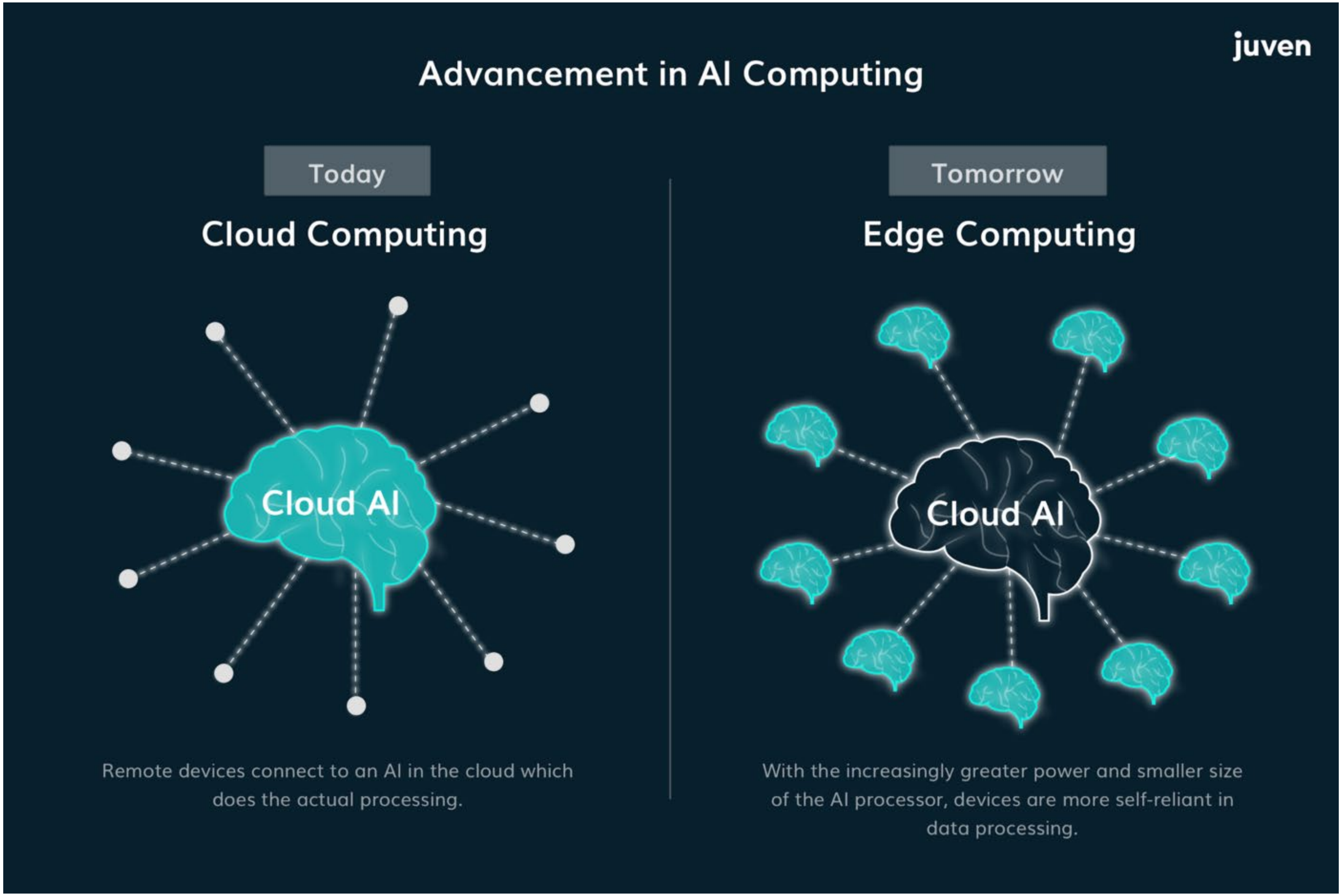
AI at the Edge

Wei Gao

AI At the Edge



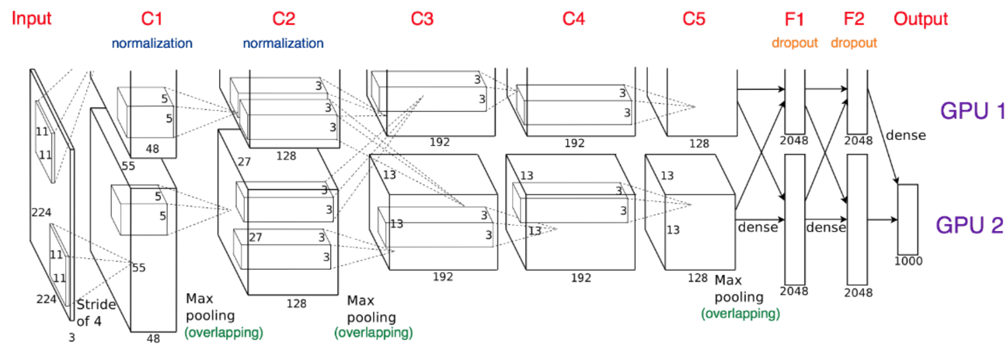
Key Challenges

- Limited computing resources at the edge devices
  - Limited computing power
  - Limited memory space

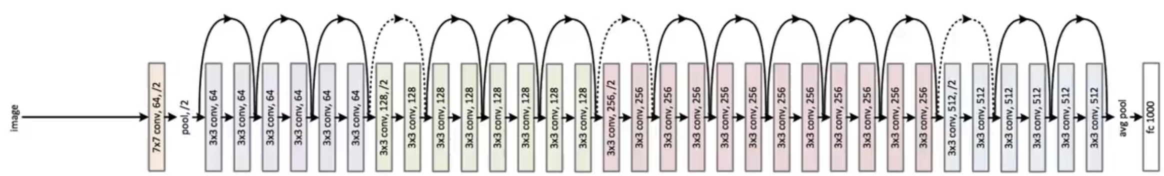
- Slow speed of training and inference

Huge NN Models

- Image Recognition
- AlexNet (2012) ILSVRC winner
  - 8 layers, 62Mparameters
  - 1.4 GFLOP inference
  - 16% error rate



- ResNet (2015) ILSVRC winner
  - 152 layers, 60Mparameters
  - 22.6 GFLOP inference
  - 6.16% error rate

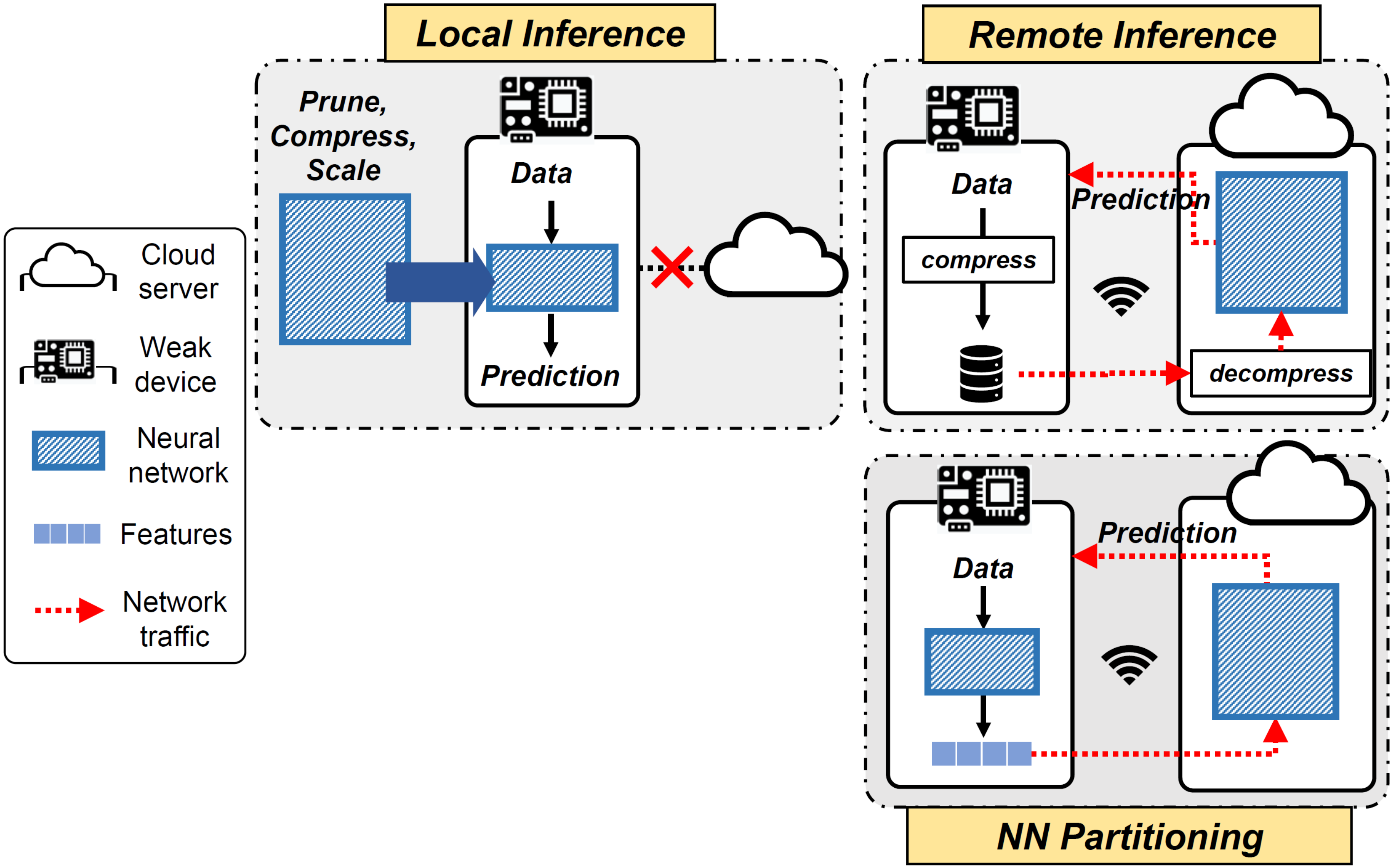


Huge NN Models

- Network design and training time have become a huge bottleneck

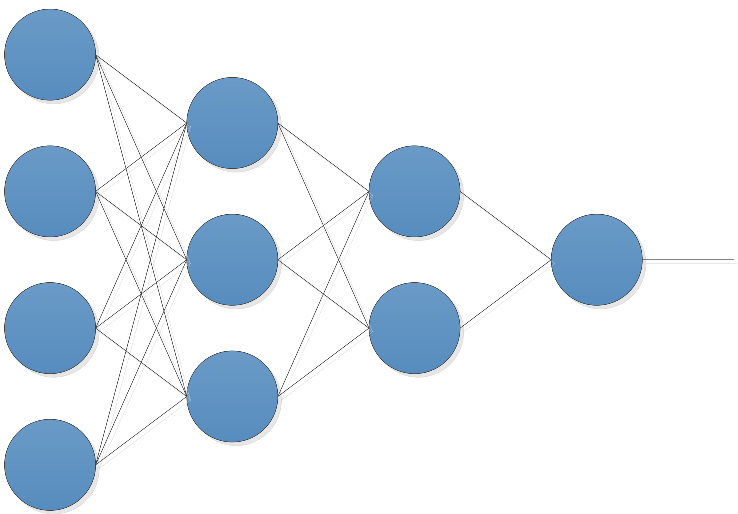
	Error rate	Training time
ResNet 18:	10.76%	2.5 days
ResNet 50:	7.02%	5 days
ResNet 101:	6.21%	1 week
ResNet 152:	6.16%	1.5 weeks

Potential Solutions

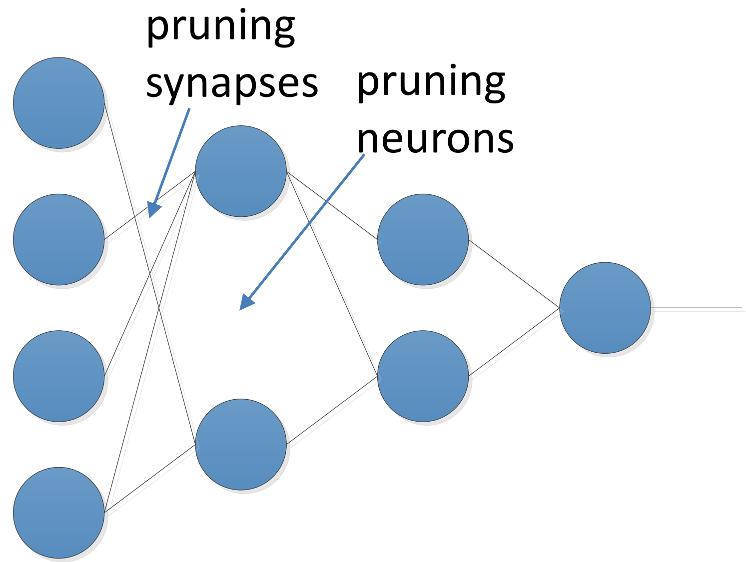


1. Local Inference – NN Pruning/Compression

before pruning



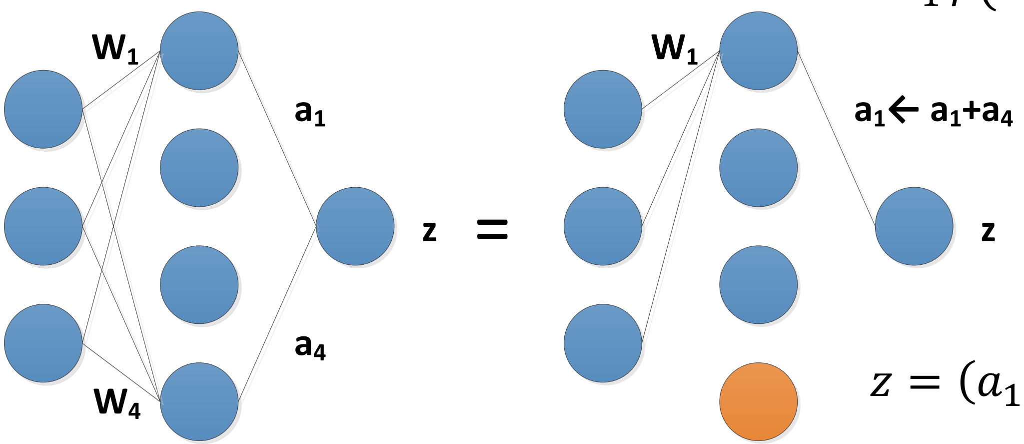
after pruning



Scenario 1: You only have a model

- Naïve pruning: Remove weights based on magnitude, weights close to zero are removed
  - No well-founded theory, error increases rapidly
- Data-Free parameter pruning based upon weight similarity

$$z = a_1\varphi(W_1^T X) + a_2\varphi(W_2^T X) + \cdots + a_n\varphi(W_n^T X)$$



Suppose, $W_1 = W_4$

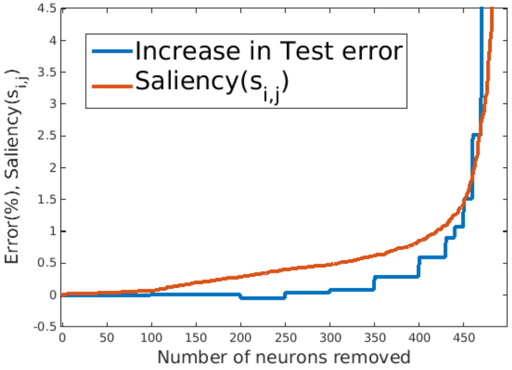
Then, $\varphi(W_1^T X) = \varphi(W_4^T X)$

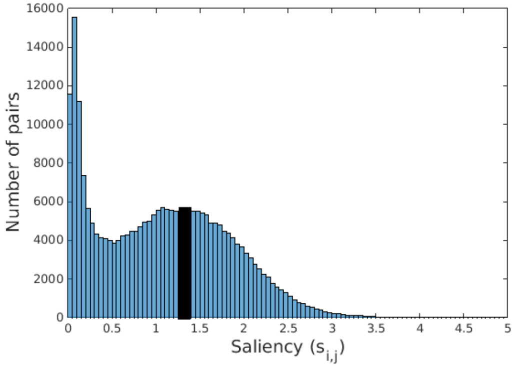
$$z = (a_1 + a_4)\varphi(W_1^T X) + a_2\varphi(W_2^T X) + \cdots$$

Data-Free pruning uses only the model sensitivity

- In practice neurons are different, $\|W_1 - W_2\| = \|\varepsilon_{1,2}\| \geq 0$
  - Compute errors for Weight replacement and naïve removal, so called **saliency** matrix ***M***
    - Pick minimum entry in the list e.g. indices (i', j'), delete the j'^{th} neuron and update $a_{i'} \leftarrow a_{i'} + a_{j'}$
    - Update ***M*** by removing j'^{th} column and row, and update the i'^{th} column for updated $a_{i'}$

- When to stop?
  - Saliency in line with test error
  - Find the mode in the gauss like curve





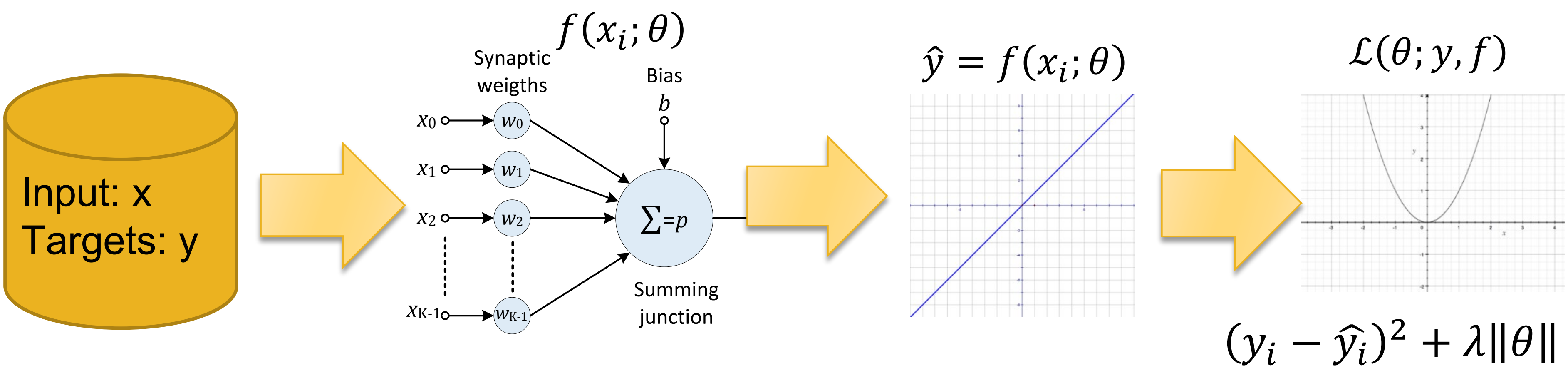
Scenario 2: You have data: how to prune aggressively?

- With access to training data, you can do a lot more

1. Train your network differently such that you have more zero weights
2. Retrain you network after pruning to fix the errors

Training a neural network: With a weight regularization

- The neural network is a function of inputs x_i and weights θ: $f(x_i;\theta)$
- Start with feed forward batch (i=1..64) through the network: $x_i \rightarrow \widehat{y_i}$
- Insert network results and desired target labels into a loss function: $\mathcal{L}(\theta; y, f)$
- Compute a score on how well the net performs, not only error also weight organization



Tune the weights by gradient descent

- Compute the error gradients
- Update the coefficients to reduce error, also taking into account regularization
- Repeat

$$\theta^{t+1} = \theta^{t} - \eta_t \nabla_\theta \, \mathcal{L}$$

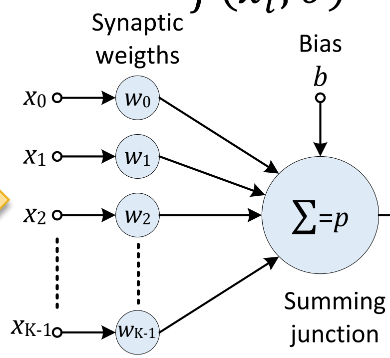
$$\frac{\partial f(x_i;\theta)}{\partial\theta}$$

$$\frac{\partial \hat{y}}{\partial f}$$

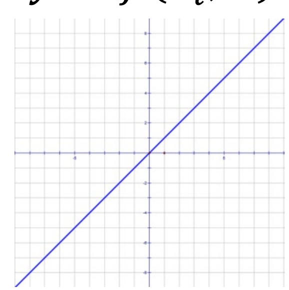
$$\frac{\partial \mathcal{L}(\theta;\hat{y},f)}{\partial \hat{y}}$$

Input: x
Targets: y

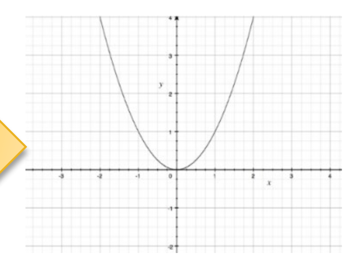
$$f(x_i;\theta)$$



$$\hat{y} = f(x_i;\theta)$$



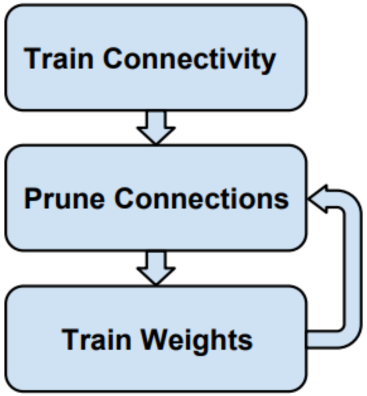
$$\mathcal{L}(\theta;y,f)$$



$$(y_i - \hat{y_i})^2 + \lambda\|\theta\|$$

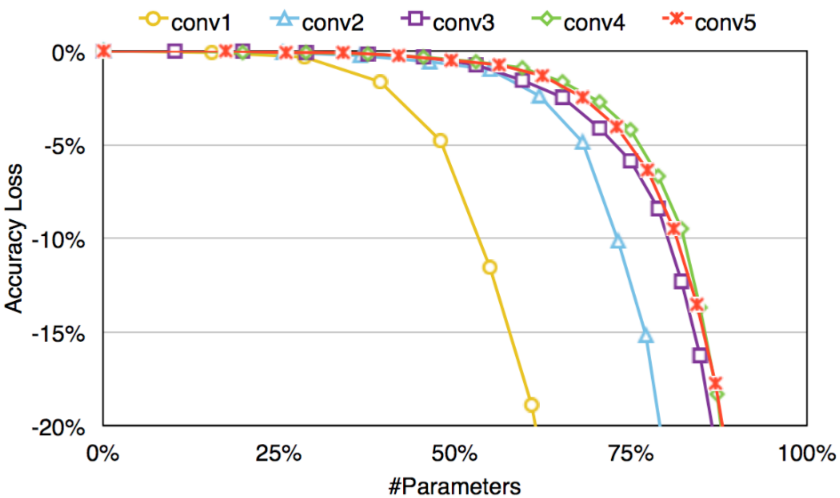
Iterative Pruning and Retraining

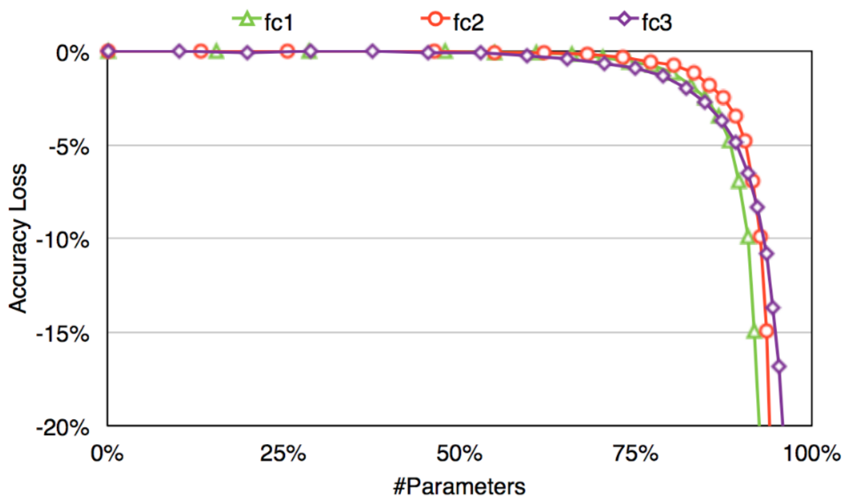
- Train a neural network until reasonable solution or download a pretrained net
  1. Prune the weights base on magnitudes that are less than a threshold
  2. Train the network until a reasonable solution is obtained
  3. Iterate to step 1



Where does pruning help the most?

- Fully connected layers

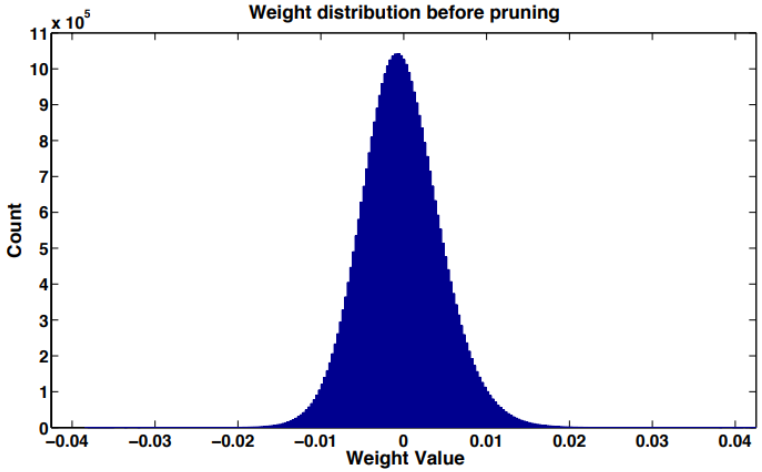


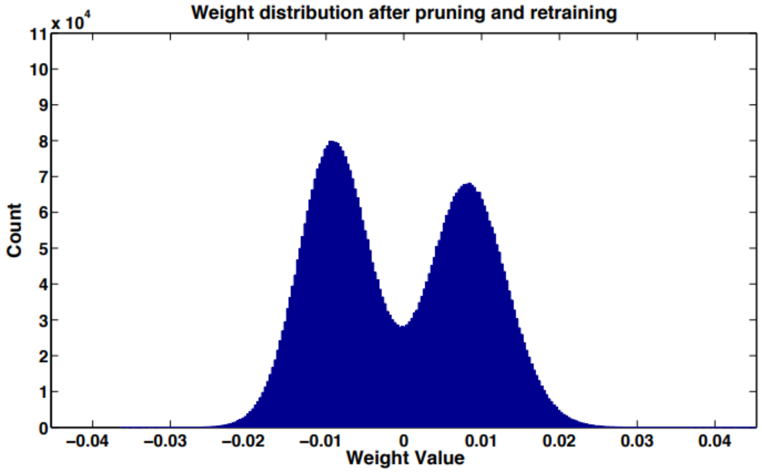


Pruning and Network Sparsity Improvements

What happens to the weight distribution?

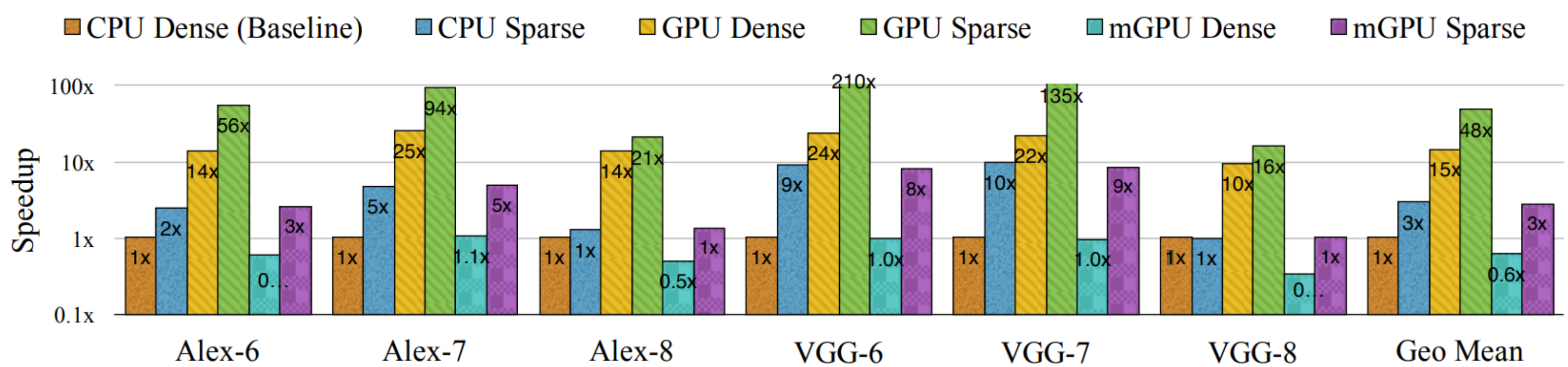
- Before: Most weights are close to zero; almost all between [-0.015, 0.015]
- After pruning: Bimodal distribution and more spread across x-axis, between [-0.025, 0.025]





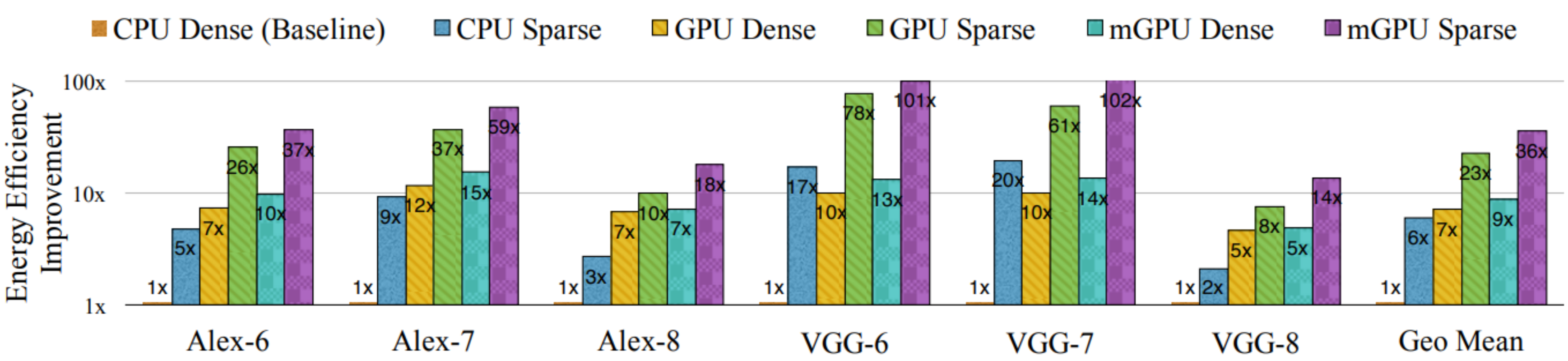
Using sparse Matrix Computations

- Use Intel Core i7 5930K, MKL CBLAS GEMV (full) vs MKS SPBLAS CSRMV (sparse)
- Use NVIDIA GTX Titan X, cuBLAS GEMV (full) vs cuSPARSE CSRMV (sparse)
- Use NVIDIA Tegra K1 as embedded GPU



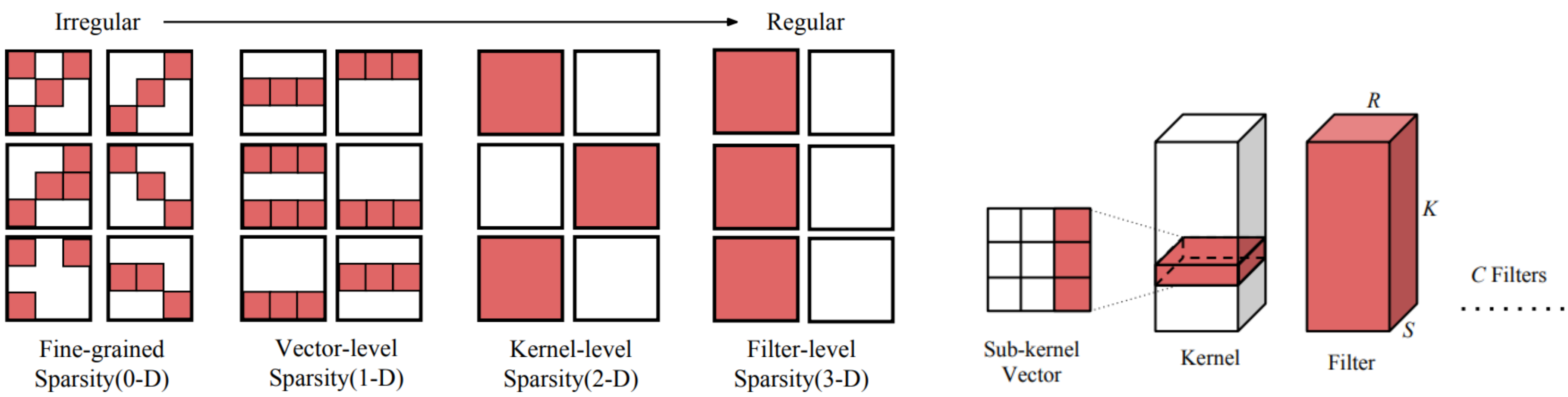
Energy Efficiency

- 6x improvement CPU 3.2x improvement GPU
  4x embedded GPU
- Difficult to exploit the large parameter reduction due to irregularity
  - Sparse matrices have also storage overhead; 16% for storing indices



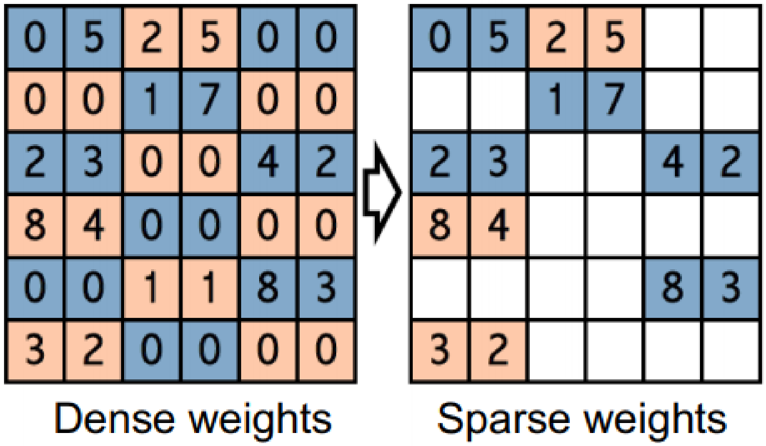
From Fine to Coarse-Grained Pruning

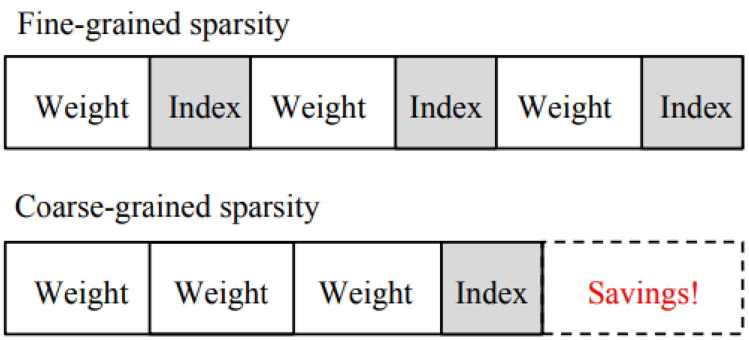
- Prune to match the underlying data-parallel hardware
  - E.g. prune by eliminating entire filter planes



Structured Pruning

- Example 2-way SIMD
- Less storage overhead

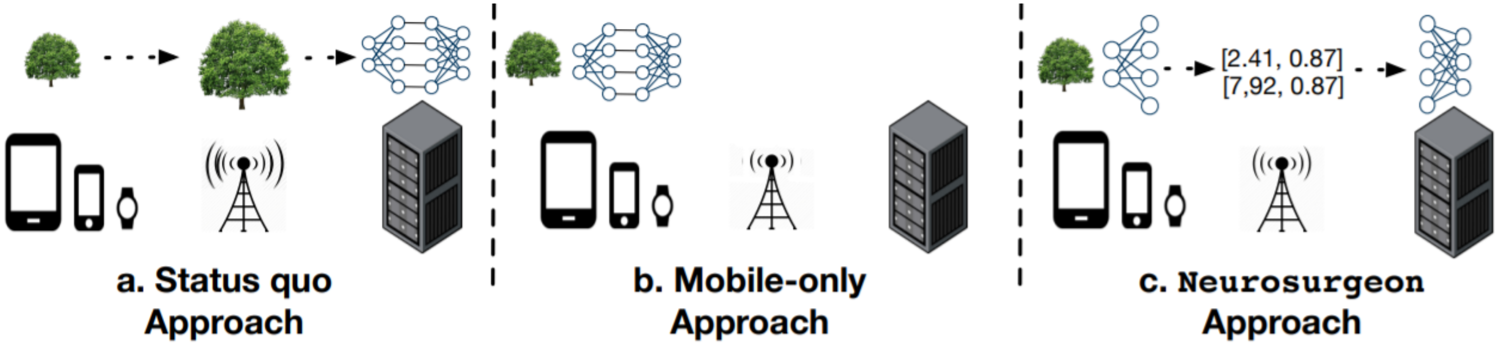




2. NN Partitioning

- Why not offloading the work to the edge?

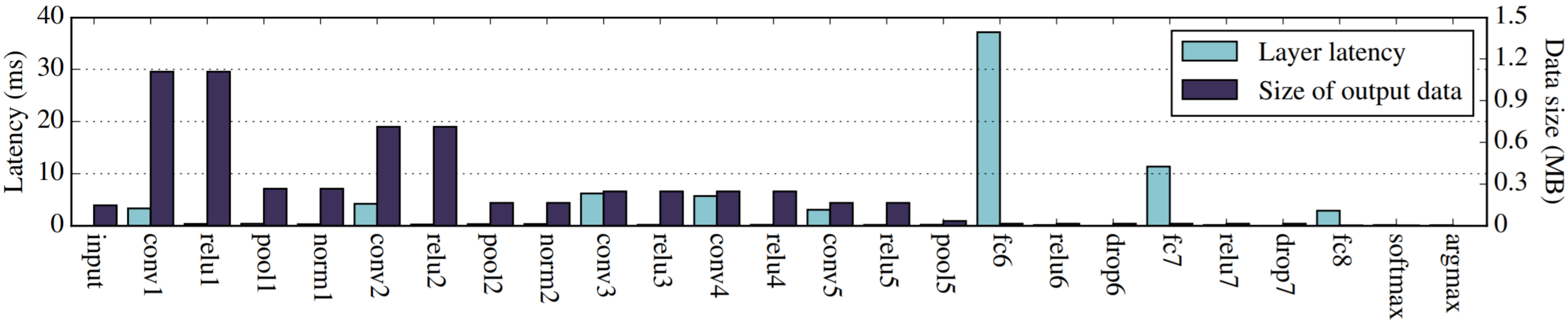
- Representative work: Neurosurgeon

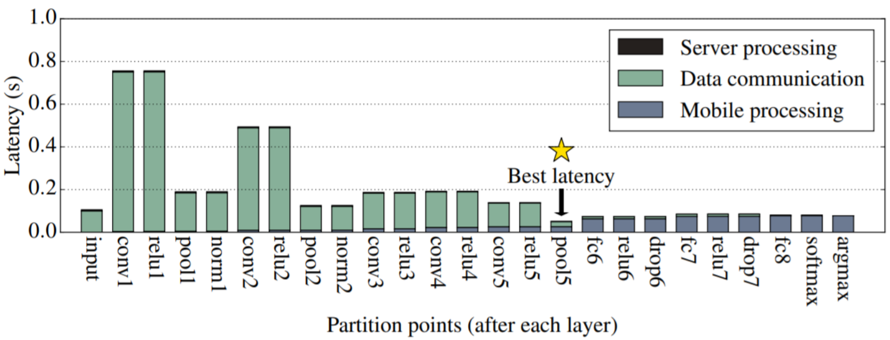


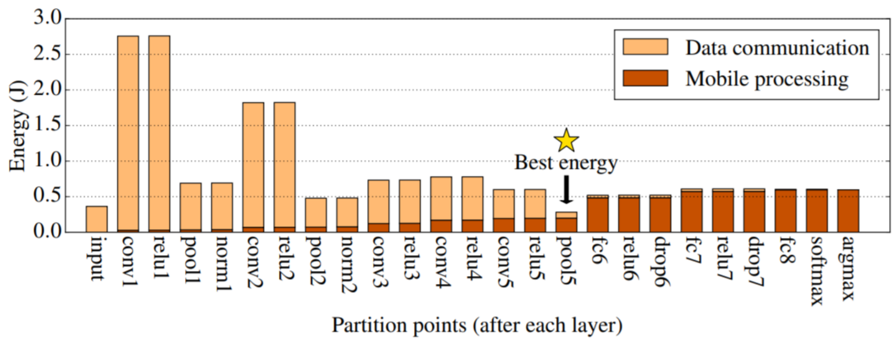
  - Partition the neural network in layers

Key Question

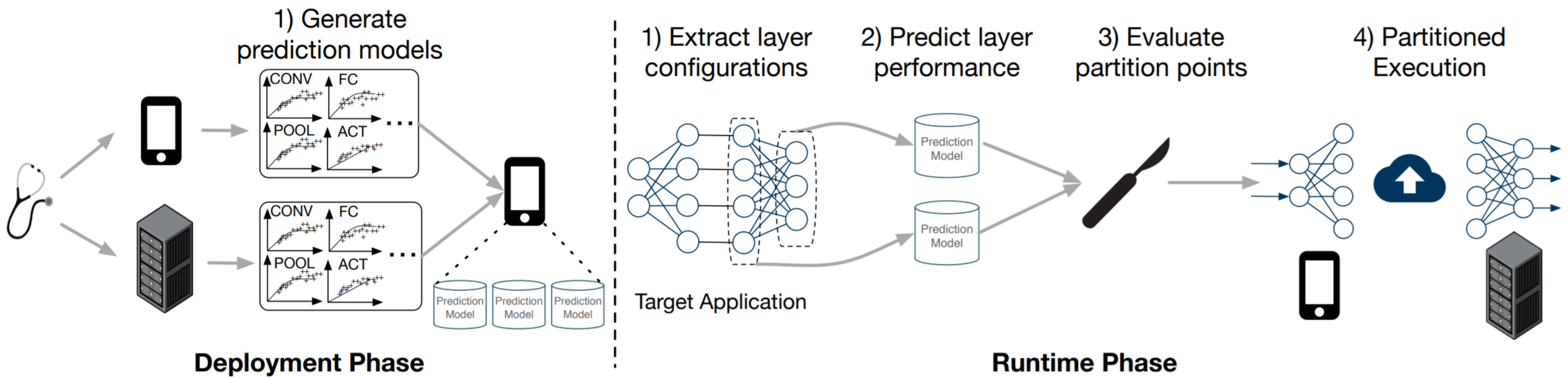
- Where to partition?







Practical Use

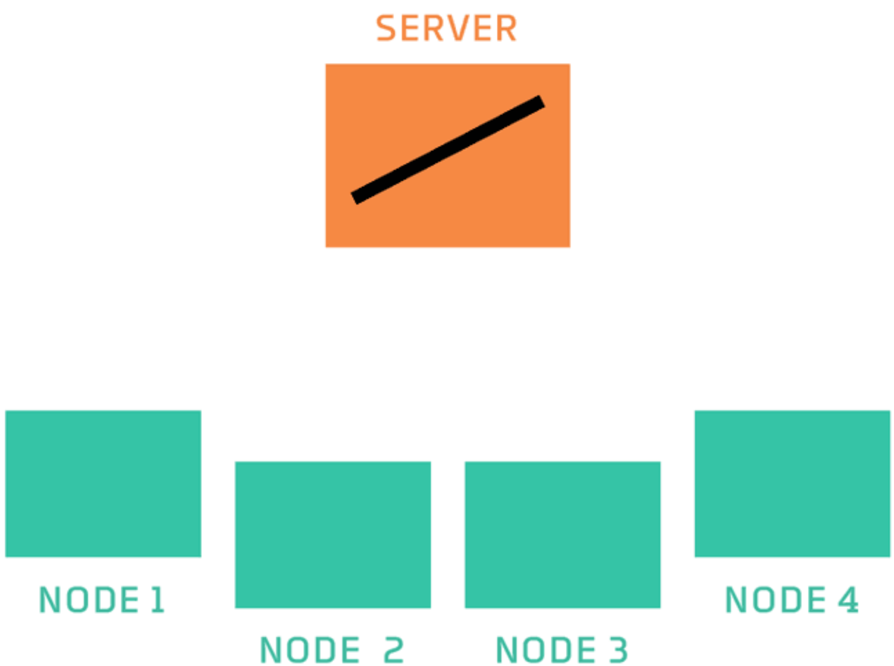


More Fine-Grained Partitioning?

- Vertical vs. Horizontal Partitioning
  - Partitioning the feature space

- Adaptive partitioning

Federated and Distributed Learning



$$\min \left[F(x) = \sum_{i=1}^{m} p_i F_i(x) \right]$$

m	n_i	n	p_i	$F_i(x)$	$F(x)$
Number of clients	Number of samples at client i	Total number of samples	n_i / n, relative sample size	Local objective function at client i	Global objective function

Challenges

Expensive Communication:
Communication in the network can be slower than local computation by many orders of magnitude.

Soultion: Smaller messages or sending less frequently

Privacy Concerns:
Sensitive information can still be revealed to third party or central server during the communication.

<u>Systems Heterogeneity</u>:
- Size of data
- Computational power
- Network stability
- Local solvers
- Learning rate

Statistical Heterogeneity:

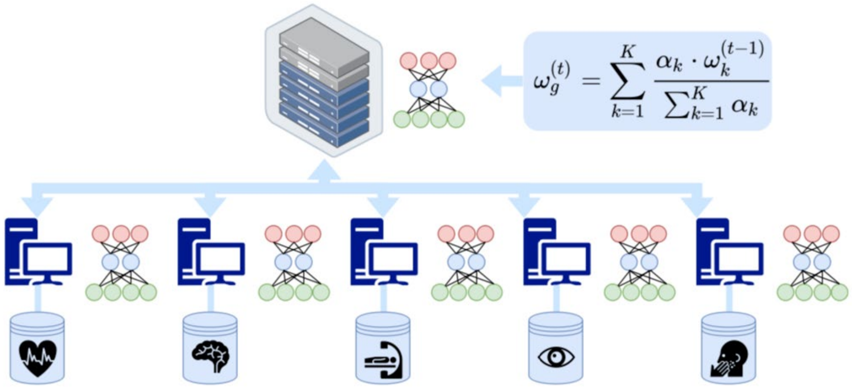
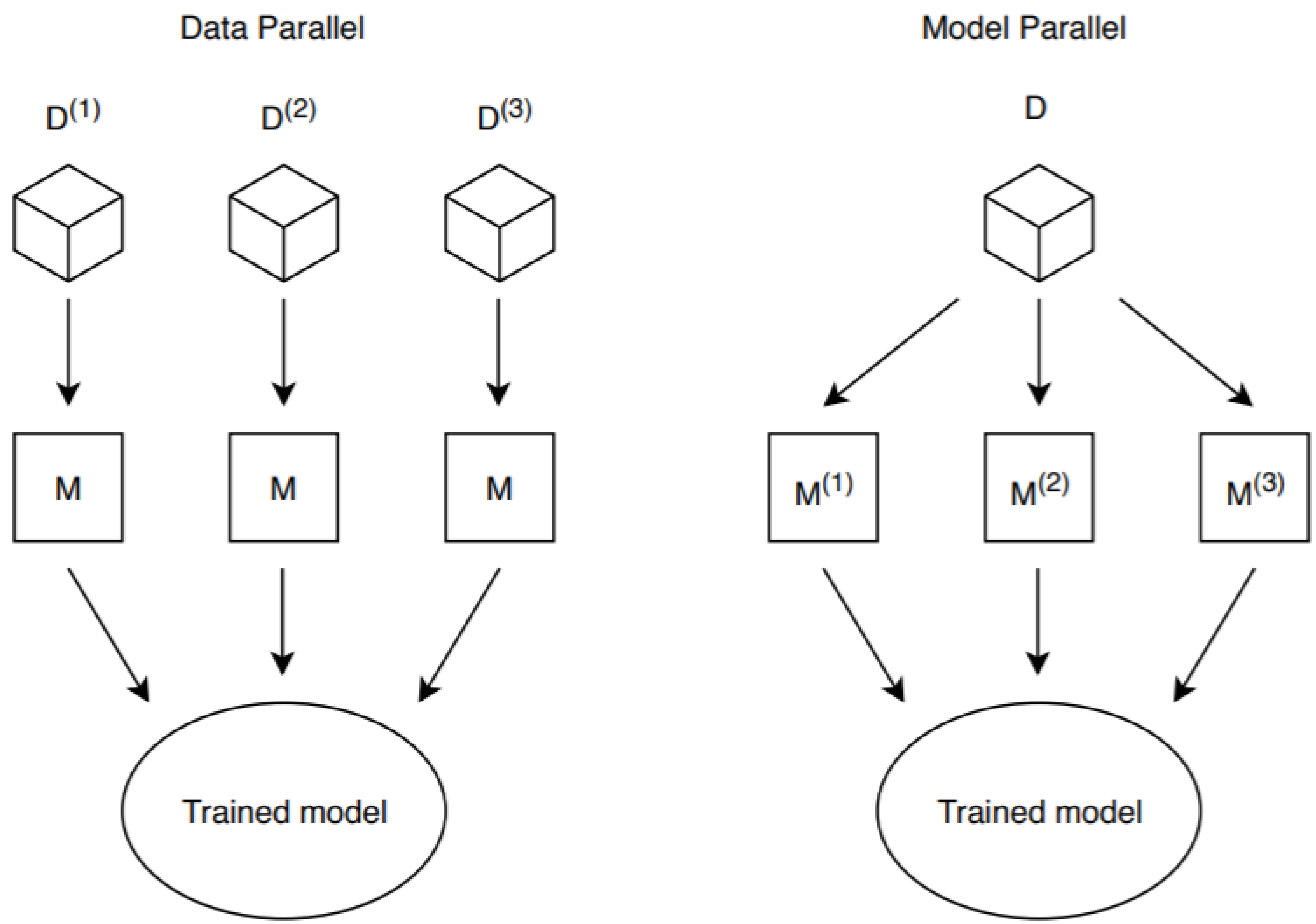
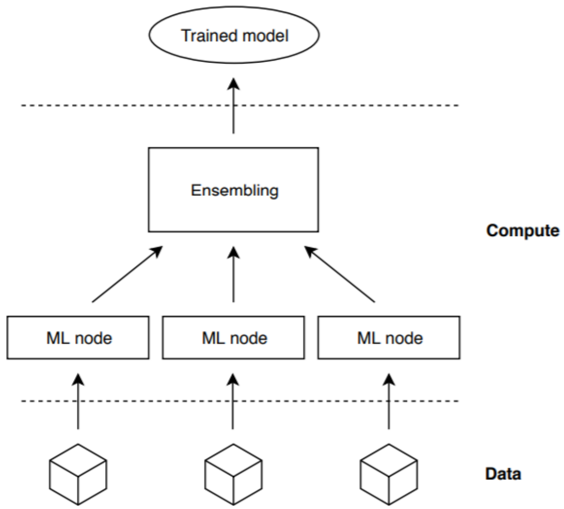


Fig. 1: Federated learning with non-iid data - The data has different distributions among clients.

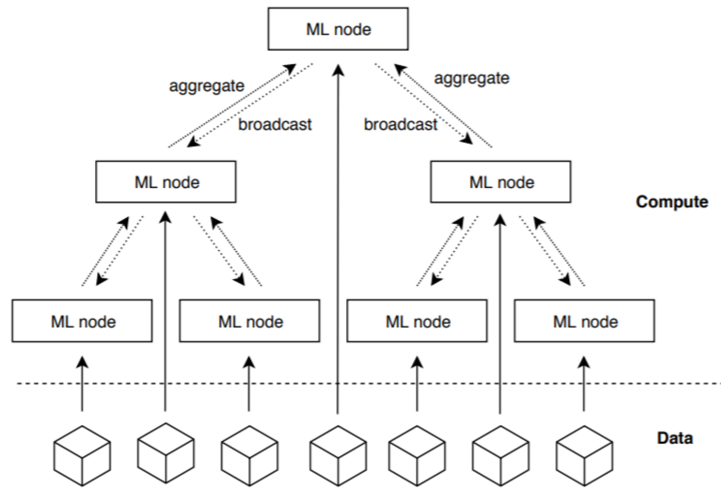
How to Achieve Parallelism



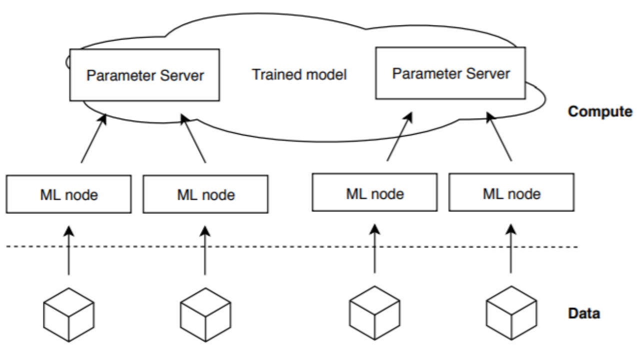
Different Operational Modes



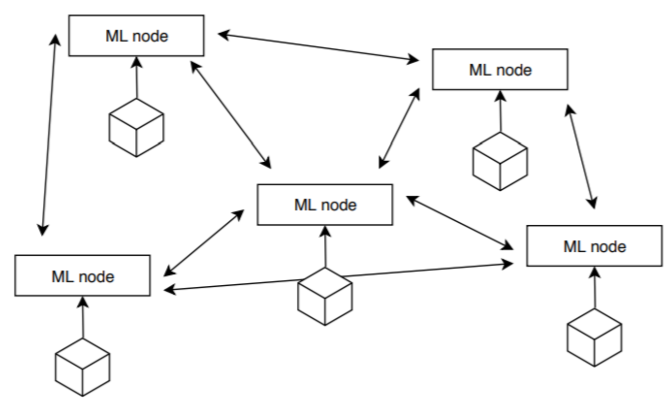
(a) Centralized (Ensembling)



(b) Decentralized (Tree)

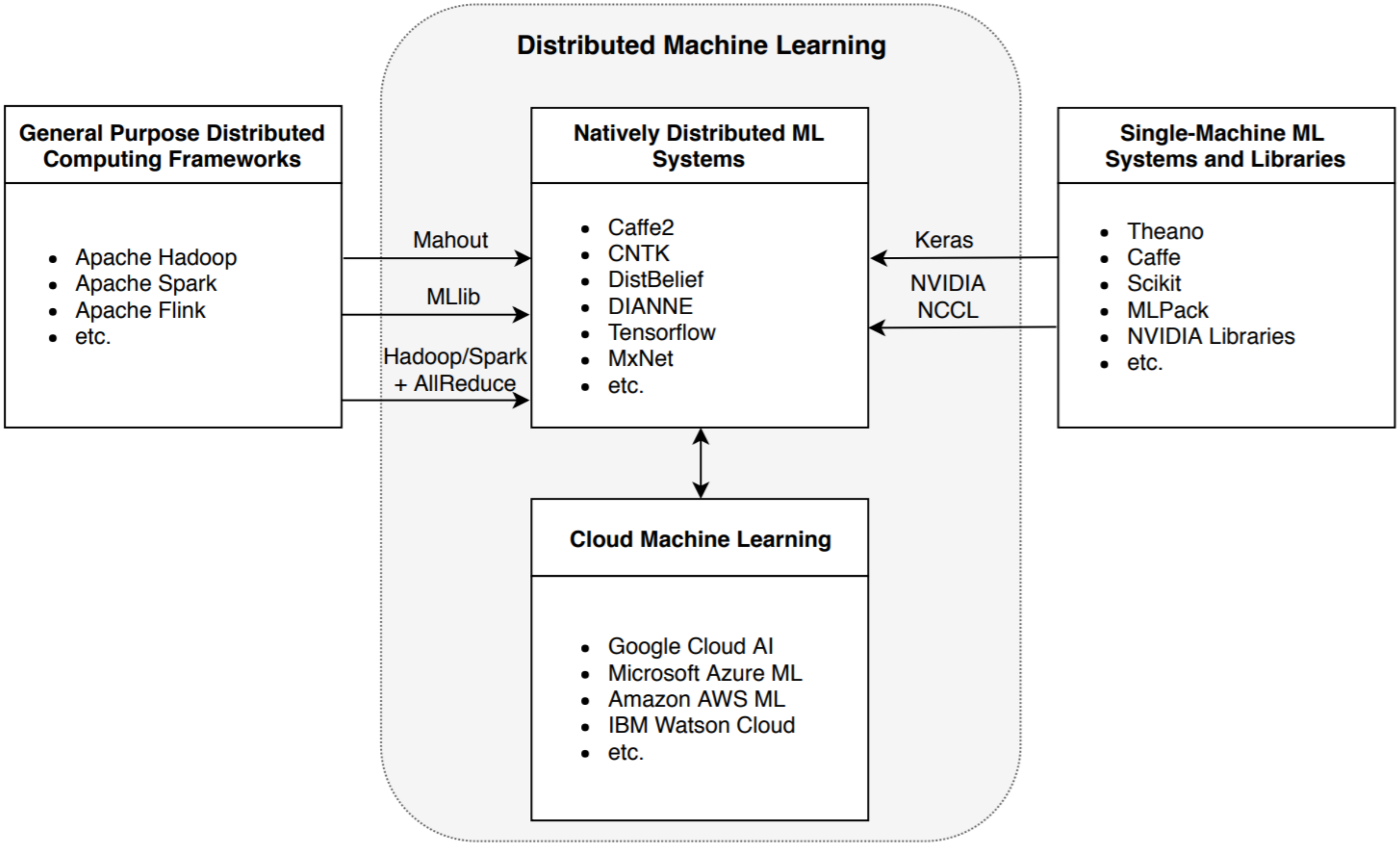


(c) Decentralized (Parameter Server)

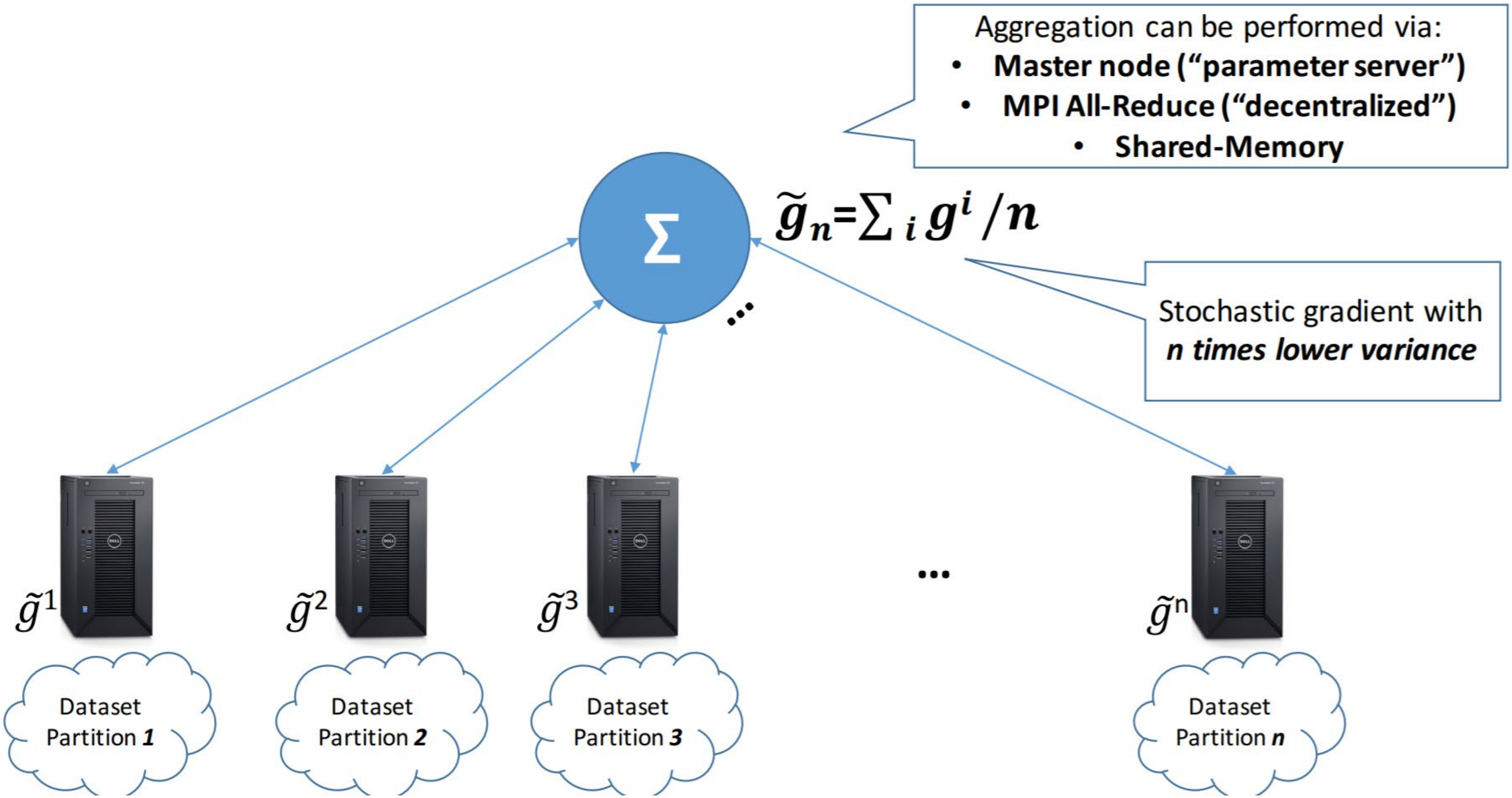


(d) Fully Distributed (Peer to Peer)

The Ecosystem



SGD Parallelization



Some more recent works

- TinyML / MCUNet
  - https://mcunet.mit.edu/
- Split learning
  - http://splitlearning.mit.edu/