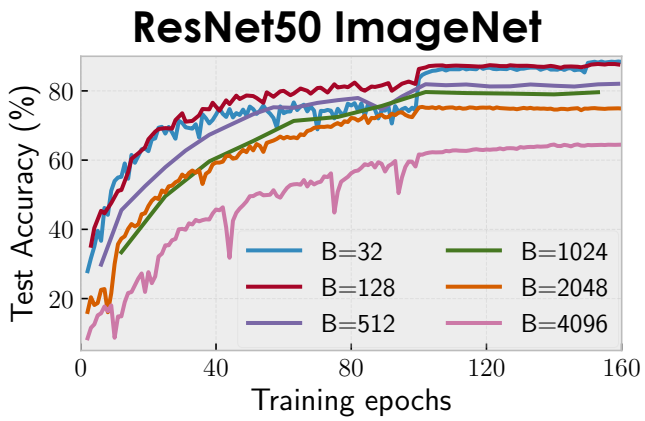


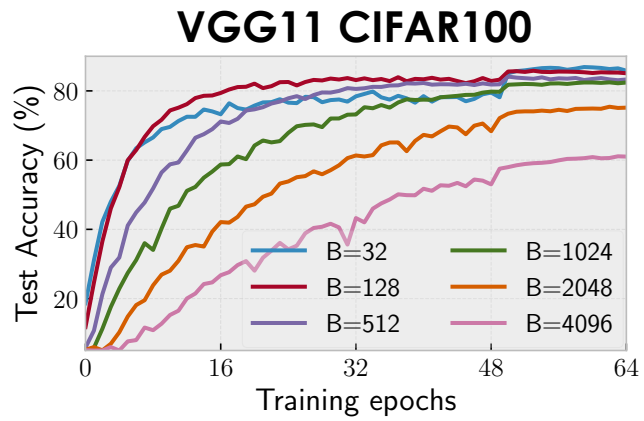
Epoch completion time vs. batch-size

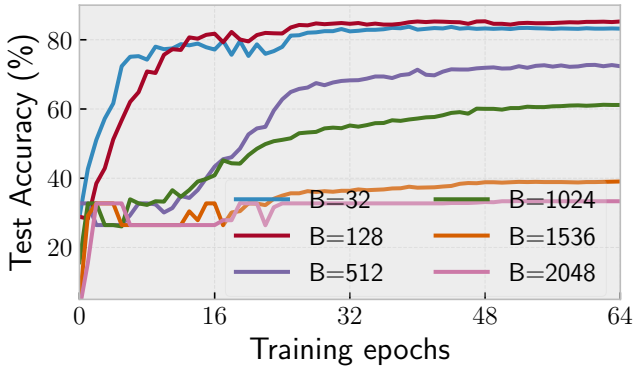
Statistical Performance Efficiency

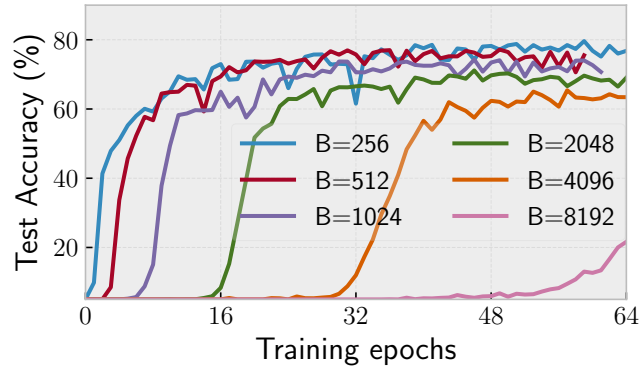
- Large-batch training improves overall throughput and training accuracy/loss
- However, large-batches observe worse performance on unseen or test data

Although large-batches improve parallel efficiency, they can substantially degrade convergence quality!









Generalization loss in large-batch training

Tula: Design Overview

Balancing Parallel & Statistical Performance

CCGRID 2026

- Parallel training in distributed settings yields speedup with diminishing returns
- Optimal training configuration depends on various system and workload characteristics
- Even with an optimal systems configuration, large-batches suffer from generalization gap
- *Tula* jointly optimizes parallel & statistical performance by:
  - Identifying optimal (N, B) configuration
  - Improving generalization with chosen config.

Parallel Performance Model

Parallel Performance Model

- Iterations are identical from a computational standpoint
- Compute cost varies with batch-size
- Synchronization cost may vary with cluster-size
- Under fixed resource-cost model, training cost rises with time

$$t_{step}^{(i)} = t_{compute}^{(i)} + t_{sync}^{(i)}$$

$$t_{comp} \propto b$$

$$t_{sync} \propto \begin{cases} N, & \text{centralized parameter server} \\ (1 - \frac{1}{N}), & \text{ring-allreduce} \end{cases}$$

$$T = I \cdot \tilde{t}_{step} \quad \Rightarrow \quad T = \frac{ED}{B} \cdot \tilde{t}_{step}$$

$$C = T \cdot N \cdot \tilde{p}$$

Parallel Performance Model

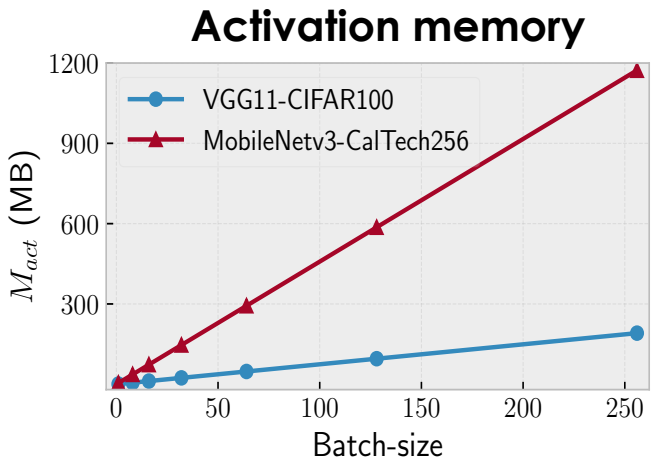
- Despite its gains, large-batches constrained by available memory

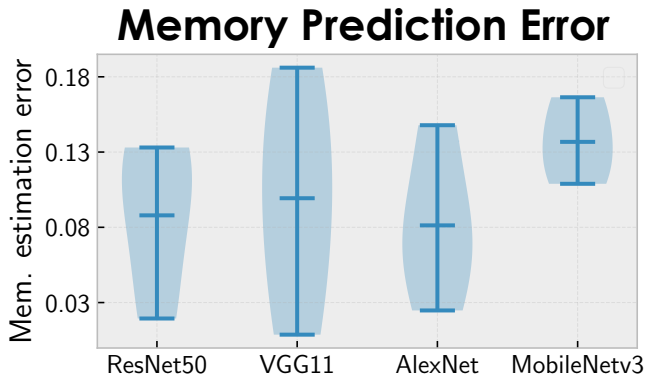
- Memory footprint in training attributed to individual components

- Intra-node large-batches constrained by available memory

$$M_{act}, M_{batch} \propto b$$

$$M_{total} = M_{param} + M_{grad} + M_{opt} + M_{act} + M_{batch}$$





Parallel Performance Model

- Objective: Infer training time/cost and estimate trade-off curves w.r.t. (N, B) configs

$$t_{step}^{(i)} = t_{compute}^{(i)} + t_{sync}^{(i)}$$

$$t_{comp} \propto b$$

$$t_{sync} \propto \begin{cases} N, & \text{centralized parameter server} \\ (1 - \frac{1}{N}), & \text{ring-allreduce} \end{cases}$$

- Perform profiling to log iteration and memory overhead:
  - *Full:* run each config. briefly and log metrics
  - *Partial:* run at extremums and interpolate

$$T = I \cdot \tilde{t}_{step} \quad \Rightarrow \quad T = \frac{ED}{B} \cdot \tilde{t}_{step}$$

$$C = T \cdot N \cdot \tilde{p}$$

- Choose (N, B) that satisfies user-specified objective: minimize time, cost or knee-point

$$M_{total} = M_{param} + M_{grad} + M_{opt} + M_{act} + M_{batch}$$

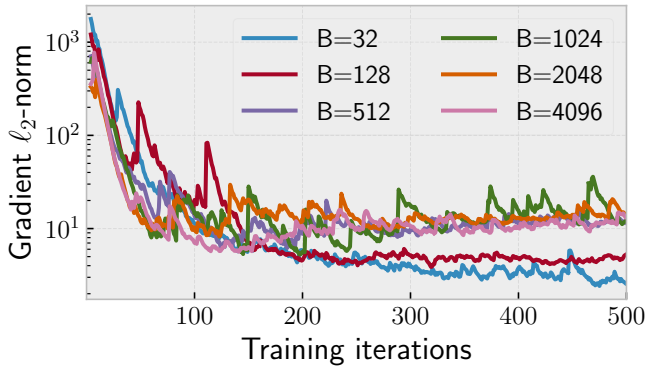
Statistical Performance Model

Gradient Sensitivity over Training

- Gradient trajectory varies throughout training due to stochasticity across batches
- Second-order information like Eigenvalues of the Hessian indicate sensitive phases
- Gradient sensitivity detected by tracking changes in the norm

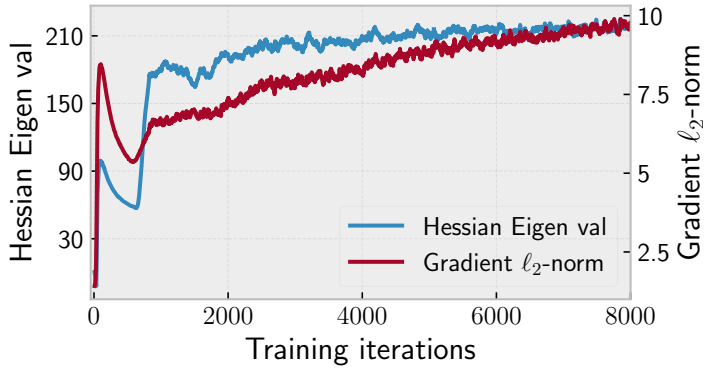
$$\triangle(G_{(b)}^{(i)}) = \left| \frac{|G_{(b)}^{(i)}|^2 - |G_{(b)}^{(i-1)}|^2}{|G_{(b)}^{(i-1)}|^2} \right|$$

ResNet50



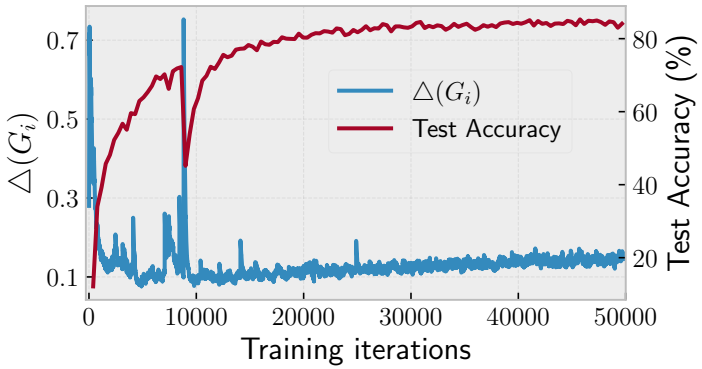
Initial gradient sensitivity

VGG11



Second vs. First-order gradient computation

ResNet50



Rate of gradient change w/ model accuracy

First-order gradients statistics offer a lightweight and accurate mechanism to detect sensitive training phases

Statistical Performance Model

- Initially, large-batch gradients tend to be smoother and less noisy than small batch-size
- Small-batch noise provides implicit regularization
- Variance between small and large-batch updates can be measured via adaptive noise

$$G^{(i)}_{b_{small}} = G^{(i)}_{b_{large}} + \gamma^{(i)}(b_{small}, b_{large})$$

Adaptive noise term

Static Gradient Scaling

- Mapping mechanism $\mathcal{M}$ transforms large-batch updates (to small-batch scale)

$$G_{b_{large}}^{(i)} = \frac{1}{|b_{large}|} \nabla f(w^{(i)}, b_{large})$$

$$\widetilde{G}^{(i)} = \mathcal{M}(G_{b_{large}}^{(i)}), \quad w^{(i+1)} = w^{(i)} - \eta \cdot \widetilde{G}^{(i)}$$

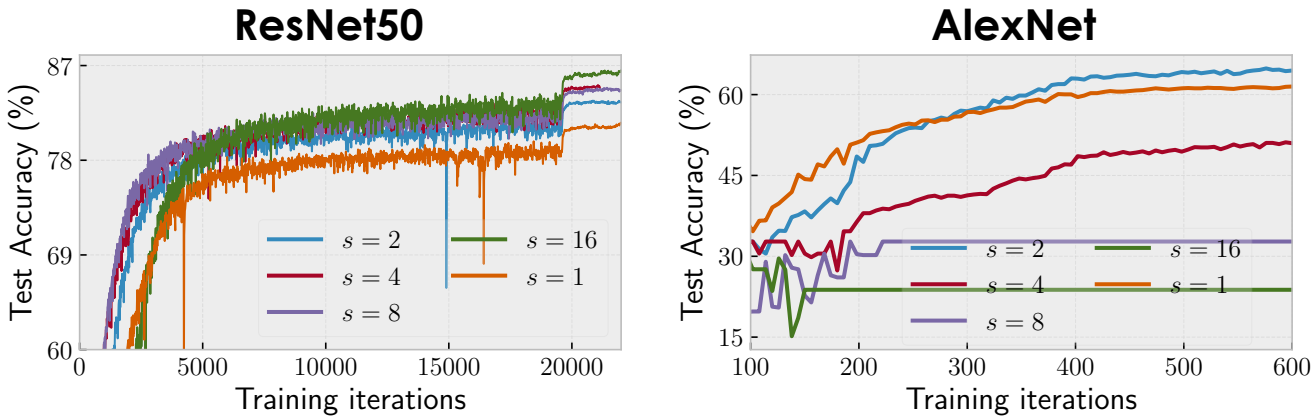
- But what about gradient sensitivity and critical phases?

$$\mathcal{U}(s) = \begin{cases} 1, & \text{sensitive phase} \\ s, & \text{otherwise} \end{cases}$$

$$\widetilde{G}^{(i)} = \mathcal{U}(s) \odot G_b^{(i)}$$

Apply mapping mechanism *out* of critical/sensitive phases!

- Example: Naive approach where updates are statically scaled by *s*



Model convergence quality w/ static scaling

Statistical Performance Model

- Static scaling may converge in some cases and degrade in others
- Scale of updates should adjust dynamically for every training parameter
- Under Lipschitz smooth loss function and bounded scaling factors $s \in [s_{min}, s_{max}]$, AGS converges in $\boldsymbol{O}\left(\frac{1}{\sqrt{I}}\right)$ when learning-rate decays $\propto 1/\sqrt{I}$ for $\boldsymbol{I}$ iterations

Algorithm 1: Adaptive Gradient Scaling (AGS)

Input: Batches (b_{small}, b_{large}), threshold δ, LR η, stability term $\epsilon = 10^{-8}$, *gradscale* $\leftarrow \mathbf{1}$ (per-parameter)

for *epoch* $e = 1, 2, \ldots$ **do**
  for *iteration* $i = 1, 2, \ldots$ **do**
    $G^{(i)}_{b_{large}} \leftarrow \nabla f(w^{(i)}, b^{(i)}_{large})$
    Compute $\Delta(G^{(i)})$ using Eq. (5)
    if $\Delta(G^{(i)}) < \delta$ **then**
      $\widetilde{G}^{(i)} \leftarrow gradscale \odot G^{(i)}_{b_{large}}$
    else
      $\widetilde{G}^{(i)} \leftarrow G^{(i)}_{b_{large}}$
    $w^{(i+1)} \leftarrow w^{(i)} - \eta \widetilde{G}^{(i)}$
  gradscale $\leftarrow$ UPDATEGRADSCALE()
function UPDATEGRADSCALE()
  $s_{max} \leftarrow \sqrt{b_{large}/b_{small}}$
  Sample $b^{(i)}_{small} \subset b^{(i)}_{large}$, $|b_{small}| < |b_{large}|$
  $G^{(i)}_{b_{small}} \leftarrow \nabla f(w^{(i)}, b^{(i)}_{small})$
  foreach *parameter* (g_{small}, g_{large}) **do**
    $s \leftarrow \min\left(\left|\frac{g_{small}}{g_{large}+\epsilon}\right|, s_{max}\right)$
    $gradscale.key(p) \leftarrow s$
  return *gradscale*

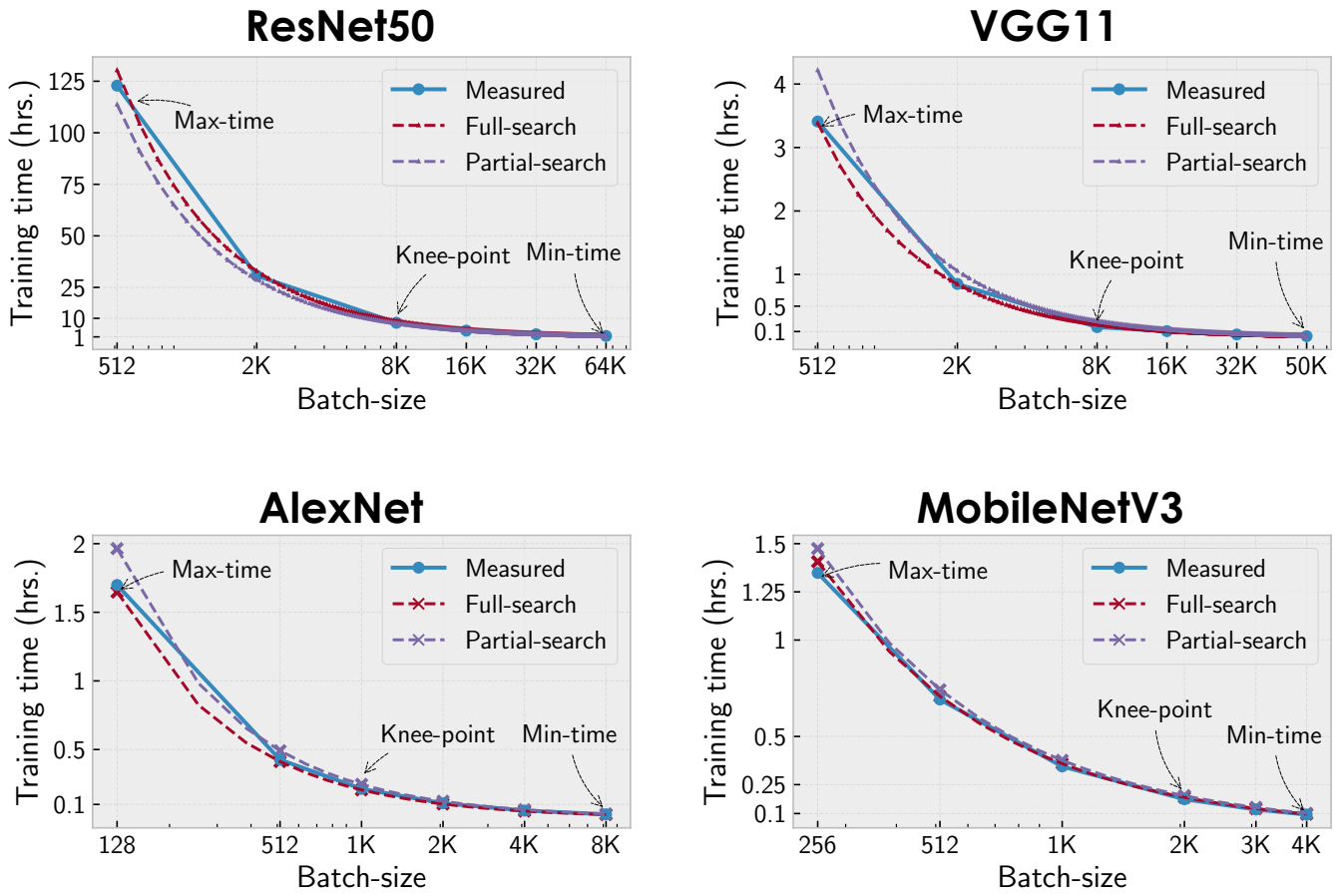
Experimental Evaluation

Implementation and Setup

- Implement over PyTorch 2.6.0
- Tested on up to 16 nodes w/ Intel Xeon CPU, 128GB memory and 32GB V100S GPU, connected via 10Gbps ethernet on NCCL 2.21.5
- Evaluate 4 models: ResNet50, VGG11, AlexNet and MobileNetV3 on 4 datasets: ImageNet-1K, CIFAR100, CalTech101 and CalTech256.
- In Tula, min-max clusters set to [2,16], with configurations scaled exponentially: [2,4,8,16], min. batch-size 32, max. batch predicted w/ memory estimation model

Estimating Training Time and Cost

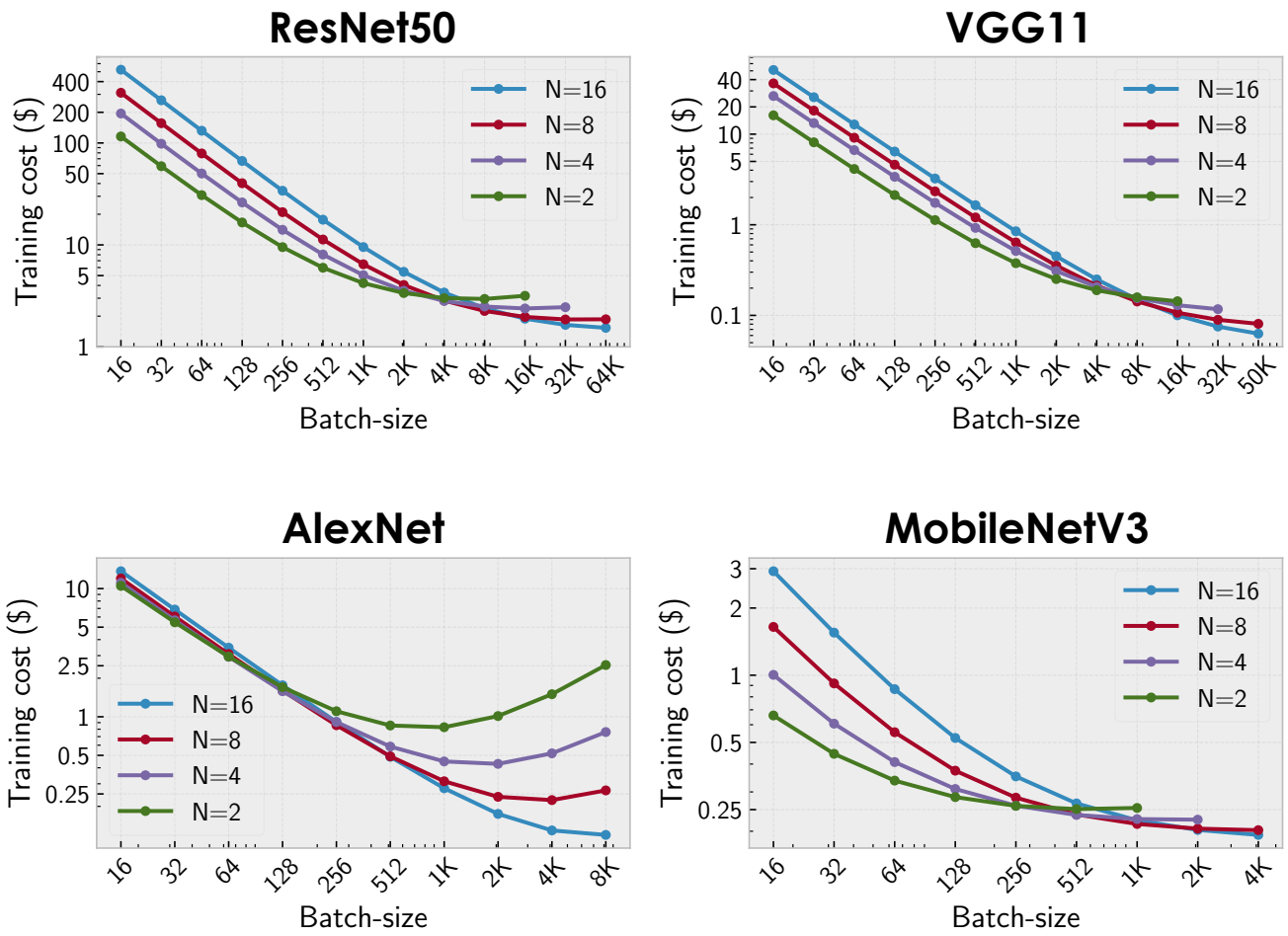
- Infer training time in *full* and *partial* search mode; compare with actual performance
- **Predicts training time within 11% and 20% error respectively**
- Diminishing returns observed for batches beyond the knee-point
- Fixed price model assumes resource cost is fixed



Training time vs. Batch-size over 16 nodes

Estimating Training Time and Cost

- Present an alternate pricing model
- Cost based on resource utilization – GPU memory
- In contrast to fixed pricing, cost rises with larger batches under pricing based on memory utilization



Training cost over memory-based pricing model

Overall Convergence Quality

- Evaluate convergence quality with adaptive gradient scaling around knee-points

- Compared with baseline large-batch training (LBT), layerwise adaptive-rate scaling (LARS), post-local SGD (PoLo) and learning-rate scaling (LRS)

Model	Batch-size	Test accuracy (%)				
		LBT	LARS	PoLo	LRS	*Tula*
ResNet50	1024	79.67	83.26	74.03	83.33	*85.28*
	2048	75.55	85.02	65.58	74.44	*84.58*
	4096	64.72	86.77	62.46	70.11	*82.16*
VGG11	1024	83.30	52.93	82.90	7.50	*84.42*
	2048	75.92	50.91	81.13	5.00	*82.58*
	4096	70.24	48.64	79.50	5.00	*81.14*
AlexNet	256	80.62	55.40	85.11	83.38	*84.44*
	512	72.80	56.00	85.95	72.30	*83.00*
	1024	62.60	57.00	85.67	32.74	*81.20*
MobileNetv3	1024	76.80	67.64	81.14	78.66	*81.56*
	2048	71.15	72.82	75.54	10.00	*79.68*
	4096	66.48	74.29	73.91	5.00	*75.70*

Top-1 Test accuracy in baseline vs. different large-batch optimization schemes

Tula converges consistently across models and batches

Limitations and Discussion

Limitations + Future Directions

- Full and partial search incur non-negligible overhead under vast exploration space
- Assumes stable communication without congestion
- Simplistic cost models; cost estimation may be non-trivial w/ spot instances or heterogeneous hardware
- Convergence behavior of adaptive gradient scaling depends on δ, b_{small} and frequency of parameter scaling op.
- Currently evaluates convolutional networks with vision-based workloads – extend to LLMs and GNNs
- Explore other knowledge-distillation mechanisms to map large-to-small batch gradient space – e.g., autoregressors, transformers, autoencoders

Conclusion

- Efficient distributed training must balance resource cost, parallel and statistical efficiency – Tula develops an online, profiling-driven method to model training performance across (N, B) configs.

- Parallel performance model captures the relative trend b/w different configs.

- With optimal batch-size, Tula applies gradient scaling to close generalization gap

- Tula attains up to 20x convergence speedup over a naively chosen config.

- Gradient scaling improves accuracy by up to 8.8% over vanilla large-batch training

Thank you!

For additional questions/discussion, please contact tyagis@ornl.gov