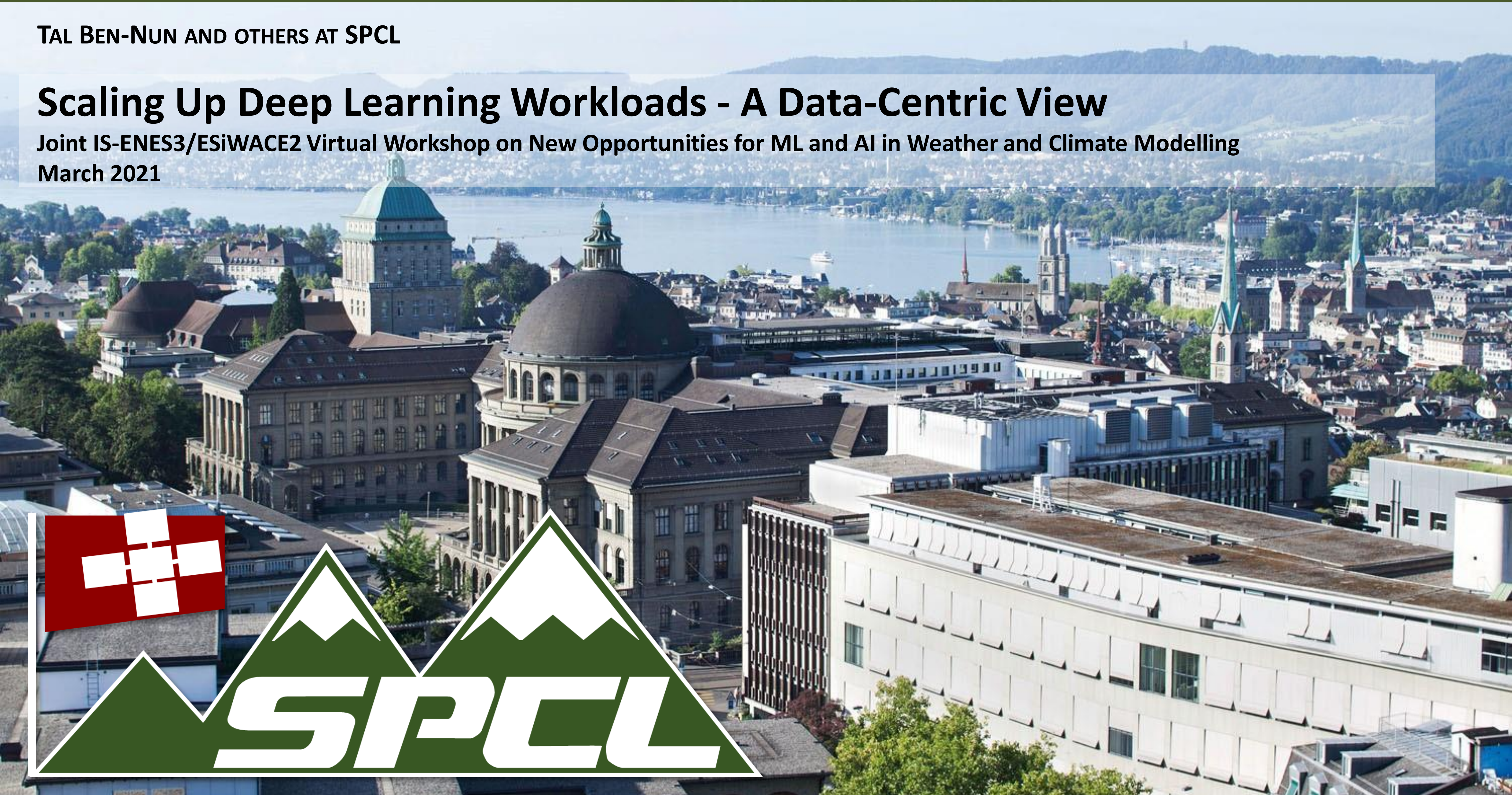
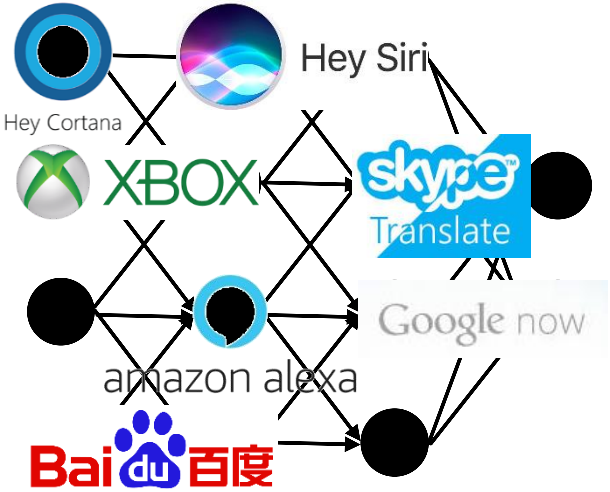
TAL BEN-NUN AND OTHERS AT SPCL

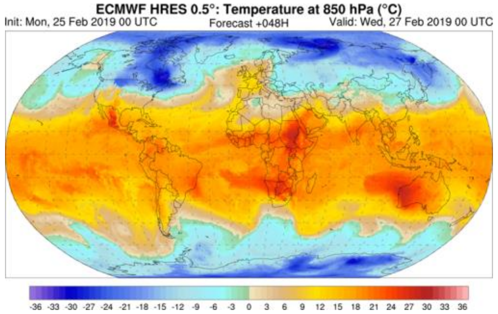
Scaling Up Deep Learning Workloads - A Data-Centric View

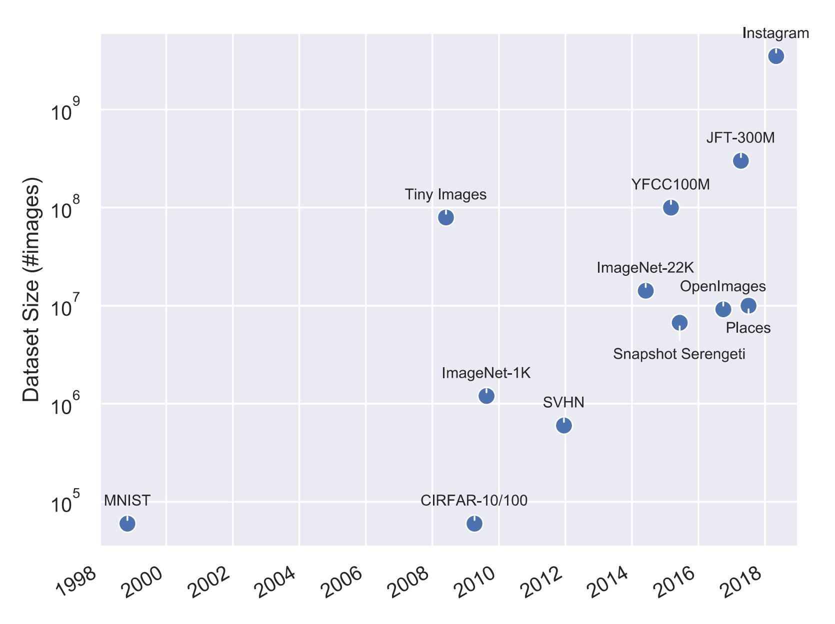
Joint IS-ENES3/ESiWACE2 Virtual Workshop on New Opportunities for ML and AI in Weather and Climate Modelling

March 2021

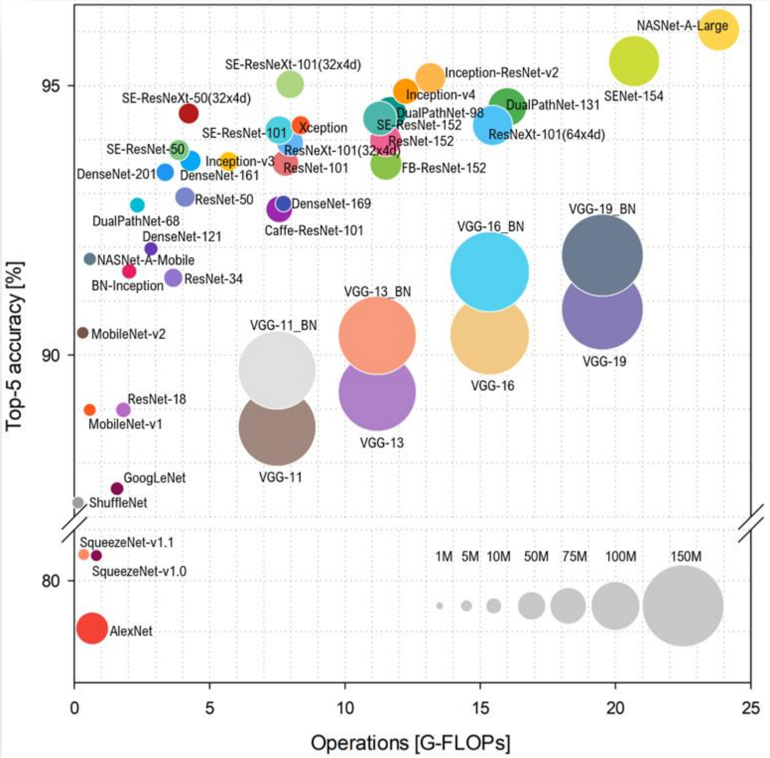






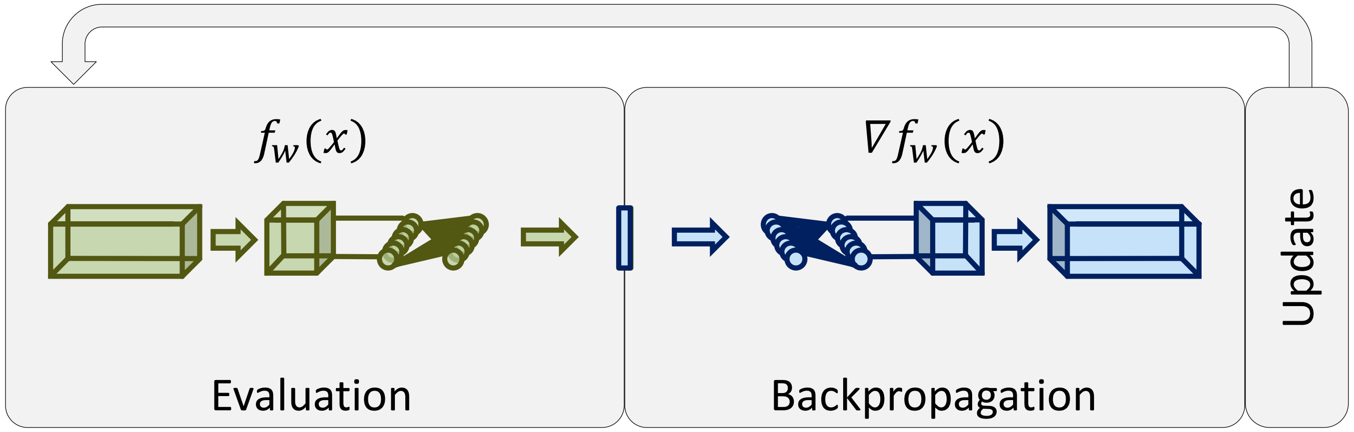


≥TBs of random access

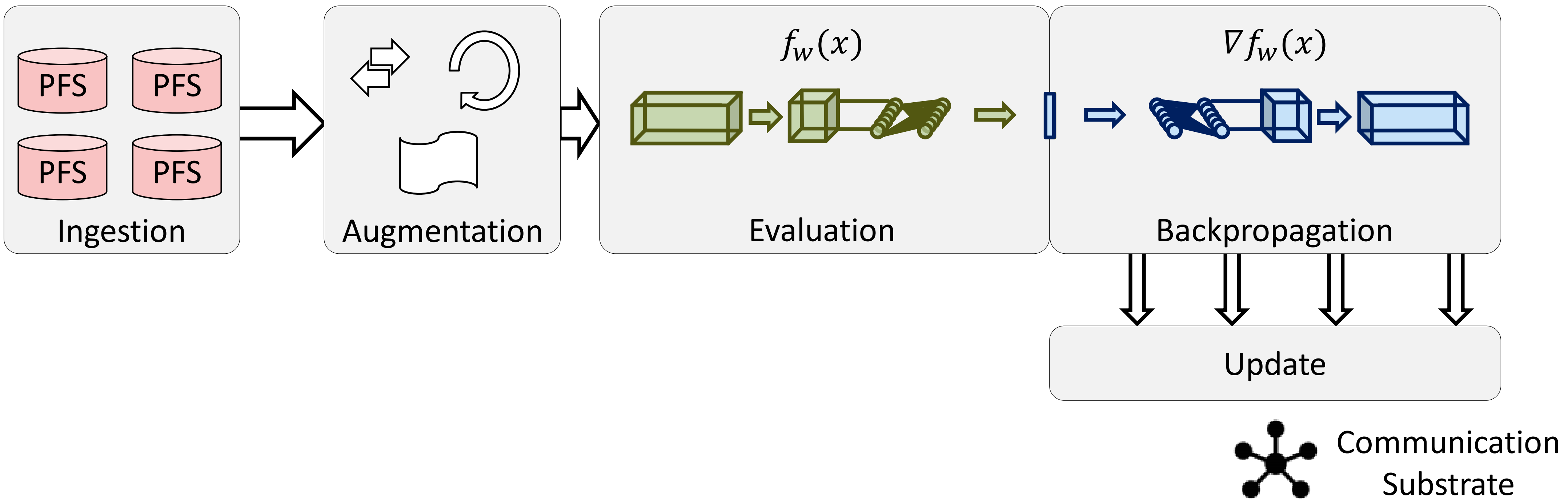


100MiB-32GiB and beyond

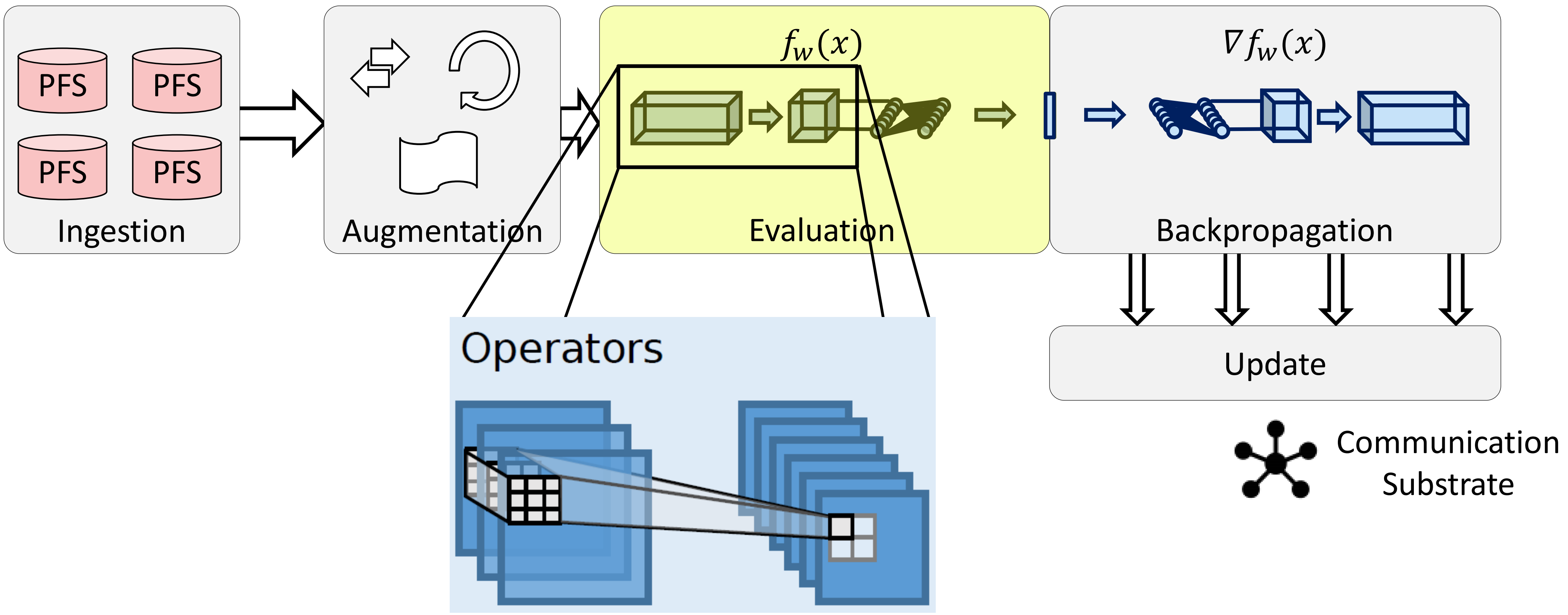
Machine Learning Pipeline



Machine Learning Pipeline



Machine Learning Pipeline



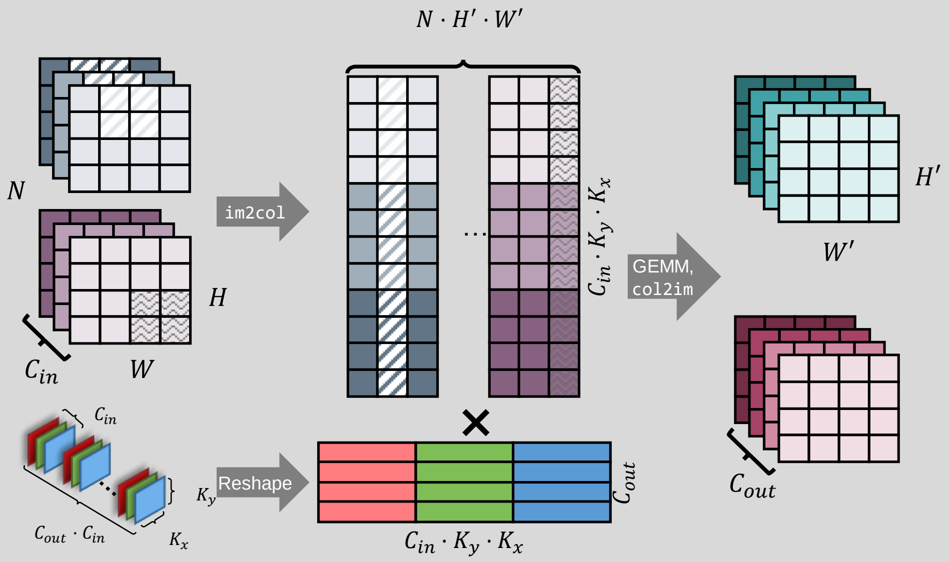
Example: Options for computing convolutional layers

Direct

Direct

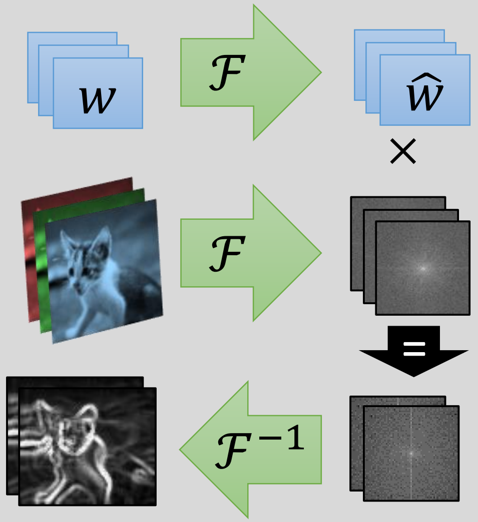
$$\begin{bmatrix}4&1&9&8\\5&9&9&8\\0&7&3&4\\2&6&3&1\end{bmatrix} * \begin{bmatrix}1&-1&0\\0.1&-2&0\\3&4&1.1\end{bmatrix} = \begin{bmatrix}21.9&59.3&53.9&43.9\\-6.3&16.8&12.3&12\\9.6&15.3&25.8&14\\0.4&7.1&52.1&53.1\end{bmatrix}$$

im2col

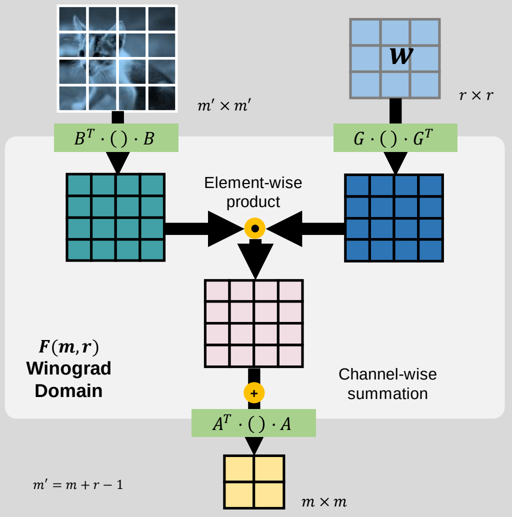


Indirect

FFT

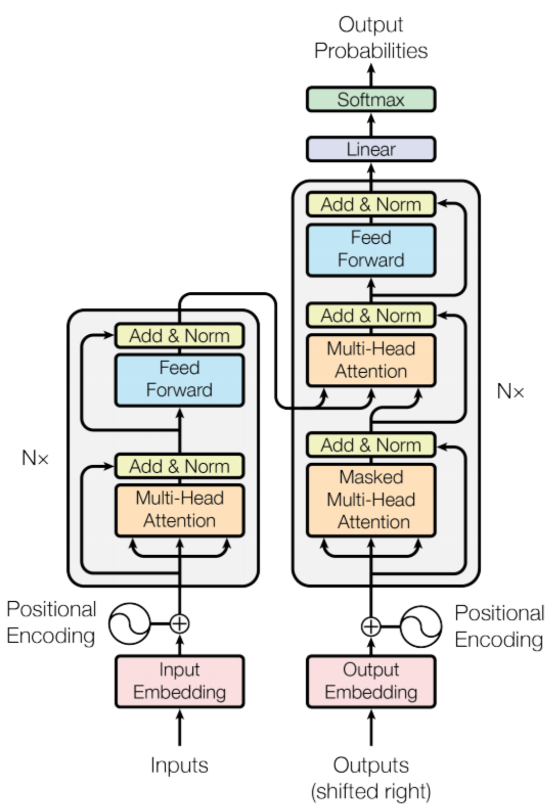


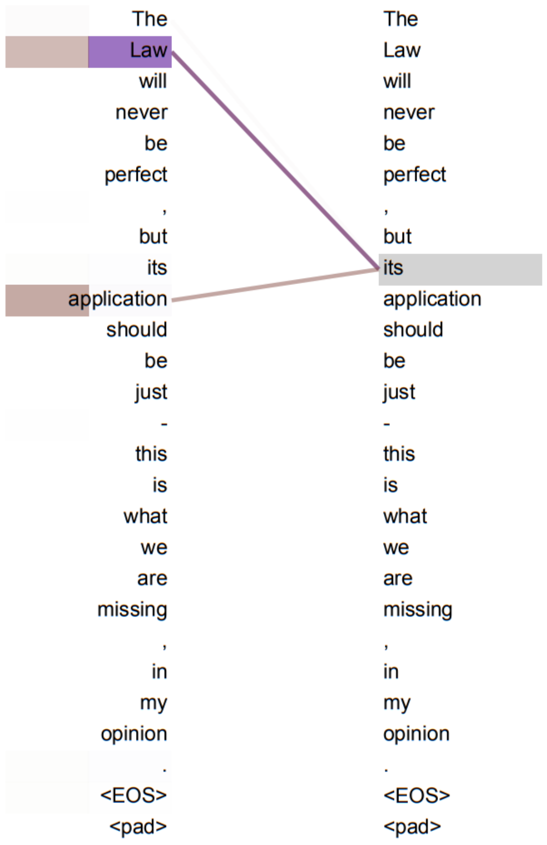
Winograd

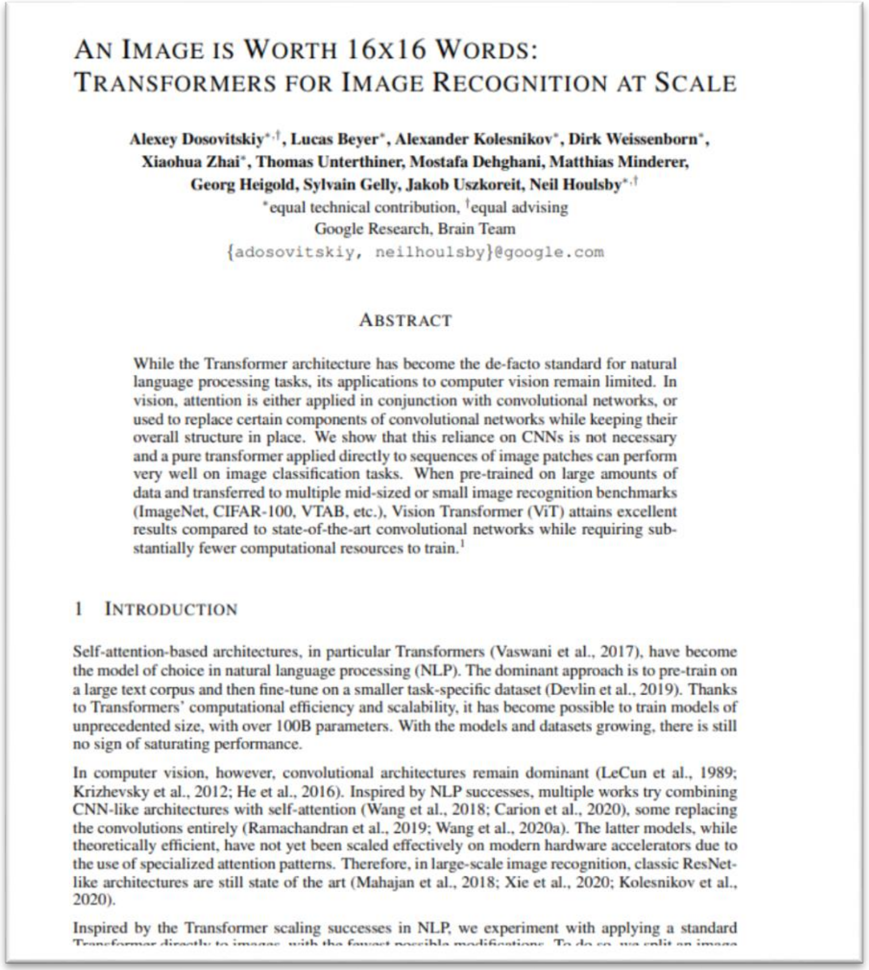


K. Chellapilla et al.: High Performance Convolutional Neural Networks for Document Processing, Int'l Workshop on Frontiers in Handwriting Recognition 2016
M. Mathieu et al.: Fast Training of Convolutional Networks through FFTs, ICLR'14
A. Lavin and S. Gray: Fast Algorithms for Convolutional Neural Networks, CVPR'16

Transformers







AN IMAGE IS WORTH 16X16 WORDS: TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE

Alexey Dosovitskiy[*,†], Lucas Beyer[*], Alexander Kolesnikov[*], Dirk Weissenborn[*], Xiaohua Zhai[*], Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, Neil Houlsby[*,†]
[*]equal technical contribution, [†]equal advising
Google Research, Brain Team
{adosovitskiy, neilhoulsby}@google.com

ABSTRACT

While the Transformer architecture has become the de-facto standard for natural language processing tasks, its applications to computer vision remain limited. In vision, attention is either applied in conjunction with convolutional networks, or used to replace certain components of convolutional networks while keeping their overall structure in place. We show that this reliance on CNNs is not necessary and a pure transformer applied directly to sequences of image patches can perform very well on image classification tasks. When pre-trained on large amounts of data and transferred to multiple mid-sized or small image recognition benchmarks (ImageNet, CIFAR-100, VTAB, etc.), Vision Transformer (ViT) attains excellent results compared to state-of-the-art convolutional networks while requiring substantially fewer computational resources to train.[1]

1 INTRODUCTION

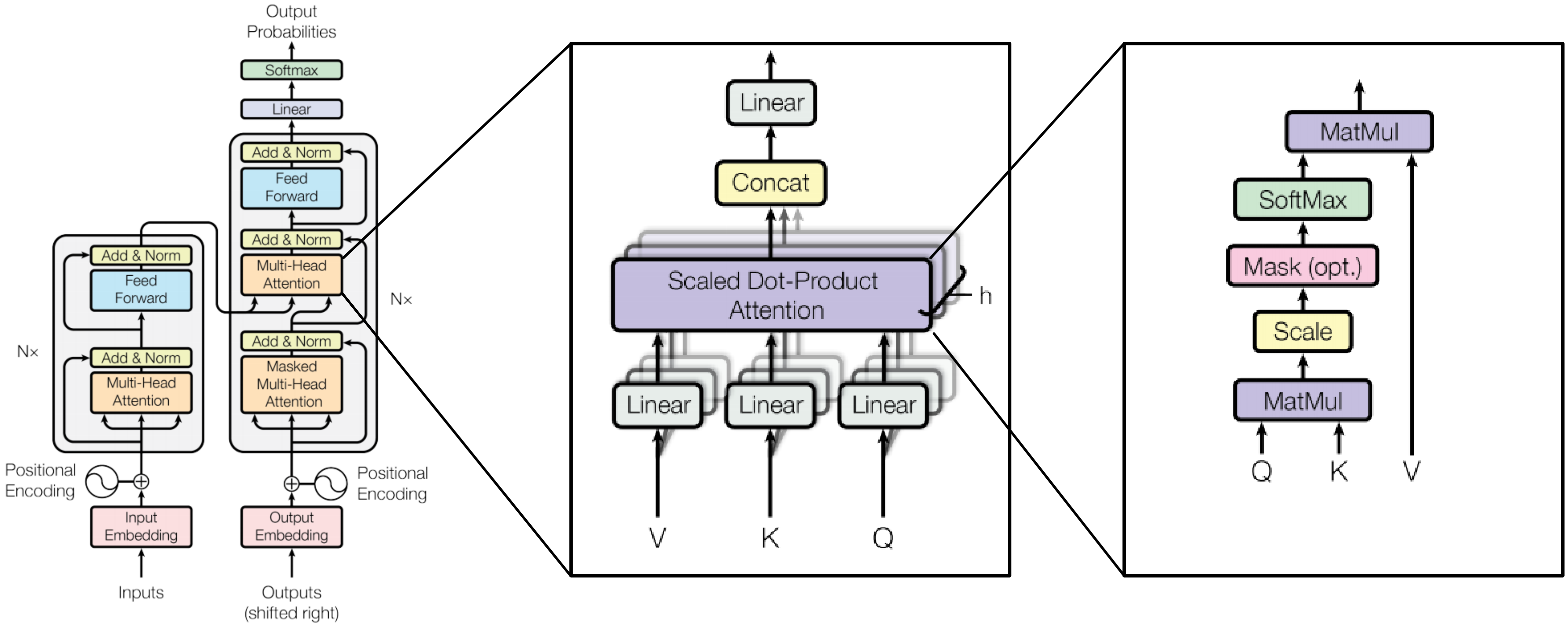
Self-attention-based architectures, in particular Transformers (Vaswani et al., 2017), have become the model of choice in natural language processing (NLP). The dominant approach is to pre-train on a large text corpus and then fine-tune on a smaller task-specific dataset (Devlin et al., 2019). Thanks to Transformers' computational efficiency and scalability, it has become possible to train models of unprecedented size, with over 100B parameters. With the models and datasets growing, there is still no sign of saturating performance.

In computer vision, however, convolutional architectures remain dominant (LeCun et al., 1989; Krizhevsky et al., 2012; He et al., 2016). Inspired by NLP successes, multiple works try combining CNN-like architectures with self-attention (Wang et al., 2018; Carion et al., 2020), some replacing the convolutions entirely (Ramachandran et al., 2019; Wang et al., 2020a). The latter models, while theoretically efficient, have not yet been scaled effectively on modern hardware accelerators due to the use of specialized attention patterns. Therefore, in large-scale image recognition, classic ResNet-like architectures are still state of the art (Mahajan et al., 2018; Xie et al., 2020; Kolesnikov et al., 2020).

Inspired by the Transformer scaling successes in NLP, we experiment with applying a standard

A. Vaswani et al. “Attention Is All You Need,” NeurIPS 2017.

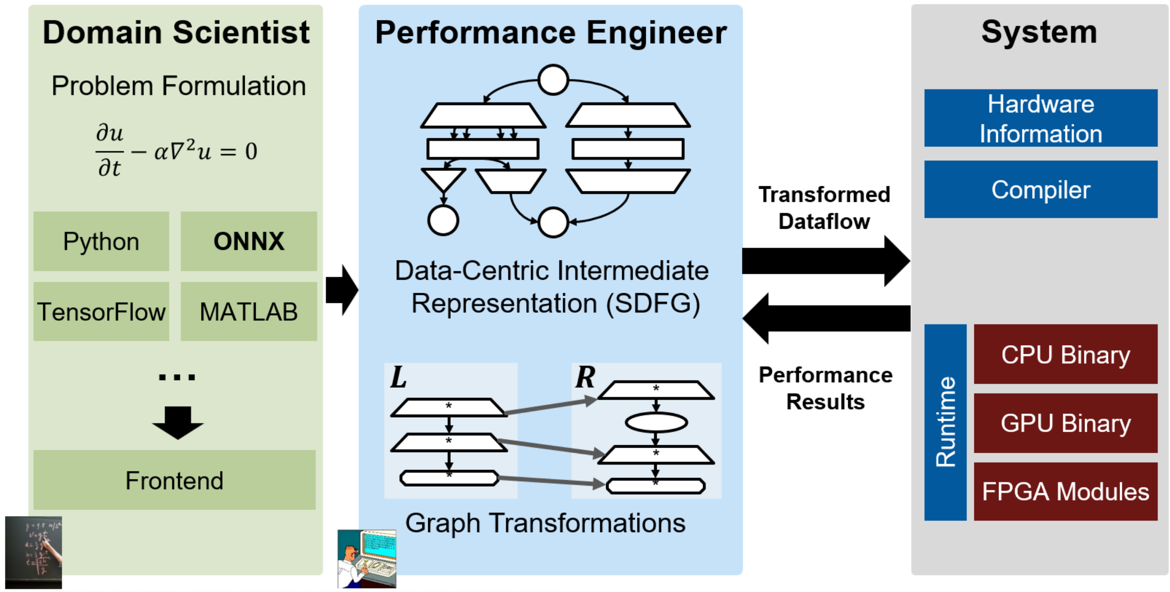
Transformers – Multi-Head Attention

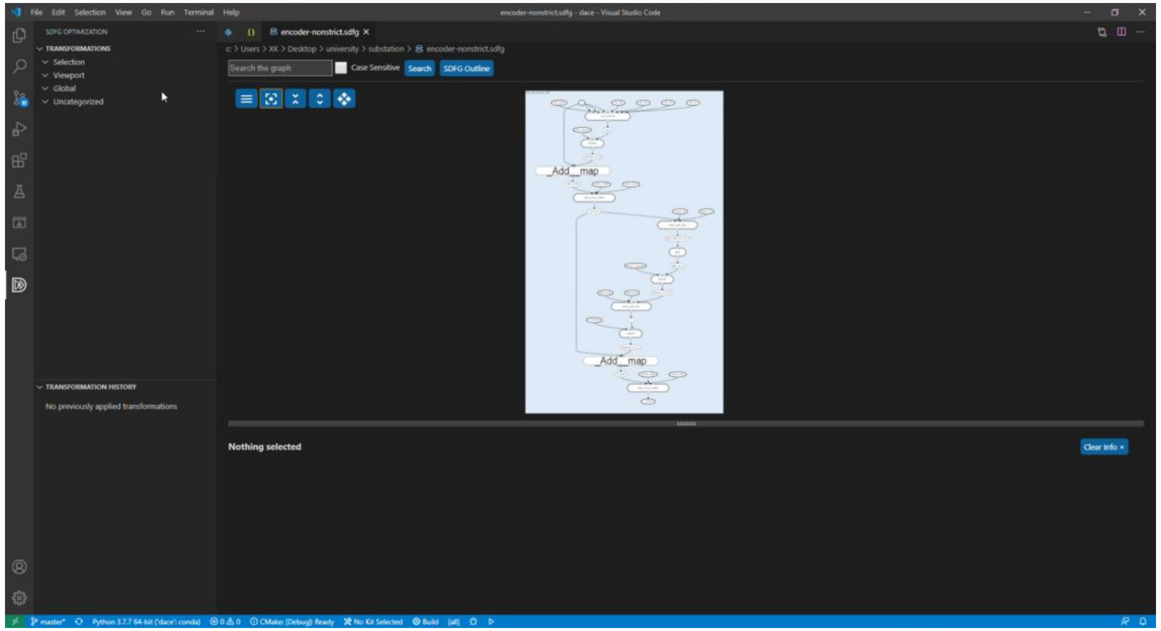


A. Vaswani et al. "Attention Is All You Need," NeurIPS 2017.

DNN Compilers

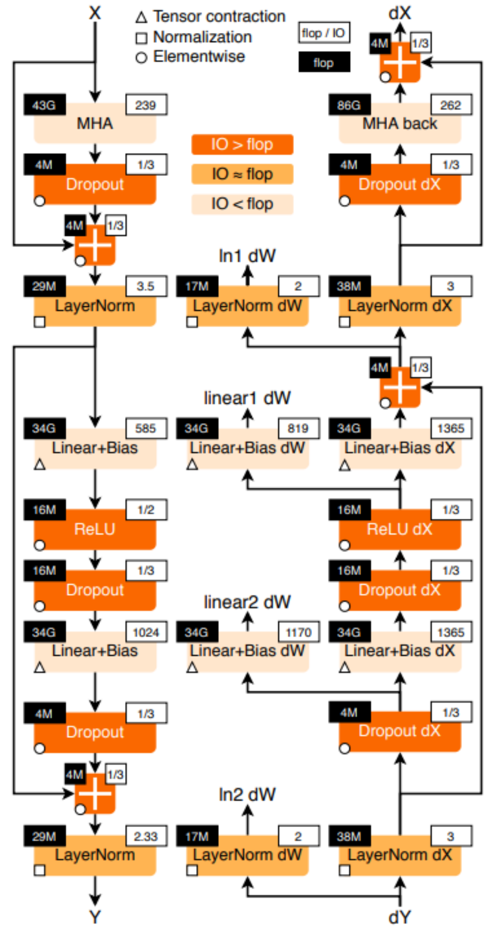
- Use techniques from compiler construction: DNN → Graph → IR → Transformations → HW Mapping



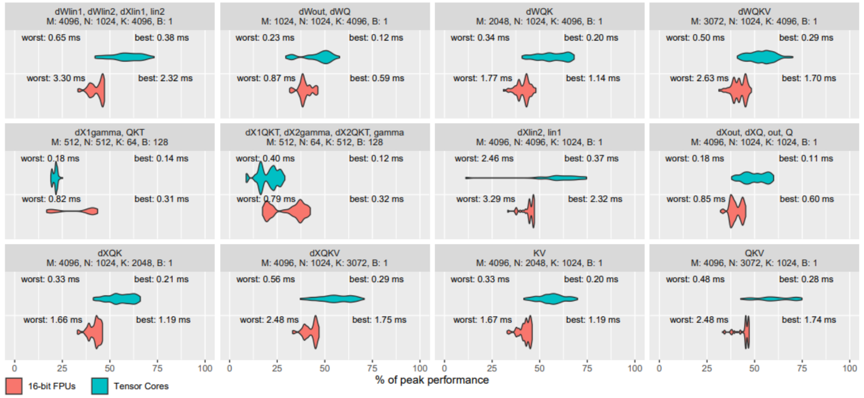


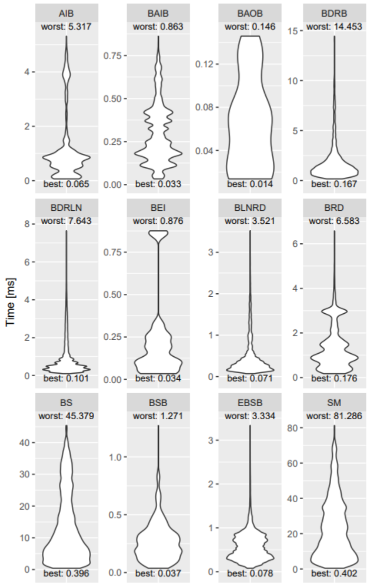
T. Ben-Nun et al. "Stateful Dataflow Multigraphs: A data-centric model for performance portability on heterogeneous architectures", SC 2019.

Data Movement is All You Need

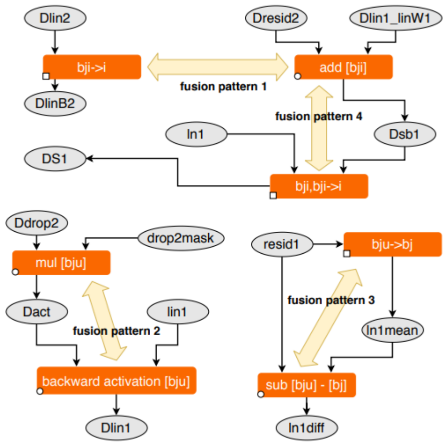


	Operator	Gflop	Input (1e6)	Output (1e6)	PyTorch Gflop	PyTorch Time (μs)	PyTorch % peak	Ours Time (μs)	Ours % peak	Ours MUE	Speedup	Kernel
Forward	▲ Q, K, V	24	7.3	12.5	24.012	333	56.2	306	**61.2**	12	1.08	—
	● Input bias	0.012	12.5	12.5	0.023	90	0.4	66	0.5	**78**	1.35	}AIB
	▲ QK^T	4	8.3	33.5	4.031	189	16.5	143	21.8	**50**	1.32	—
	■ Scaled softmax	0.188	33.5	100.6	0.89	453	1.3	433	1.3	**32**	1.04	}SM
	▲ Gamma	4	37.7	4.1	8.008	142	21.9	160	**19.4**	6	0.88	—
	▲ Out	8	5.2	4.1	8.09	136	45.9	120	**52**	10	1.13	—
	● Output bias	0.004	4.1	4.1	0.008	34	0.4	102	0.1	**42**	1.68	DRLN
	○ Dropout	0.004	4.1	8.3	0.013	37	0.3					
	○ Residual	0.004	8.3	4.1	0.008	36	0.3					
	□ LayerNorm	0.027	4.1	4.1	0.048	63	1.3					
	△ Linear	32	8.3	16.7	32.016	451	55.4	402	**62.1**	12	1.12	—
	○ Bias	0.016	16.7	16.7	0.031	116	0.4	183	0.3	**76**	1.90	BRD
	○ ReLU	—	16.7	16.7	—	112	0					
	○ Dropout	0.016	16.7	33.5	0.048	120	0.4					
	△ Linear	32	20.9	4.1	32.09	449	55.6	369	**67.6**	6	1.21	—
	○ Bias	0.004	4.1	4.1	0.008	35	0.3	101	0.1	**43**	1.70	BDRLN
	○ Dropout	0.004	4.1	8.3	0.013	37	0.3					
	○ Residual	0.004	8.3	4.1	0.008	37	0.3					
	□ LayerNorm	0.027	8.3	4.1	0.048	63	1.3					
Backward	□ LayerNorm dW	0.016	8.3	<0.01	0.02	184	0.3	150	0.3	**6**	1.22	}BSB
	□ LayerNorm dX	0.035	8.3	4.1	0.06	78	1.4	71	1.5	**37**	1.58	BLNRD
	○ Dropout dX	0.004	8.3	4.1	0.008	34	0.4					
	△ Linear+Bias dX	32	8.3	16.7	32.016	427	58.4	414	**60.3**	5	1.03	—
	△ Linear dW	32	20.9	4.1	32.027	424	58.9	378	**66**	13	1.11	—
	□ Bias dW	0.004	4.1	<0.1	0.005	24	0.5	362	<0.1	**38**	1.05	BDRB
	○ Dropout dX	0.016	33.5	16.7	0.031	129	0.4					
	○ ReLU dX	—	33.5	16.7	—	166	0					
	□ Bias dW	0.016	16.7	<0.1	0.02	61	0.8					
	△ Linear+Bias dX	32	20.9	4.1	32.027	417	59.9	398	**62.7**	6	1.04	—
	△ Linear dW	32	20.9	4.1	32.09	437	57.2	372	**67.2**	6	1.17	—
	○ Residual	0.004	8.3	4.1	0.008	36	0.3	250	<0.1	**17**	0.89	EBSB
	□ LayerNorm dW	0.016	8.3	<0.1	0.02	186	0.3					
	□ LayerNorm dX	0.035	8.3	4.1	0.06	80	1.4	69	1.6	**37**	1.64	BLNRD
	○ Dropout dX	0.004	8.3	4.1	0.008	34	0.4					
	■ Output bias dW	0.004	4.1	<0.1	0.005	23	0.5	38	0.3	**22**	0.60	}BAOB
	▲ Out dX	8	4.3	4.1	8.044	131	47.6	119	**52.2**	10	1.09	—
	▲ Out dW	8	8.3	1.0	8.09	136	45.9	113	**54.8**	5	1.19	—
	▲ Gamma dX1	4	8.3	33.5	8.008	136	22.8	147	**21.2**	7	0.93	—
	▲ Gamma dX2	4	67.1	33.5	4.031	188	16.6	123	**25.2**	8	1.52	—
	■ Scaled softmax dX	0.156	12.5	4.1	0.199	790	0.6	426	1.1	**49**	1.85	}BS
	▲ QK^T dX1	4	37.7	4.1	4.004	135	23.1	155	**20**	7	0.86	—
	▲ QK^T dX2	4	37.7	4.1	8.008	139	22.3	115	**26.9**	9	1.20	—
	▲ Q, K, V dX	24	15.7	4.1	24.027	344	54.4	274	**68.2**	6	1.25	—
	▲ Q, K, V dW	24	20.4	1.0	24.132	329	57	293	**64**	6	1.12	—
	■ Input bias dW	0.012	12.5	<0.1	0.015	52	0.7	39	0.9	**66**	1.32	}BAIB
	○ Residual	0.004	8.3	4.1	0.008	35	0.3	31	0.4	**83**	1.14	}BEI
	△ Tensor contractions	312	—	—	324.75	4951	43.1	4411	48.5	—	1.12	
	□ Stat. normalizations	0.535	—	—	1.389	2063	0.9	1591	0.6	—	1.29	
	○ Element-wise	0.098	—	—	0.223	1096	0.3	735	0.1	—	1.49	
	Total	312.633	—	—	326.362	8110	31.1	6739	35	—	1.20	



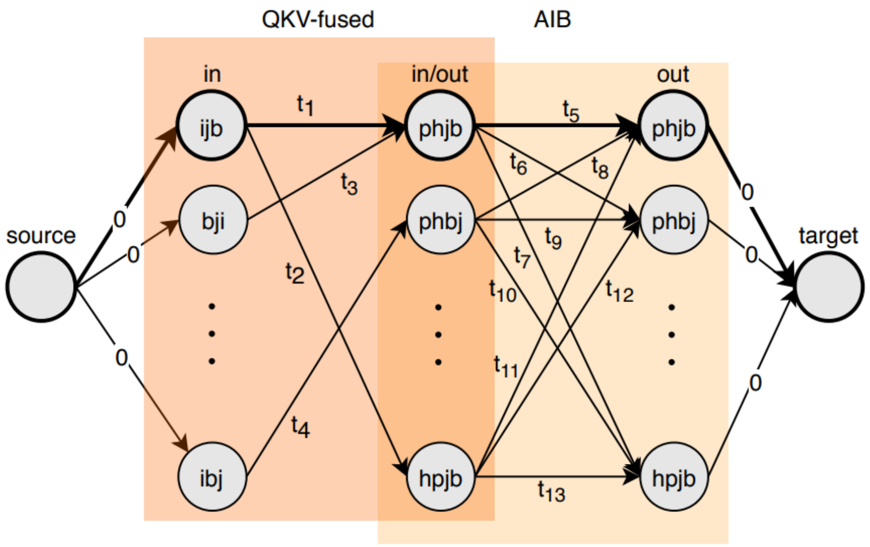


Data Movement is All You Need



	TF+XLA	PyTorch	cuDNN	Ours
Forward (ms)	1.60	1.90	131	**1.25**
Backward (ms)	2.25	2.77	652	**1.86**

Multi-Head Attention

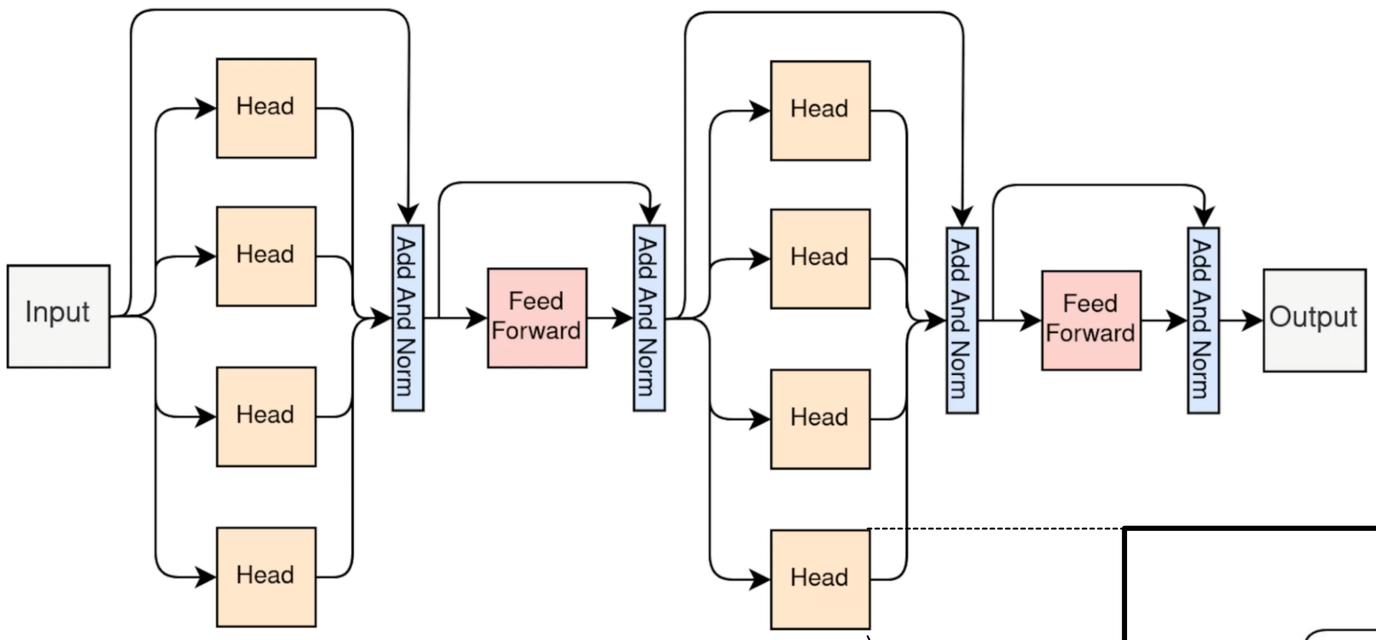


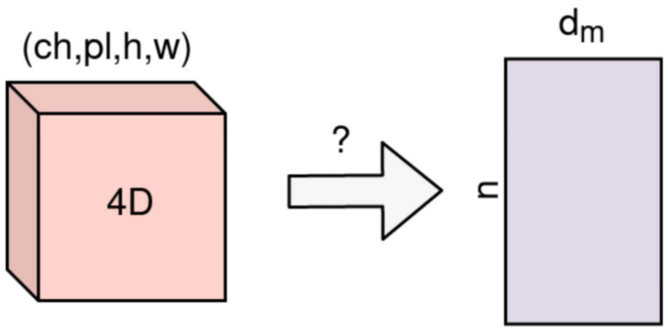
	TF+XLA	PyTorch	DeepSpeed	Ours
Forward (ms)	3.20	3.45	2.80	**2.63**
Backward (ms)	5.20	5.69	4.80	**4.38**

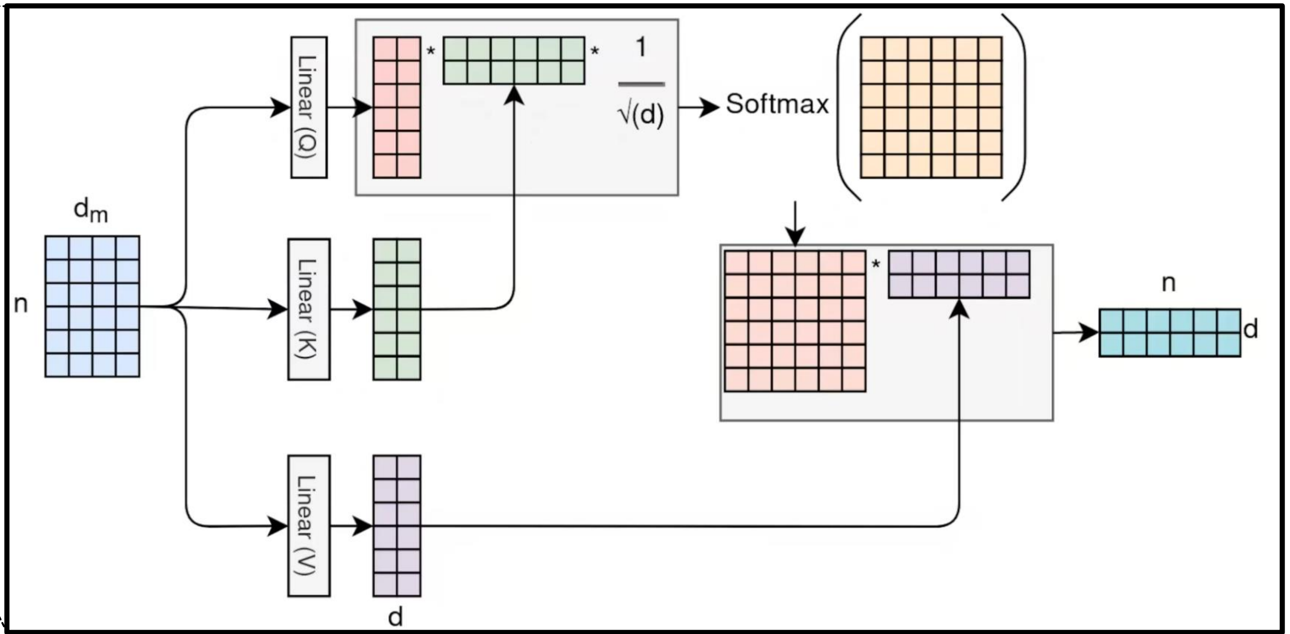
Full BERT Encoder Layer

A. Ivanov et al. "Data Movement Is All You Need: A Case Study on Optimizing Transformers", MLSys 2021. https://arxiv.org/abs/2007.00072

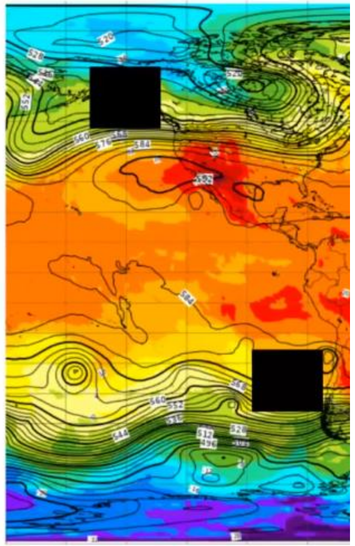
Transformers



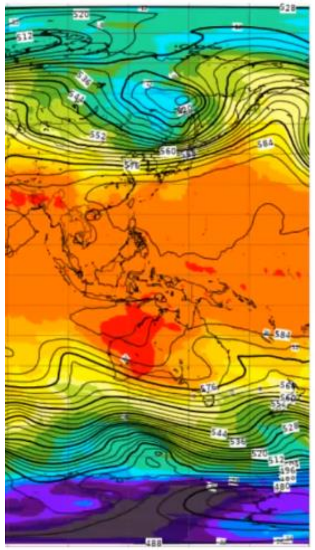




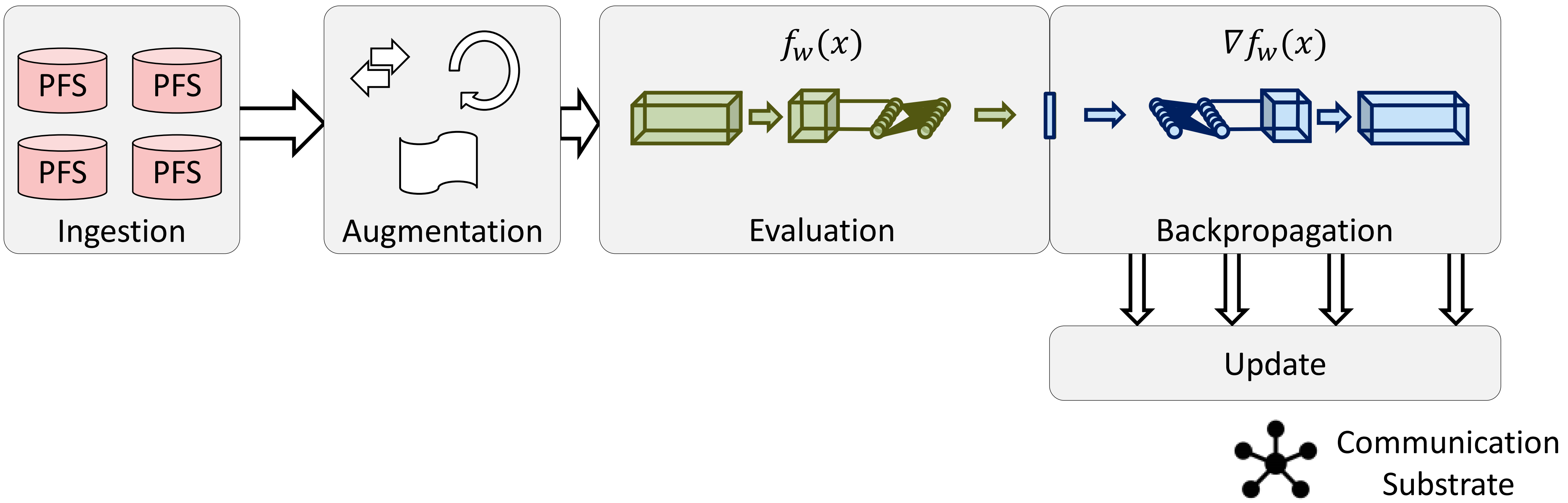
Pretraining Transformers on Weather Data



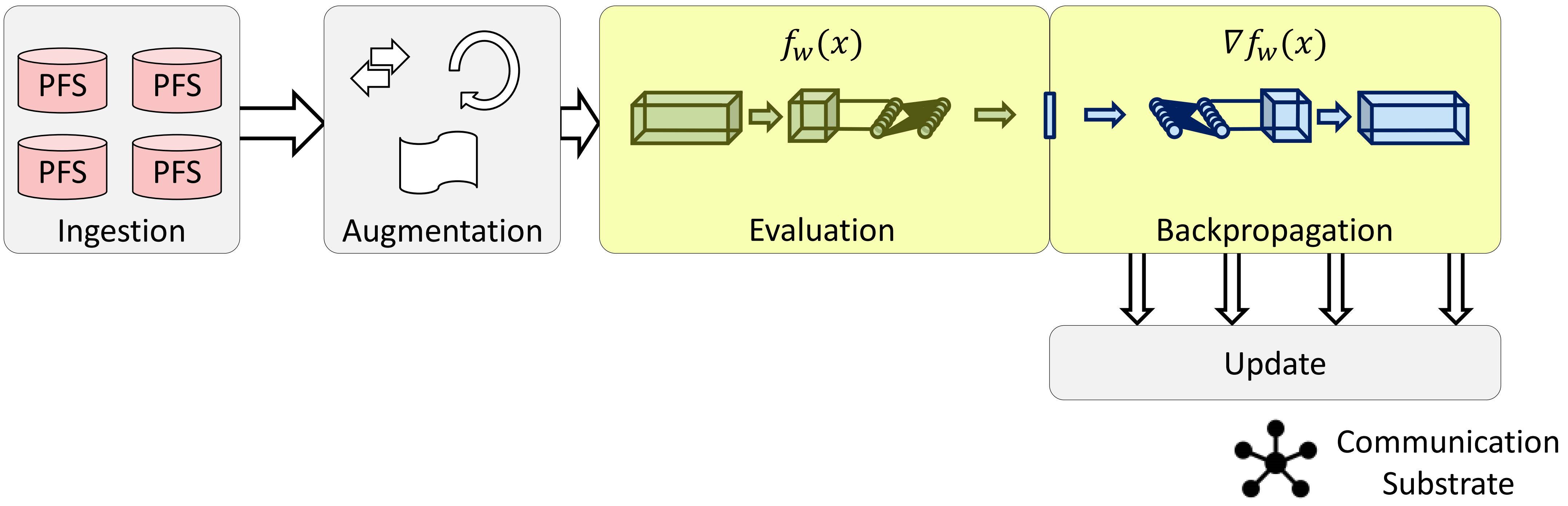
Input Indices	Error	$\sigma(10^{-3})$	Epochs	Parameters	Notes
ch	0.2091/0.2155	4.13/4.55	6.2	1.08B	No FF
pl	**0.1944/0.2028**	22.34/24.08	10.8	849M	No FF
h	0.2010/0.2082	0.39/0.39	**16.6**	**291M**	No FF
w	0.2013/0.2085	**0.02/0.02**	7.0	957M	
ch,pl	0.1539/0.1594	3.00/3.07	8.0	802M	
ch,h	0.1367/0.1419	0.43/0.50	27.8	57.8M	
ch,w	0.1385/0.1437	0.23/0.29	23.8	4.89M	
pl,h	0.1356/0.1403	**0.19/0.27**	**28.4**	93.6M	
pl,w	0.1341/0.1391	0.50/0.60	27.2	7.93M	
h,w	0.1330/0.1378	4.26/4.23	27.8	**575K**	
h,w	0.1388/0.1434	2.84/2.71	21.0	4.15M	No FF
h,w	**0.1324/0.1377**	1.72/1.76	24.4	604K	Linformer
ch,pl,h	**0.1431/0.1482**	0.11/**0.07**	**32.4**	483K	
ch,pl,w	0.1499/0.1555	**0.09**/0.08	30.2	44.5K	Linformer
pl,h,w	0.1479/0.1532	0.14/0.16	25.2	5.52K	Linformer
ch,h,w	0.1502/0.1559	0.11/0.15	19.4	**3.39K**	Linformer



Machine Learning Pipeline

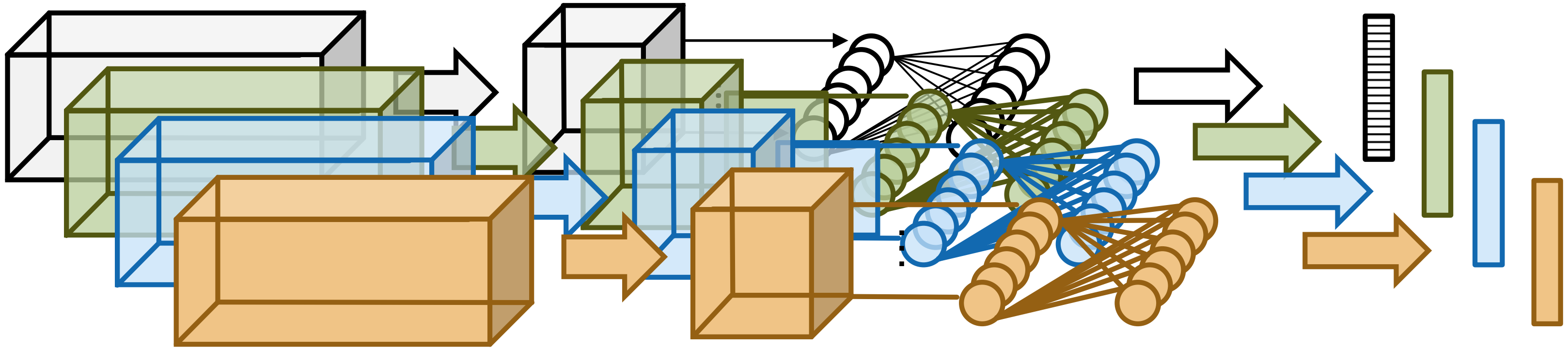


Machine Learning Pipeline



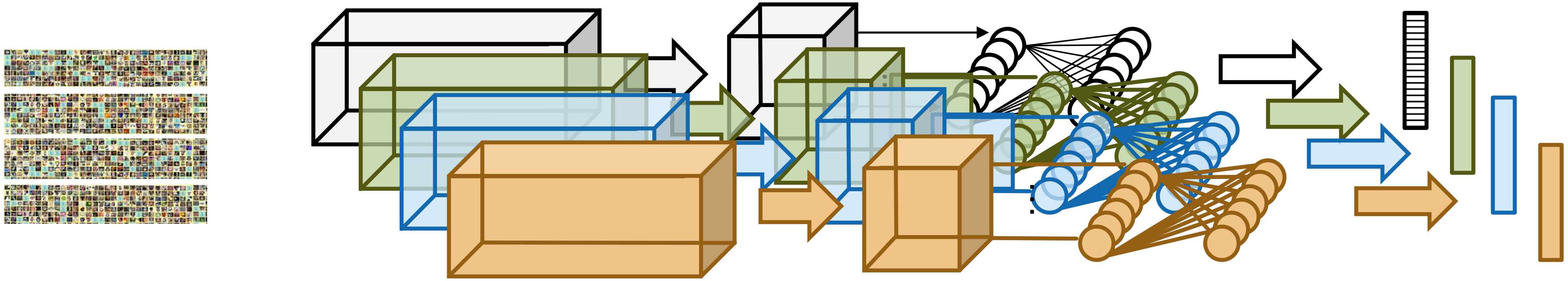
Partitioning Computation?

Data Parallelism

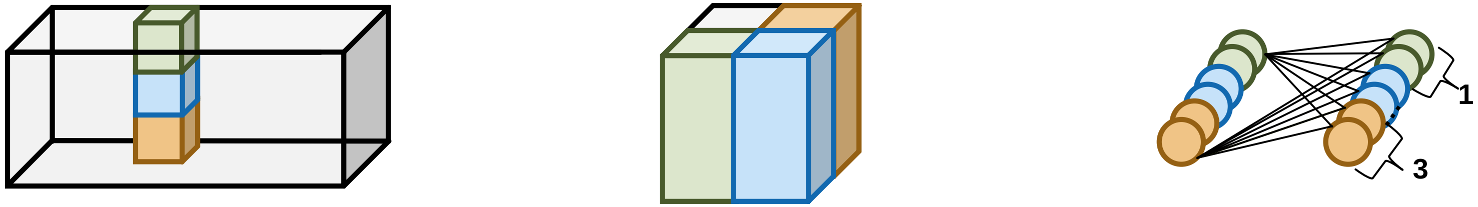


Partitioning Computation?

Data Parallelism

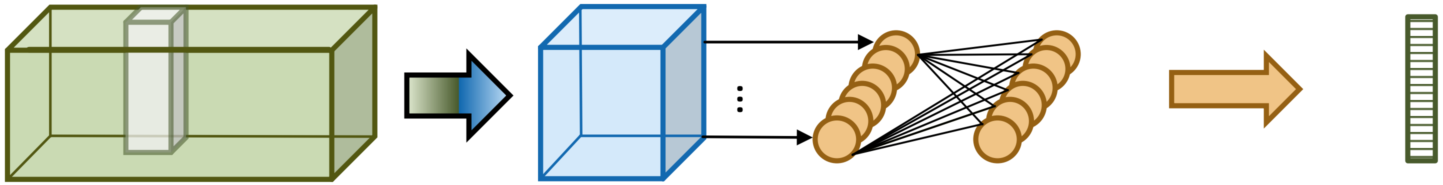


Model Parallelism



Channel/Filter   Spatial   Layer

Pipeline Parallelism



N. Dryden et al. “Improving strong-scaling of CNN training by exploiting finer-grained parallelism”, IPDPS 2019.
N. Dryden et al. “Channel and filter parallelism for large-scale CNN training”, SC 2019.

Partitioning Computation?

Data Parallelism

- Simple and efficient solution, easy to implement
- Duplicate parameters at all processors
- Affects generalization

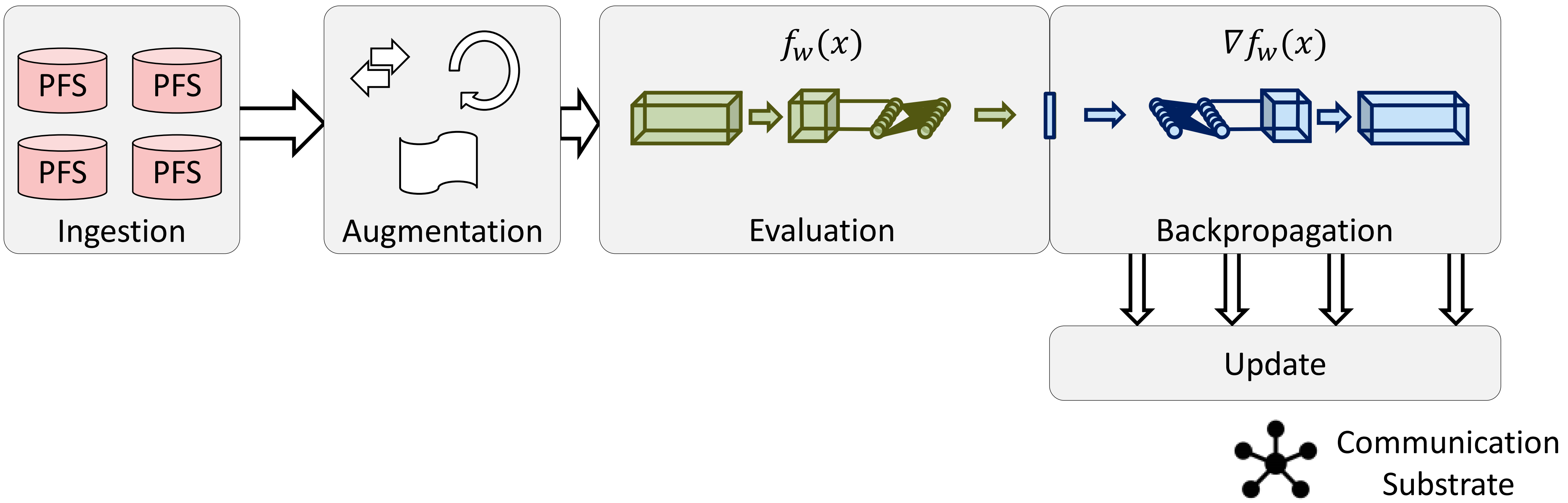
Model Parallelism

- Parameters/domain can be distributed across processors
  - Good for: large inputs, wide networks
- Complex communication per-layer
- Performance hinges on implementation

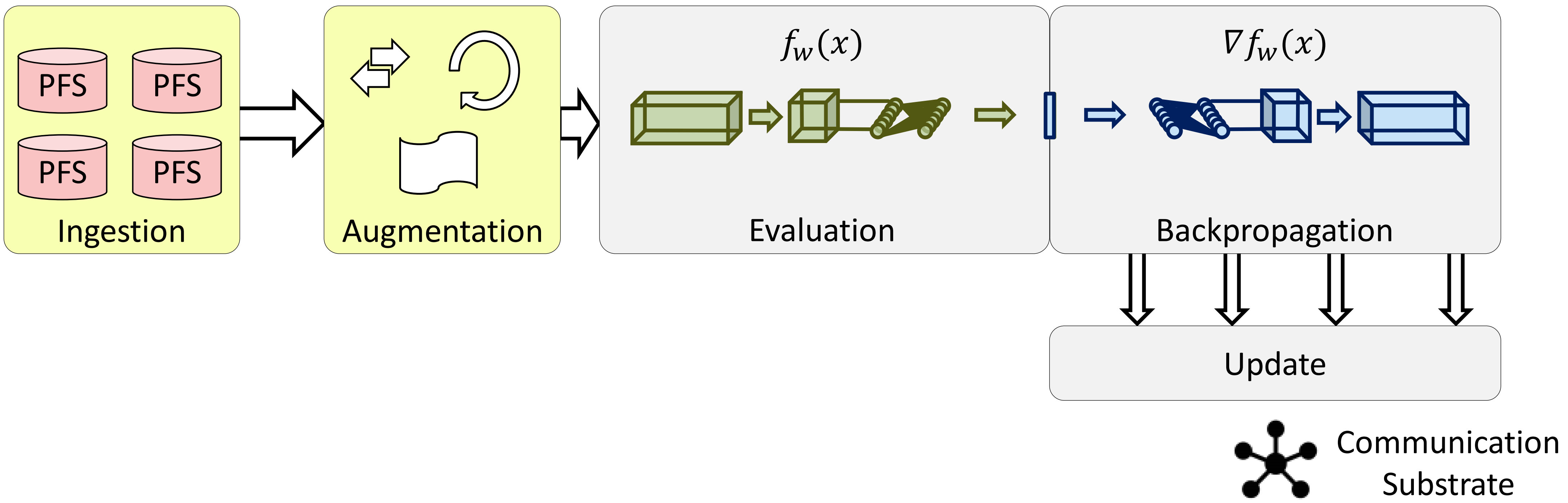
PIpeline Parallelism

- Parameters can be distributed across processors
  - Good for: deep models, few activations
- Sparse communication pattern (only pipeline stages)
- Consistent model introduces idle-time “Bubble”

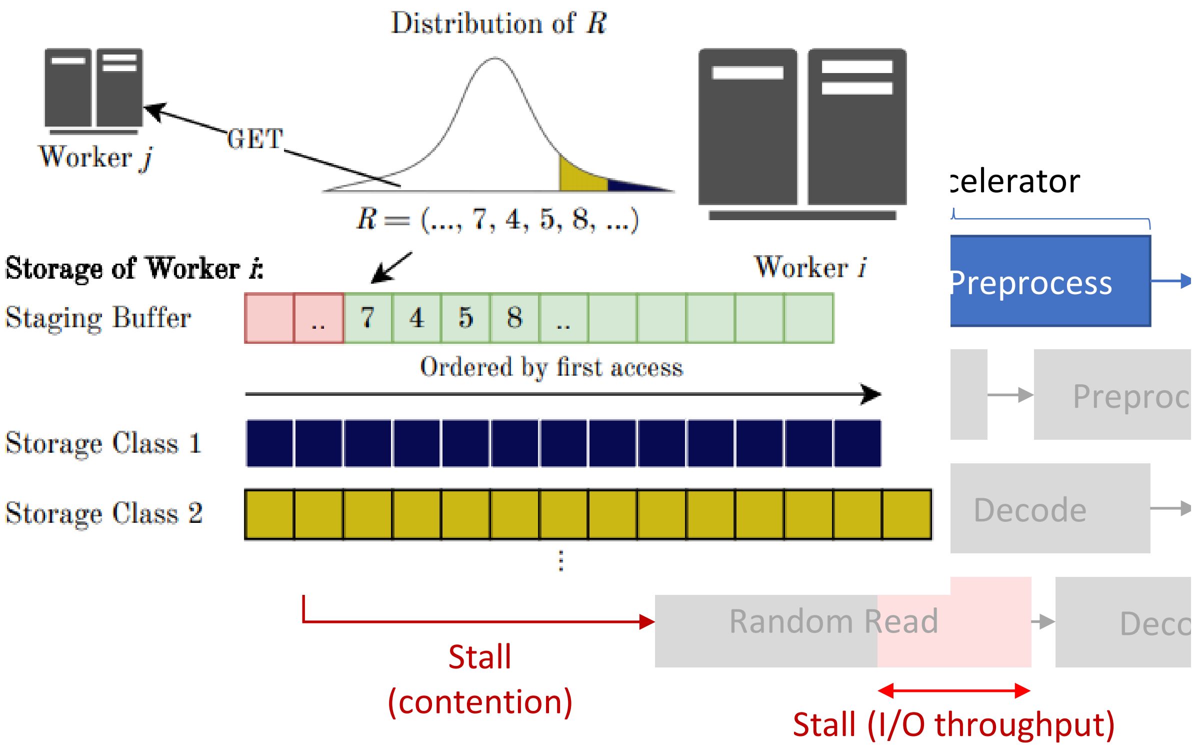
Machine Learning Pipeline

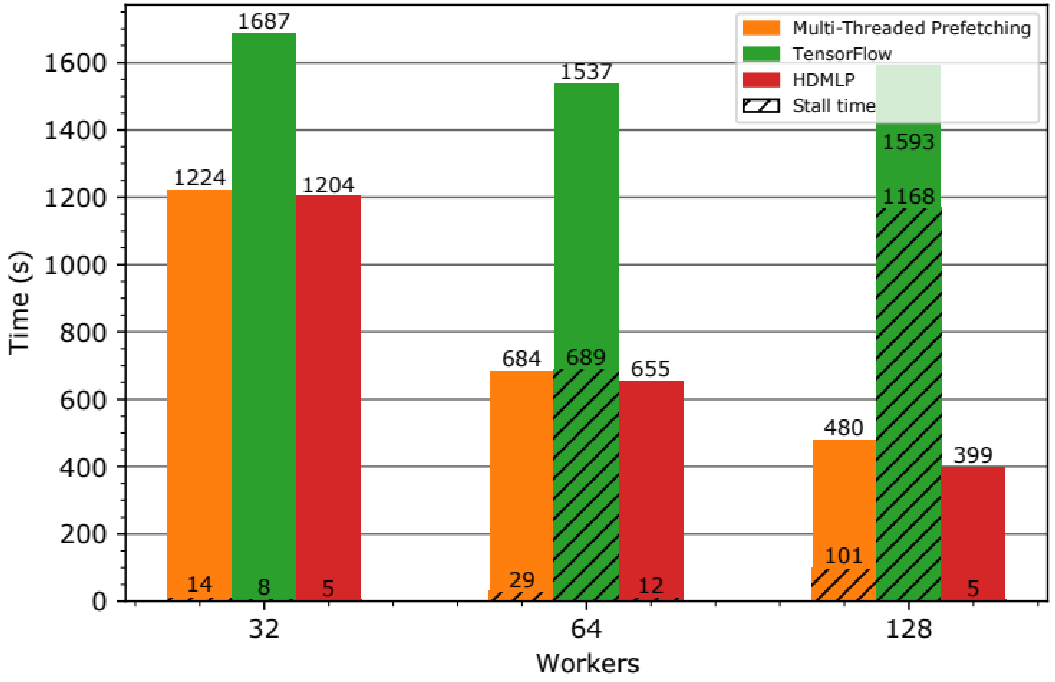


Machine Learning Pipeline



Training is not just Learning





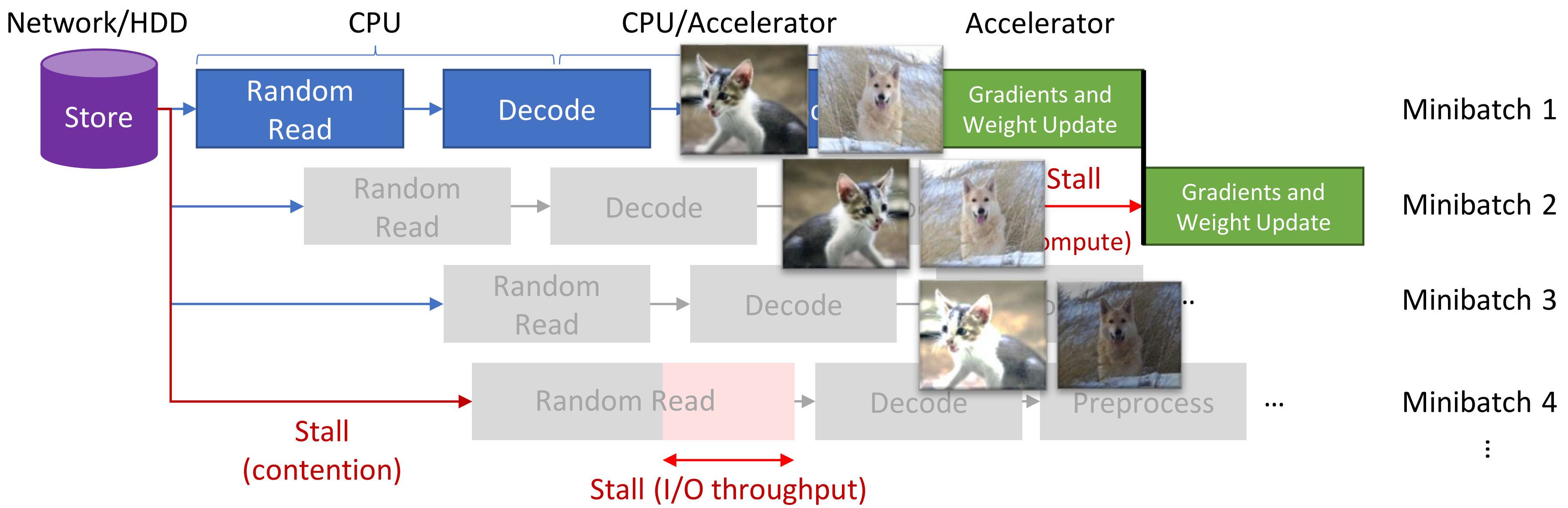
HDMLP

R. Böhringer et al., "Clairvoyant Prefetching for Distributed Machine Learning I/O", arXiv 2021.

Training Pipeline – Data and Batch Augmentation

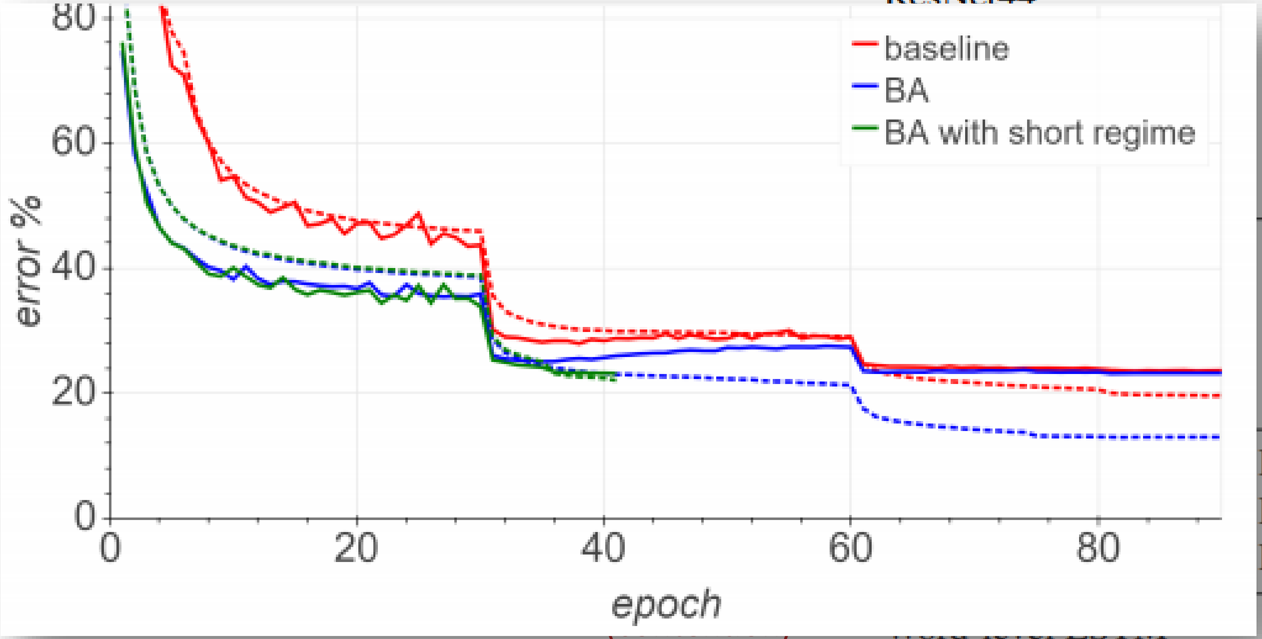
- **Data augmentation is a determining factor in end accuracy**
- **Affects I/O and the training pipeline**

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \frac{1}{B} \sum_{n \in \mathcal{B}(k(t))} \nabla_{\mathbf{w}} \ell\left(\mathbf{w}_t, T(\mathbf{x}_n), \mathbf{y}_n\right)$$



E. Hoffer et al. "Augment Your Batch: Improving Generalization Through Instance Repetition," CVPR 2020.

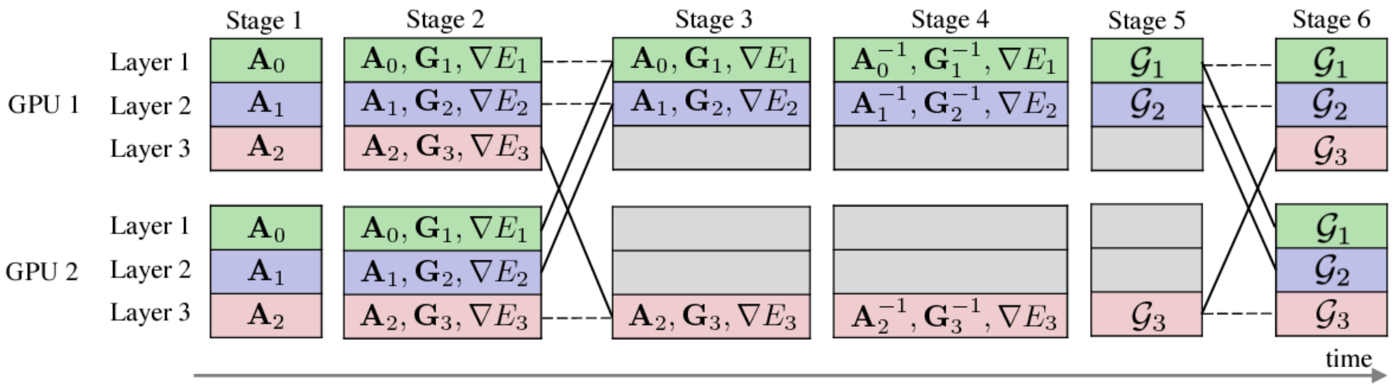
Training Pipeline – Batch Augmentation



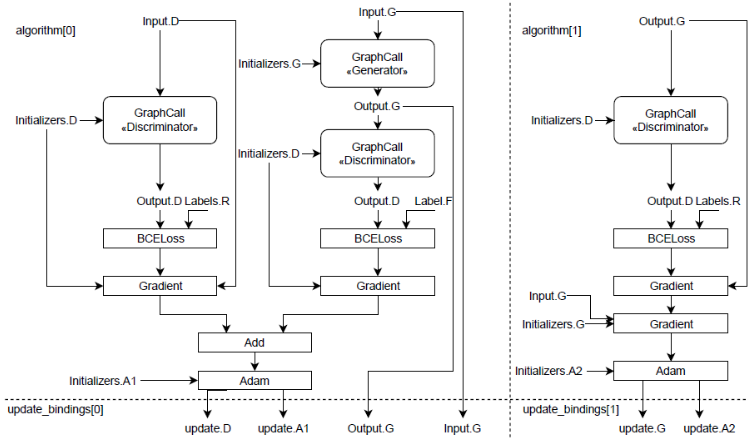
Network	Dataset	M	Baseline Fixed #Steps	Baseline + Fixed #Samples	BA
ResNet44	Cifar10	40	93.70%	93.80%	**95.43%**
	Cifar10	32	93.82%	94.49%	**95.32%**
	Cifar10	6	96.60%	96.60%	**97.15%**
	Cifar10	8	97.65%	97.63%	**97.85%**
	Cifar10	8	98.16%	98.10%	**98.24%**
	Cifar100	40	72.97%	70.30%	**74.13%**
	Cifar100	32	73.03%	67.20%	**75.50%**
	Cifar100	10	79.85%	80.12%	**83.45%**
	Cifar100	4	77.73%	75.35%	**78.80%**
	ImageNet	8	58.25%	57.60%	**62.31%**
	ImageNet	4	70.60%	69.50%	**71.40%**
	ImageNet	4	76.30%	75.70%	**76.86%**
	PTB	10	58.8 ppl	58.8 ppl	**58.6 ppl**
Transformer (base)	WMT En-De	4	26.88 BLEU	27.13 BLEU	**27.49** BLEU

E. Hoffer, T. Ben-Nun et al. “Augment Your Batch: Improving Generalization Through Instance Repetition,” CVPR 2020.

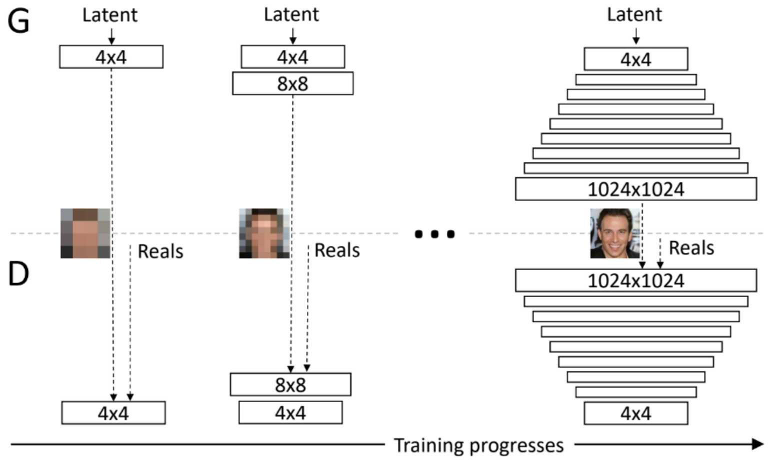
Training is not just Learning



Data/compute redistribution



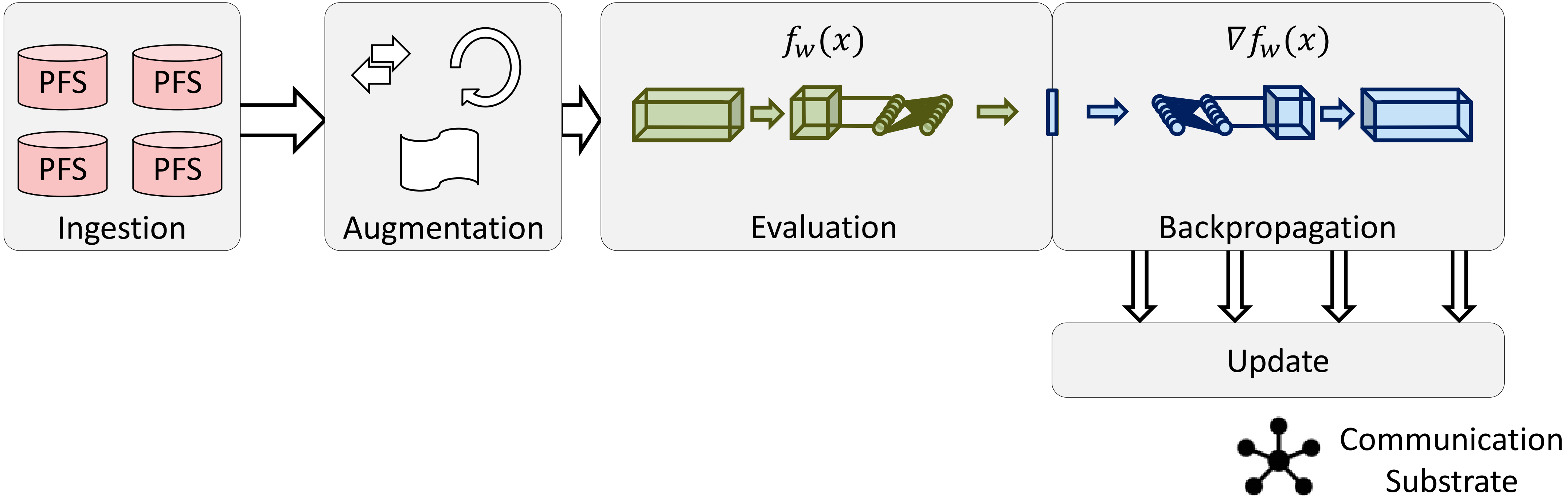
Nontrivial gradient aggregation



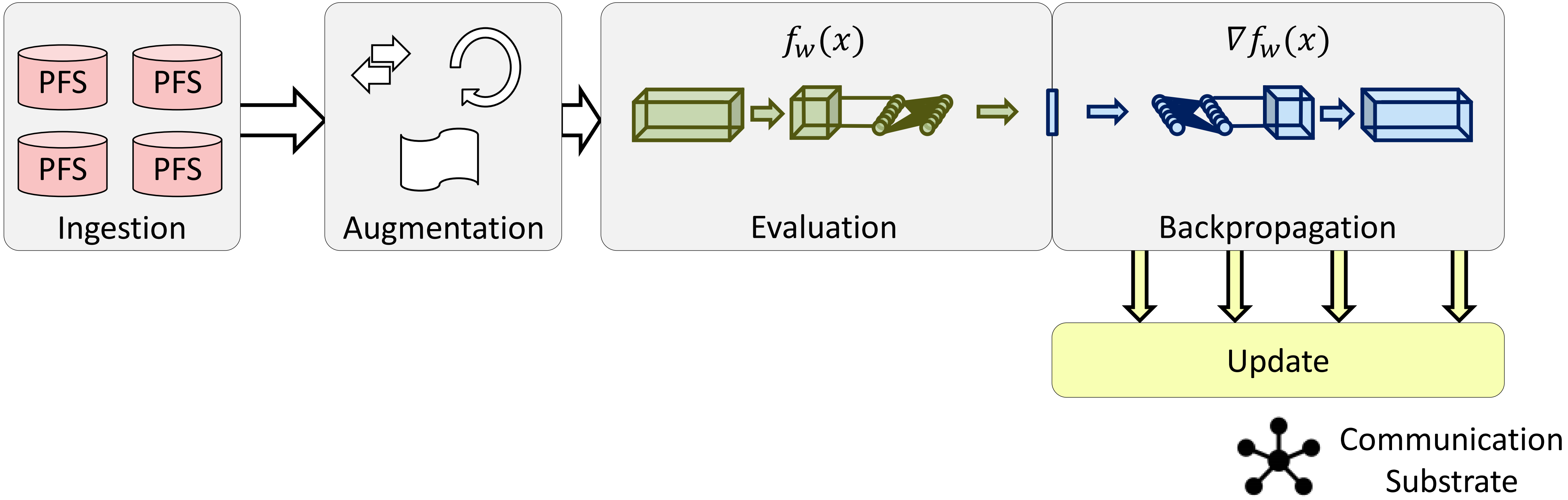
Imbalanced workload over time

K. Osawa et al., “Second-order Optimization Method for Large Mini-batch: Training ResNet-50 on ImageNet in 35 Epochs”, arXiv 2018.
T. Karras et al., “Progressive Growing of GANs for Improved Quality, Stability, and Variation”, arXiv 2017.

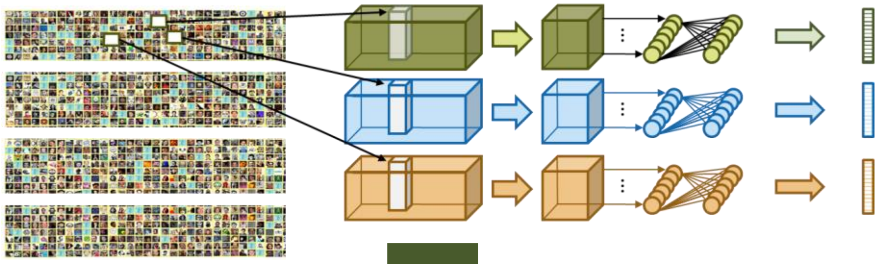
Machine Learning Pipeline

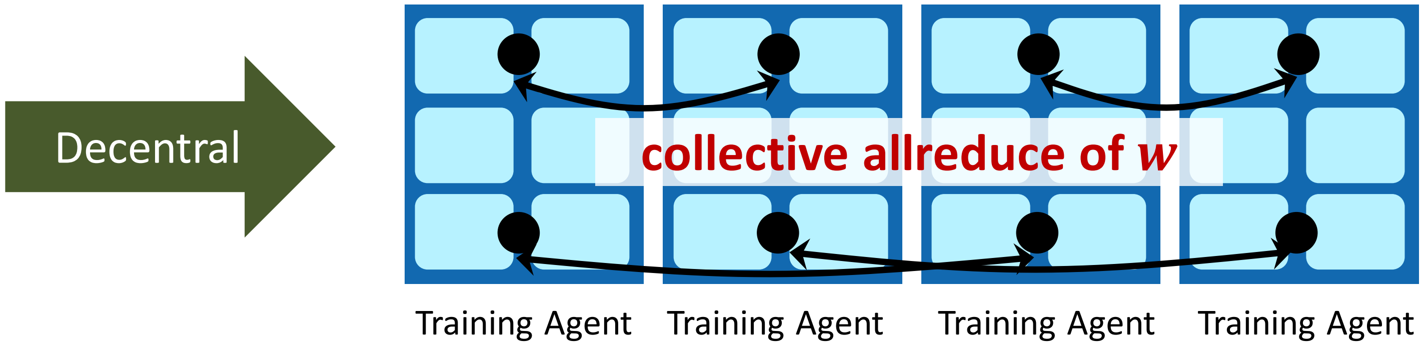


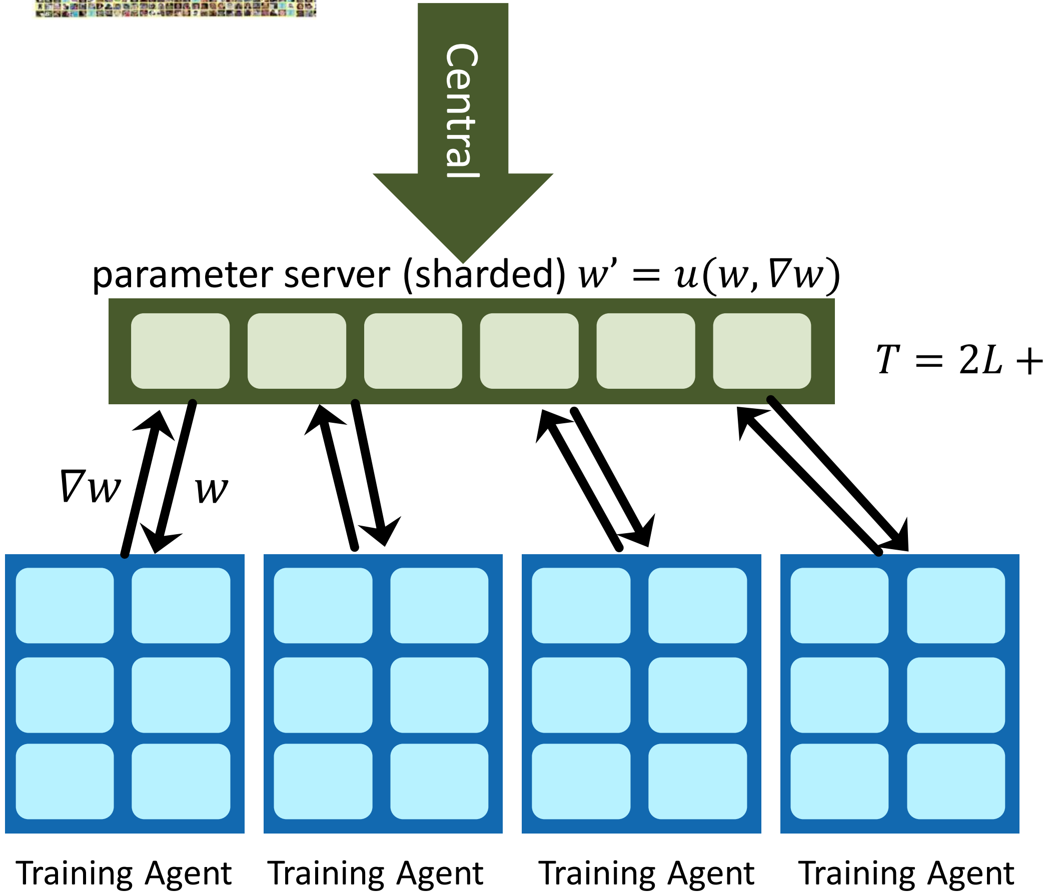
Machine Learning Pipeline



Centralization



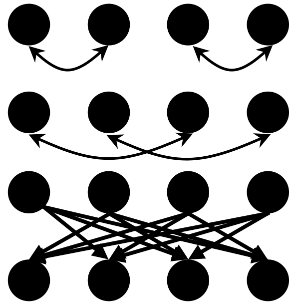




$T = 2L + 2P\ \gamma m/s\ G$

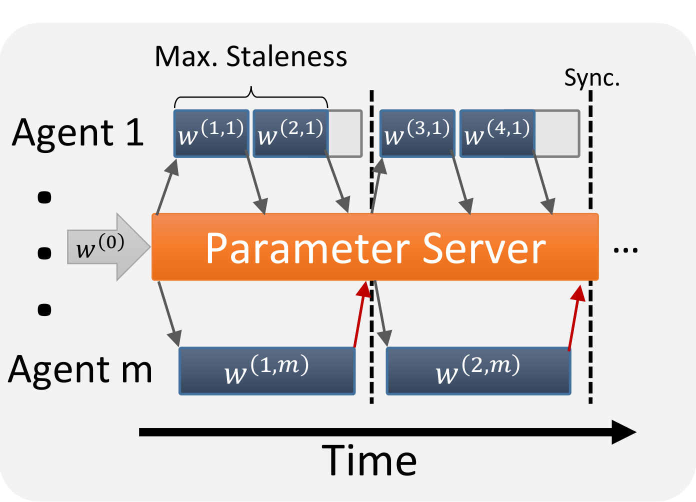
MPI

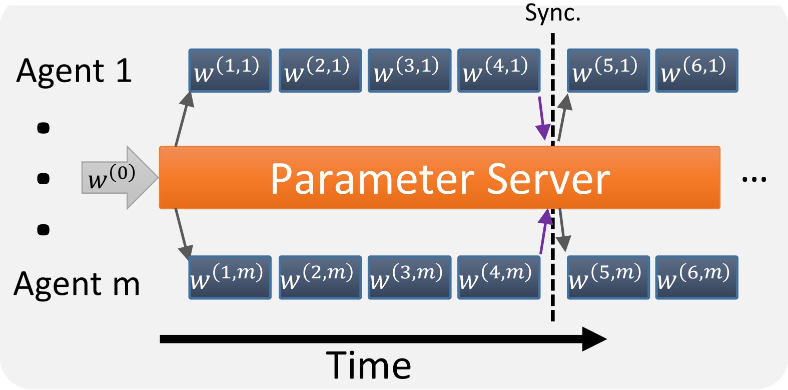
- Collective operations
- Topologies
- Neighborhood collectives
- RMA?

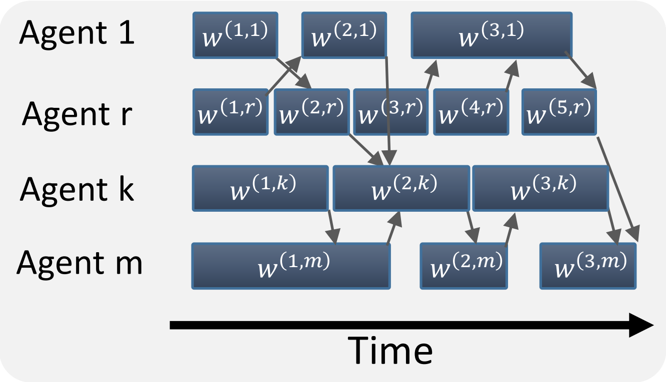


$T = 2L \log_2 P + 2\gamma m G(P-1)/P$

Relaxing parameter consistency







Synchronous SGD

Stale-Synchronous SGD

Asynchronous SGD (HOGWILD!)

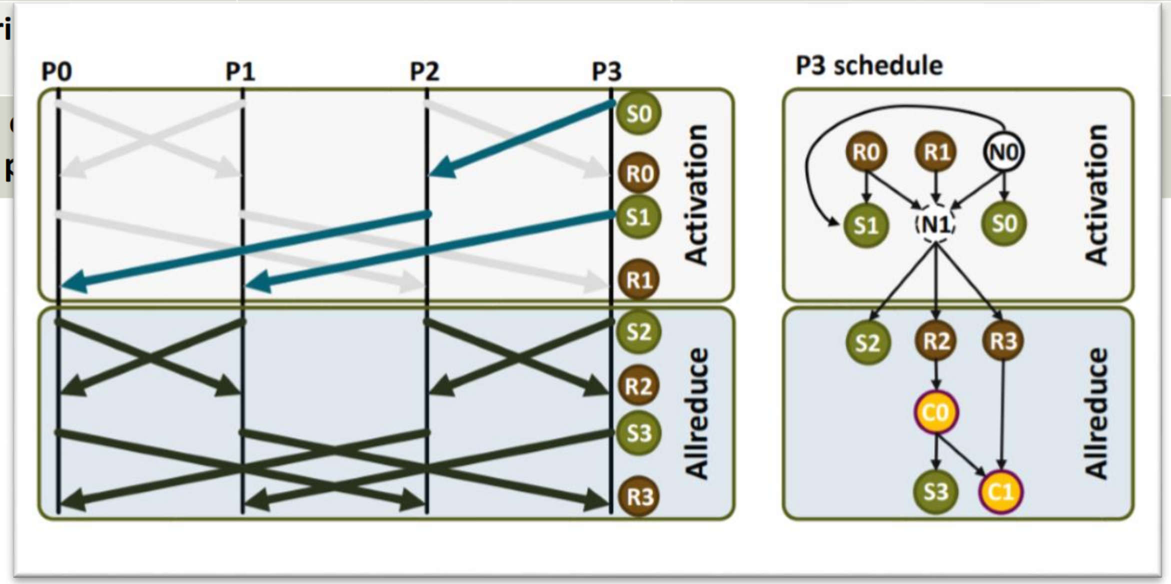
Model Averaging (e.g., elastic)

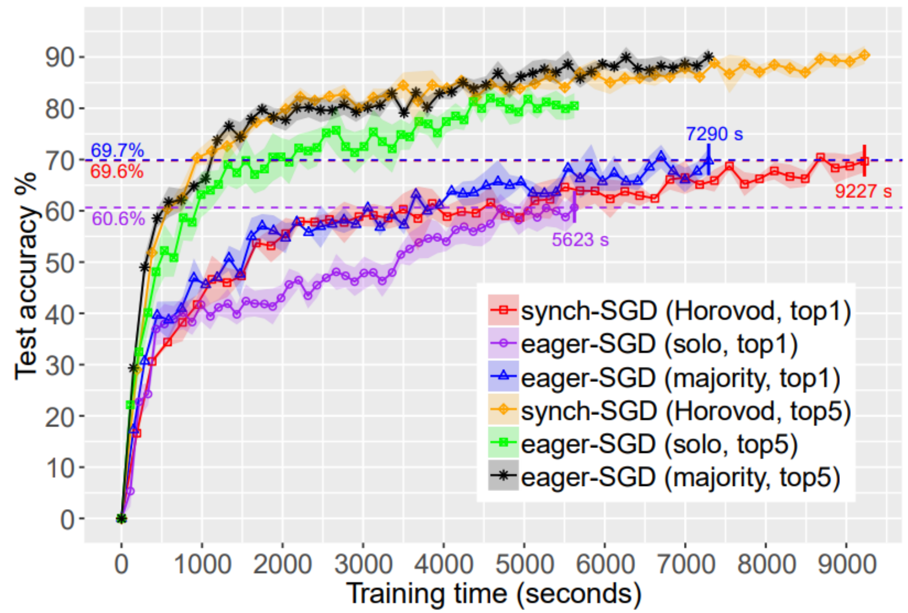
Ensemble Learning

Consistent → **Inconsistent**

Parameter (and Model) consistency - decentralized

	Solo allreduce	Majority allreduce
Initiator	The fastest process	A randomly specified process
Attri		
The the p		





Top-1 test accuracy and runtime for LSTM on UCF101 using 8 GPUs.

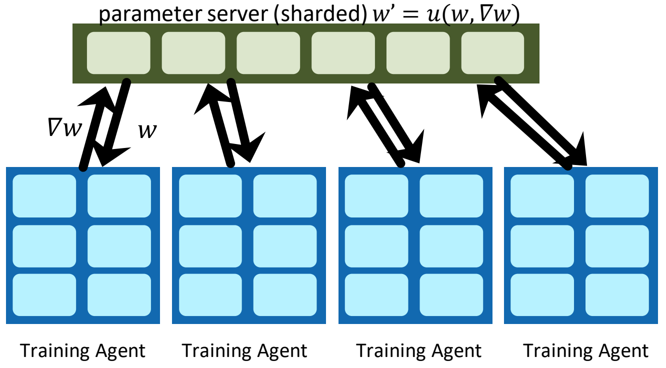
Synchronous SGD | Stale-Synchronous SGD | Asynchronous SGD (HOGWILD!) | Model Averaging (e.g., elastic) | Ensemble Learning

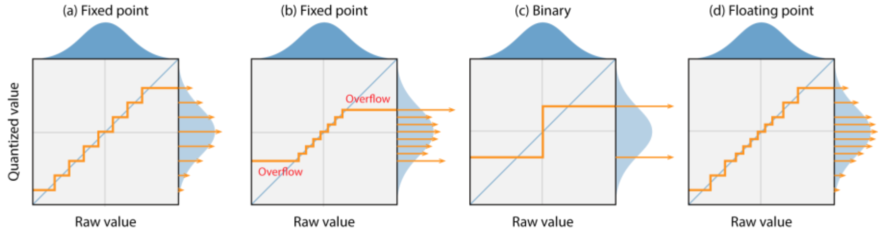
Consistent → **Inconsistent**

Shigang Li et al., “Taming unbalanced training workloads in deep learning with partial collective operations”, PPoPP 2020

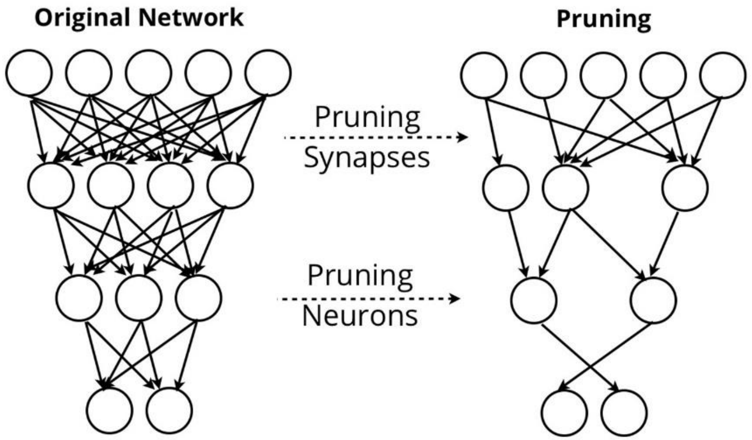
Communication optimizations

- **Different options how to optimize updates**
  - Send ∇w, receive w
  - Send FC factors (o_{l-1}, o_l), compute ∇w on parameter server
    Broadcast factors to not receive full w
  - Use lossy compression when sending, accumulate error locally!
- **Quantization**
  - Quantize weight updates and potentially weights
  - Main trick is stochastic rounding [1] – expectation is more accurate
    Enables low precision (half, quarter) to become standard
  - TernGrad - ternary weights [2], 1-bit SGD [3], …
- **Sparsification**
  - Do not send small weight updates **or** only send top-k [4]
    Accumulate omitted gradients locally





source: ai.intel.com



[1] S. Gupta et al. Deep Learning with Limited Numerical Precision, ICML'15
[2] F. Li and B. Liu. Ternary Weight Networks, arXiv 2016
[3] F. Seide et al. 1-Bit Stochastic Gradient Descent and Application to Data-Parallel Distributed Training of Speech DNNs, In Interspeech 2014
[4] C. Renggli et al. SparCML: High-Performance Sparse Communication for Machine Learning, arXiv 2018

Summary

- **Scientific ML workloads are atypical and heterogeneous**
- **Scaling is not a luxury, but a requirement**
- **Requires efficient utilization of all resources – including the single node**
- **Training is a complex pipeline, dependent on software/hardware stacks**
  - but can be exploited for performance