The Scalability and Security in Federated Leaning

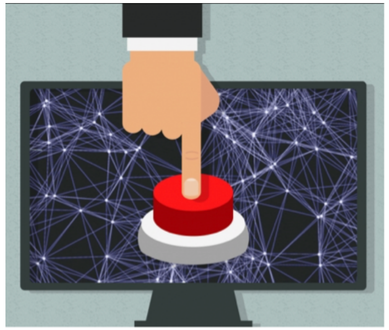
Ligeng Zhu

ligeng.zhu@gmail.com

Brief Bio

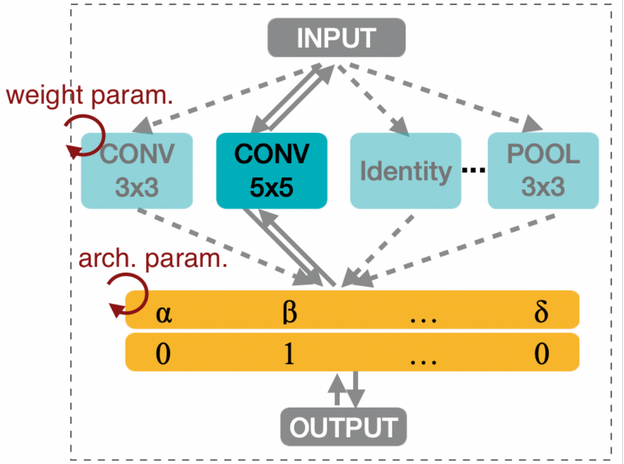
- CS Ph.D. applicant this year.
- Obtained B.Sc from Zhejiang University and Simon Fraser University in 2019.
- Visit MIT (host: Song Han) in 18-19.
- Currently work at Intel AI Labs as a Data Scientist.
- Research interests include automated and scalable machine learning system.
- Enthusiastic about open source

AutoML accelerates Inference on Edge

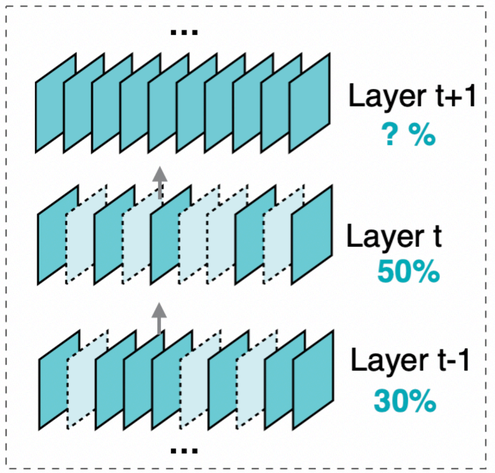


All codes are public at

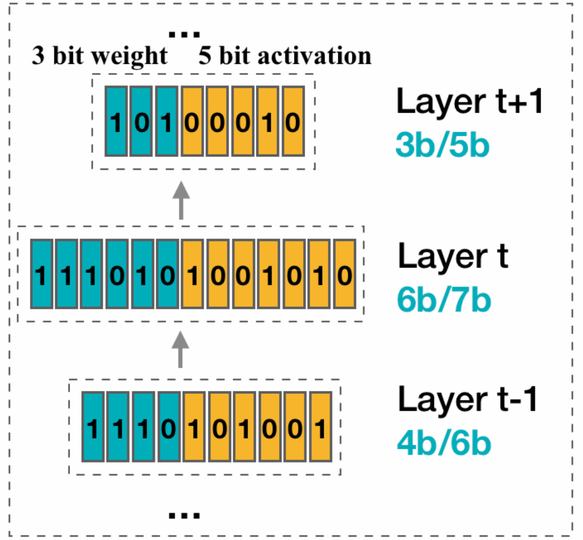
https://github.com/MIT-HAN-LAB



Proxyless Neural Architecture Search
[ICLR 2019]

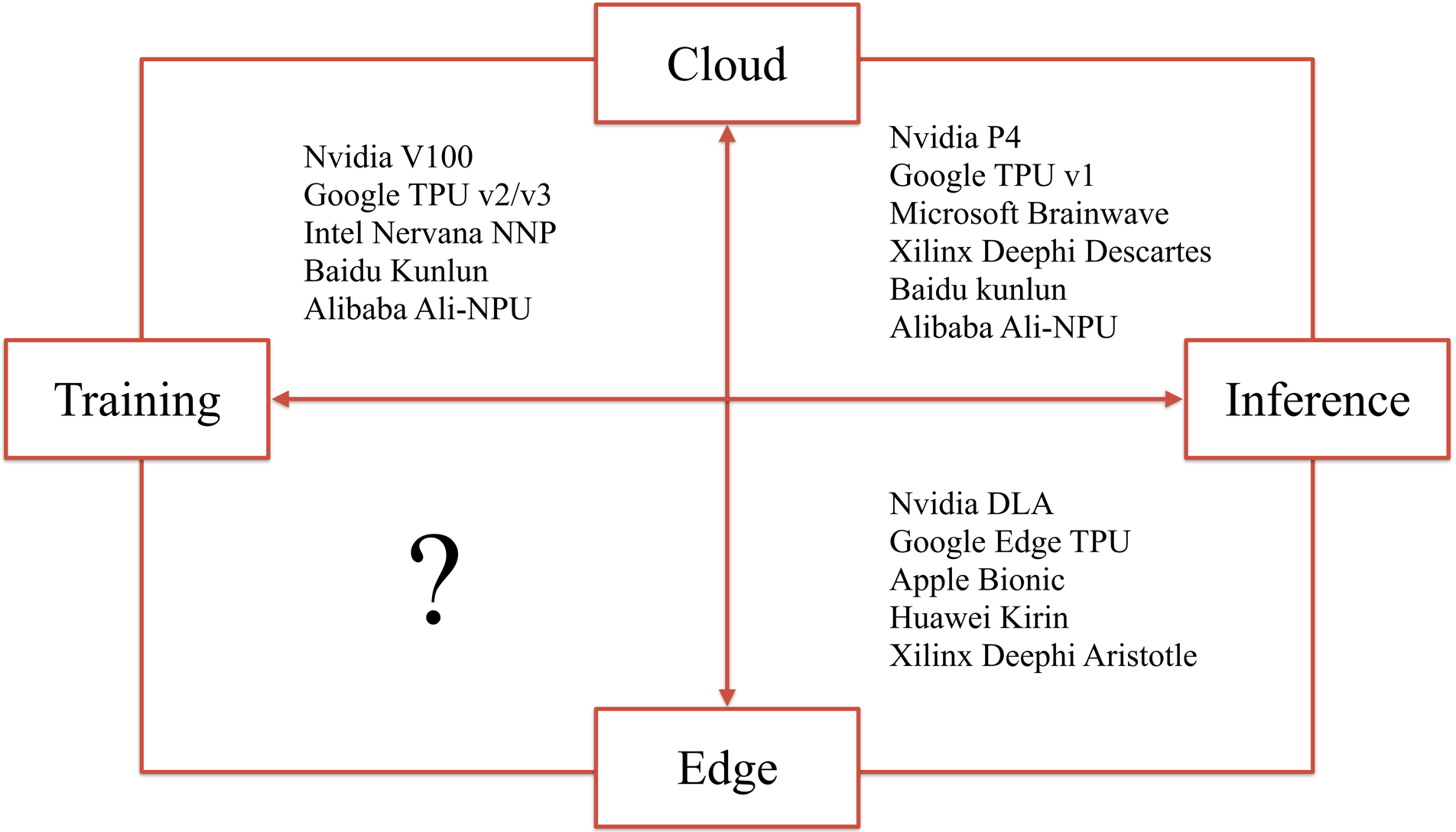


AMC: AutoML for Model Compression
[ECCV 2018]



HAQ: Hardware-aware Automated Quantization
[CVPR 2019] Oral

- First place in the Low Power Computer Vision Challenge (LPCVC), DSP track at ICCV'19
- First place in the LPCVC'20 (both classification and detection track)


Cloud
Nvidia V100
Google TPU v2/v3
Intel Nervana NNP
Baidu Kunlun
Alibaba Ali-NPU
Nvidia P4
Google TPU v1
Microsoft Brainwave
Xilinx Deephi Descartes
Baidu kunlun
Alibaba Ali-NPU
Training
Inference
?
Nvidia DLA
Google Edge TPU
Apple Bionic
Huawei Kirin
Xilinx Deephi Aristotle
Edge

Why learning on the edge?

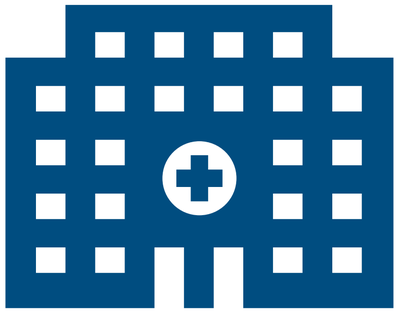
- Customization
  - I.e., Different users will have a different tone for speech recognition

Amazon
Alexa

Apple
Home Pod

Google
Home

- Security
  - Data cannot leave device because of security and regularization.



Why leaning on the edge?

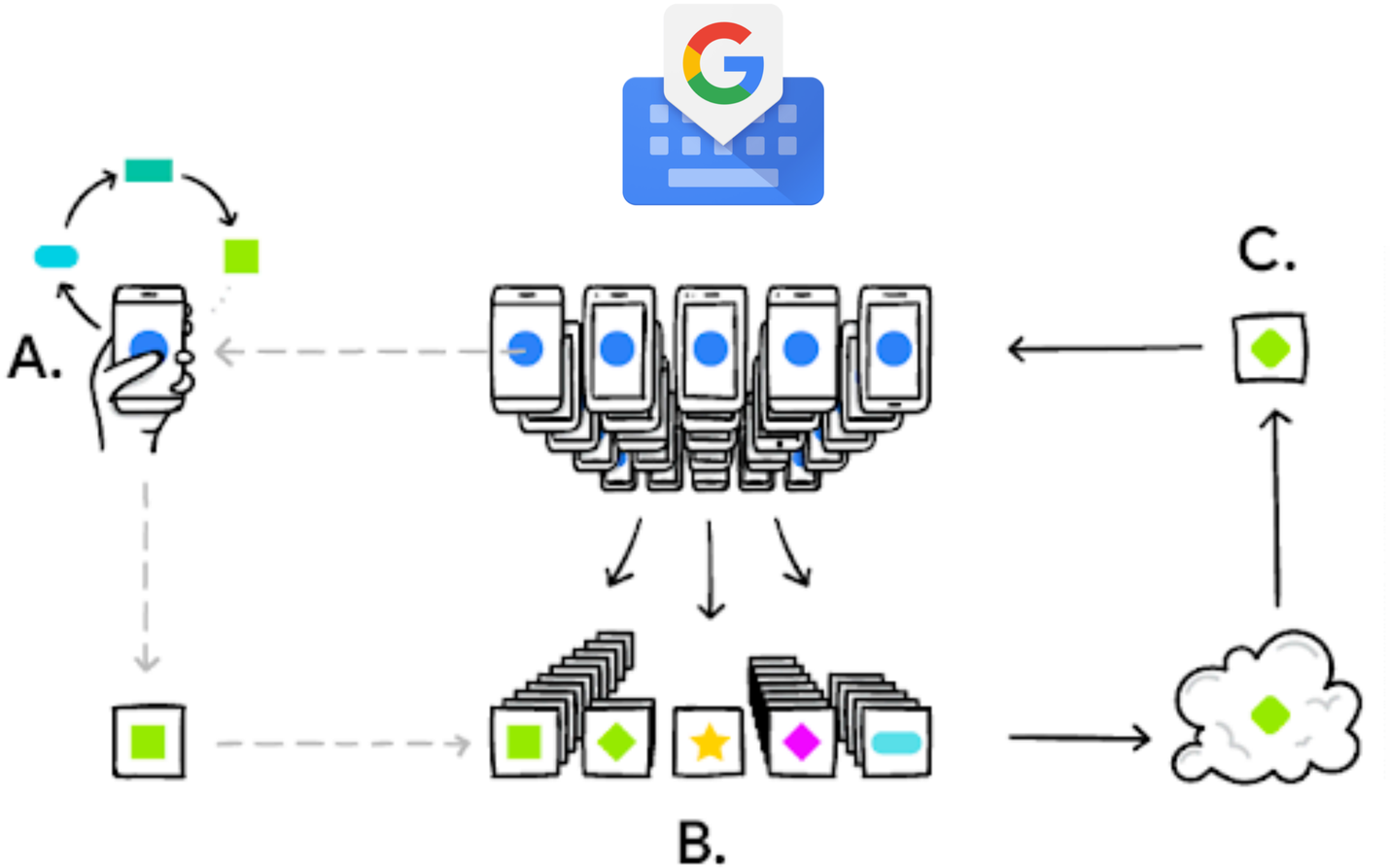
Because of **DATA**.

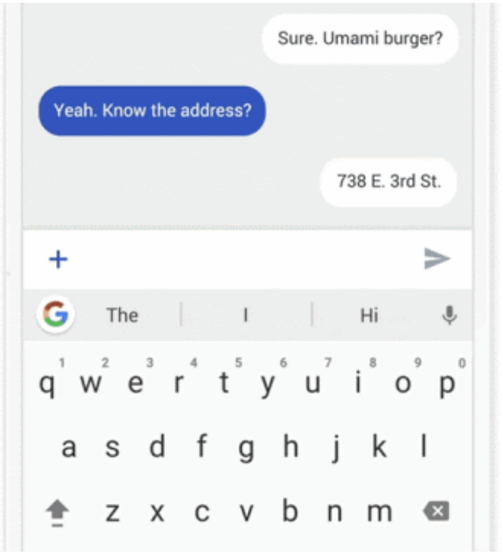
> It's not who has the best algorithm that wins.
> It’s who has the most data”
>
> —Andrew Ng

Data is everywhere (but isolated).

Federated learning provides a way to utilize them without centralizing.

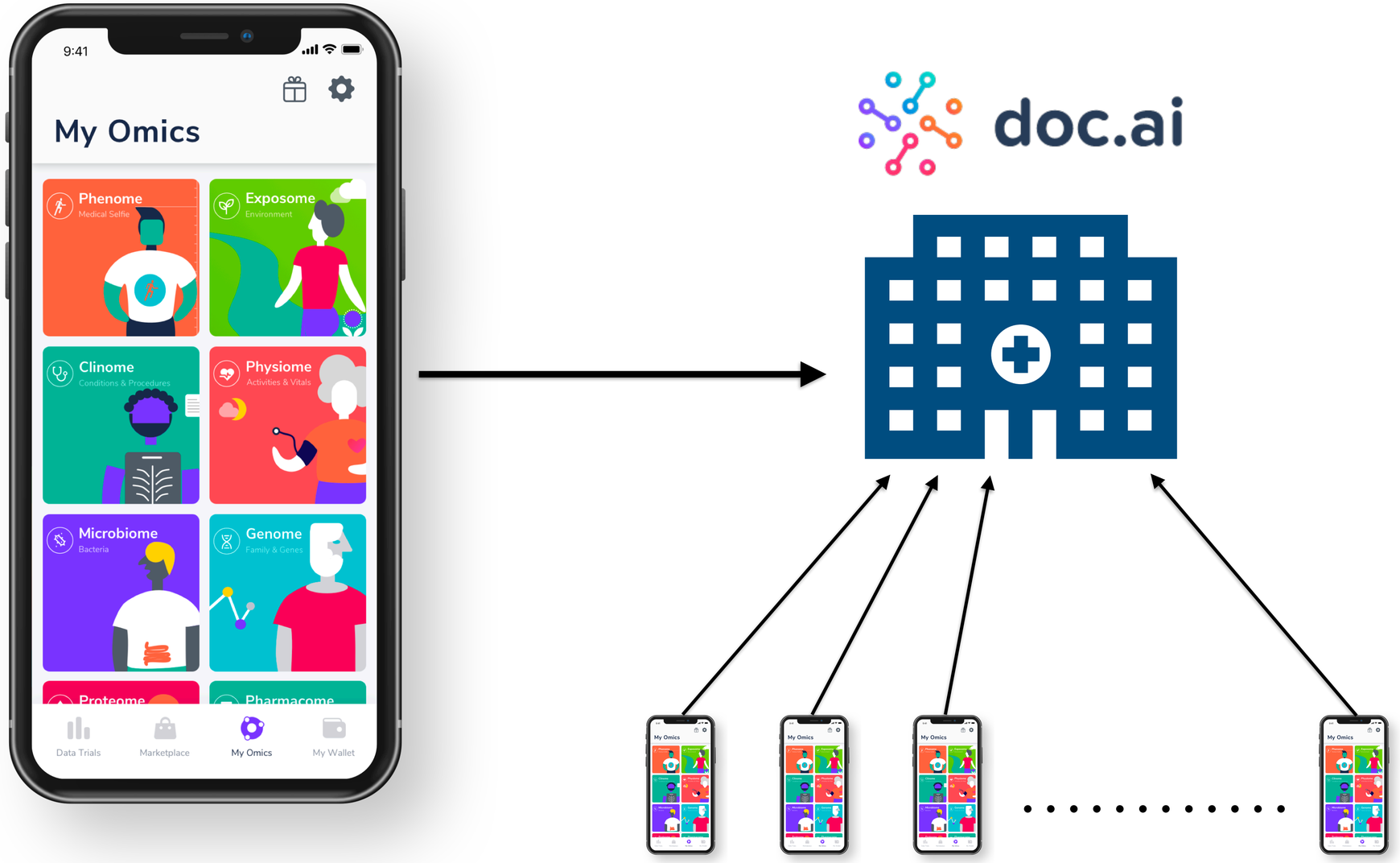
Federated Learning improves Gboard





Only gradients are sharing across, the user data never leaves local device.

Federated Learning accelerates medical research



Conventional Distributed Training

…… ……

[2011] Niu et al. *Hogwild!: A Lock-Free Approach to Parallelizing Stochastic*
[2012] Google. *Large Scale Distributed Deep Networks*
[2012] Ahmed et al. *Scalable inference in latent variable models*.
[2014] Li et al. *Scaling Distributed Machine Learning with the Parameter Server.*
[2017] Facebook. *Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour.*

…… ……

Almost all of them are performed **within a cluster.**

But federated leaning may involve users **across the world.**

Distributed Training Across the World

Ligeng Zhu, Yao Lu, Hangzhou Lin, Yujun Lin, Song Han

Limitation on Scalability (across clusters)

Bandwidth

- Infinity band: up to 100 Gb/s
- Normal ethernet: up to 10 Gb/s
- Mobile network: 100 Mb/s (4G), 1Gb/s (5G)

Latency

- Infinity band: < 0.002 ms
- Normal ethernet: ~0.200 ms
- Mobile network: ~50ms (4G) / ~10ms (5G)

What we need

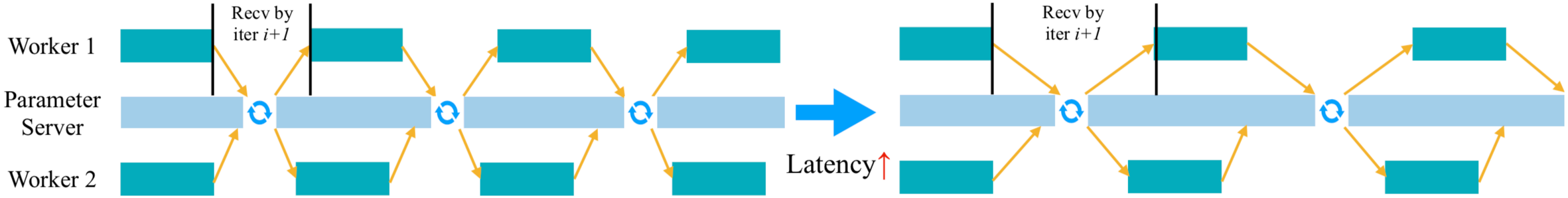
- ResNet 50: 24.37MB, 0.3s / iter (v100)
- At least 600 Mb/s bandwidth and 1ms latency.

Limitation on Scalability (across clusters)

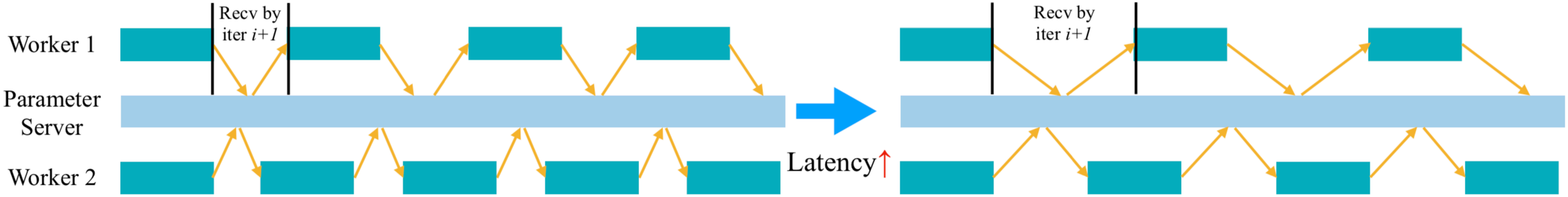
- Bandwidth can be always improved by
  - Hardware upgrade (Wired: fiber, Wireless: 5G)
  - Gradient sparsification (e.g., DGC, one-bit)

- Latency is hard to reduce because **physical laws**.
  - I.e. Shanghai to Boston, 11,725km, even considering the speed of light, still takes 78ms.

Conventional Algos suffer from high latency

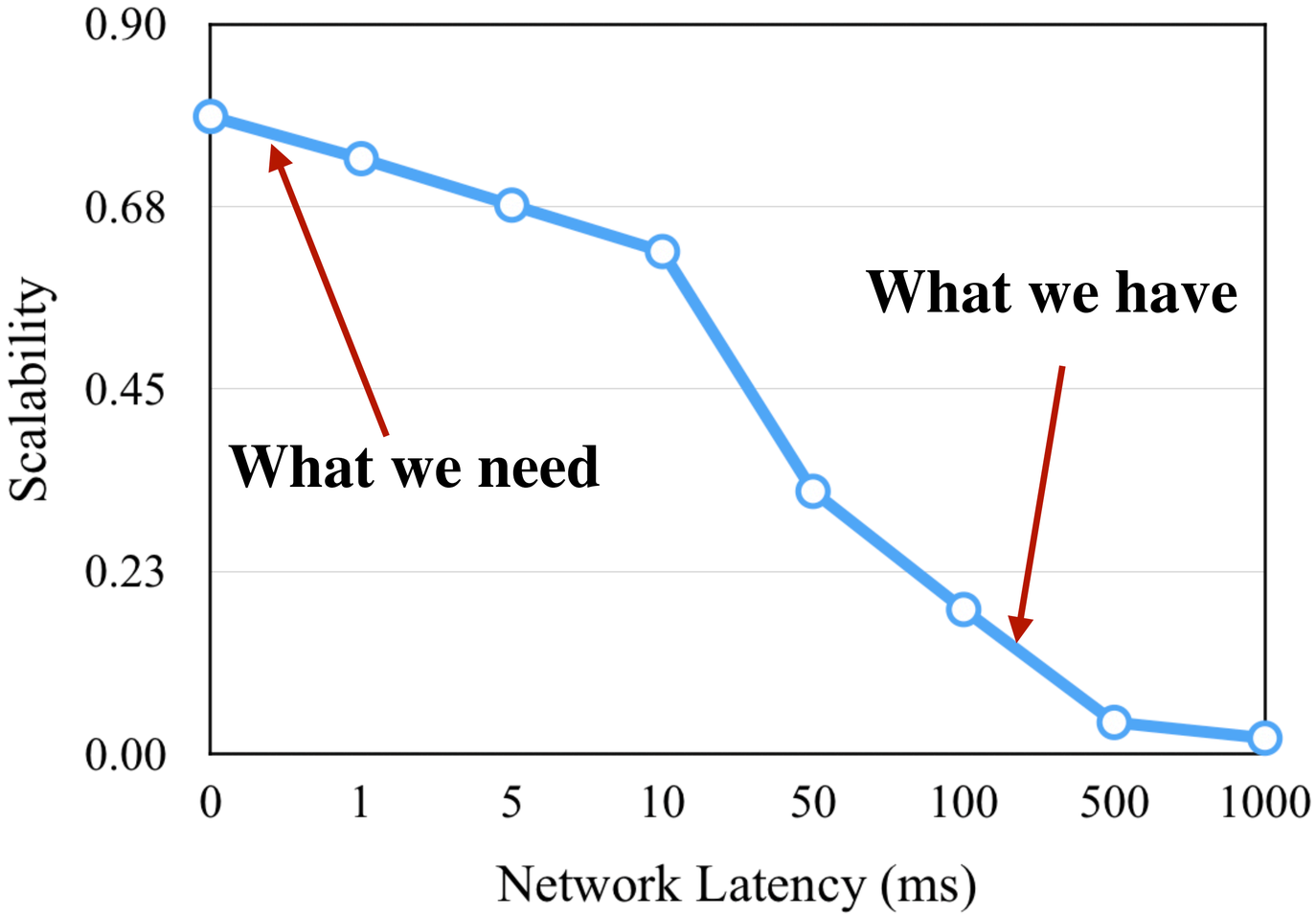


(a) Synchronous SGD synchronizes current gradients before next round of computation, thus not scalable when the latency increases.



(b) Asynchronous SGD has no synchronization barrier but each worker still needs to wait a long time when pulling latest weights.

Scalability degrades quickly with latency



Synchronous SGD

1. $\nabla w_{(i,j)} = \dfrac{\partial F(x_{(i,j)}, y_{(i,j)}; w)}{\partial w}$ ← Sample data from dataset and compute the local gradients

2. $\overline{\nabla w_{(i)}} = \dfrac{1}{J} \sum_{j=1}^{J} \nabla w_{(i,j)}$ ← Synchronize gradients to obtain the average

3. $w_{(i+1)} = w_{(i)} - \eta \overline{\nabla w_{(i)}}$ ← Apply the updates

Where i is the iterations,
j is the machine index
J is the total number of machines.

Delayed Synchronous SGD

Conventional Synchronous SGD

1. $\nabla w_{(i,j)} = \dfrac{\partial F(x_{(i,j)}, y_{(i,j)}; w)}{\partial w}$

2. $\overline{\nabla w_{(i)}} = \dfrac{1}{J} \sum_{j=1}^{J} \nabla w_{(i,j)}$

3. $w_{(i+1)} = w_{(i)} - \eta \overline{\nabla w_{(i)}}$

Our Algorithm

1. $\nabla w_{(i,j)} = \dfrac{\partial F(x_{(i,j)}, y_{(i,j)}; w)}{\partial w}$

2. $\overline{\nabla w_{(i-d)}} = \dfrac{1}{J} \sum_{j=1}^{J} \nabla w_{(i-d,j)}$

3. $w_{(i+1)} = w_{(i)} - \eta(\nabla w_{(i,j)} - \nabla w_{(i-d,j)} + \overline{\nabla w_{(i-d)}})$

Keypoints: synchronize stale gradients (d steps before).

Synchronous SGD

$$w_{(i+1)} = w_{(i)} - \overline{\nabla w_{(i)}}$$

Globally averaged gradients

Locally calculated gradients

Delayed Synchronous SGD (d=4)

$$w_{(i+1,j)} = w_{(i,j)} - \eta(\nabla w_{(i,j)} - \nabla w_{(i-d,j)} + \overline{\nabla w_{(i-d)}})$$

Synchronous SGD

$$\overline{w_{(i+1)}} = w_{(0)} - \eta \sum_{i=0}^{n} \overline{\nabla w_{(i)}}$$

$$\overline{w_{(i+1)}} = w_{(0)} - \eta \sum_{i=0}^{n-d} \overline{\nabla w_{(i)}} - \eta \sum_{i=n-d+1}^{n} \nabla w_{(i,j)}$$

Only differs in a small range

Delayed Synchronous SGD

DSSGD guarantees the convergence

- Assumption 1: the loss function $F(w; x, y)$ is L-Lipchitz smooth

$$\nabla f_j(x) - \nabla f_j(y)\,|| \leq L\,||x - y\,||. \quad \forall x, y \in \mathbb{R}^d$$

- Assumption 2: Bounded gradients and variances

$$\mathbb{E}_{\zeta_j}\,||\,\nabla F_j(w; \zeta_i)\,||^2 \leq G^2, \forall w, \forall j,\ \mathbb{E}_{\zeta_j}\,||\,\nabla F_j(w; \zeta_j) - \nabla f_j(w)\,||^2 \leq \sigma^2, \forall w, \forall j\,.$$

The convergence rate of DSSGD is $O(\frac{\Delta + \sigma^2}{\sqrt{JN}} + \frac{Jd^2}{N})$ (details in paper)

(N:iterations, J:number of machines)

DSSGD guarantees the convergence

The convergence rate of DSSGD is

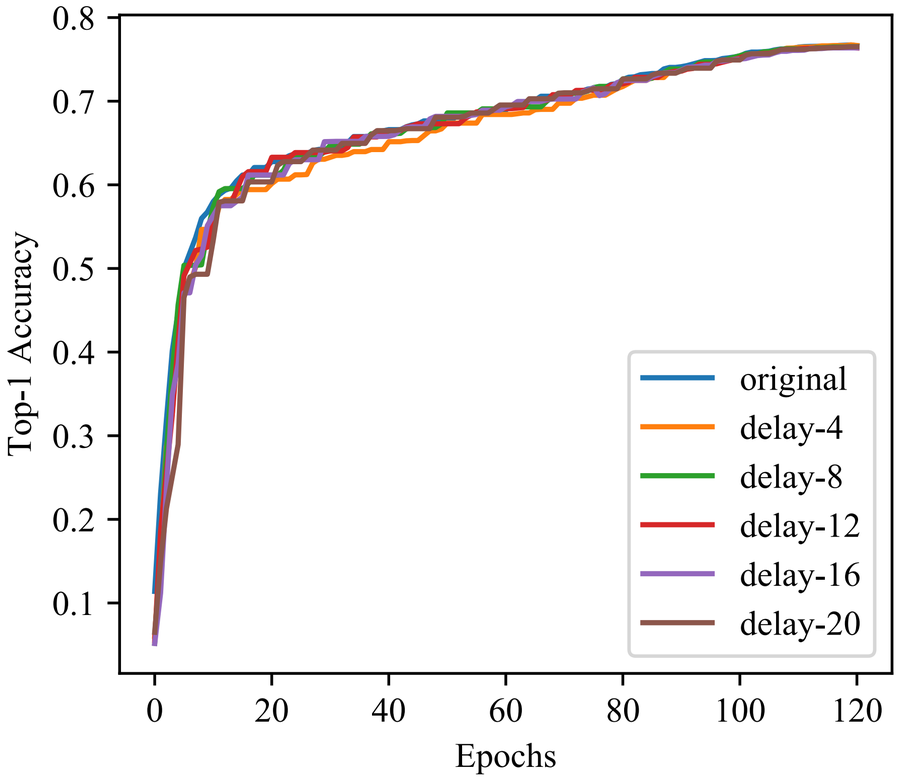
$$O(\frac{\Delta + \sigma^2}{\sqrt{JN}} + \frac{Jd^2}{N})$$

When $d < O(N^{\frac{1}{4}}J^{-\frac{3}{4}})$, the first term $O(\frac{\Delta + \sigma^2}{\sqrt{JN}})$ dominates.

DSSGD converges as fast as original SGD which is $O(\frac{\Delta + \sigma^2}{\sqrt{JN}})$.

(N:iterations, J:number of machines)

DSSGD provides the same accuracy



Consider training ResNet-50 on V100

Forward / backward -> 300ms

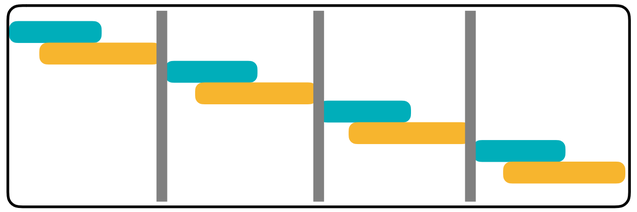
20 delay -> tolerate 6s latency.

6s ≈ 45 circles across earth

DSSGD tolerates high latency

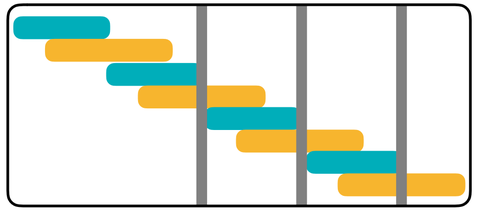
Naive Distributed SGD

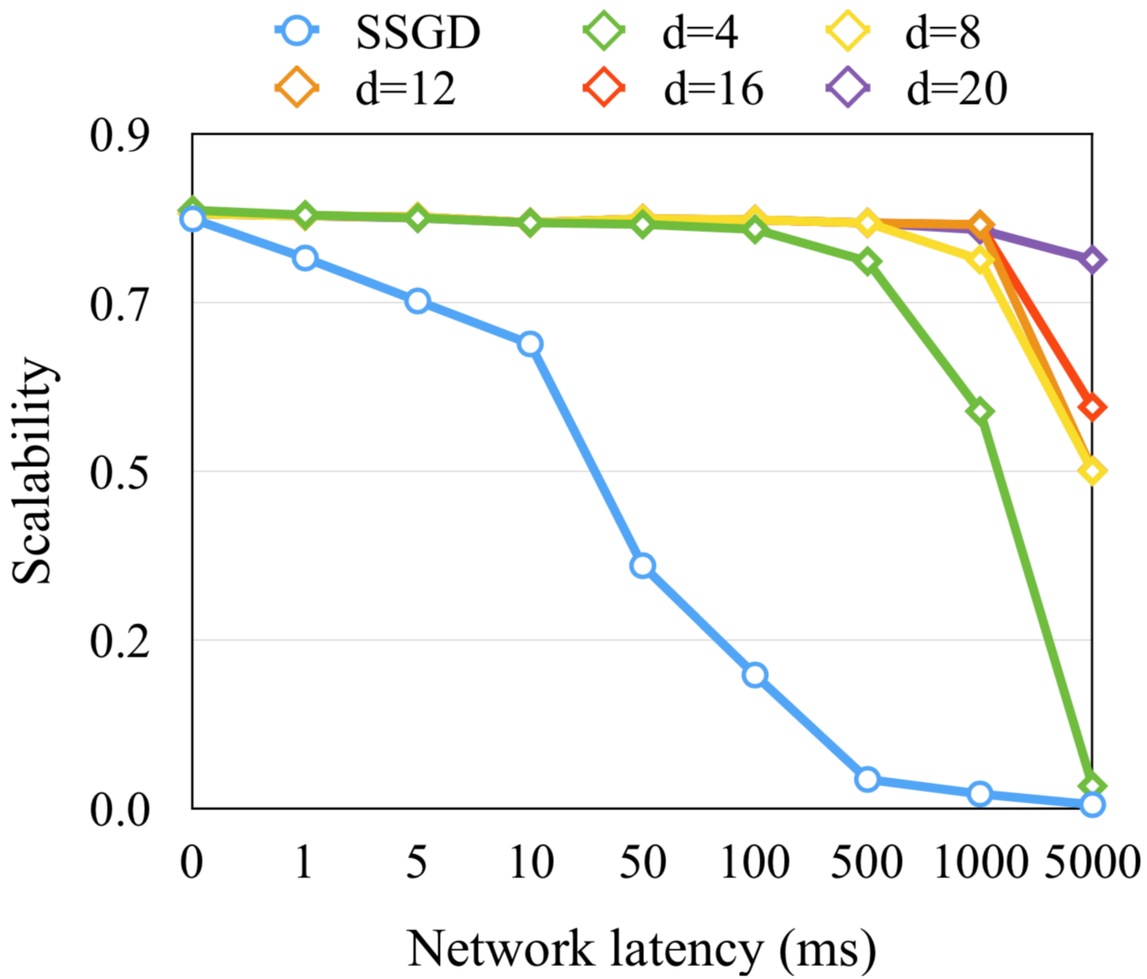
Wait time: $T_{\text{communicate}} - T_{\text{overlap}}$



Delayed Update

$$\max(0, T_{\text{communicate}} - T_{\text{overlap}} - t \times T_{\text{compute}})$$



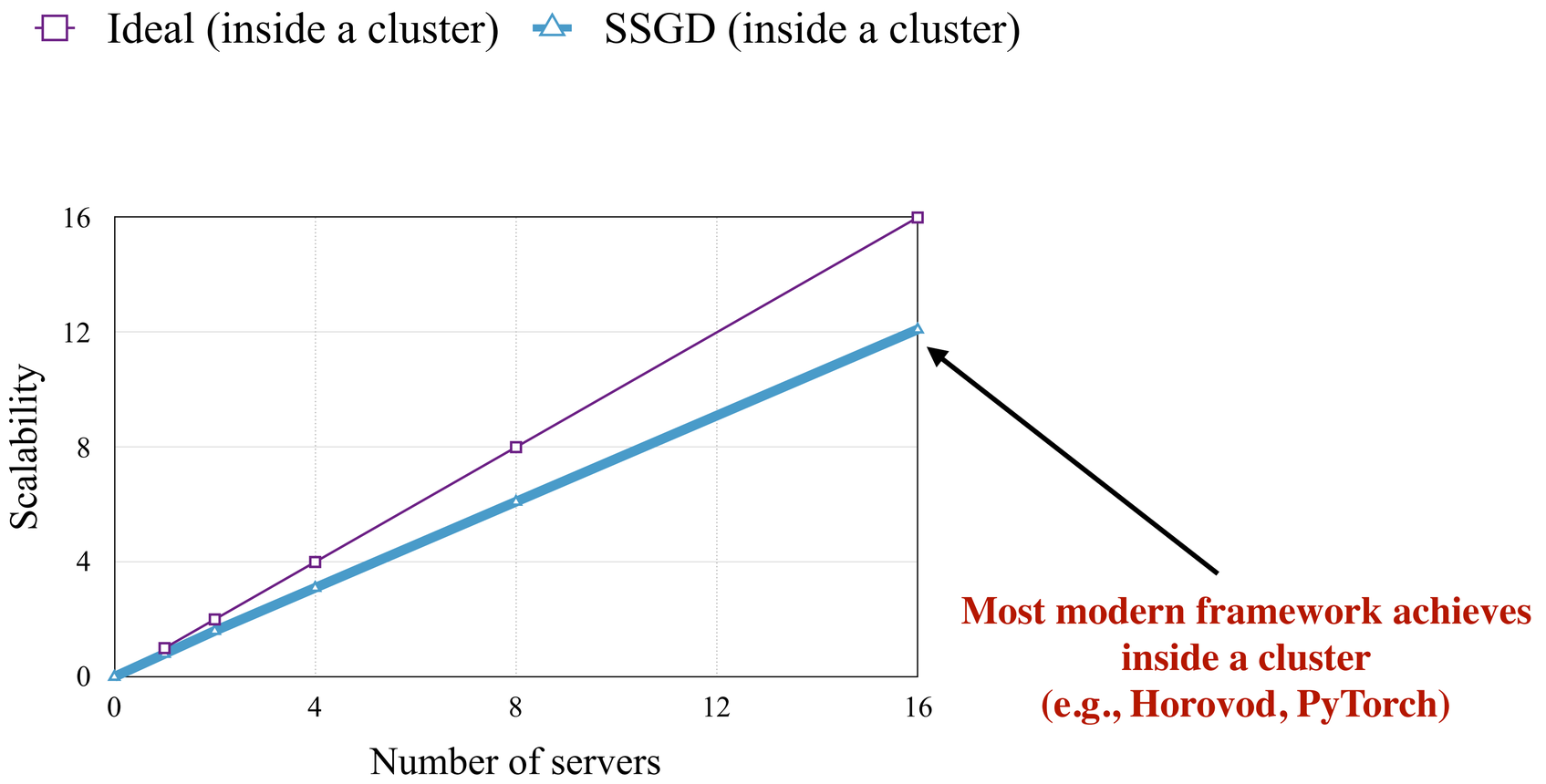


Distributed Training Across the World

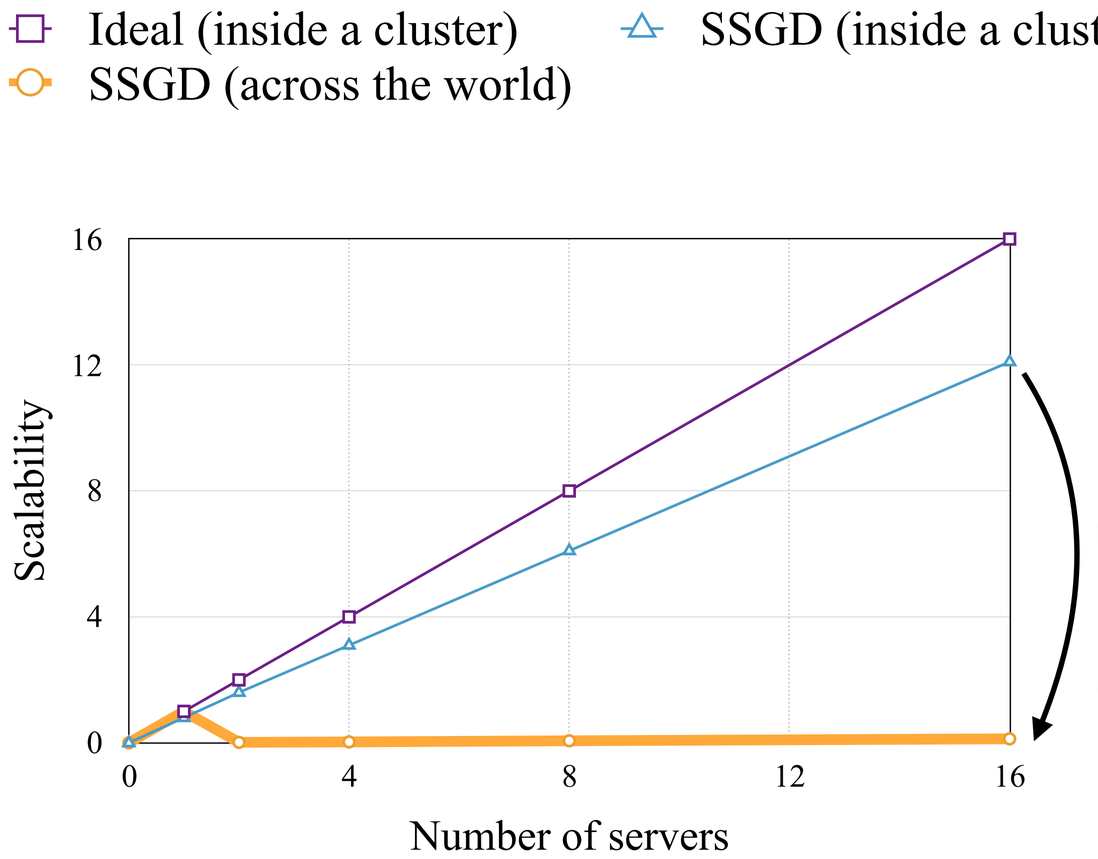
Experiment environments

- p3.16x Instances on AWS (8 x V100)
- 8 instances at 4 different geographical locations
  - Ohio, Oregon, London, Tokyo
  - Latency: ~480ms (based on ring all reduce)
- The scalability of naive training: 0.008
  - Training on 100 machines is slower than single one.

Scalability of SSGD when inside a cluster



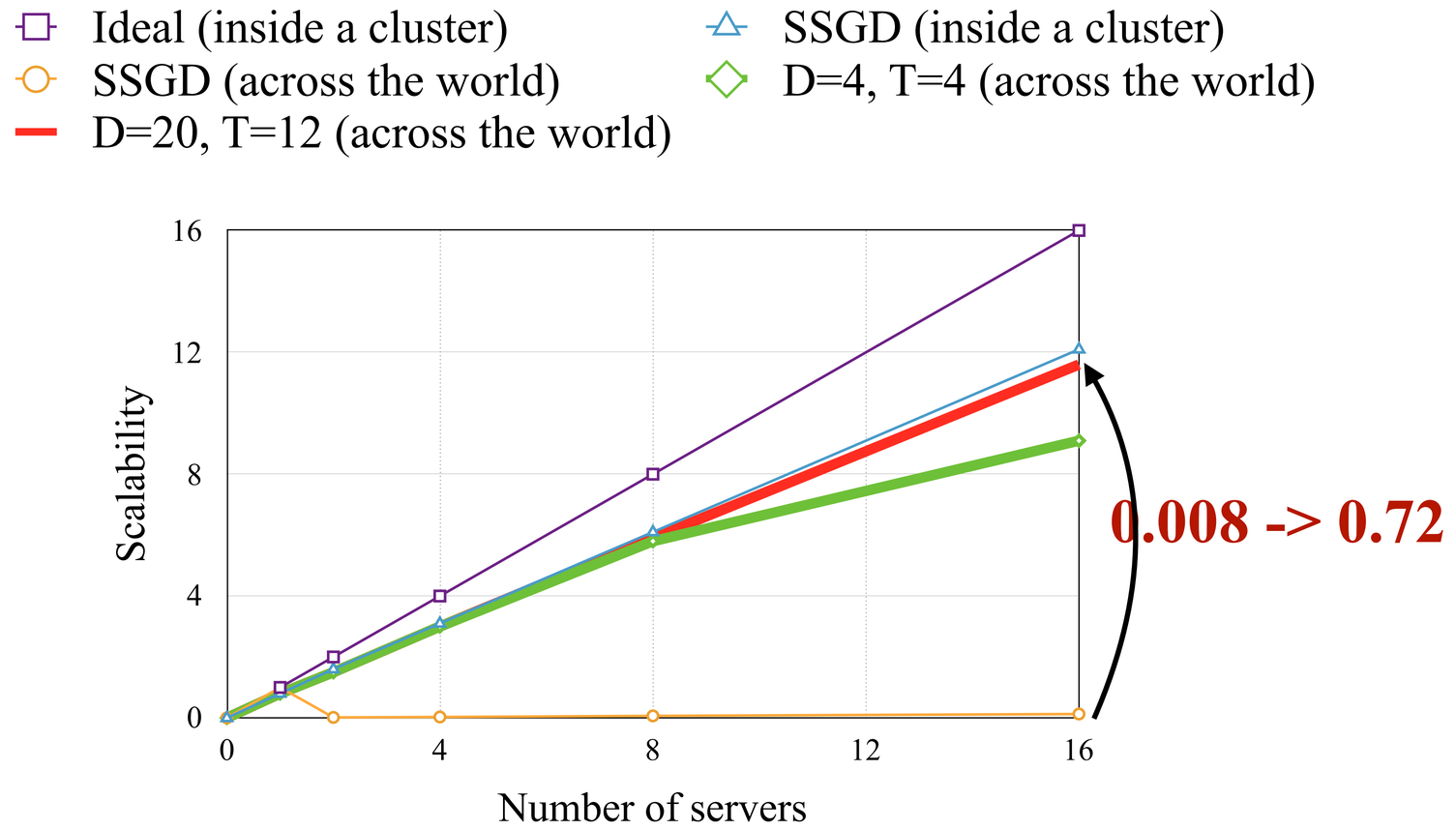
Scalability of SSGD when across the world



0.8 -> 0.008

Conventional algorithms

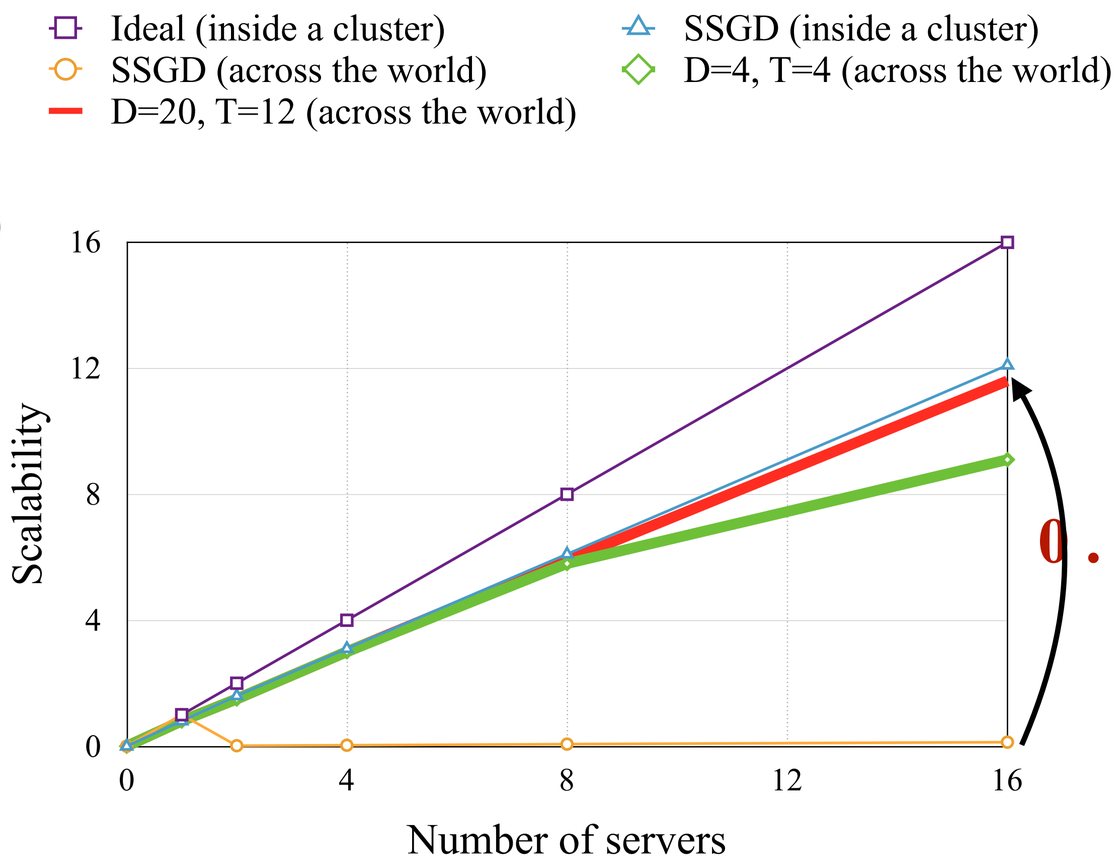
Scalability of SSGD when across the world



Scalability of DTS when across the world

- Delayed update (tolerate latency)
- Temporally sparse update (amortize latency)
- Gradient compression[1] (reduce transferred data)

	Top-1%
Original SGD	76.63
D=4, T=4, C=1%	76.15
D=8, T=8, C=1%	76.32
D=12, T=8, C=1%	76.18
D=20, T=12, C=1%	75.81



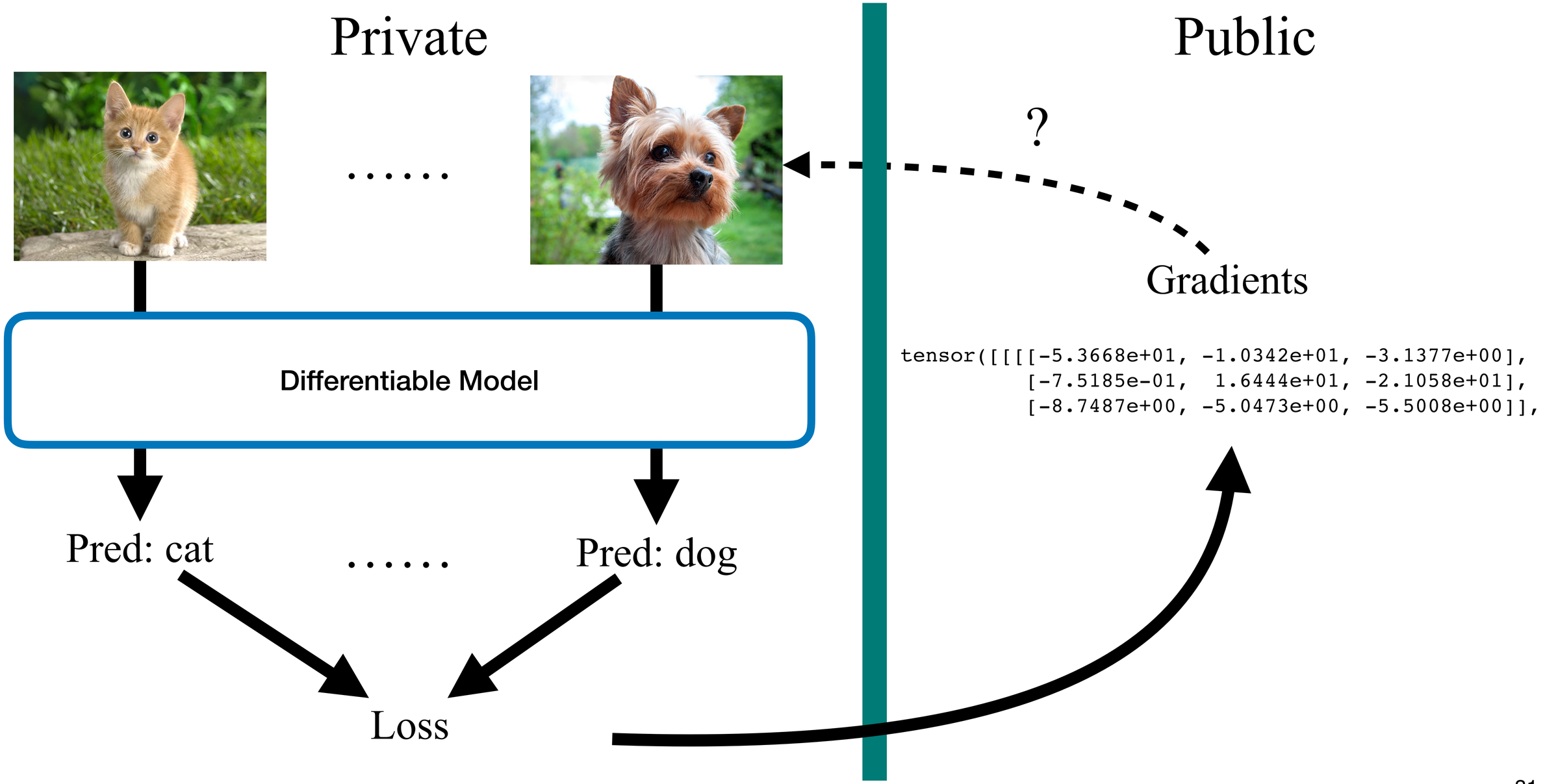
[1] *Deep Gradient Compression: Reducing the Communication Bandwidth for Distributed Training.* Yujun Lin, Song Han, Yu Wang, Bill Dally. ICLR 18

Deep Leakage from Gradients

Ligeng Zhu, Zhijian Liu, Song Han

NeurIPS’19

Is gradient safe to share?



Existing Shallow Leakage

Gradients

```
tensor([[[[-5.3668e+01, -1.0342e+01, -3.1377e+00],
          [-7.5185e-01,  1.6444e+01, -2.1058e+01],
          [-8.7487e+00, -5.0473e+00, -5.5008e+00]],
```

Membership Inference

Whether a record is used in the batch.

Property Inference

Whether a sample with certain property is in the batch.

But, can we obtain **the original training data**?

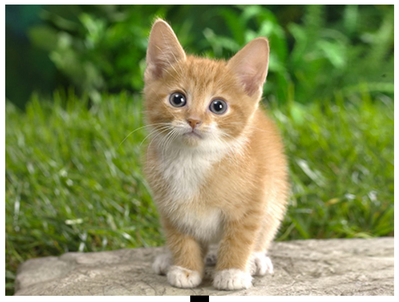
[1] L. Melis, C. Song, E. D. Cristofaro, and V. Shmatikov. *Exploiting unintended feature leakage in collaborative learning*.

[2] R. Shokri, M. Stronati, C. Song, and V. Shmatikov. *Membership inference attacks against machine learning models*.

Gradient is not safe to share!

Private

Public



.....

Differentiable Model

Pred: cat

.....

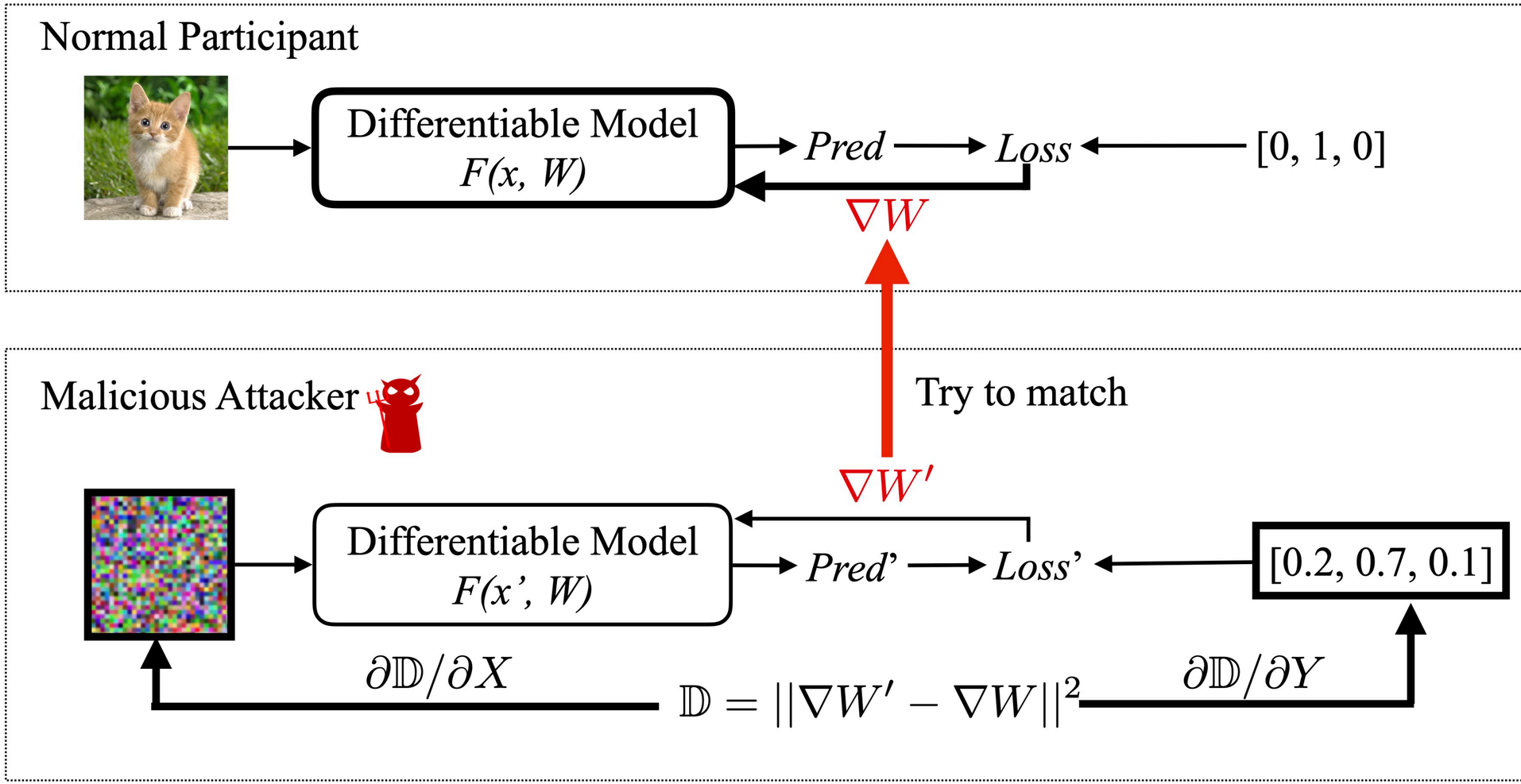
Pred: dog

Loss

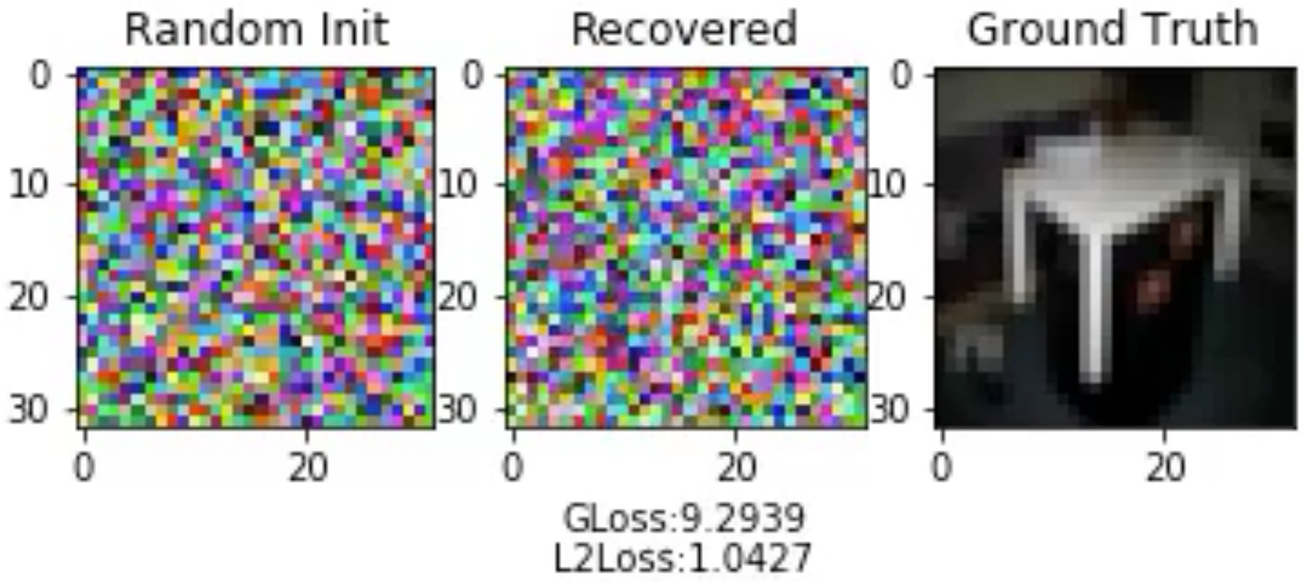
Gradients

```
tensor([[[[-5.3668e+01, -1.0342e+01, -3.1377e+00],
          [-7.5185e-01,  1.6444e+01, -2.1058e+01],
          [-8.7487e+00, -5.0473e+00, -5.5008e+00]],
```

Deep Leakage by Gradient Matching

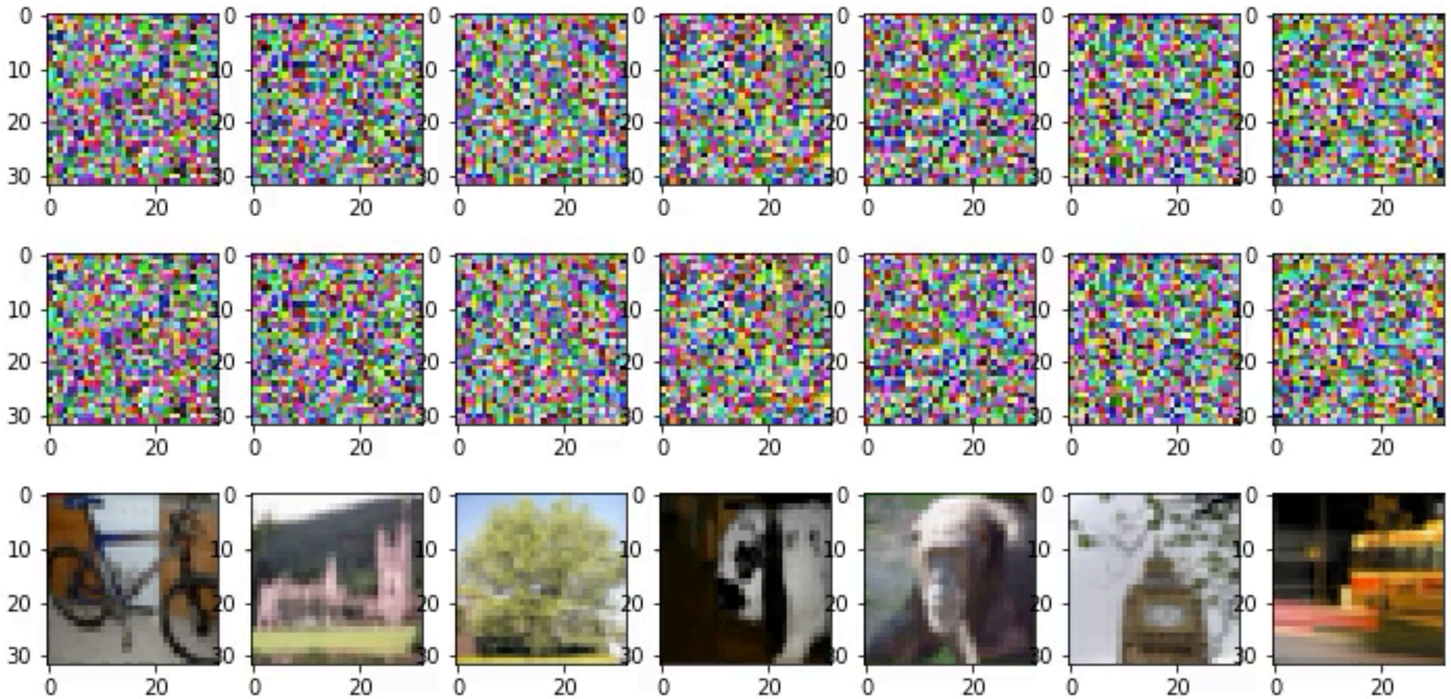


Recovering Visualization (bs=1)



Model: ResNet18 Dataset: CIFAR100 Optimizer: LBFGS 300 iters

Recovering Visualization (bs=8)



Model: ResNet18 Dataset: CIFAR100 Optimizer: LBFGS 300 iters

Experiment on Bert

Iters=0: tilting fill given **less word **itude fine **nton overheard living vegas **vac **vation *f forte **dis cerambycidae ellison **don yards marne **kali

Iters=10: tilting fill given **less full solicitor other ligue shrill living vegas rider treatment carry played sculptures lifelong ellison net yards marne **kali

Iters=20: registration , volunteer applications , at student travel application open the ; week of played ; child care will be glare .

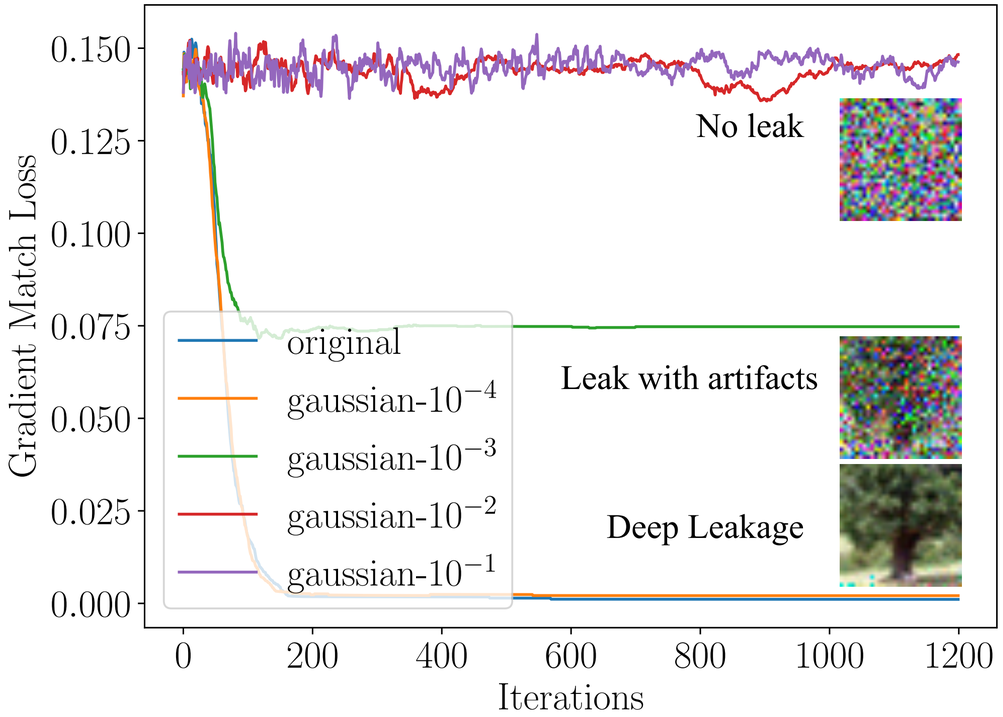
Iters=30: registration, volunteer applications, and student travel application open the first week of september . child care will be available

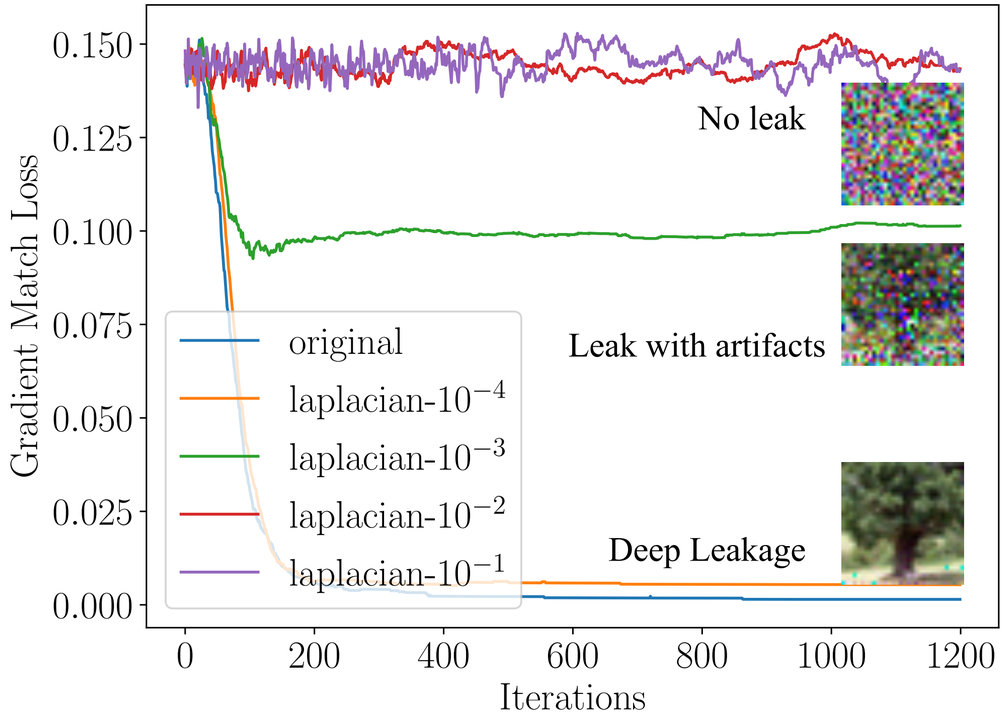
Original text: Registration, volunteer applications, and student travel application open the first week of September. Child care will be available.

Implementation in 20 lines (PyTorch)

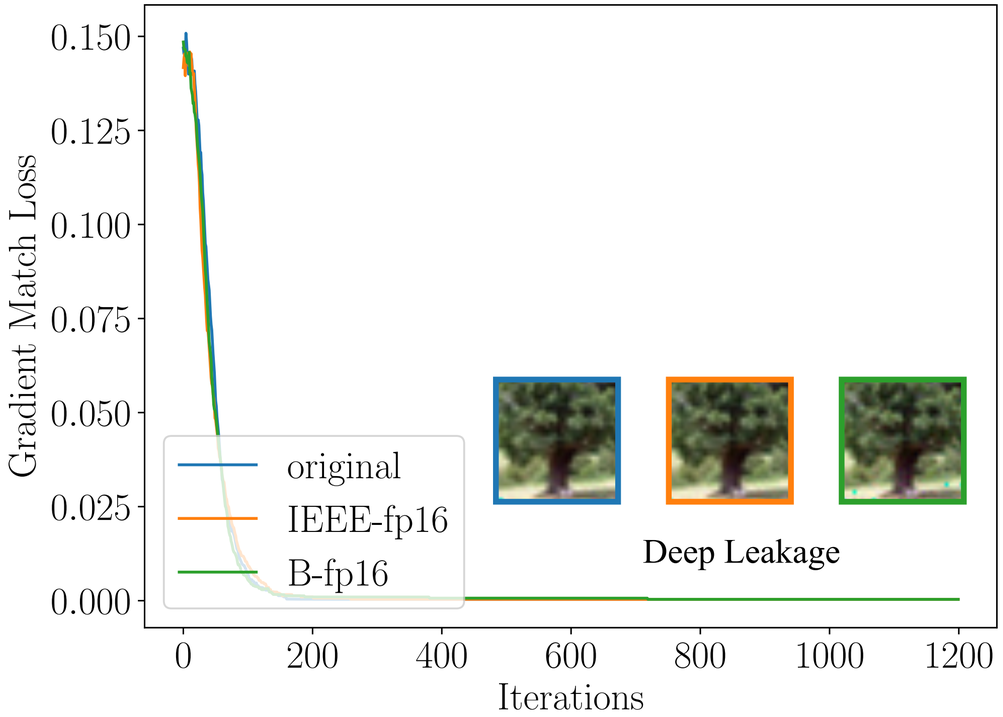
```
def deep_leakage_from_gradients(model, origin_grad):
  dummy_data = torch.randn(origin_data.size())
  dummy_label =  torch.randn(dummy_label.size())
  optimizer = torch.optim.LBFGS([dummy_data, dummy_label] )

  for iters in range(300):
    def closure():
      optimizer.zero_grad()
      dummy_pred = model(dummy_data)
      dummy_loss = criterion(dummy_pred, dummy_label)
      dummy_grad = grad(dummy_loss, model.parameters(), create_graph=True)

      grad_diff = sum(((dummy_grad - origin_grad) ** 2).sum() \
        for dummy_g, origin_g in zip(dummy_grad, origin_grad))

      grad_diff.backward()
      return grad_diff

    optimizer.step(closure)

  return  dummy_data, dummy_label
```

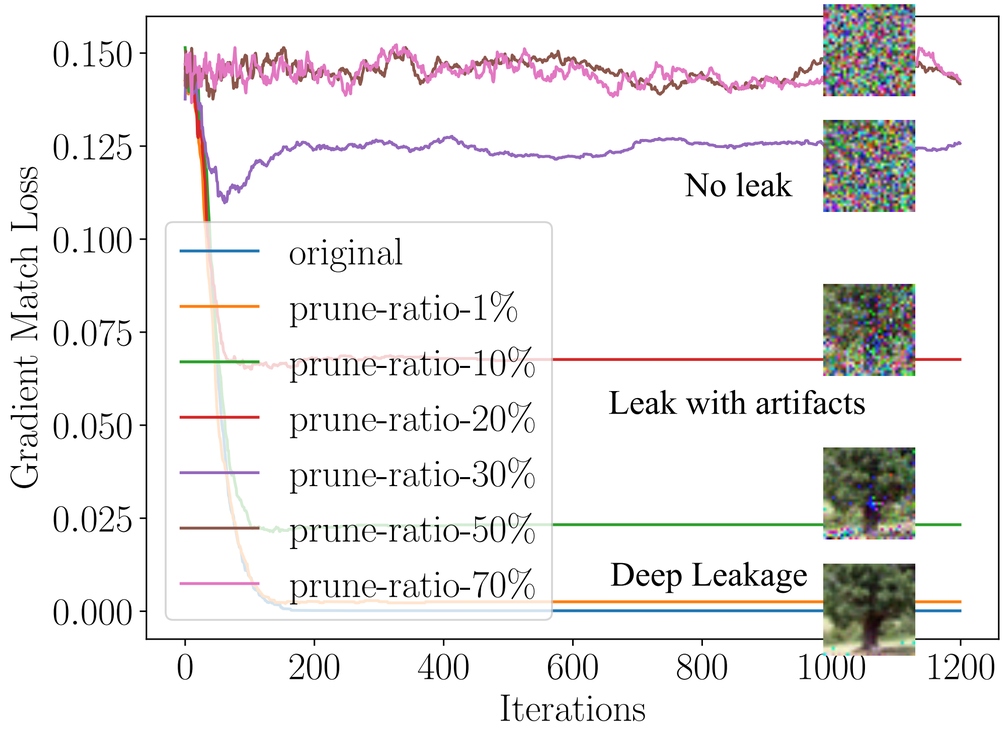
Defense Strategy





Defense Strategy





Thank you!