Towards Efficient and Reliable Deep Learning - Research Insights

Hongxu (Danny) Yin

Senior Research Scientist, NVIDIA Research

CVPR’23 Tutorial on Full-stack GPU Based Acceleration of Neural Networks

Pervasive Usage of Deep Learning Models

Efficient & Reliable Deep Learning

Understanding Networks Better!

(photos from web)

Evolution of Deep Learning (e.g., Computer Vision)

Transformers

CNNs

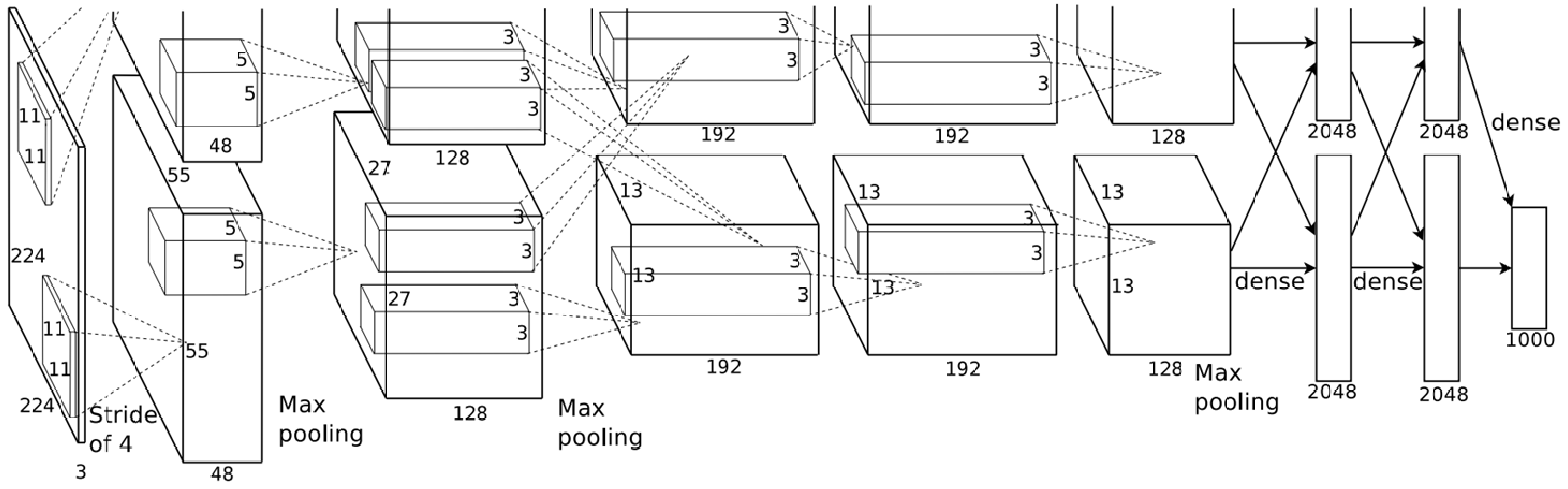


Photo from AlexNet paper (NeurIPS’12)

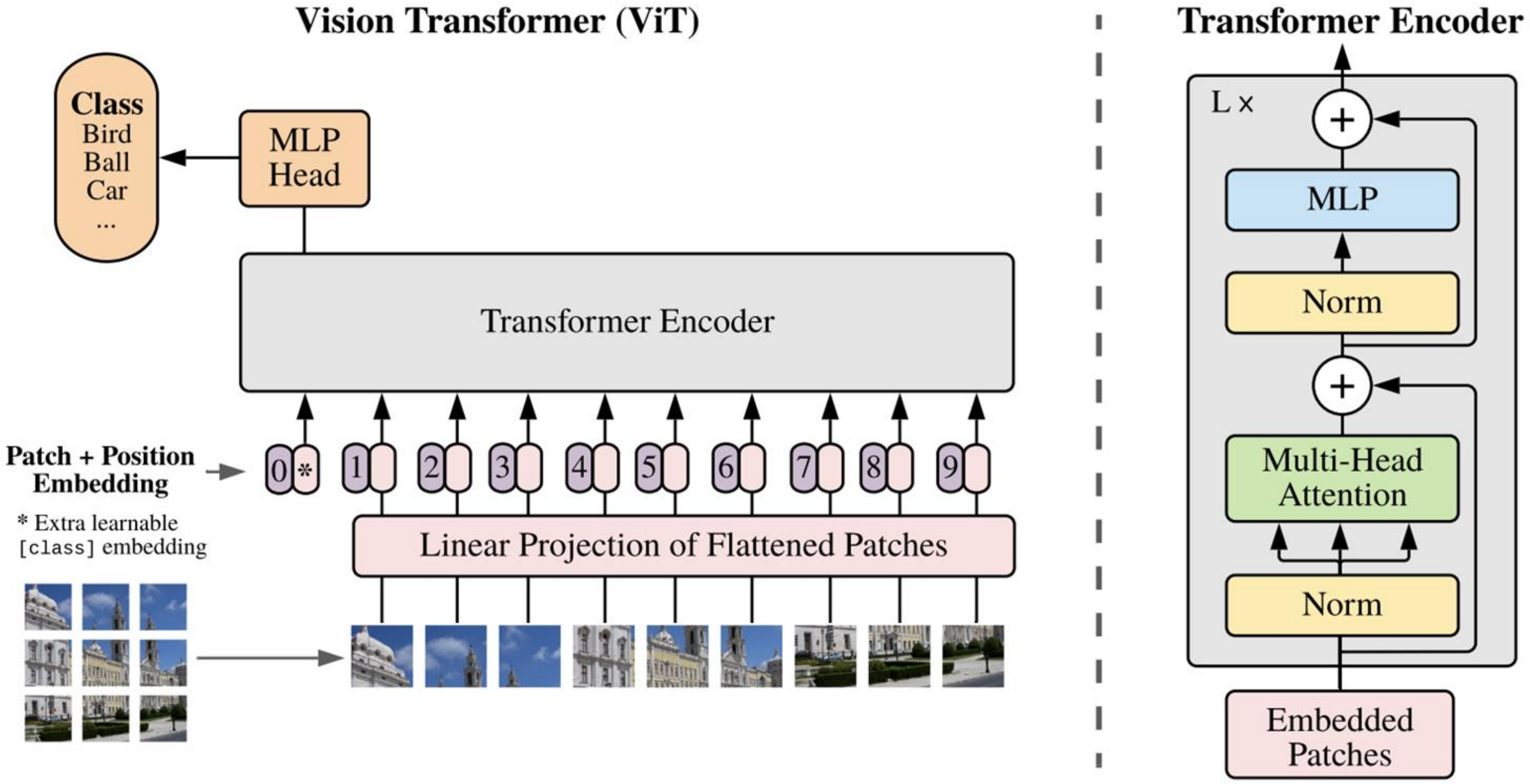


Photo from Google Research (ICLR’21)

Understanding DNNs: From CNNs into Transformers

Network Efficiency

CNNs

NAS, Pruning, Dynamic Inference, Quant, etc.
(CVPR'19, ICLRW'29, CVPR'22, ECCV'22, etc.)

ViT/LLM

A-ViT'22 (CVPR'22 Oral)

NViT (CVPR'23)

SmoothQuant (ICML'23)

Data Efficiency & Security

CNNs

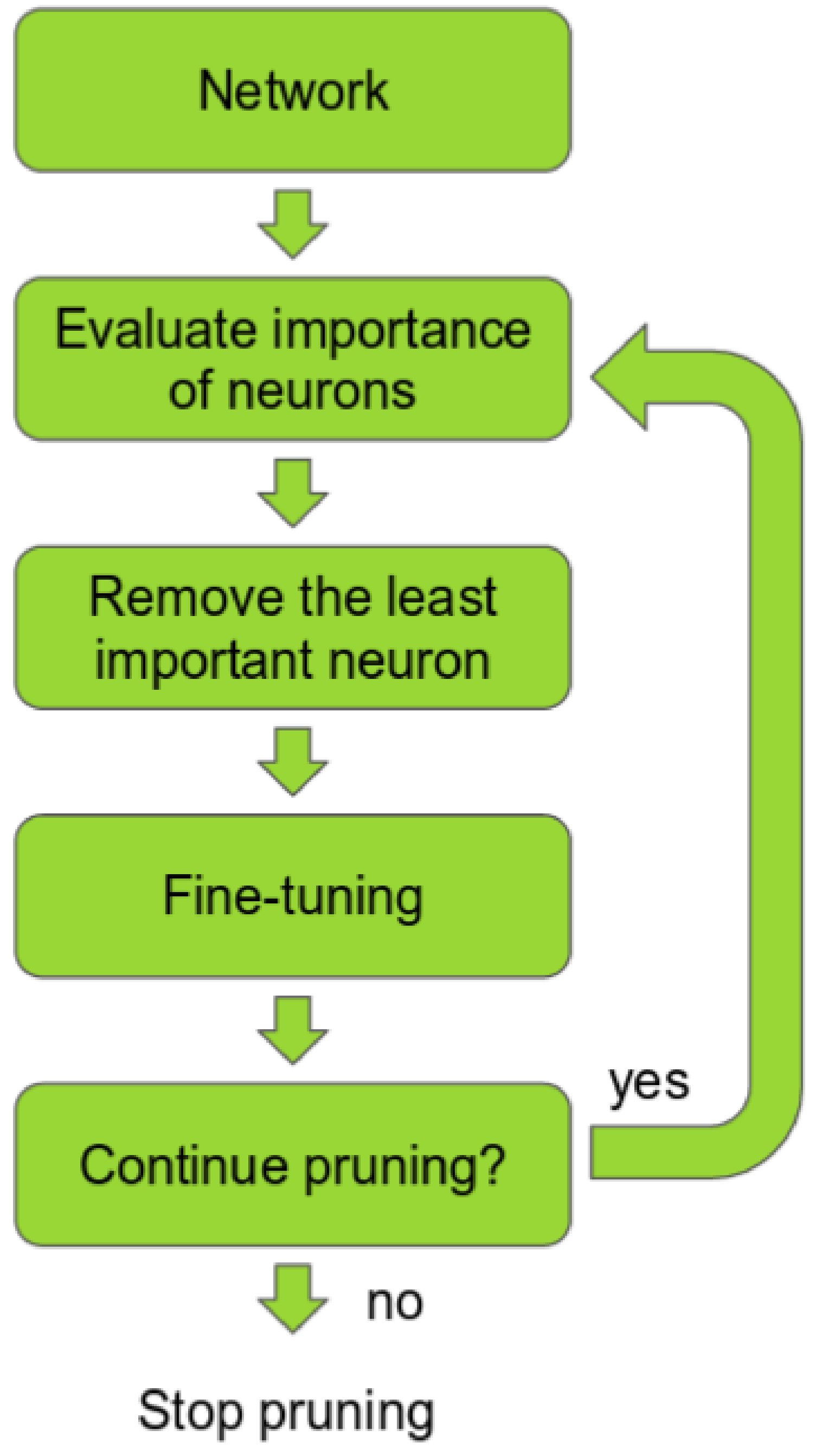
DeepInversion (CVPR'20 Oral)

GradInversion (CVPR'21)

ViT

GradViT (CVPR'22)

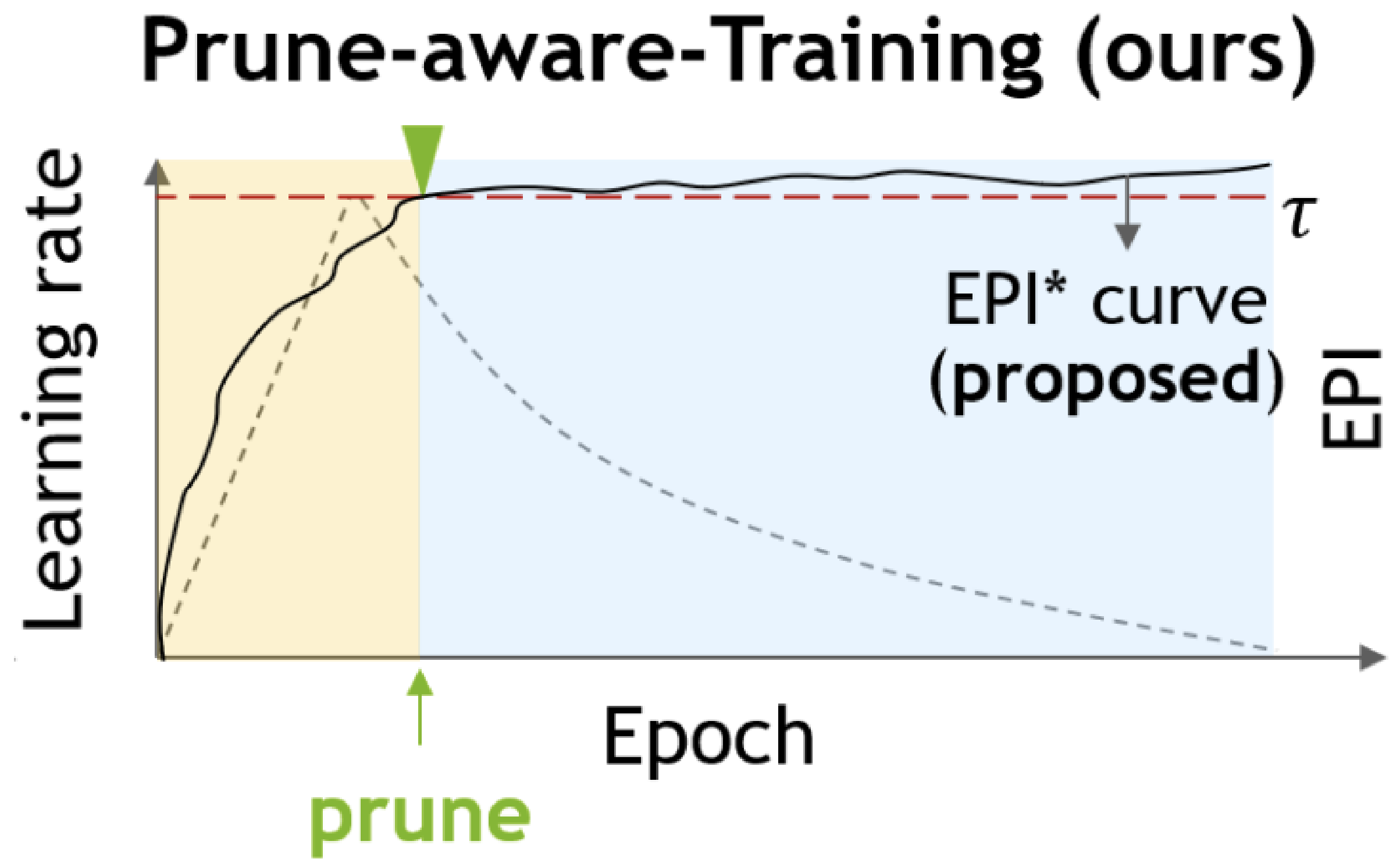
Making CNNs Efficient on Hardware



Network
Evaluate importance of neurons
Remove the least important neuron
Fine-tuning
Continue pruning?
yes
no
Stop pruning

a) Post-training

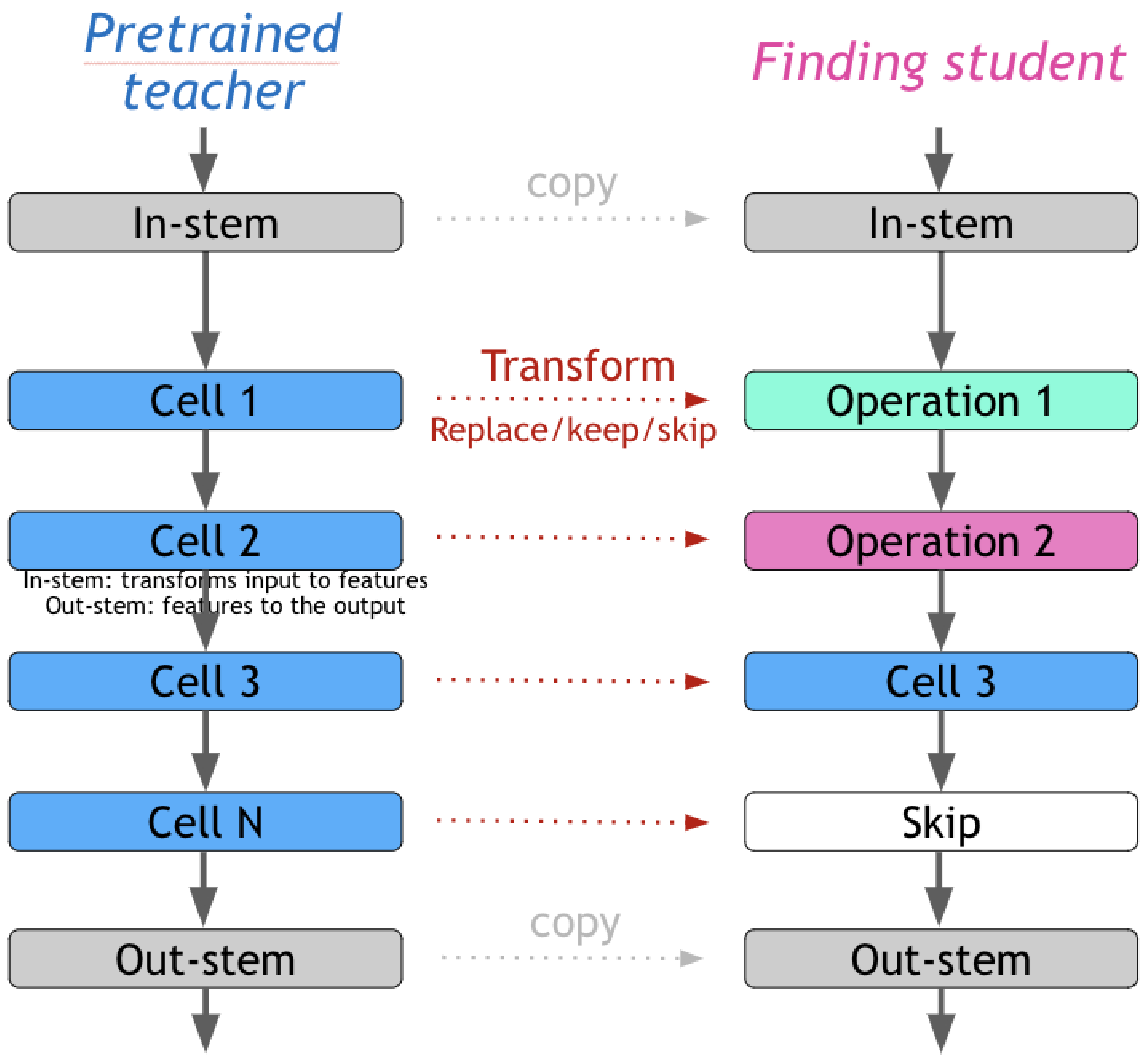
(CVPR’19)



b) During-training

(CVPR’22)

Filter pruning



In-stem: transforms input to features
Out-stem: features to the output

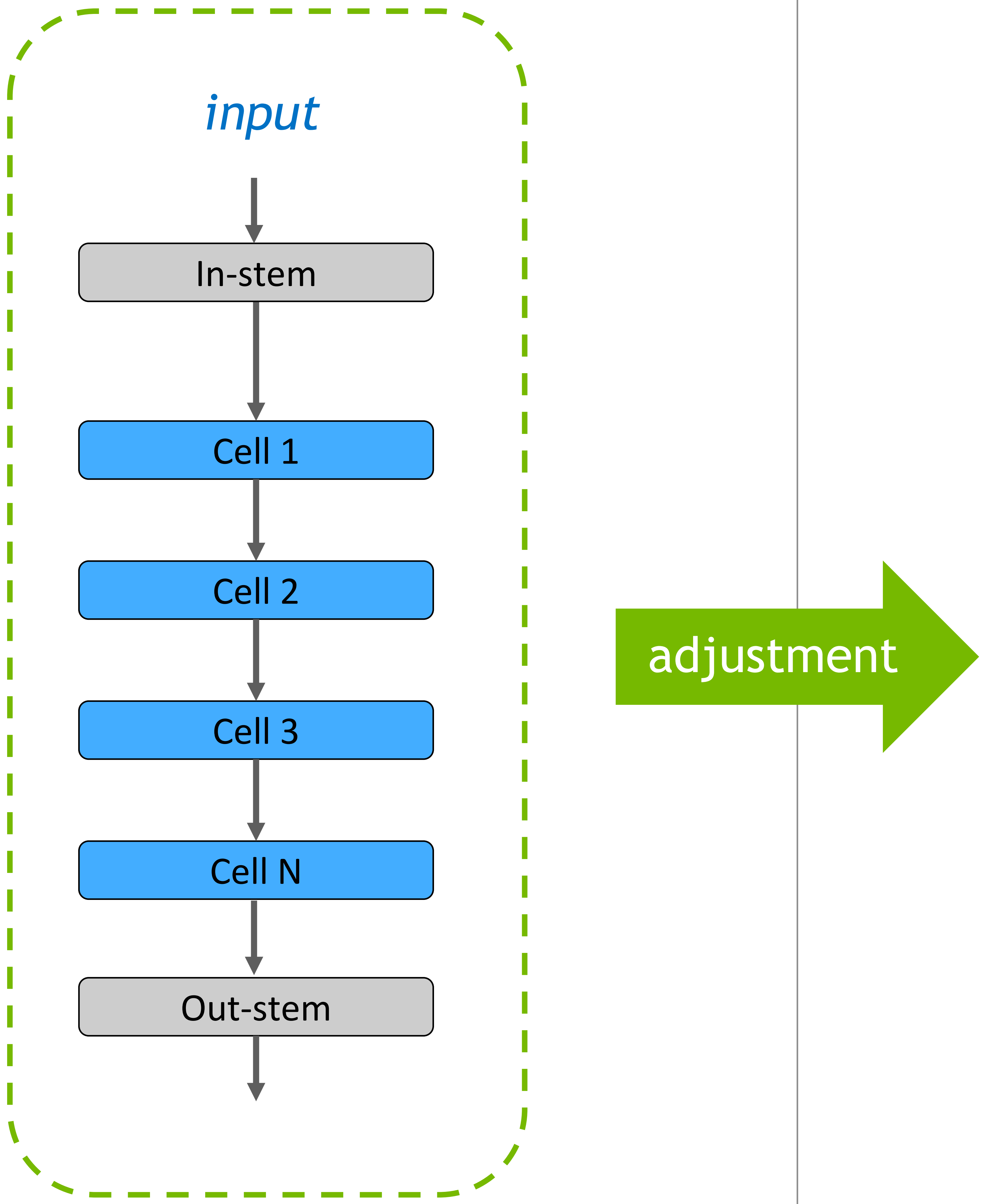
c) LANA – **L**atency-**a**ware **N**etwork **A**daptation

(ECCV’22)

Network adaptation

a) Molchanov, Mallya, Tyree, Irui, & Kautz, *Importance estimation for neural network pruning*, CVPR’19
b) Shen, Molchanov, Yin, Jose, *When to prune?*, CVPR’22
c) Molchanov*, Hall*, Yin, Kautz, Fusi, Vahdat, *LANA: Latency-aware network adaptation*, ECCV’22

LANA - Latency-aware Network Acceleration



Training

Larger - higher accuracy

GPU type 1, latency < a ms

GPU type 2, latency < b ms

CPU type 1, latency < c ms

CPU type 2, latency < d ms

Accelerator type 1, latency < f ms

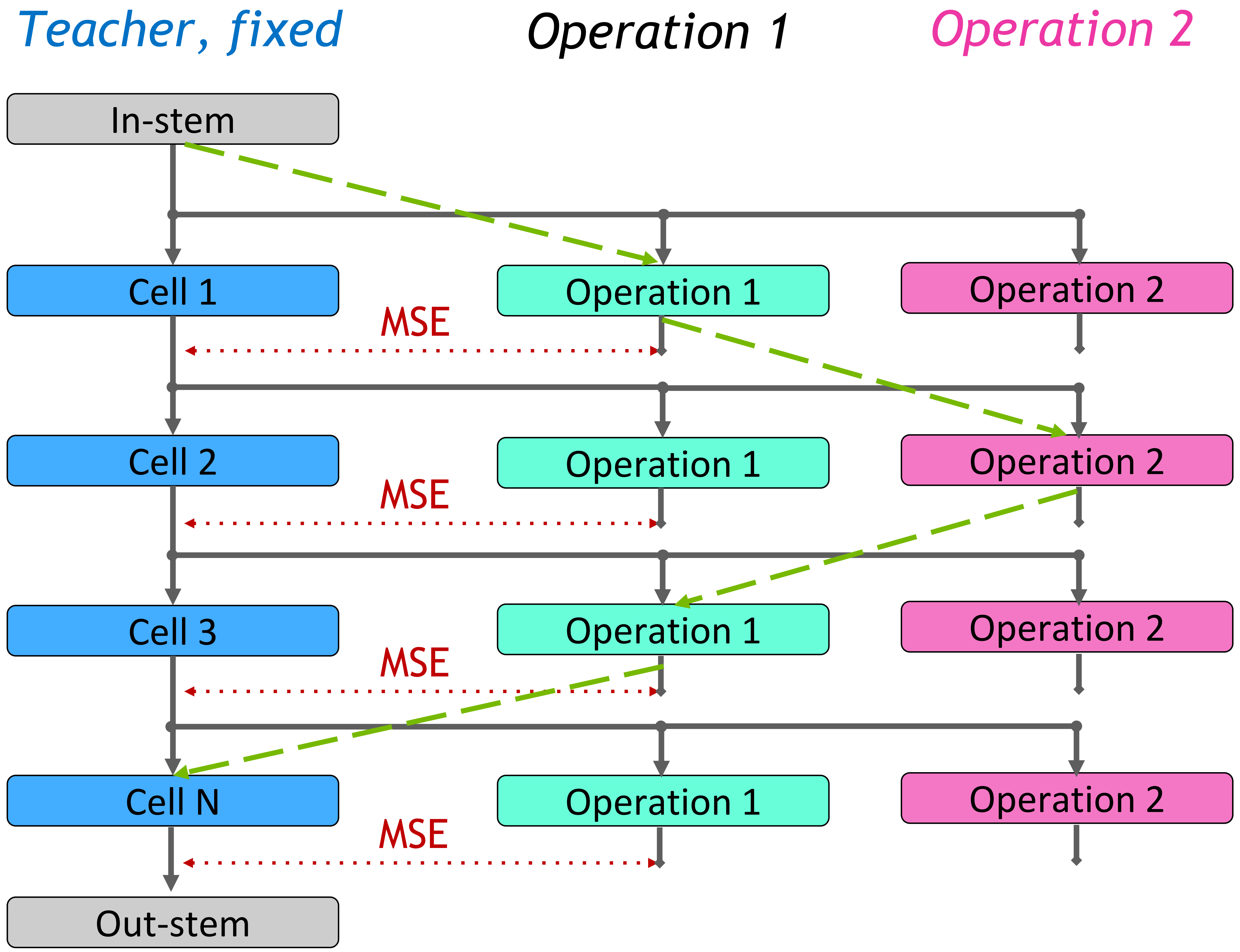
Inference

Varying platform, budgets

- Pruning, NAS, quantization
  - Original/similar arch.
  - Slow and computation intensive
- Our hypothesis:
  - Train-One-Large-Swap-Faster

LANA - Latency-aware Network Acceleration

Train One Large, Swap Faster



Training one large model - use as teacher
(once, higher accuracy)

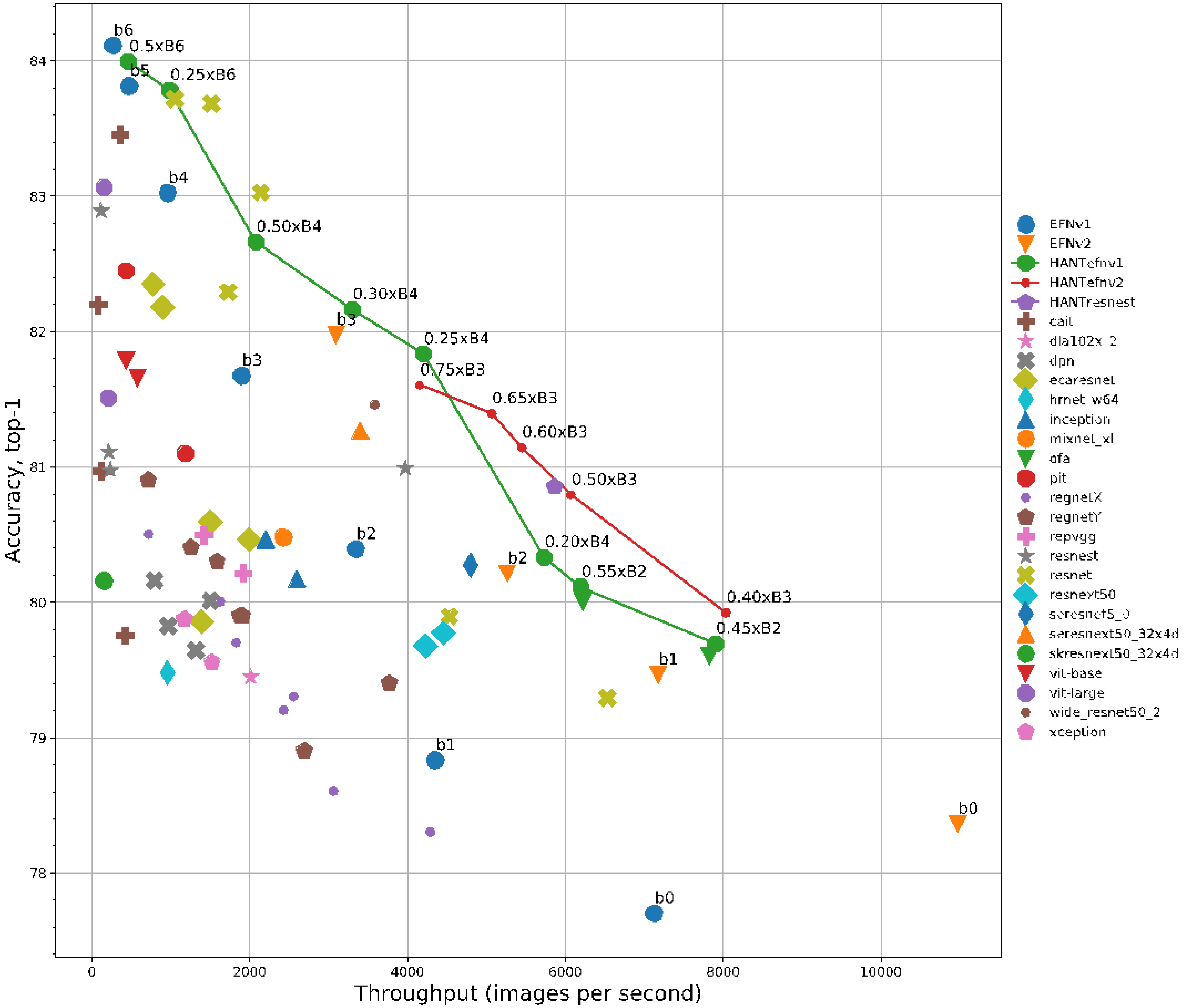
Preparing ops via distillation
(parallelable, one epoch)

Combinatory problem
(solvable in CPU seconds)

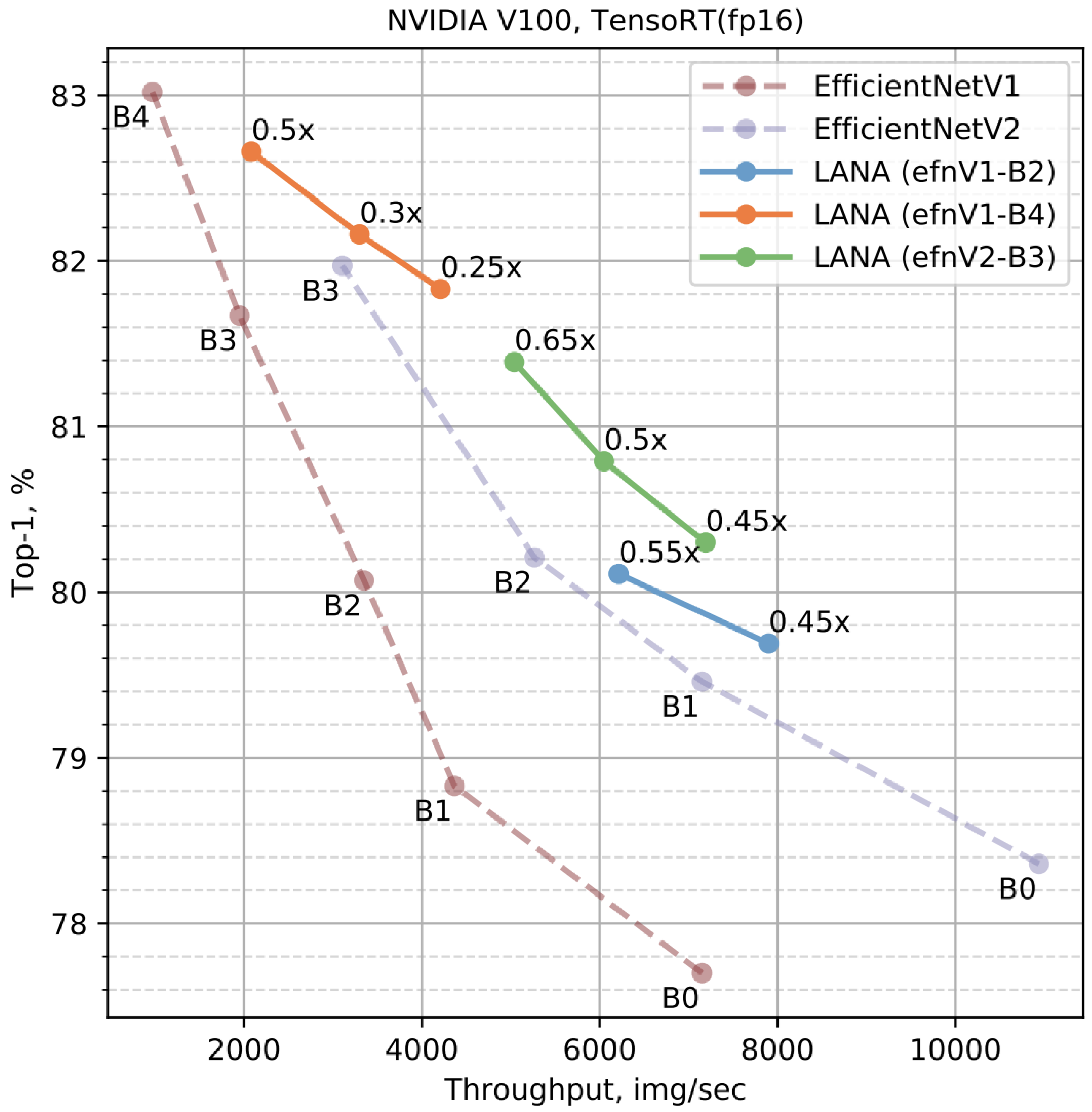
Quick finetuning
(per hardware-latency)

Molchanov, Hall, Yin, Kautz, Fusi, Vahdat, *LANA: Latency-aware network adaptation*, ECCV'22

ImageNet Results - Pareto Front



Adapting EfficientNets cover almost all CNNs
(30+ SOTAs from TIMM)



Adapting larger better than smaller from scratch

How about Vision Transformers (ViTs)

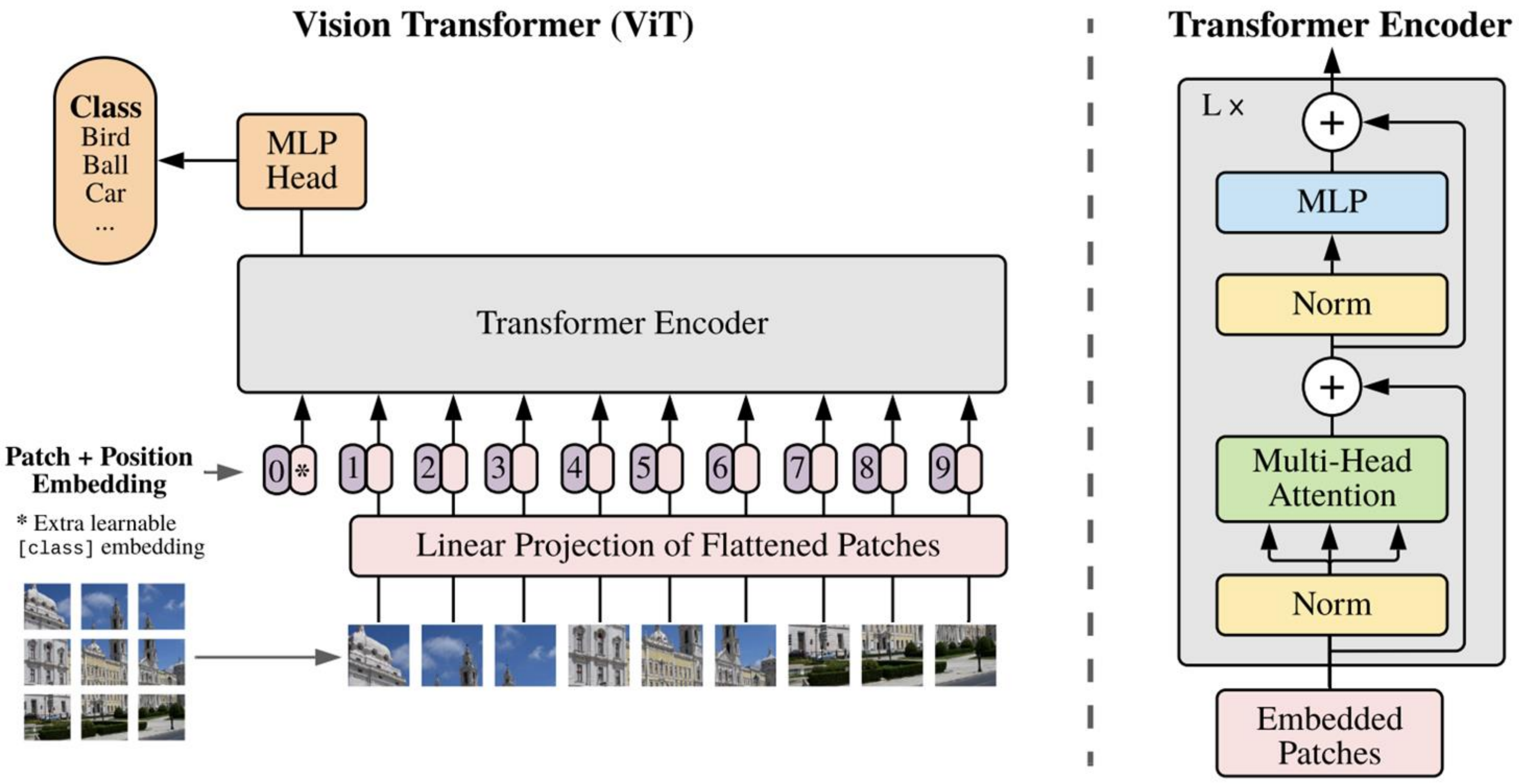
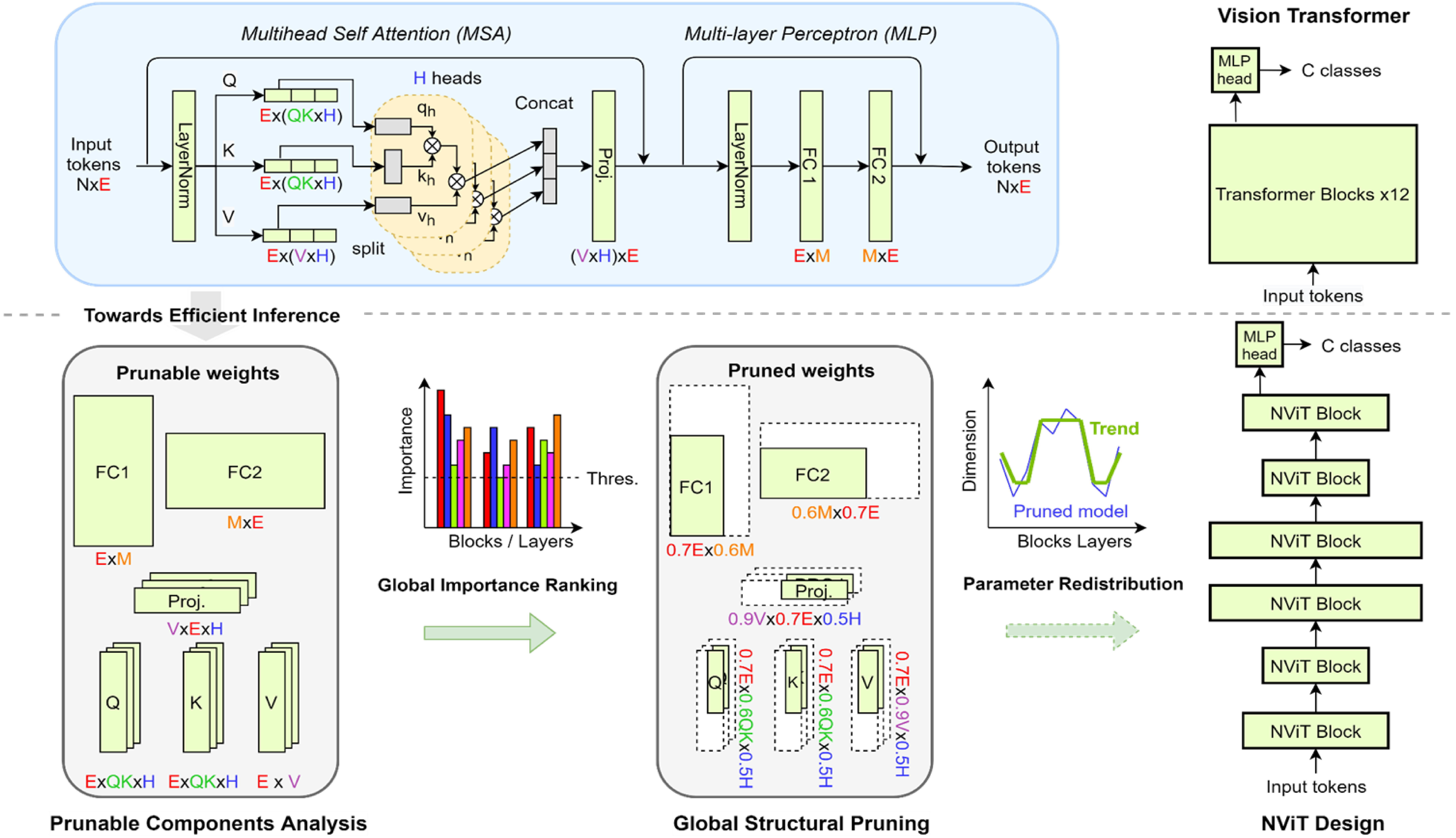


Photo from Google Research (ICLR’21)

- Pros
  - Stronger representation ability
  - Achieving higher accuracy
  - Large data
  - Unified structure
- Cons
  - Lacks inductive bias
  - Data hungry
  - More parameters and lower throughput
- This talk: Make ViTs Fast
  - Compression (NViT, CVPR’23)
  - Adaptive Inference (A-ViT, CVPR’22 oral)
  - Quantization (SmoothQuant, ICML’23)

NViT - Pruning & Parameter Redistribution

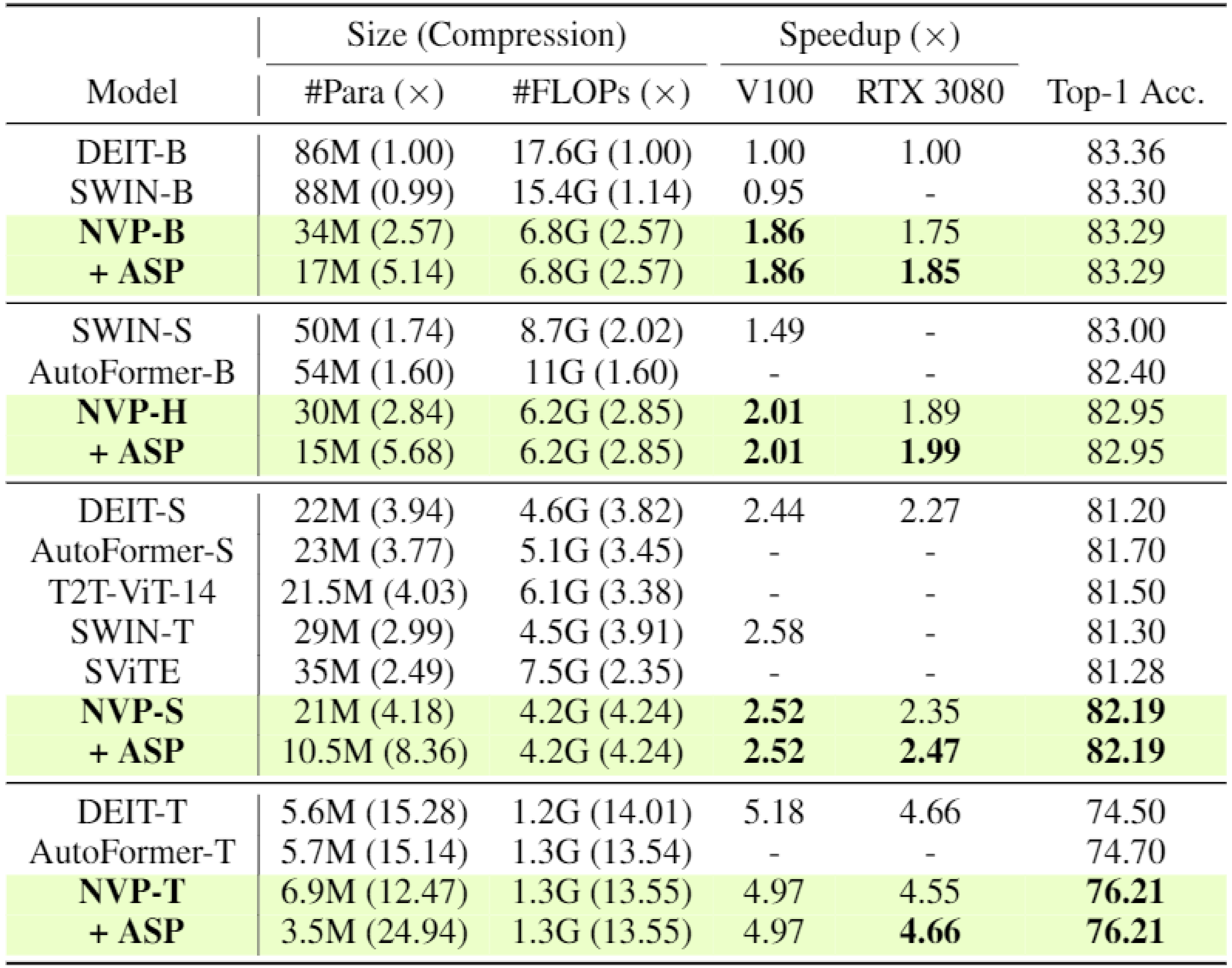


- Global, Structural pruning of all paremeter across all ViT layers, in latency-aware manner

Yang, Yin, Molchanov, Li, & Kautz, *NViT: Vision Transformer Compression and Parameter Redistribution*, CVPR'23

Key Pruning Results

- **Detailed performance (ImageNet1K DEIT)**
  - **Lossless ref.**: 1.86x speedup with -0.07% acc.
  - **2x ref.**: 2x speedup with -0.4% acc over DEIT-B, 1.4x faster than SWIN-S
  - **NVP-S/T**: +1% / +1.7% acc over DEIT-S/T
- lossless speedup with Ampere-sparsity



Model	Size (Compression)		Speedup (×)		
	#Para (×)	#FLOPs (×)	V100	RTX 3080	Top-1 Acc.
DEIT-B	86M (1.00)	17.6G (1.00)	1.00	1.00	83.36
SWIN-B	88M (0.99)	15.4G (1.14)	0.95	-	83.30
NVP-B	34M (2.57)	6.8G (2.57)	**1.86**	1.75	83.29
+ ASP	17M (5.14)	6.8G (2.57)	**1.86**	**1.85**	83.29
SWIN-S	50M (1.74)	8.7G (2.02)	1.49	-	83.00
AutoFormer-B	54M (1.60)	11G (1.60)	-	-	82.40
NVP-H	30M (2.84)	6.2G (2.85)	**2.01**	1.89	82.95
+ ASP	15M (5.68)	6.2G (2.85)	**2.01**	**1.99**	82.95
DEIT-S	22M (3.94)	4.6G (3.82)	2.44	2.27	81.20
AutoFormer-S	23M (3.77)	5.1G (3.45)	-	-	81.70
T2T-ViT-14	21.5M (4.03)	6.1G (3.38)	-	-	81.50
SWIN-T	29M (2.99)	4.5G (3.91)	2.58	-	81.30
SViTE	35M (2.49)	7.5G (2.35)	-	-	81.28
NVP-S	21M (4.18)	4.2G (4.24)	**2.52**	2.35	**82.19**
+ ASP	10.5M (8.36)	4.2G (4.24)	**2.52**	**2.47**	**82.19**
DEIT-T	5.6M (15.28)	1.2G (14.01)	5.18	4.66	74.50
AutoFormer-T	5.7M (15.14)	1.3G (13.54)	-	-	74.70
NVP-T	6.9M (12.47)	1.3G (13.55)	4.97	4.55	**76.21**
+ ASP	3.5M (24.94)	1.3G (13.55)	4.97	**4.66**	**76.21**

NViT - Pruning-Inspired Parameter Redistribution

Pruned models

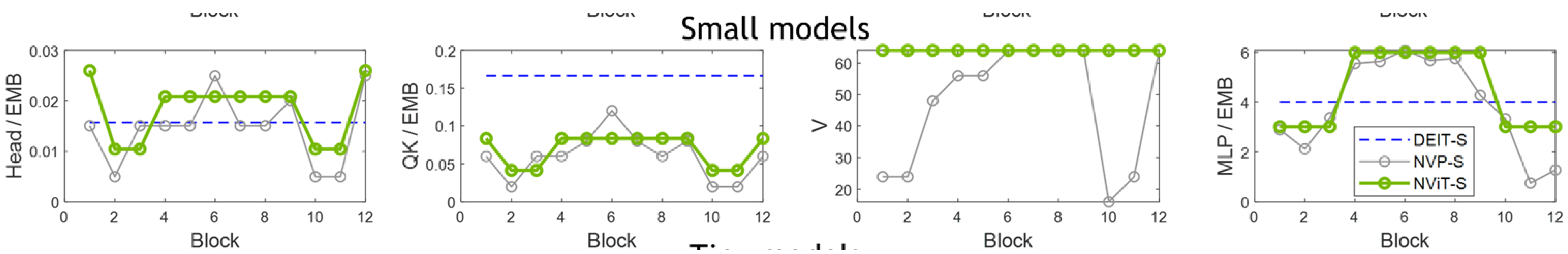
(inspires)

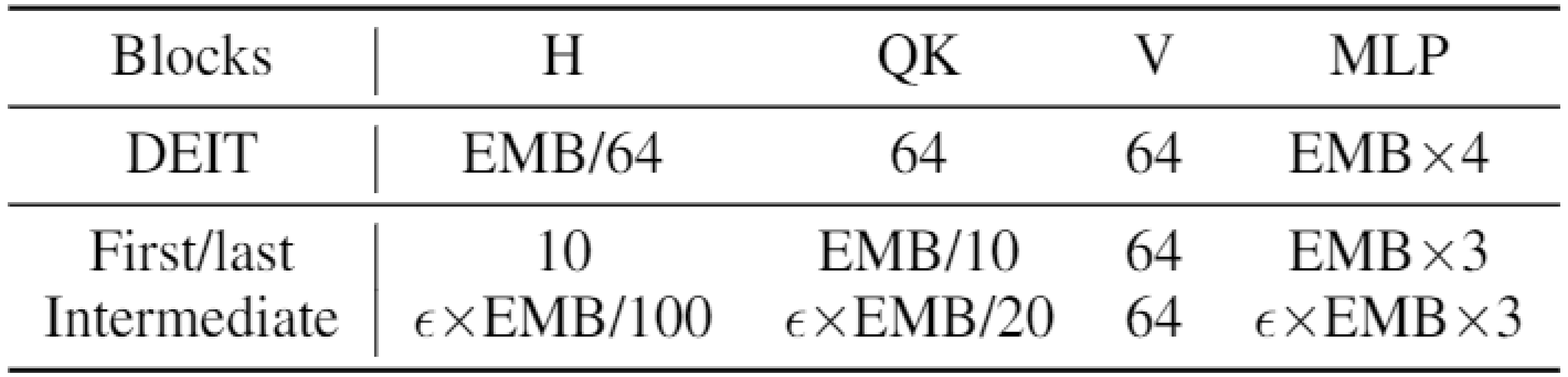
Embedding-based distribution rule

(yields)

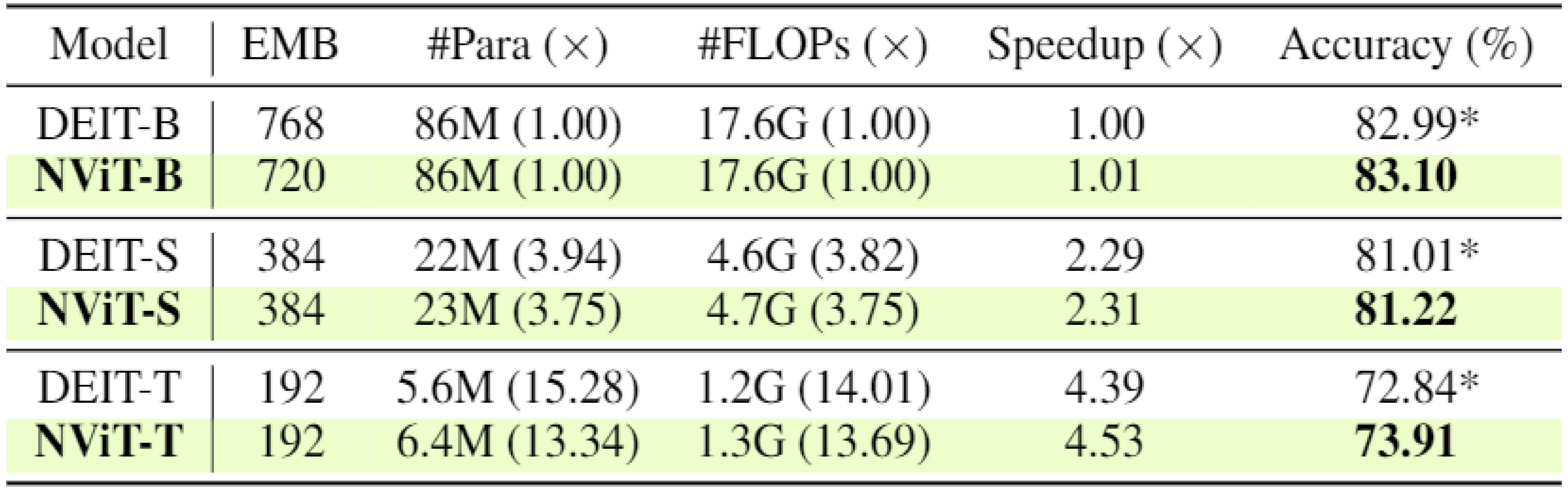
Consistent Improvements over Hand Designed (ImageNet1K)

(scales to downstream)





Blocks	H	QK	V	MLP
DEIT	EMB/64	64	64	EMB×4
First/last	10	EMB/10	64	EMB×3
Intermediate	$\epsilon\times$EMB/100	$\epsilon\times$EMB/20	64	$\epsilon\times$EMB×3



Model	EMB	#Para (×)	#FLOPs (×)	Speedup (×)	Accuracy (%)
DEIT-B	768	86M (1.00)	17.6G (1.00)	1.00	82.99*
NViT-B	720	86M (1.00)	17.6G (1.00)	1.01	**83.10**
DEIT-S	384	22M (3.94)	4.6G (3.82)	2.29	81.01*
NViT-S	384	23M (3.75)	4.7G (3.75)	2.31	**81.22**
DEIT-T	192	5.6M (15.28)	1.2G (14.01)	4.39	72.84*
NViT-T	192	6.4M (13.34)	1.3G (13.69)	4.53	**73.91**

Human - Adaptive Effort vs. Network - Fixed Effort

A-ViT – Adaptive Tokens for Efficient Vision Transformer

Tokenization
Tokens remain: 197
0.0
0.0
0.0
0.0
Embedding
0.0
Halting probability
Layer 1
Transformer Block
0.0
0.1
0.0
1.0
0.7
Token halting
Token removal & reorg.
Adaptive Halting
Tokens remain: 190
0.0
0.1
0.0
0.7
Layer 2
Transformer Block
Adaptive Halting
Tokens remain: 10
0.2
0.2
Layer K
Transformer Block
Adaptive Halting
Class token memory
Mean-field aggregation
Task head
Token Depths
ImageNet1K Examples for Adaptive Tokens

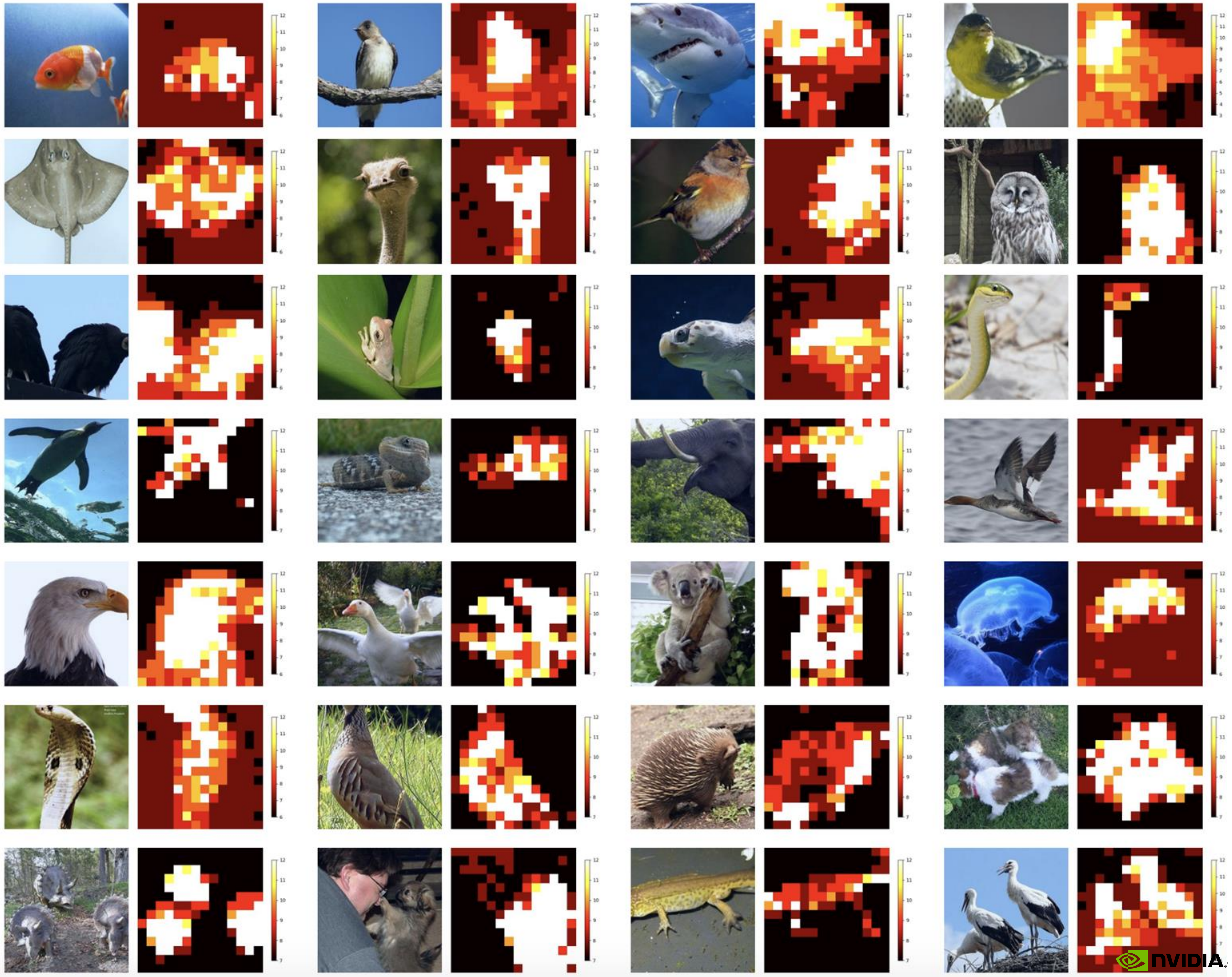
- Not all tokens are informative! Let the network decide which ones to halt, adaptively for varying inputs

Yin, Vahdat, Jose, Mallya, Jan, Pavlo, *A-ViT: Adaptive Inference for Efficient Vision Transformers*, **CVPR’22 oral**

ADAPTIVE TOKENS

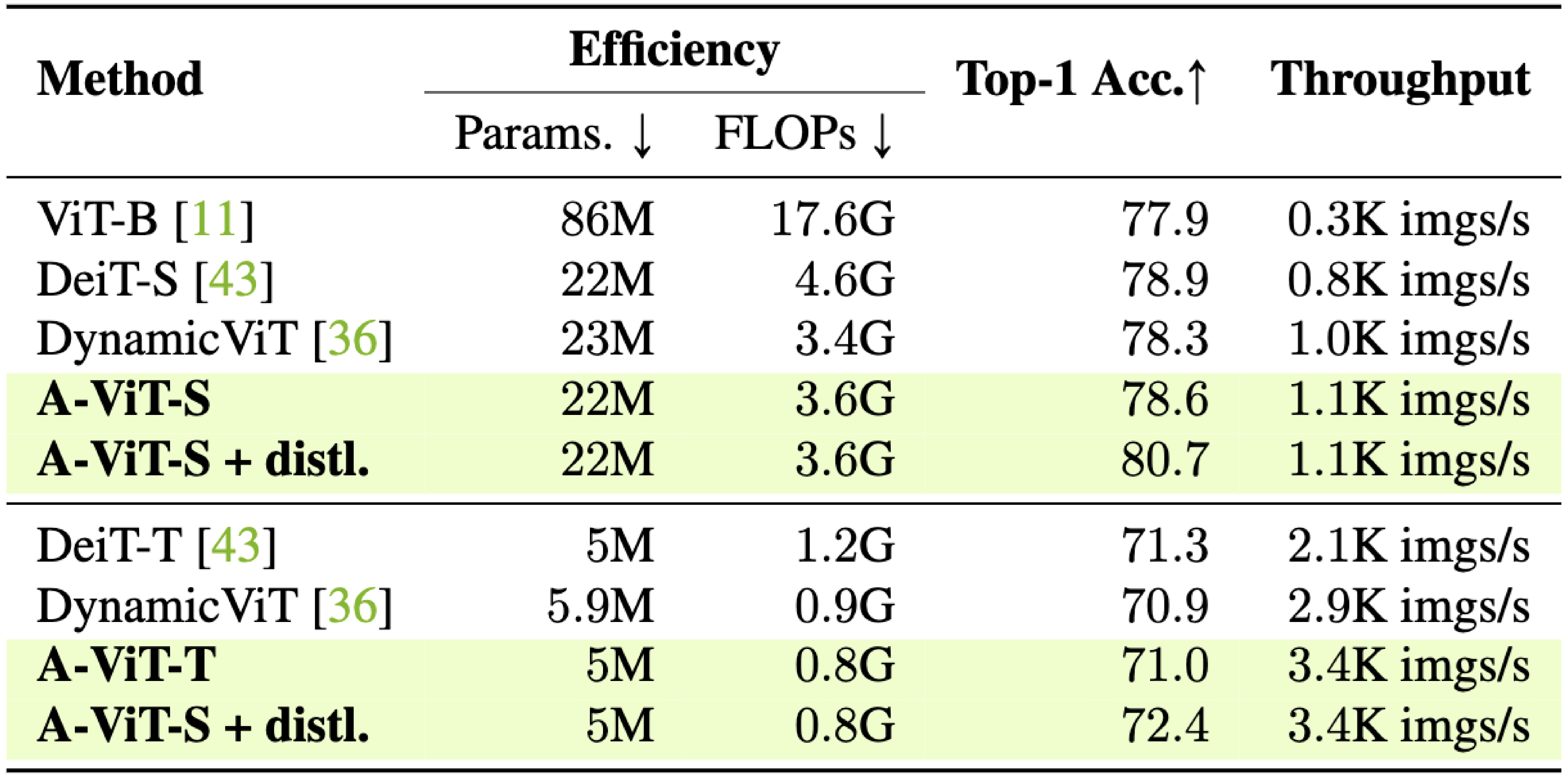
IMAGENET1K

Intuitive distribution of computation!



Direct Speed-up on Existing Platform

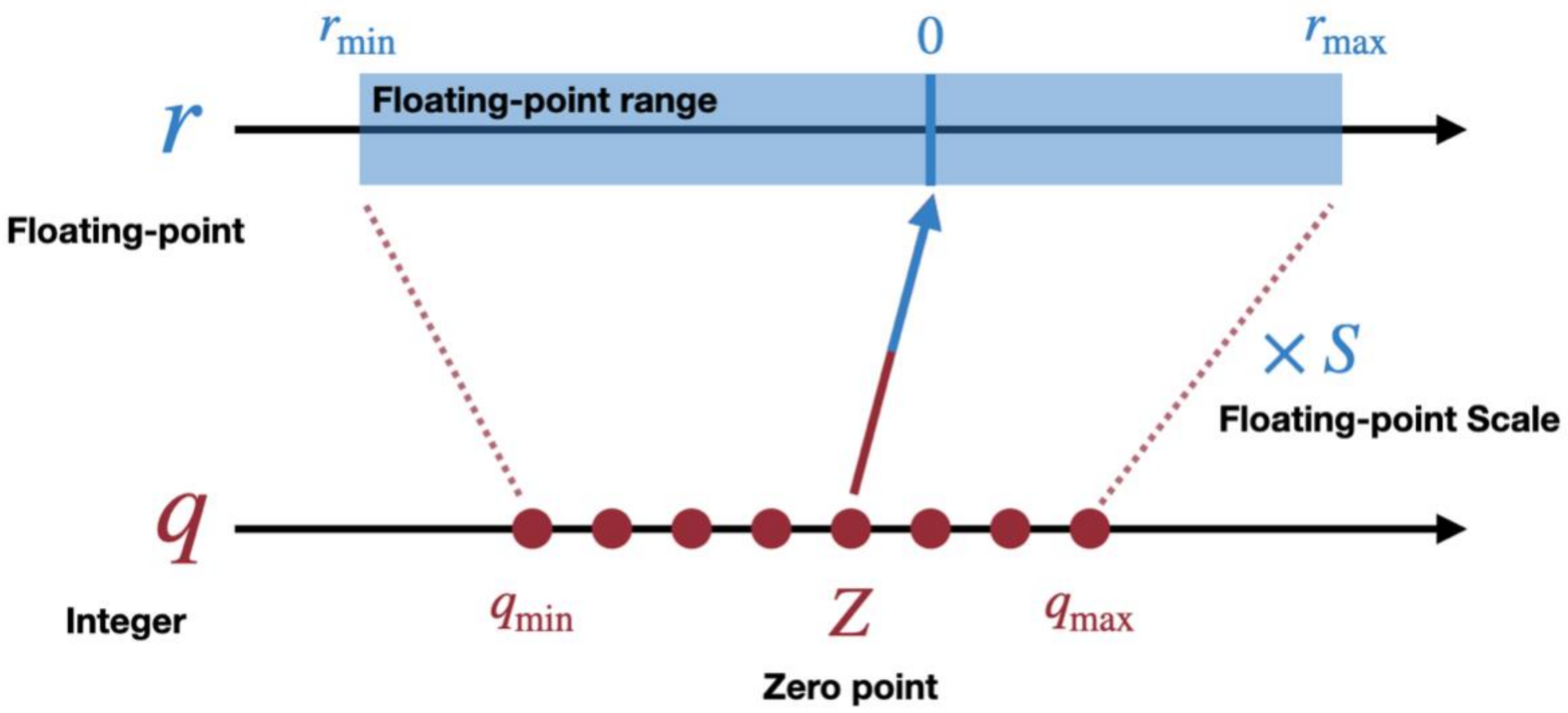
- DeiT family
  - 38%-62% throughput impr. with only 0.3% acc. drop
  - Off-the-shelf platform (GPU)
  - Direct speedup without changing DeiT cell



Method	Efficiency		Top-1 Acc.↑	Throughput
	Params. ↓	FLOPs ↓		
ViT-B [11]	86M	17.6G	77.9	0.3K imgs/s
DeiT-S [43]	22M	4.6G	78.9	0.8K imgs/s
DynamicViT [36]	23M	3.4G	78.3	1.0K imgs/s
A-ViT-S	22M	3.6G	78.6	1.1K imgs/s
A-ViT-S + distl.	22M	3.6G	80.7	1.1K imgs/s
DeiT-T [43]	5M	1.2G	71.3	2.1K imgs/s
DynamicViT [36]	5.9M	0.9G	70.9	2.9K imgs/s
A-ViT-T	5M	0.8G	71.0	3.4K imgs/s
A-ViT-S + distl.	5M	0.8G	72.4	3.4K imgs/s

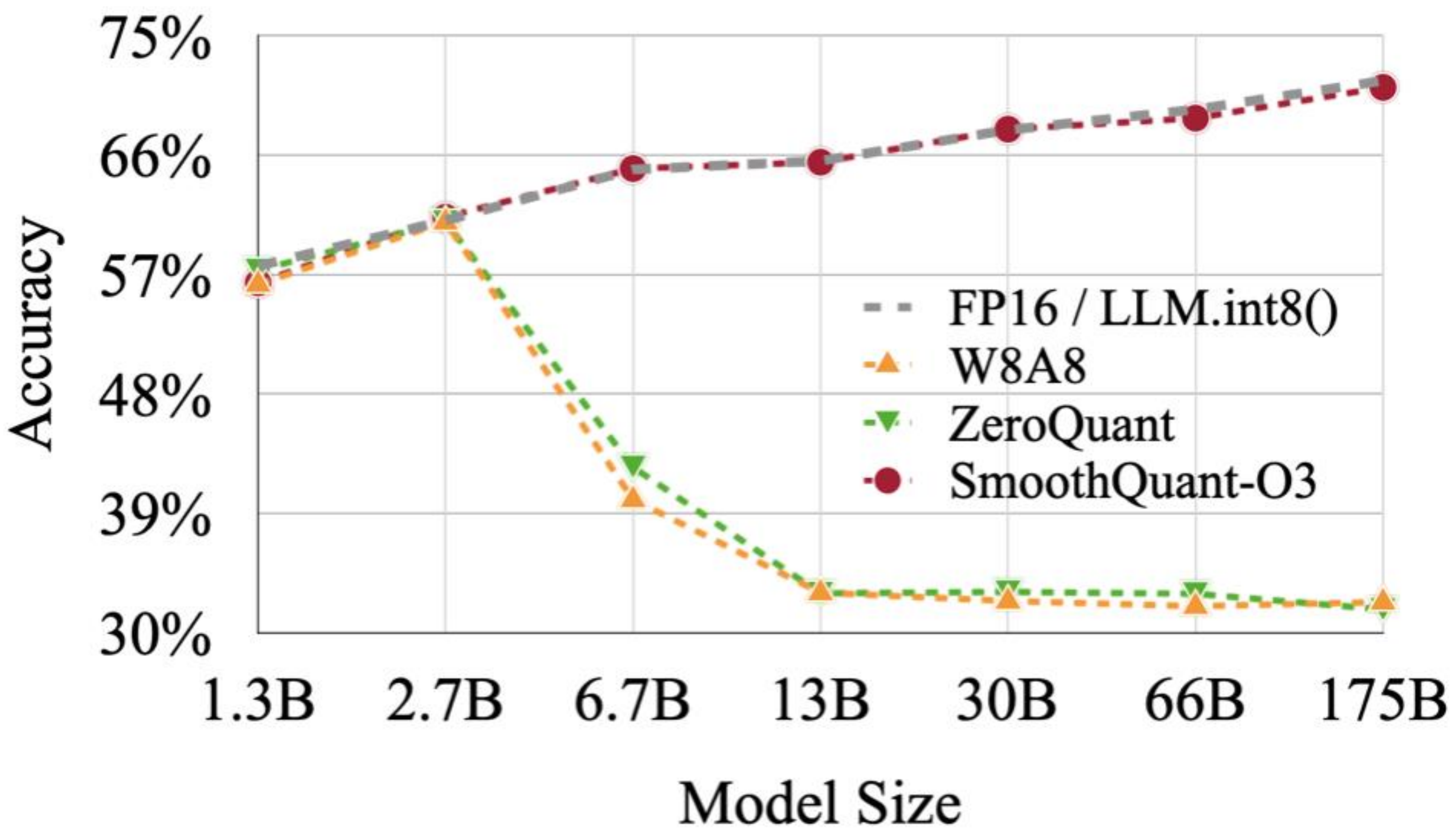
How about Lower Precision?

- LLMs are eerily large (*e.g.*, >100B params. range).
- Models scale up faster than hardware capacity.
    - Serving a 175B GPT-3 model at least requires:
        - FP16: 350GB memory ➡️ 5 x 80GB A100 GPUs
        - **INT8:** 175GB memory ➡️ **3 x 80GB A100 GPUs**



From CNN to Transformer: Shift in Pain Point

Activation outliers destroy quantized performance

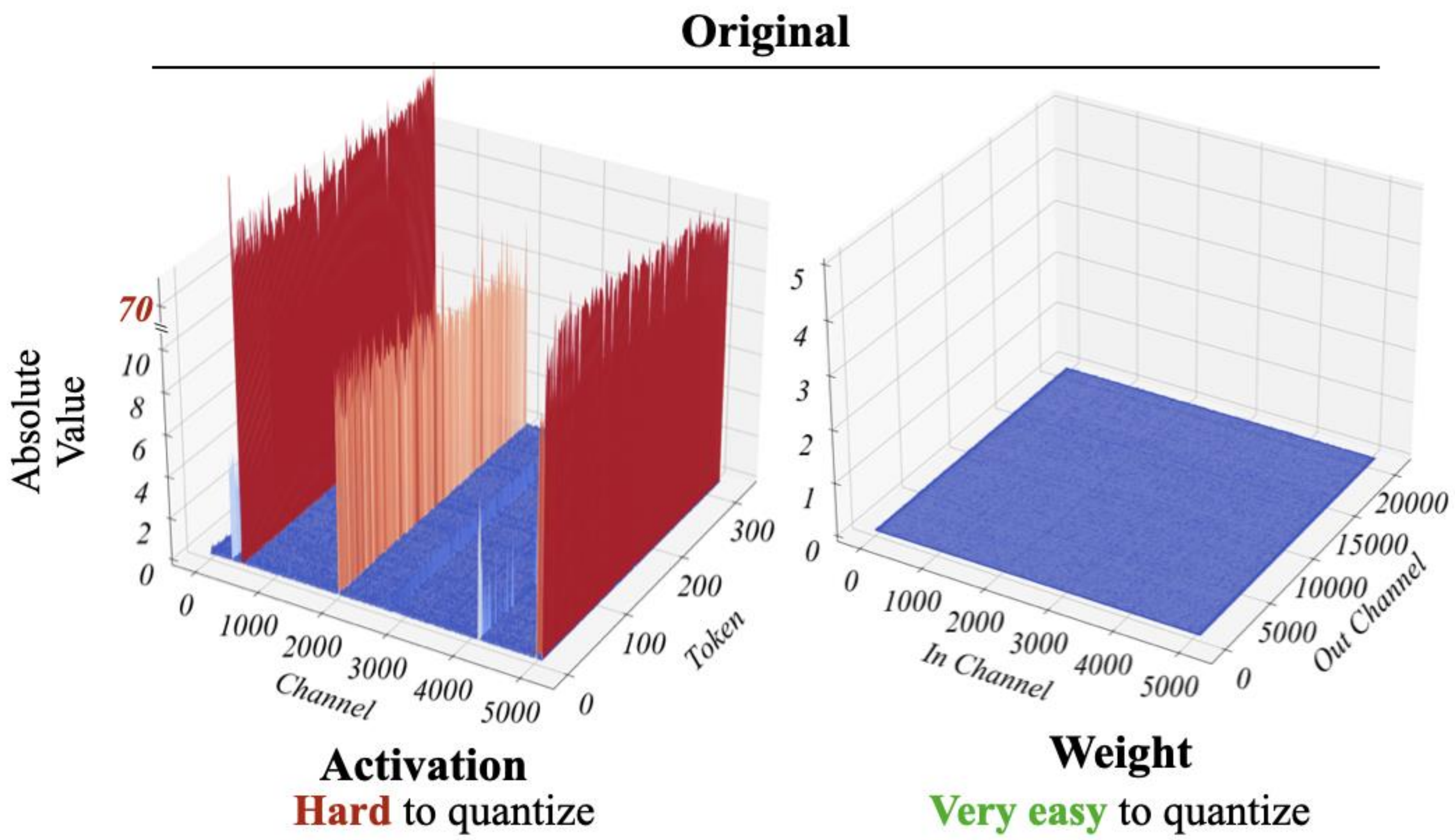


- W8A8 quantization has been an industrial standard for CNNs, but not LLM. Why?
- Systematic outliers emerge in **activations** when we scale up LLMs beyond 6.7B. Traditional CNN quantization methods will destroy the accuracy.

LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale (Dietters et al., 2022)

SmoothQuant

Smoothing Activation to Reduce Quantization Error



- Weights are easy to quantize, but activation is hard due to outliers
- Luckily, outliers persist in fixed channels

SmoothQuant

Smoothing Activation to Reduce Quantization Error

Absolute Value
70
10
8
6
4
2
0
Channel
0
1000
2000
3000
4000
5000
Token
100
200
300
In Channel
Out Channel
5000
10000
15000
20000
5
4
3
2
1

Original

Smoothed

Migrate the quantization difficulty

Activation
Hard to quantize

Weight
Very easy to quantize

Activation
Easy to quantize

Weight
Harder but still easy to quantize

- Weights are easy to quantize, but activation is hard due to outliers
- Luckily, outliers persist in fixed channels
- Migrate the quantization difficulty from activation to weights, so both are easy to quantize

Lin*****, Xiao*, et al., *SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models, ICML'23*

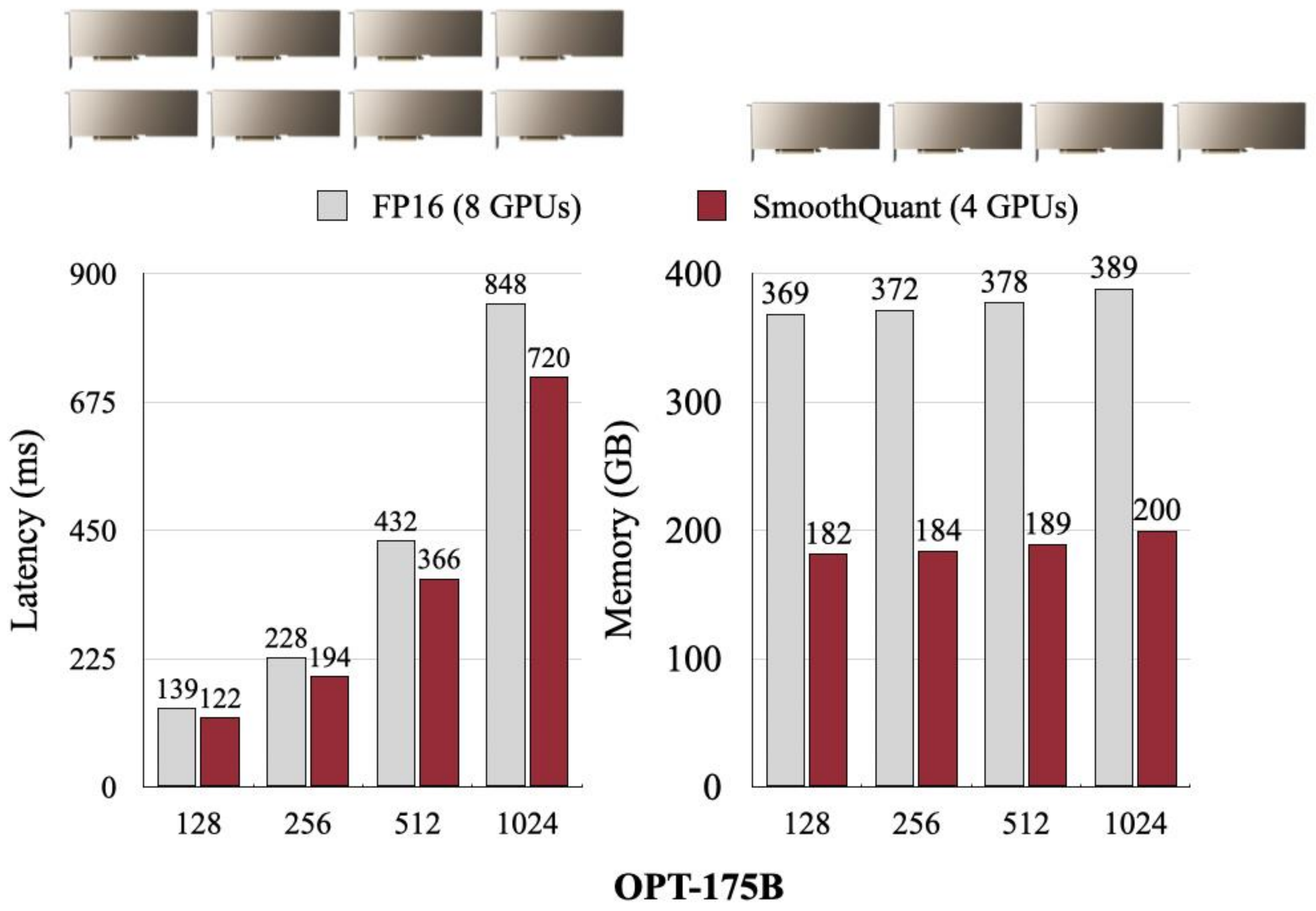
SmoothQuant (W8A8)

Smoothing Activation to Reduce Quantization Error

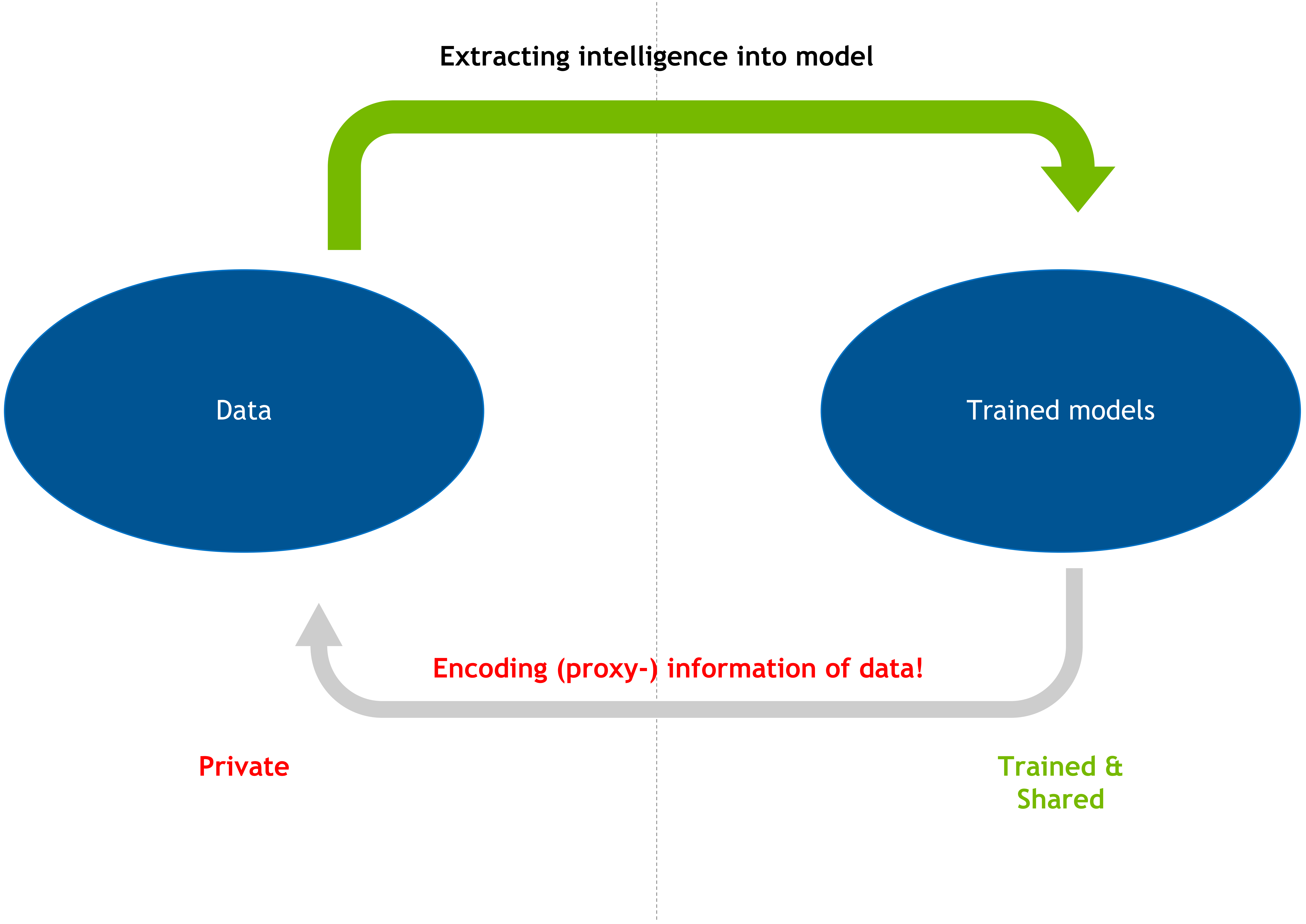
- SmoothQuant well maintains the accuracy without fine-tuning.
- SmoothQuant can both accelerate inference and halve the memory footprint.

LAMBADA Accuracy

	OPT-175B	BLOOM-176B	GLM-130B
FP16	71.6%	68.2%	73.8%
SmoothQuant	71.2%	68.3%	73.7%



Data Access Dilemma



Extracting intelligence into model

Data

Trained models

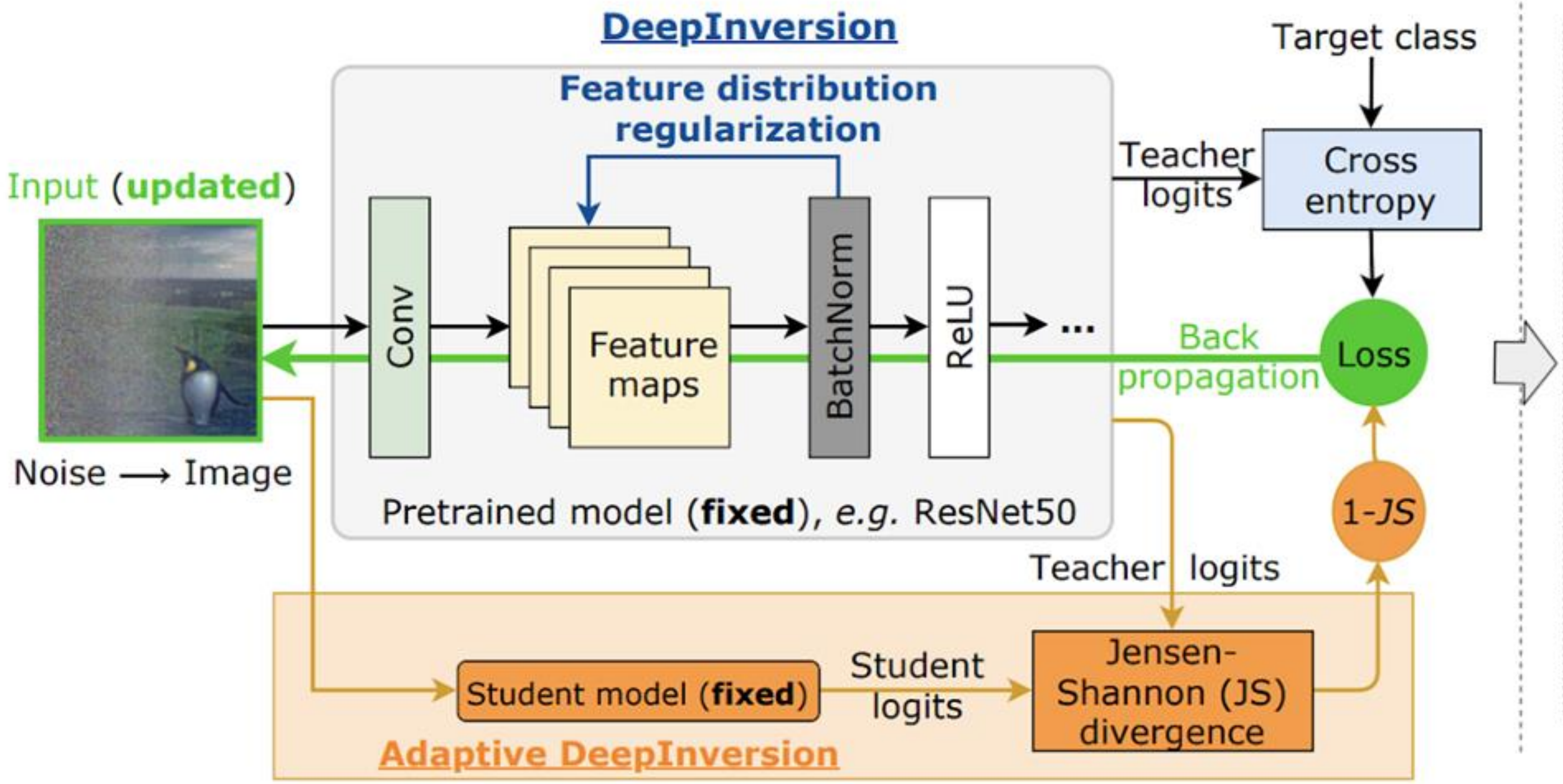
Encoding (proxy-) information of data!

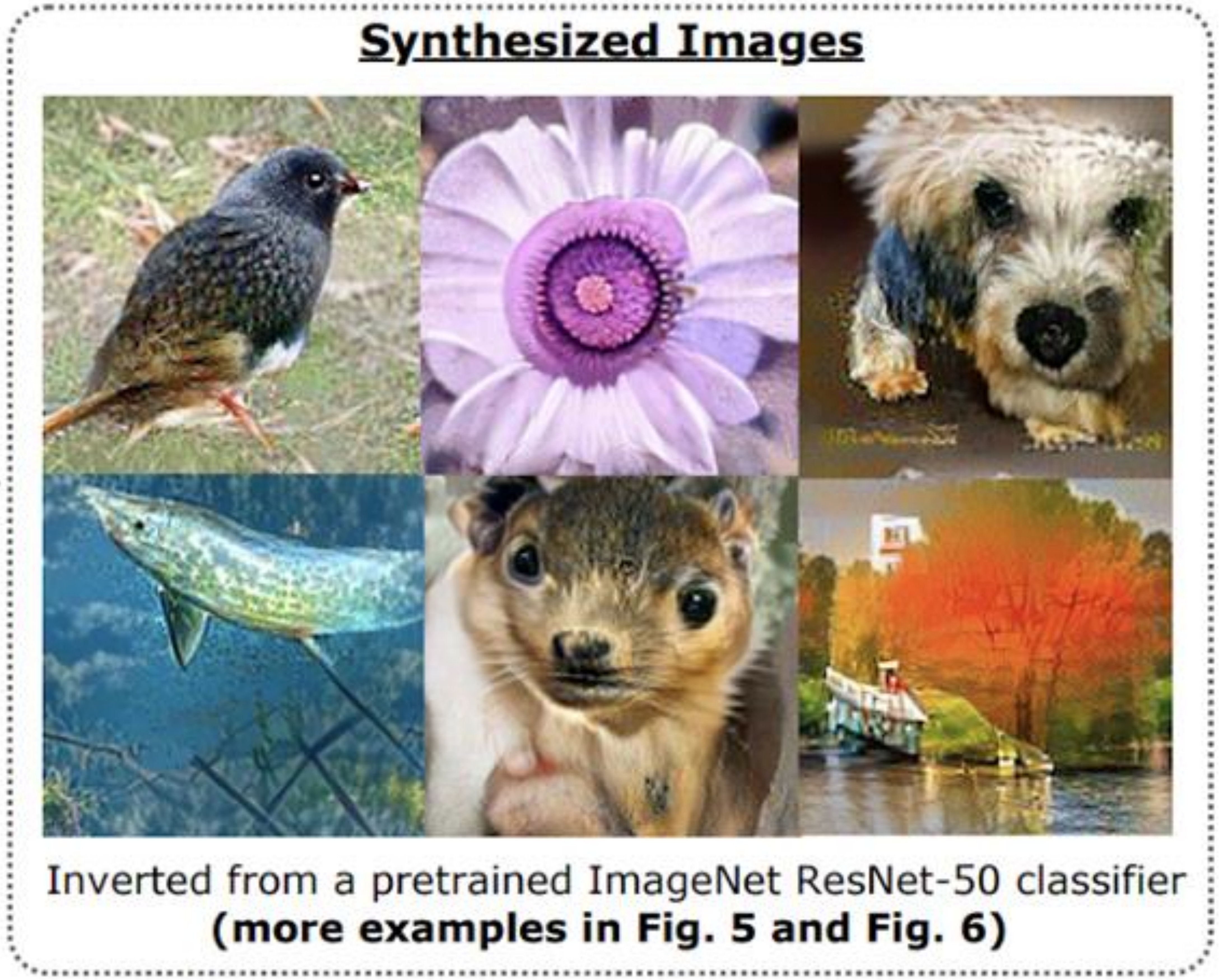
Private

Trained &
Shared

DeepInversion (CVPR'20 Oral)

Optimize Noise to Natural Images (Distribution Synthesis)





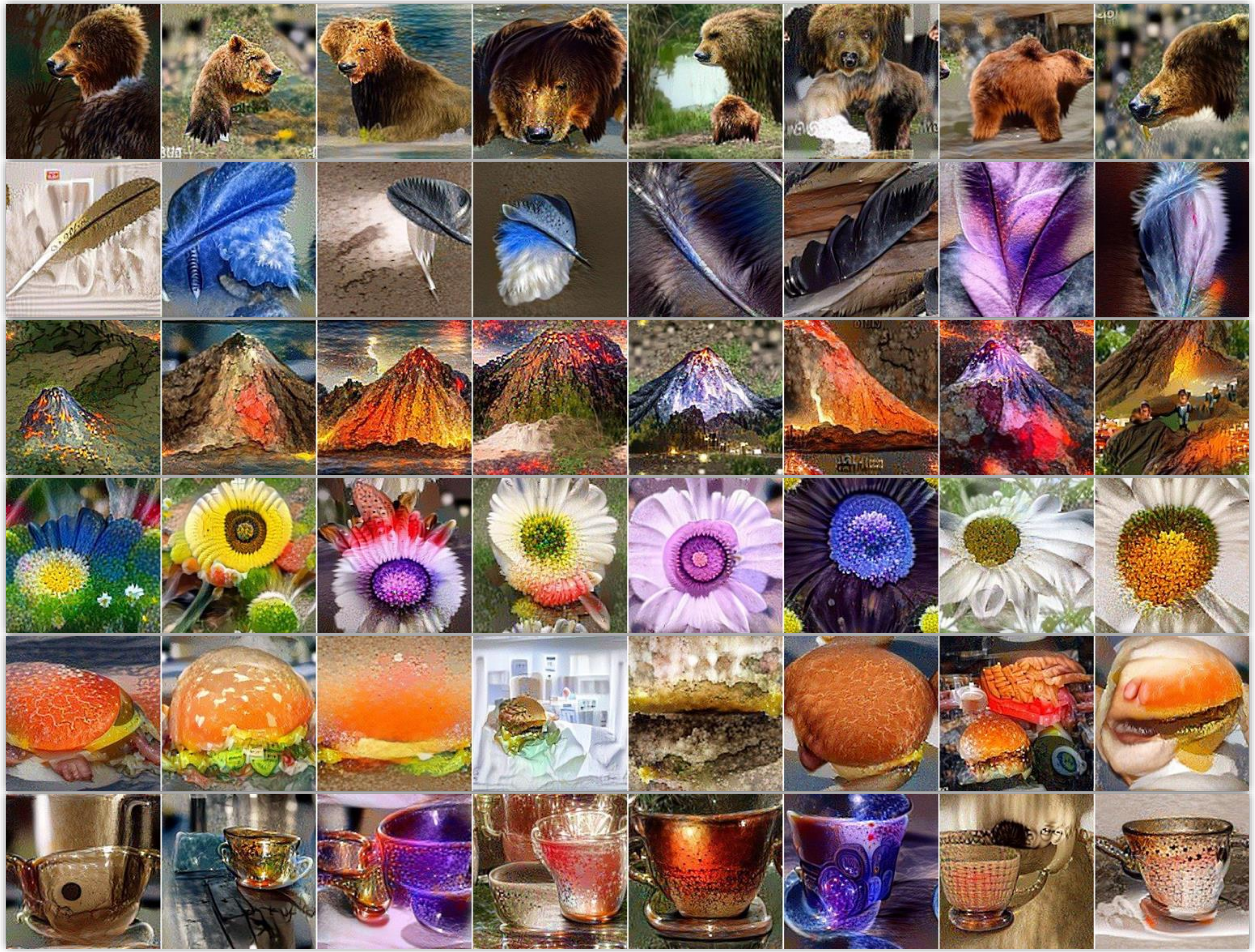
Inverted from a pretrained ImageNet ResNet-50 classifier
(more examples in Fig. 5 and Fig. 6)

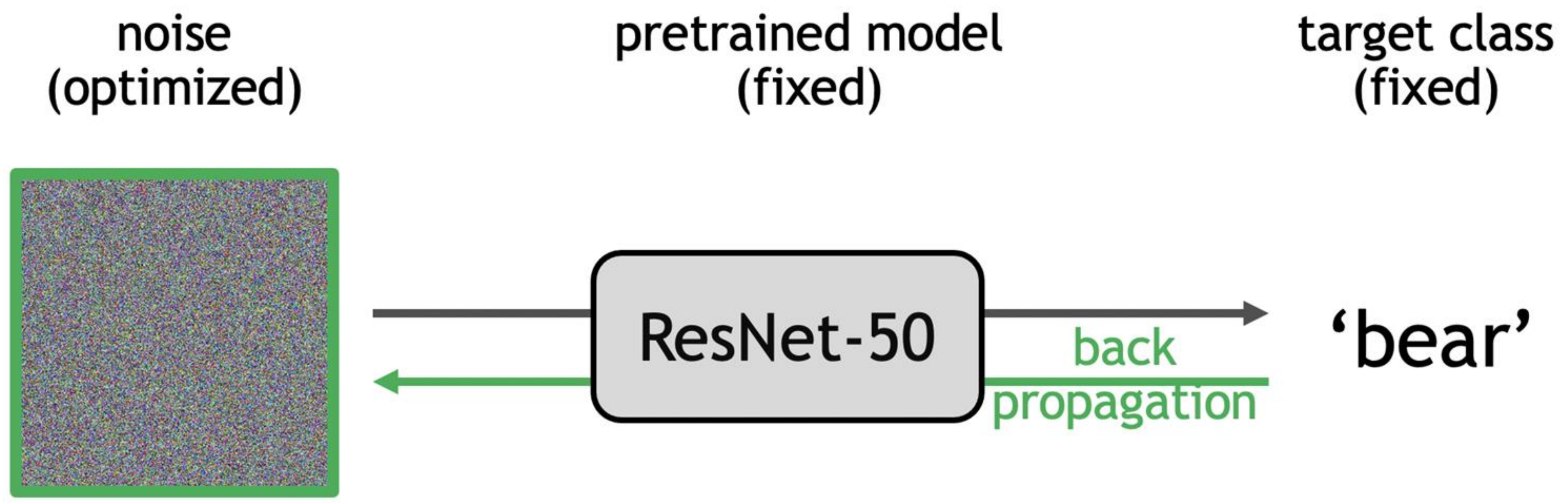
Trained Models <-> Datasets!

Yin*****, Molchanov*, et al., *Dreaming to Distill: Data-free Knowledge Transfer via DeepInversion*, *CVPR* 2020 oral

DeepInversion Image Analysis

What did we learn from inverting a ResNet-50 on ImageNet?





class-conditional

high resolution

high fidelity

high diversity

DeepInversion (CVPR’20 Oral)

Optimize Noise to natural Images (Distribution Synthesis)

DeepInversion
Feature distribution regularization
Input (updated)
Conv
Feature maps
BatchNorm
ReLU
...
Target class
Teacher logits
Cross entropy
Back propagation
Loss
1-JS
Noise ⟶ Image
Pretrained model (fixed), e.g. ResNet50
Teacher logits
Student model (fixed)
Student logits
Jensen-Shannon (JS) divergence
Adaptive DeepInversion
Synthesized Images
Inverted from a pretrained ImageNet ResNet-50 classifier
(more examples in Fig. 5 and Fig. 6)
Knowledge Distillation
Pretrained model (fixed)
Student model (updated)
Kullback–Leibler divergence
Back propagation
Loss

Zero real image, zero label!

- Data-free compression (pruning/quantization)
- Data-free knowledge distillation
- Data-free continual learning

Yin*, Molchanov*, et al., *Dreaming to Distill: Data-free Knowledge Transfer via DeepInversion*, *CVPR* 2020 oral

Data-free Applications

Zero real image, zero label

- **Data-free compression (pruning/quantization)**
- Data-free knowledge distillation
- Data-free continual learning

ImageNet ResNet-50 filter pruning, 20% filter pruned

Method	GFLOPs	top-1 accuracy	Training data needed
(base model)	4.1	76.1	-
Taylor-FO-BN-81 (CVPR-19)	2.7	75.5	1.2M image/label
SSS (ECCV-18)	2.8	74.2	1.2M image/label
ThiNet-70 (ICCV-17)	2.6	72.0	1.2M image/label
NISP-50-A (CVPR-18)	3.0	72.8	1.2M image/label
Ours (Data-free)	**2.7**	**73.3**	**0 image/label**

Data-free Applications

Zero real image, zero label

- Data-free compression (pruning/quantization)
- **Data-free knowledge distillation**
- Data-free continual learning

ResNet50v1.5 training on ImageNet

Setup	Training data	Loss	top-1 accuracy
Original (teacher)	1.2M ImageNet images/labels	Cross-entropy	77.2%
Data-free distillation (to student)	140K synthesized images	KL loss	**73.8%**

Data-free Applications

Zero real (old) image, zero (old) label

- Data-free compression (pruning/quantization)
- Data-free knowledge distillation
- **Data-free continual learning**

ImageNet ResNet-18, adding CUB and Flowers classes
(1000 to 1200 to 1302 output classes)

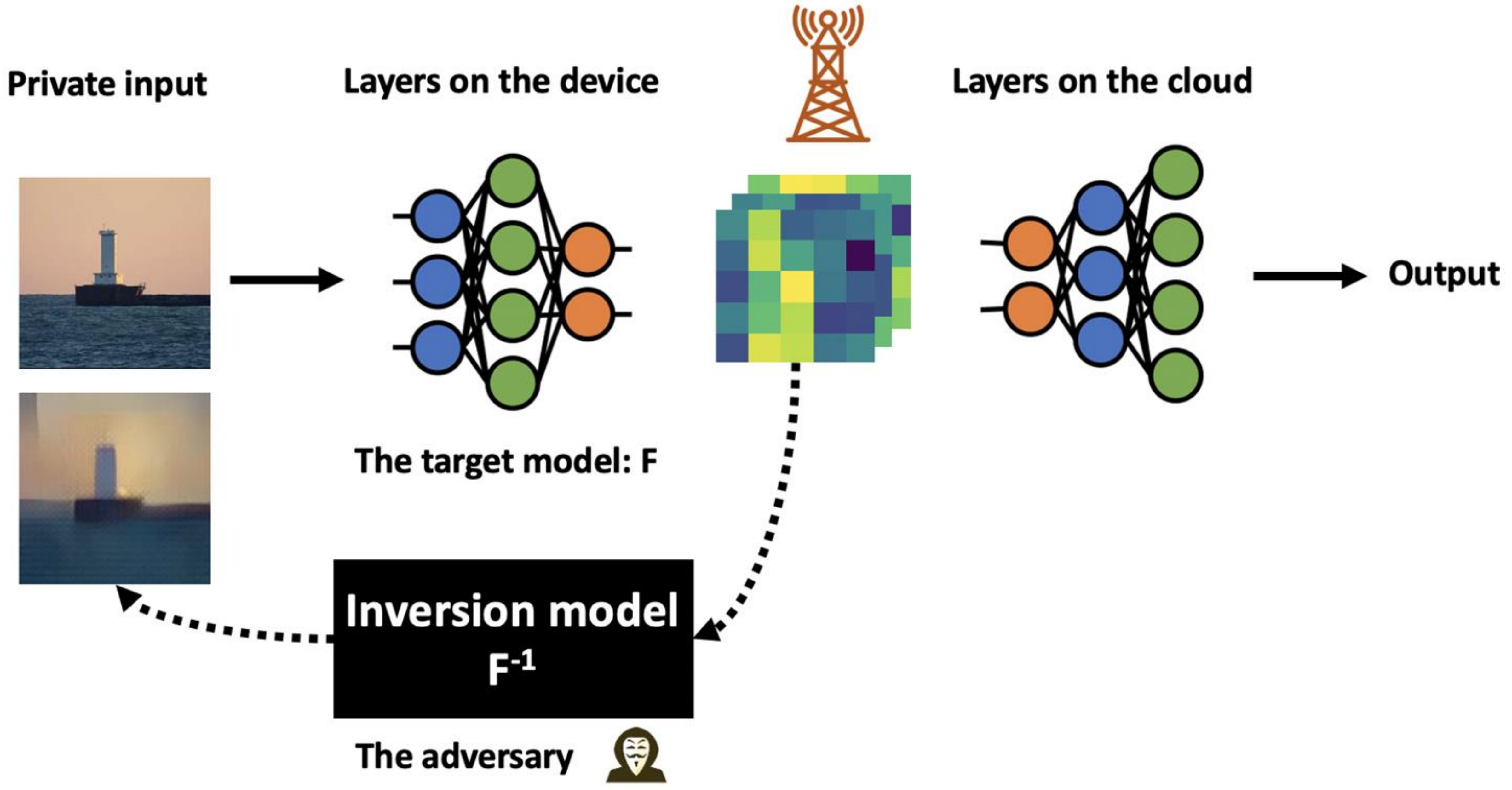
Methods	Combined*	ImageNet	CUB	Flowers
Oracle (distill)	76.2	67.2	69.6	91.8
Oracle (classify)	74.7	66.3	66.6	91.1
LwF.MC (CVPR-17)	41.7	40.5	26.6	58.0
Ours	**74.6**	**64.1**	**66.6**	**93.2**

* Performance averaged over all datasets

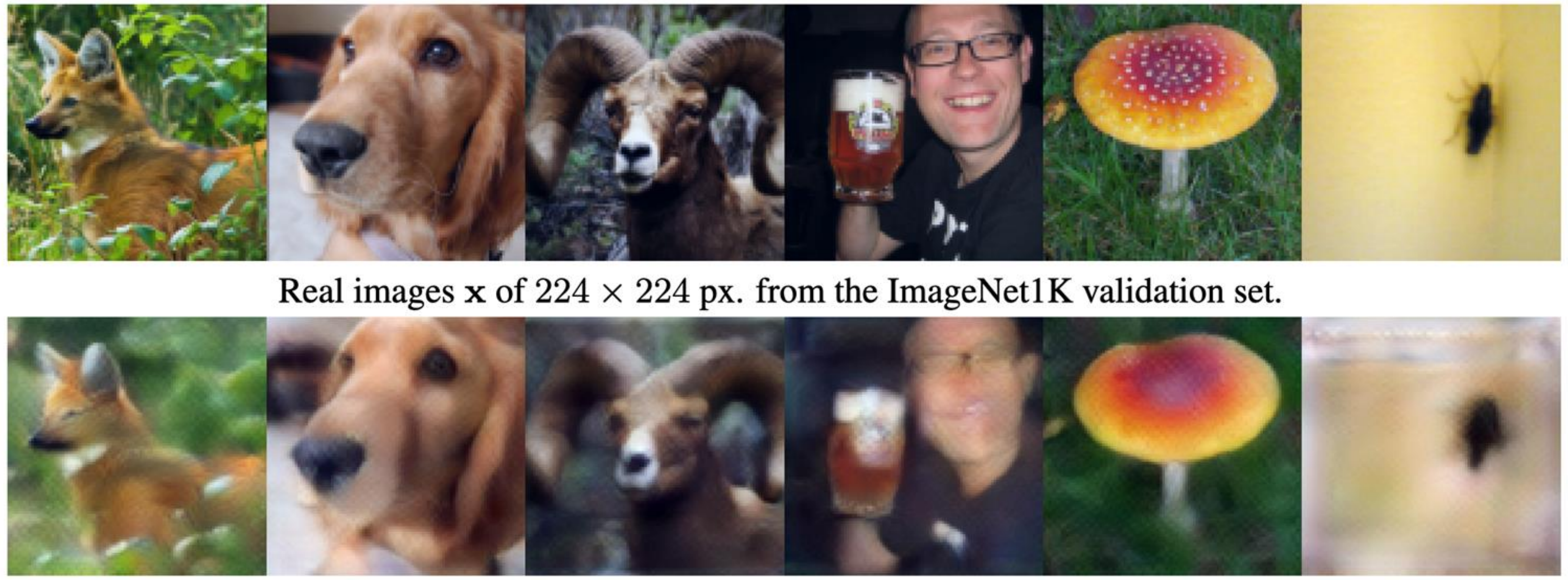
Networks encode dataset priors.

Security indication?

Inverting Feature Maps as in Split Computing



ResNet50 Feature Map Inversion - ImageNet



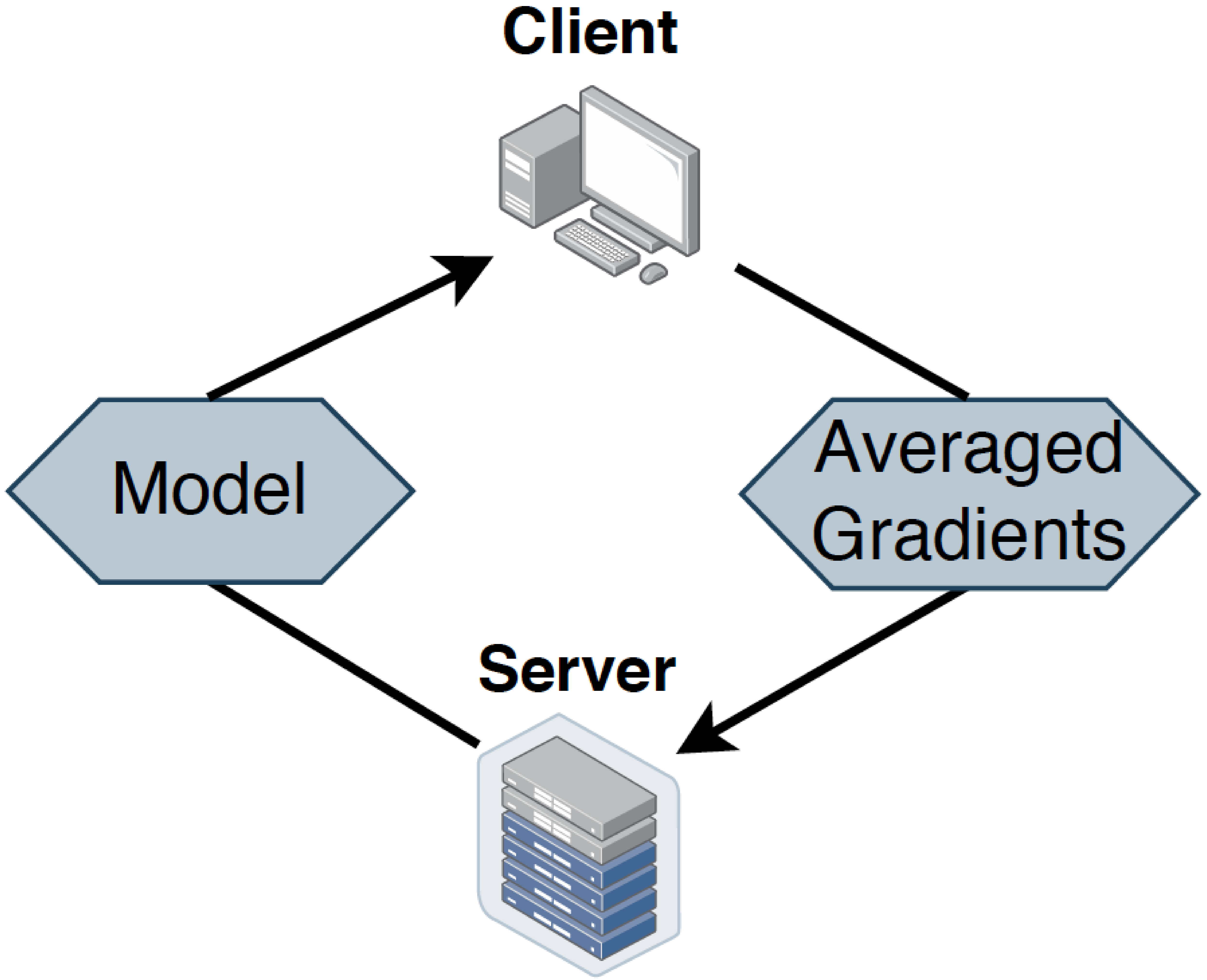
Real images $\mathbf{x}$ of 224×224 px. from the ImageNet1K validation set.

Recovered images from ResNet-50 (standard) feature embeddings after 42 convolution layers $\mathcal{F}^{-1}_{1:42}(\mathcal{F}_{1:42}(\mathbf{x}))$.

Dong, Yin, Alvarez, Kautz, Molchanov, *Deep Neural Networks are Surprisingly Reversible, BMCV*'22.

Inverting Gradients as in Gradient Sharing

Central Idea behind collaborative, distributed, and federated learning



Constraints

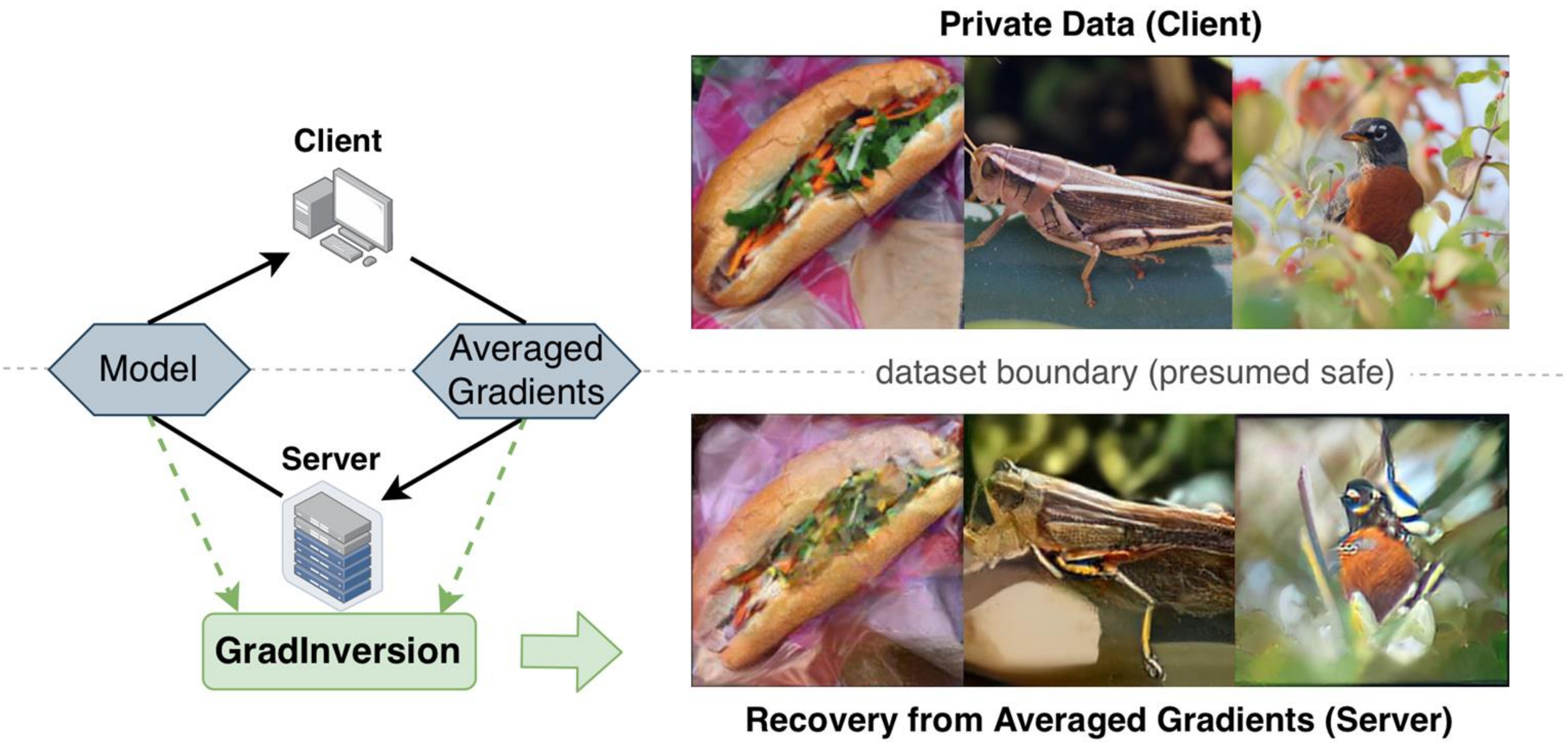
- CIFAR (NeurIPS’19)
- Sigmoid gates (NeurIPS’19)
- Batch size one (NeurIPS’20)

Sharing averaged gradients -> Assumed safe

Zhu et al., “Deep leakage from gradients,” *NeurIPS*, 2019
Geiping et al., “Inverting gradients–How easy is it to break privacy in federated learning?,” *NeurIPS*, 2020

GradInversion (CVPR’21)

Invert Averaged Gradients to Recover (Original) Images



Private Data (Client)
Client
Model
Averaged Gradients
Server
GradInversion
dataset boundary (presumed safe)
Recovery from Averaged Gradients (Server)

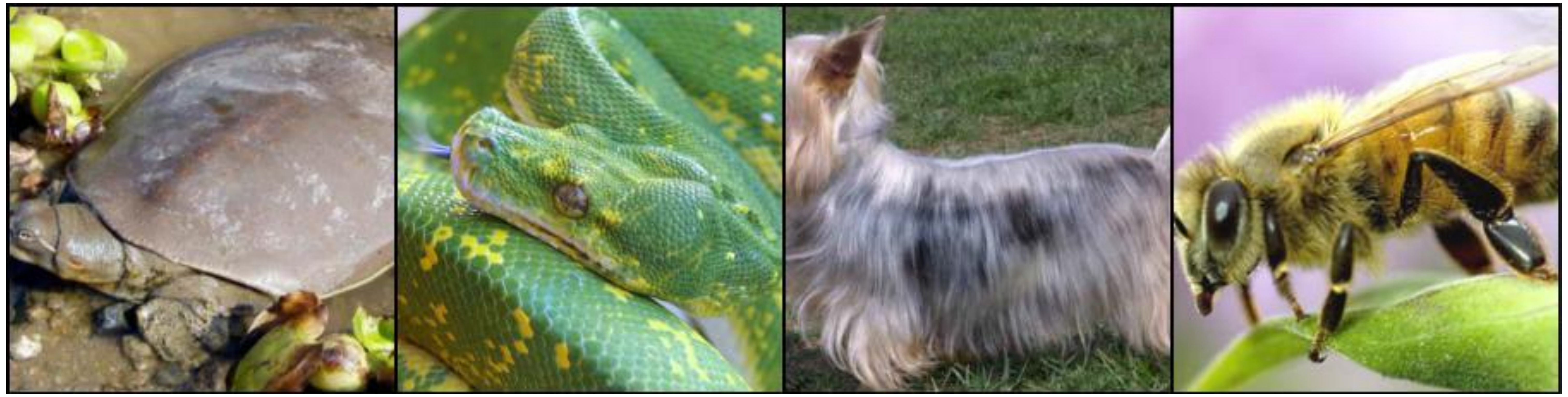
- **Larger batch size, e.g., 48**
- **ResNets (50 layers)**
- **ImageNet (1K classes, 224x224 px)**

Yin, Mallya, Vahdat, Alvarez, Kautz, Molchanov, *See through Gradients: Image Batch Recovery via GradInversion*, *CVPR*, 2021

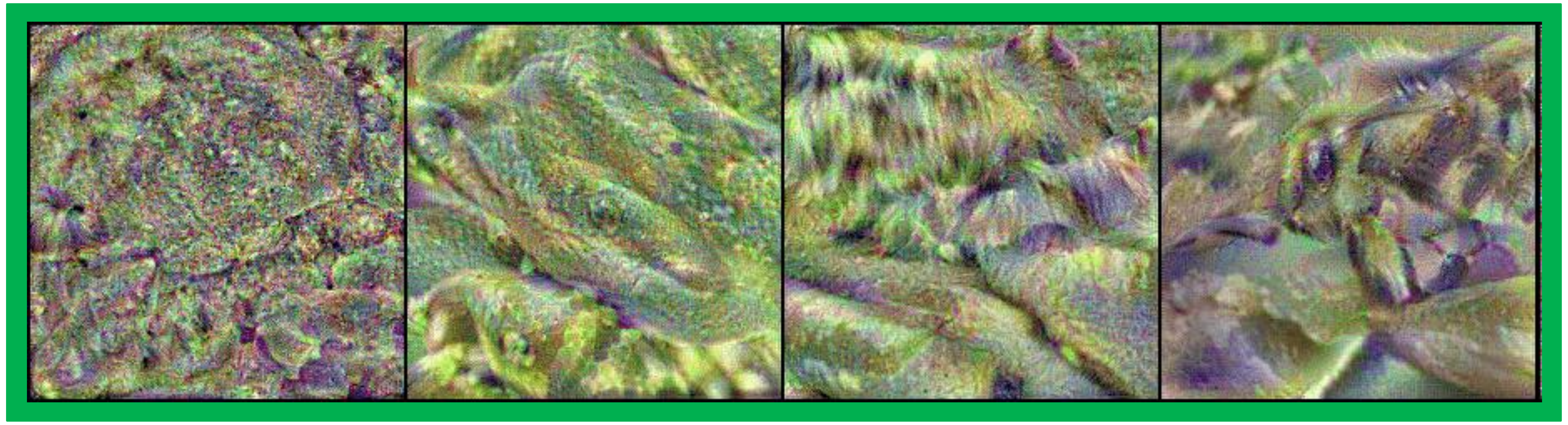
GradInversion (CVPR'21)

A Quick Demo - Inverting Gradients from ResNet-50 on ImageNet

Private Batch



Gaussian Noise $\mathcal{N}(0, \mathcal{I})$

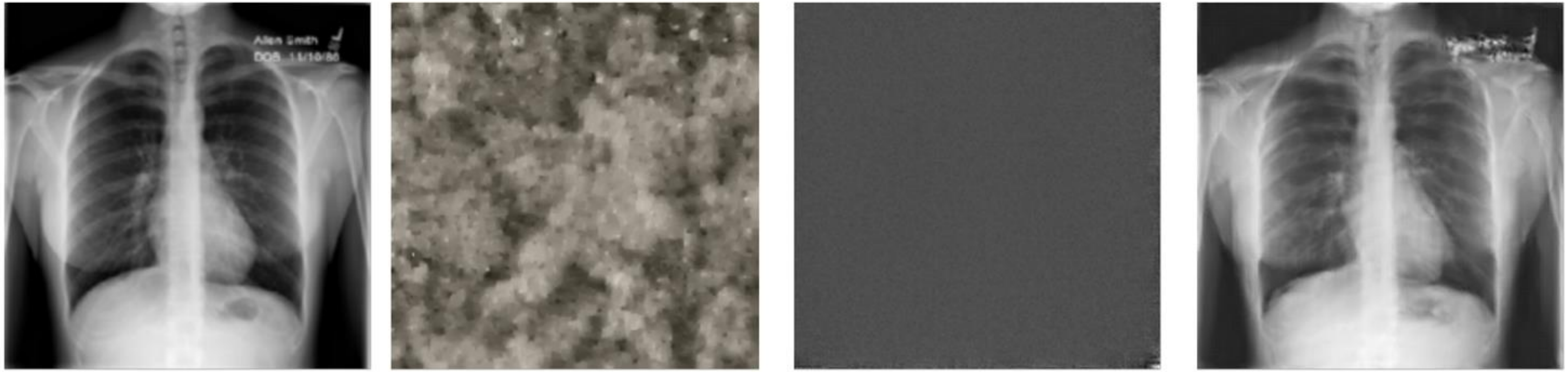


(Optimized by GradInversion from ImageNet-trained ResNet-50)

- **Off-the-shelf ResNets**
- **No GAN needed**
- **No meta-data on original dataset needed**

Yin, Mallya, Vahdat, Alvarez, Kautz, Molchanov, *See through Gradients: Image Batch Recovery via GradInversion, CVPR*, 2021

Other Domains



(a) Original

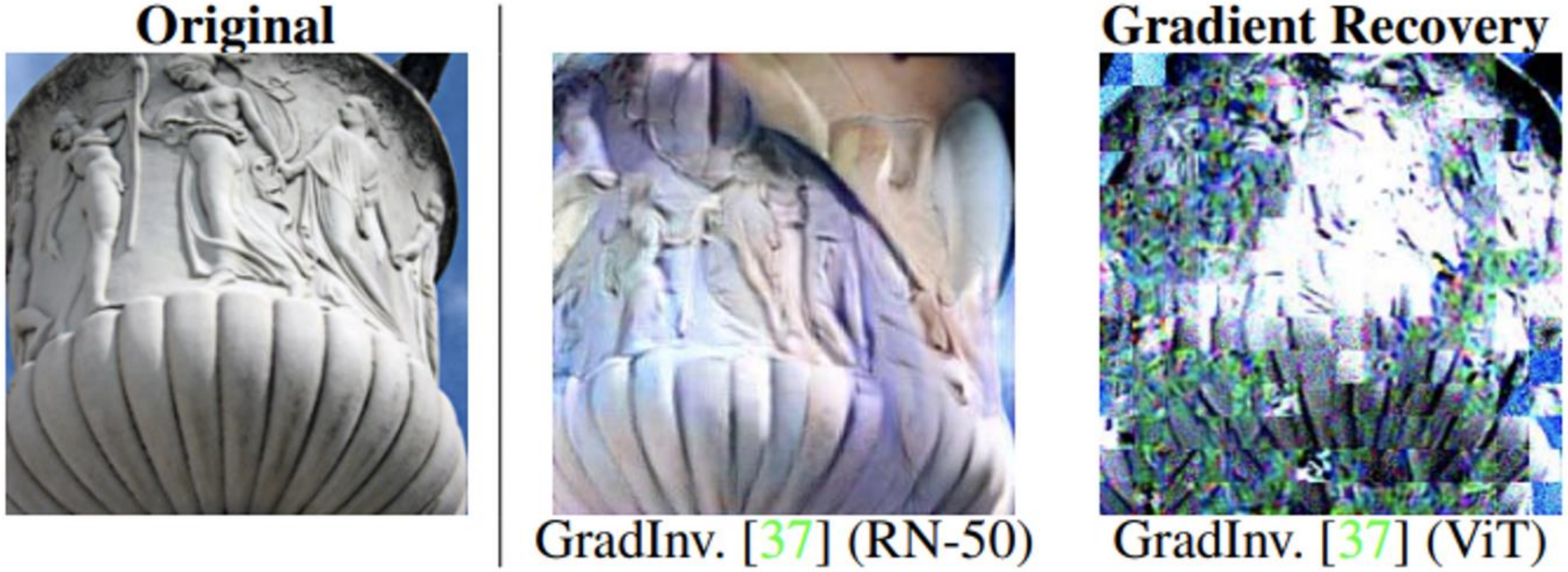
(b) w/o BN loss, w global ckpt [13]

(c) w BN loss - w/o global ckpt

(d) **Ours**: w BN loss, w global ckpt

Hatamizadeh, Yin, Molchanov, et al., *Do Gradient Inversion Attacks Make Federated Learning Unsafe? IEEE TBI 2022.*

How about Vision Transformers Gradient Inversion?



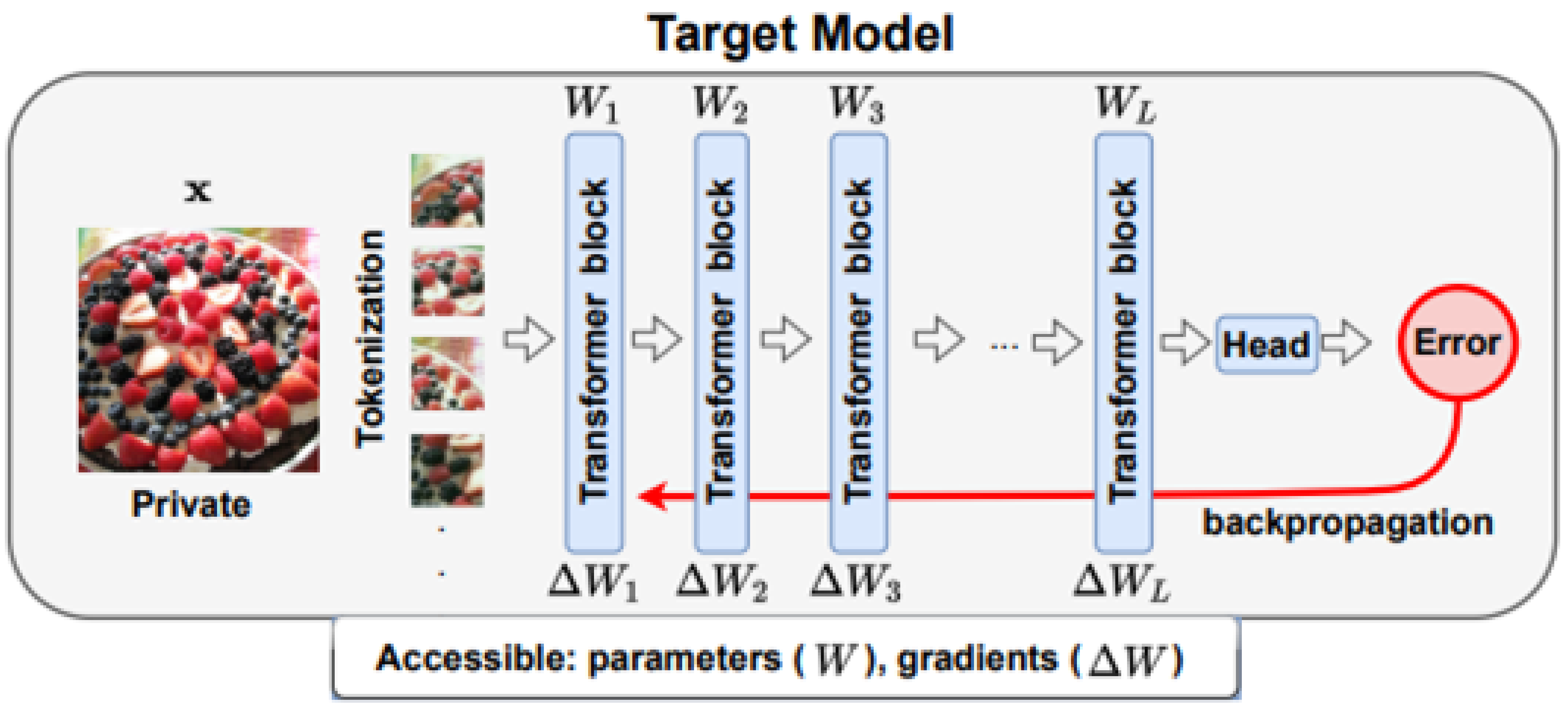
Vision Transformers Gradient Inversion - GradViT

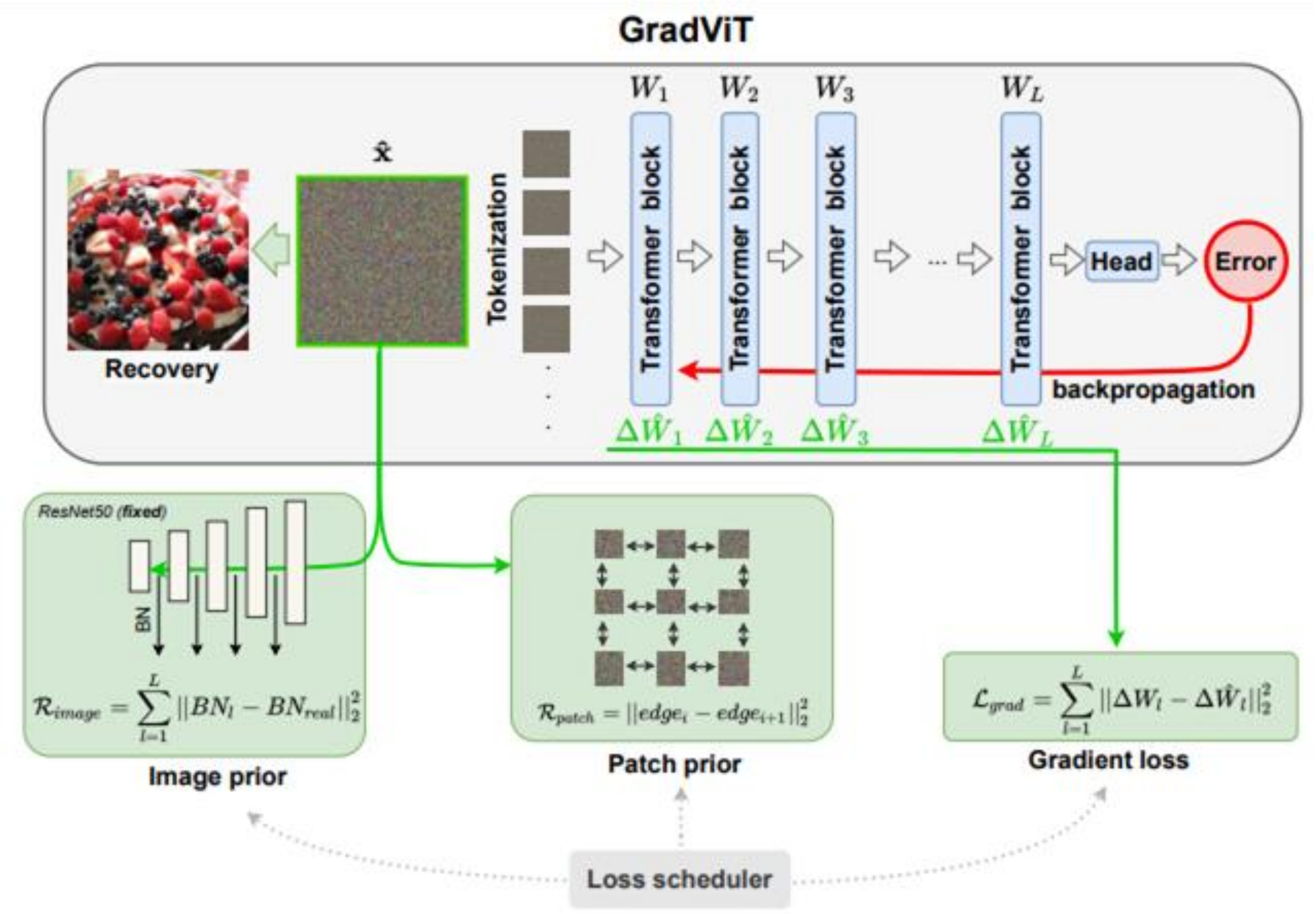
Hatamizadeh*, Yin*, et al., *GradViT: Gradient Inversion for Vision Transformers, CVPR*, 2022

Vision Transformers Gradient Inversion - GradViT (CVPR'22)

Hatamizadeh*, Yin*, et al., *GradViT: Gradient Inversion for Vision Transformers, CVPR*, 2022

GradViT


Target Model
x
Private
Tokenization
W_1
W_2
W_3
W_L
Transformer block
Transformer block
Transformer block
Transformer block
...
Head
Error
backpropagation
ΔW_1
ΔW_2
ΔW_3
ΔW_L
Accessible: parameters (W), gradients (ΔW)


GradViT
Recovery
$\hat{\mathbf{x}}$
Tokenization
W_1
W_2
W_3
W_L
Transformer block
Transformer block
Transformer block
Transformer block
...
Head
Error
backpropagation
$\Delta \hat{W}_1$
$\Delta \hat{W}_2$
$\Delta \hat{W}_3$
$\Delta \hat{W}_L$
ResNet50 (fixed)
BN
$\mathcal{R}_{image} = \sum_{l=1}^{L} ||BN_l - BN_{real}||_2^2$
Image prior
$\mathcal{R}_{patch} = ||edge_i - edge_{i+1}||_2^2$
Patch prior
$\mathcal{L}_{grad} = \sum_{l=1}^{L} ||\Delta W_l - \Delta \hat{W}_l||_2^2$
Gradient loss
Loss scheduler

Results: Face Domain, MS-CELEB-1M

Original images of 112 × 112 px.

Recovery from Face-Transformer [42] gradient by GradViT (ours)

Figure 4. Qualitative comparison of reconstructed images from MS-Celeb-1M dataset using batch gradient inversion of Face-Transformer [42]. GradViT is able to recover detailed and identical facial features as in original. Recovery at batch size 4. Best viewed in color.

Main Takeaways - CNN Insights Scale to ViTs

Network Efficiency

CNNs

NAS & Pruning & LANA

ViT

NViT’22 (pruning scales)

A-ViT’22 (adaptive inference scales better)

SmoothQuant’23 (quantization scales)

Data Efficiency & Security

CNNs

DeepInversion (model is dataset)

GradInversion (proxy info. not proxy)

ViT

GradViT (ViT more vulnerable)

Links at NVLabs

https://github.com/NVlabs/Taylor_pruning

https://github.com/NVlabs/A-ViT

https://github.com/NVlabs/DeepInversion

https://github.com/NVlabs/NViT

https://github.com/NVlabs/HALP

(more to come)

Thank You!
Q & A

Hongxu (Danny) Yin
dannyy@nvidia.com

joint with

MIT Duke UNIVERSITY Berkeley UNIVERSITY OF CALIFORNIA HARVARD UNIVERSITY PRINCETON UNIVERSITY Microsoft