Efficient Deep Learning Models

Rowel Atienza, PhD

University of the Philippines

2023

Roadmap

Define & Quantify Model Efficiency

Techniques for Improving Model Efficiency

Assumption: Knowledge of deep learning (& PyTorch)

Why does it matter?

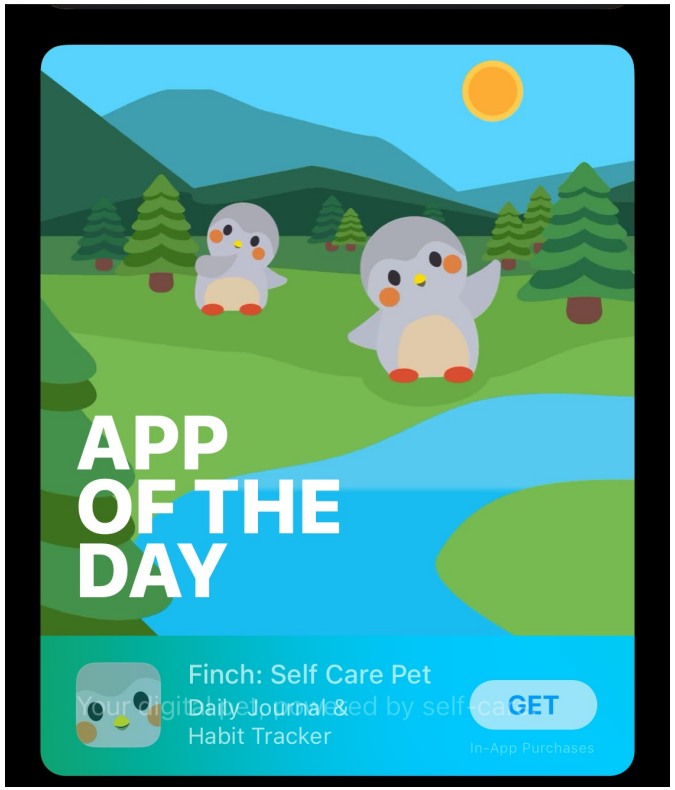
Efficient model → Good user experience

amazon

1% sales drop
per 1 sec delay

https://semupdates.com/

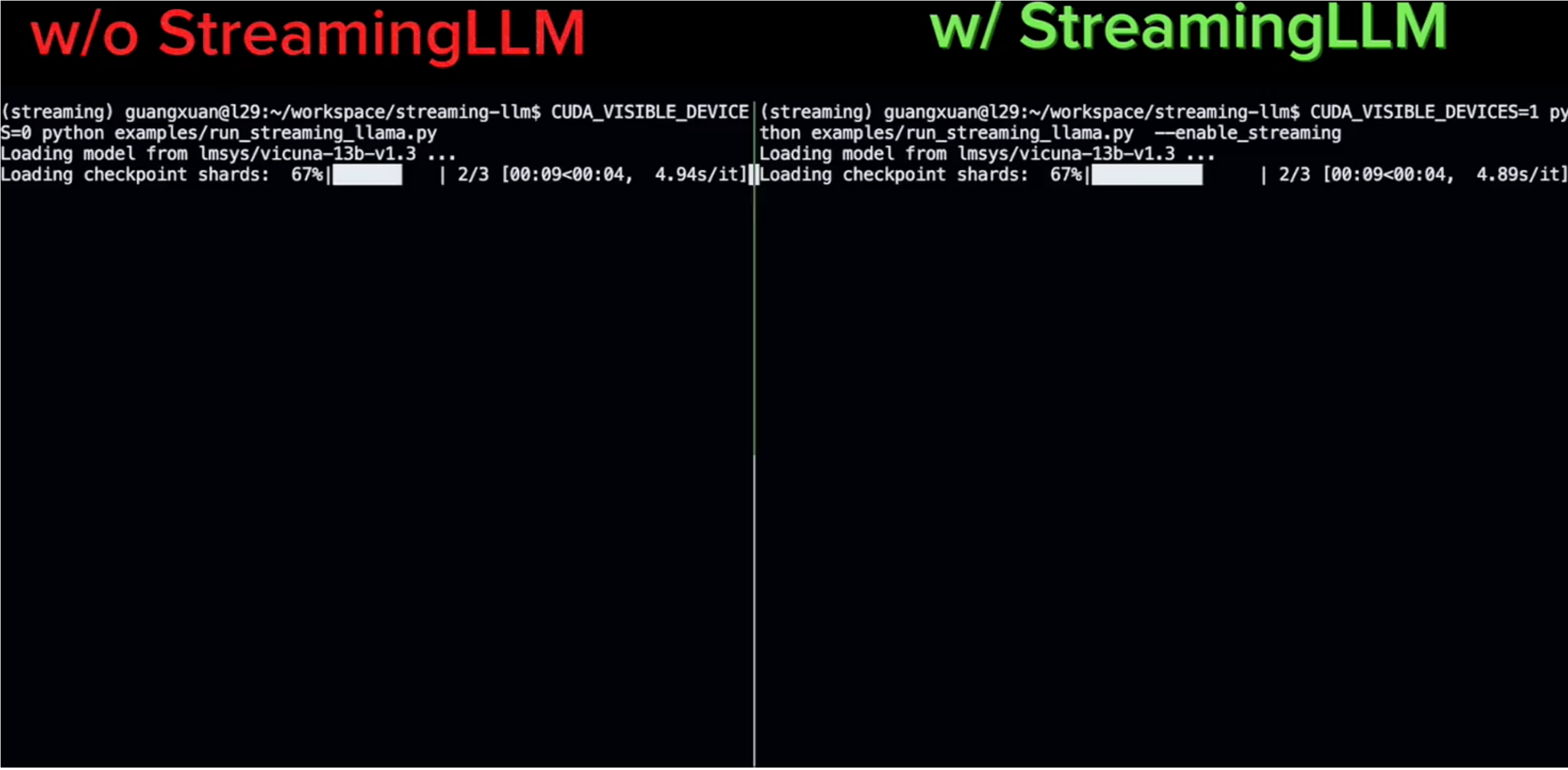
SEO penalizes
slow sites



Slow apps receive
poor ratings

Why does it matter?

Intelligence is equated to speed


w/o StreamingLLM
w/ StreamingLLM
(streaming) guangxuan@l29:~/workspace/streaming-llm$ CUDA_VISIBLE_DEVICES=0 python examples/run_streaming_llama.py
Loading model from lmsys/vicuna-13b-v1.3 ...
Loading checkpoint shards: 67%| | 2/3 [00:09<00:04, 4.94s/it]
(streaming) guangxuan@l29:~/workspace/streaming-llm$ CUDA_VISIBLE_DEVICES=1 python examples/run_streaming_llama.py --enable_streaming
Loading model from lmsys/vicuna-13b-v1.3 ...
Loading checkpoint shards: 67%| | 2/3 [00:09<00:04, 4.89s/it]

Why does it matter?

Efficient model → Less environmental impact

GPT-3 175B param : 1,287 MWh, 552 tons CO_2

Typical US household: ~1 MWh / month

https://www.scientificamerican.com/article/a-computer-scientist-breaks-down-generative-ais-hefty-carbon-footprint/

Why does it matter?

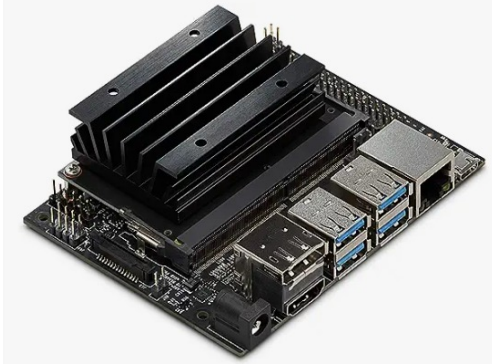
Efficient model → Less overall operational cost

CPU: $0.01/hr

LambdaLabs

Efficient models can run on a cheaper device

Purchased May 2023



NVIDIA Jetson Nano Developer Kit

4.5 out of 5 stars 533

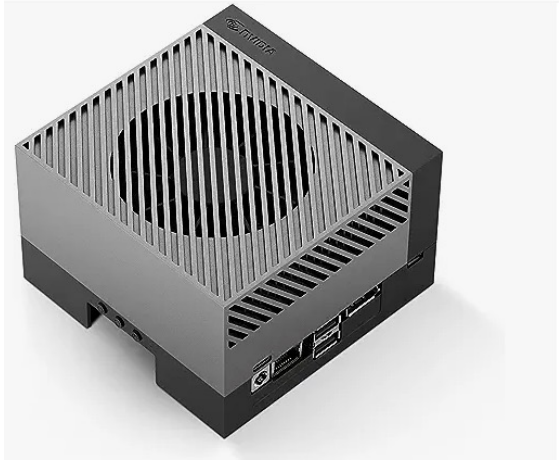
$148.94

FREE delivery

Ships to Philippines

More Buying Choices

$148.93 (11 new offers)



NVIDIA Jetson AGX Orin 64GB Developer Kit

5.0 out of 5 stars 2

$1,999.00

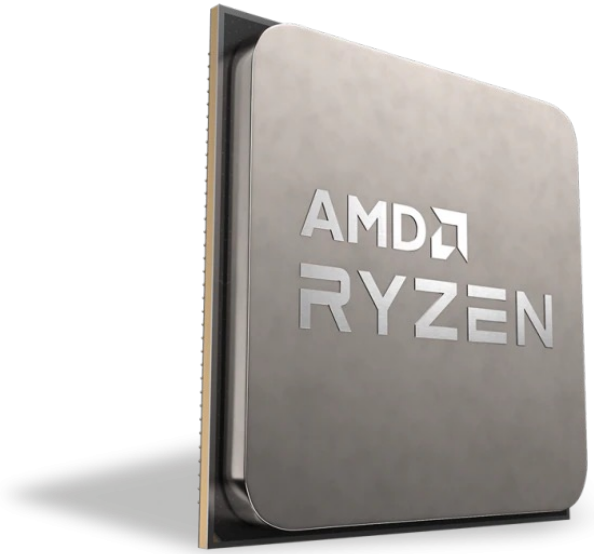
$33.52 delivery **Thu, Nov 9**

Ships to Philippines

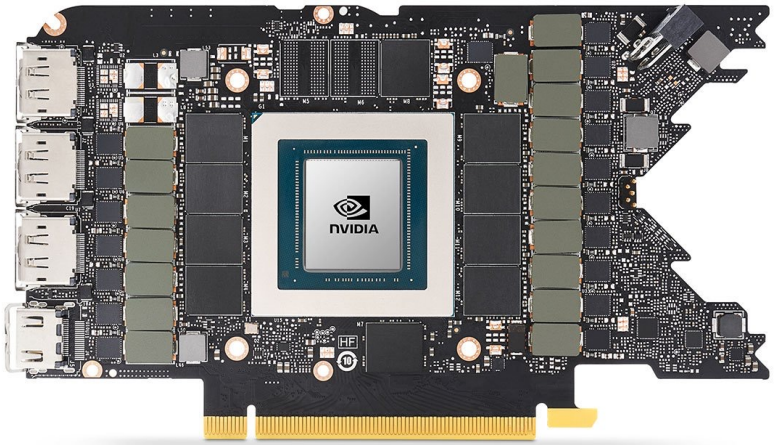
$150 Nano instead of $2k Orin

Efficient model can run in any processor

Embedded CPU



CPU



GPU

Our Efficient AI Models

1. De Leon, Josen Daniel, and Rowel Atienza. "Depth Pruning with Auxiliary Networks for Tinyml." *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2022.
2. EfficientSpeech: An On-device Text-to-Speech Model. *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2023.
3. Bautista, Darwin, and Rowel Atienza. "Scene text recognition with permuted autoregressive sequence models." European Conference on Computer Vision. Cham: Springer Nature Switzerland, 2022.

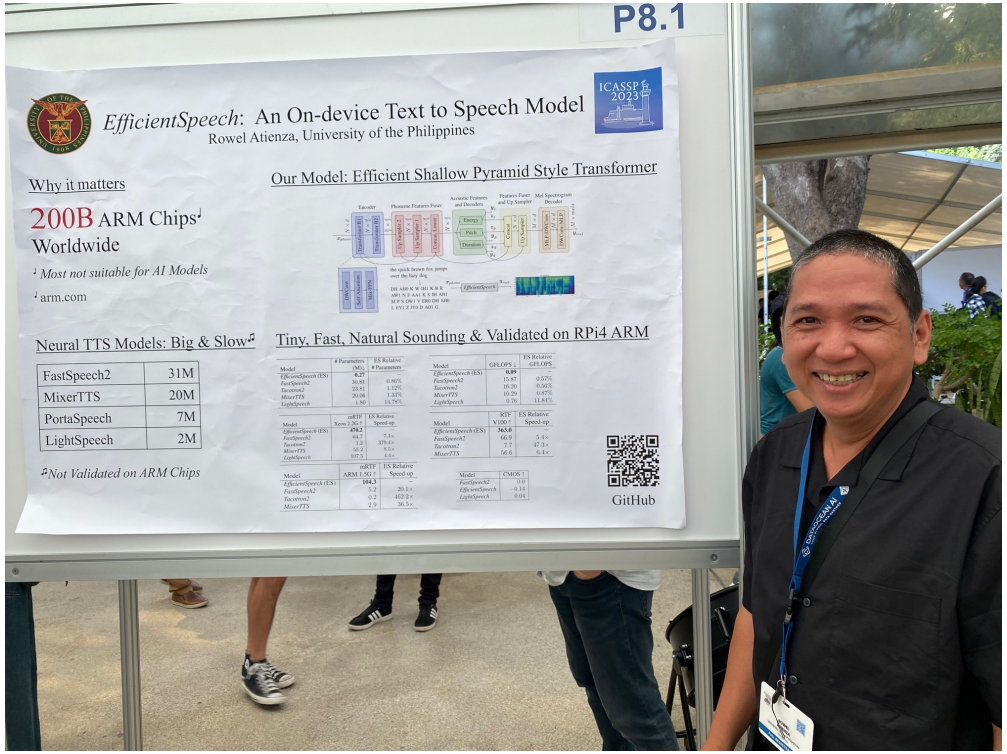
Visual Wake Words (Person Detection) on Cortex-M0 Microcontroller

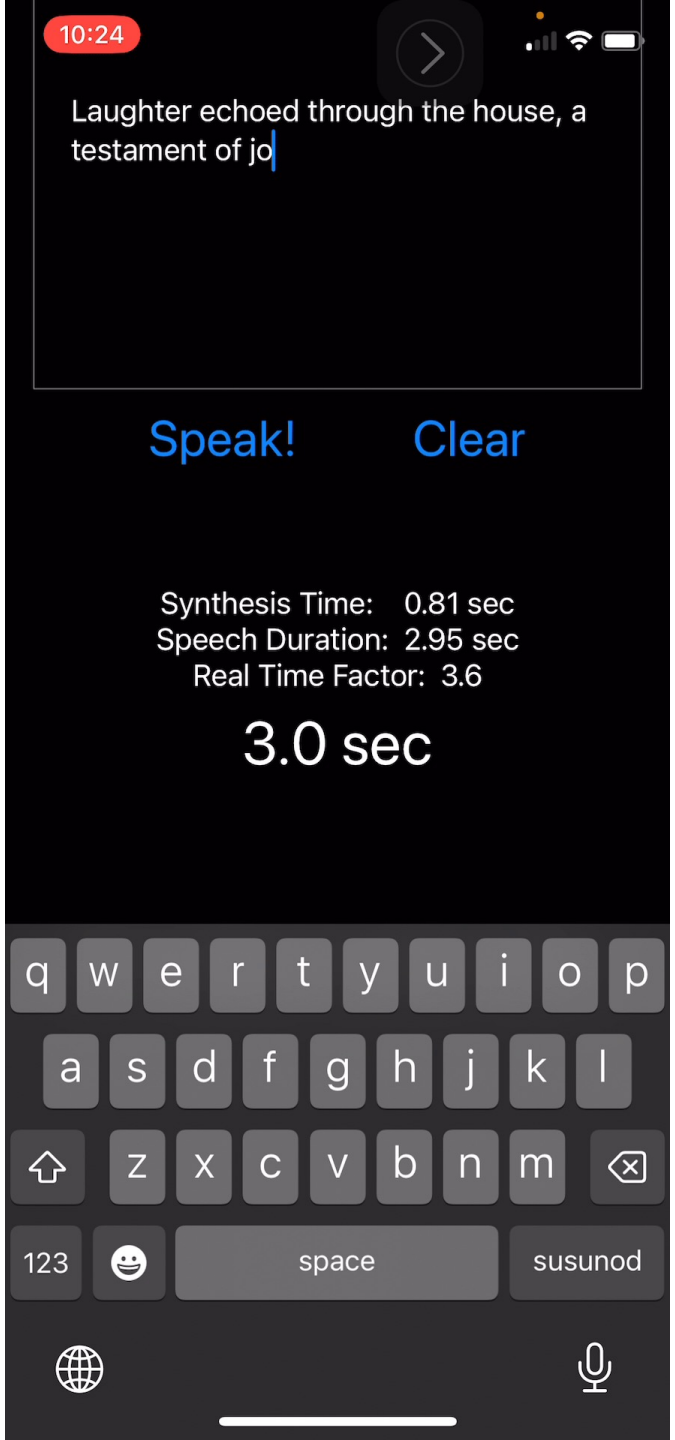
"Depth Pruning using Auxiliary Networks for TinyML"

By: De Leon, Josen Daniel and Atienza, Rowel
University of the Philippines - Diliman

Our Work on Standalone Speech Synthesis

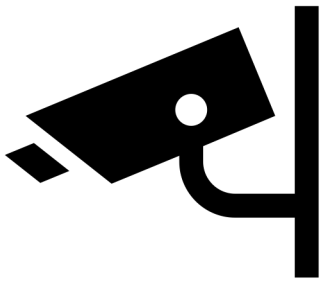
50x smaller that Apple's On-Device AI Model





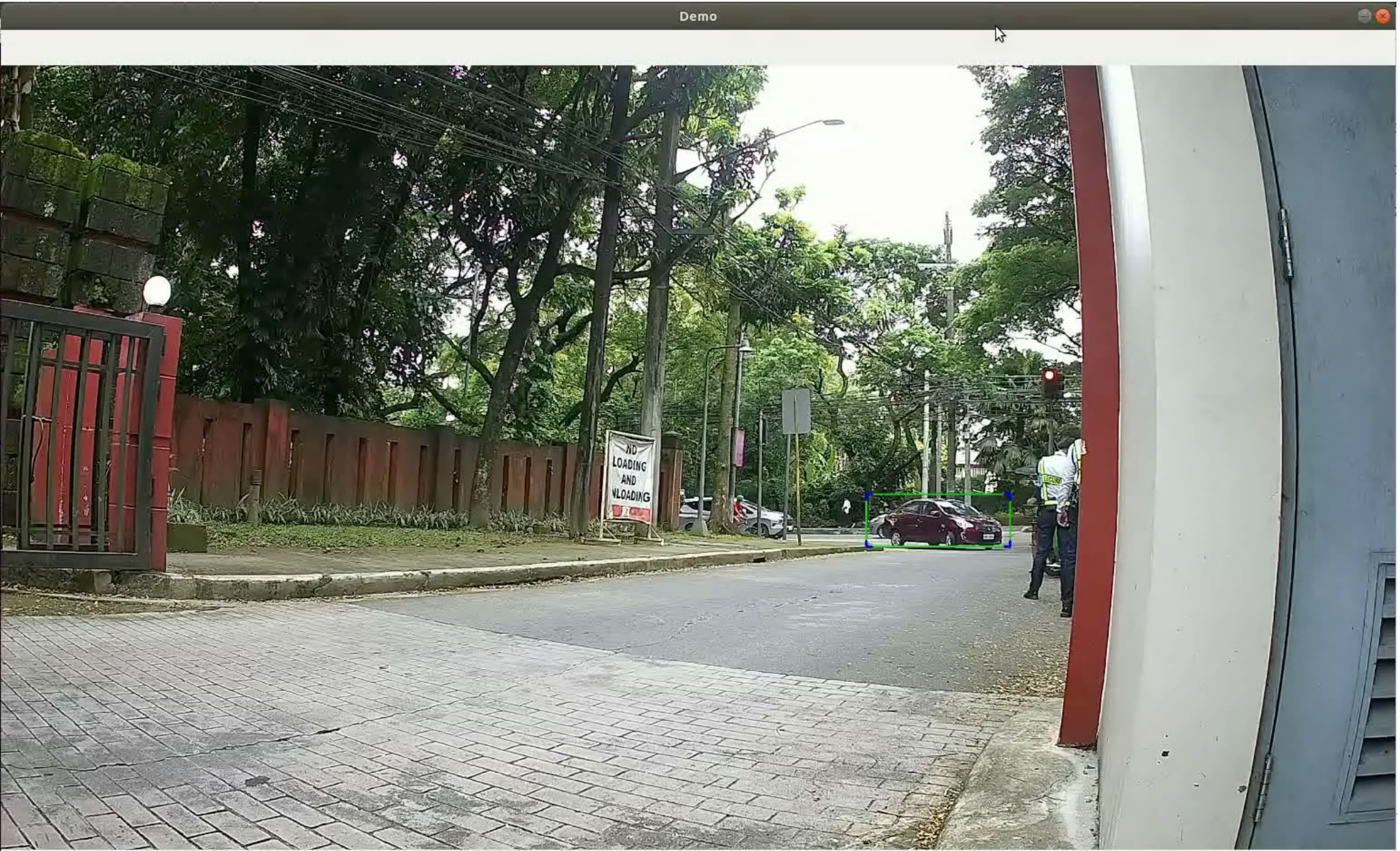
Atienza, Rowel *EfficientSpeech: An On-device Text to Speech Model*. IEEE Intl Conference on Acoustics, Speech and Signal Processing (ICASSP 2023)

Proof of Concept:
Automatic Car Plate Recognition at UP Portals



NVIDIA Jetson Orin

Yolov8 for Car Detection (Ultralytics) + PARSeq (Ours) for Plate Recognition


Demo
ND
LOADING
AND
VLOADING

PARSeq: UP's Scene Text Recognition AI Model – Best in the World

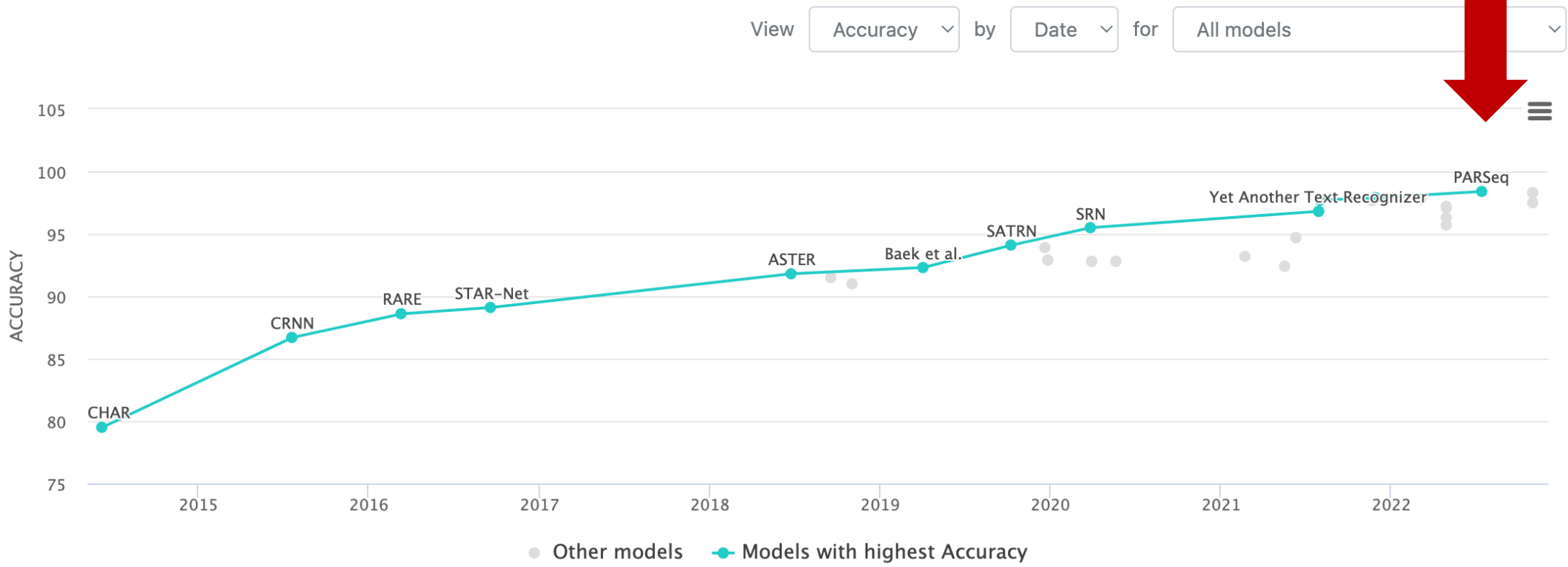
Darwin Bautista &
Rowel Bautista,
Scene text recognition with permuted autoregressive sequence models, European Conference on Computer Vision (ECCV 2022)

Paperswithcode.com

World's Best is PARSeq (Ours)

Scene Text Recognition on ICDAR2013

Leaderboard Dataset



Efficiency - Definition

Efficiency

Effective operation as measured by a comparison of production with cost (as in energy, time, and money)

Merriam Webster

The ratio of the useful work performed by a machine or in a process to the total energy expended or heat taken in.

Oxford Dictionary

Model **Inference** Efficiency

The amount of computing resources consumed by a model to perform a useful inference

$$Efficiency = \frac{Inference}{Computing\ Resource}$$

Computing Resources

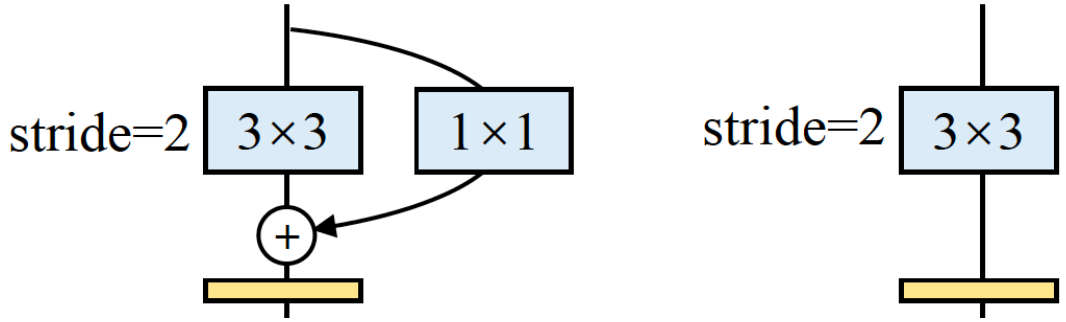
1. Memory Access Cost (MAC)
   Correlated: Parameters
2. FLOPS or Computation
3. Inference Latency

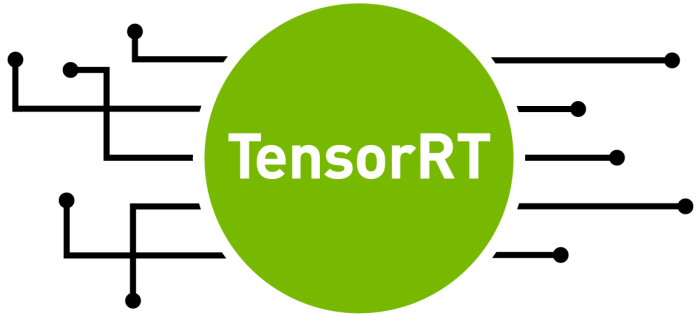
[Efficiency Misnomer, ICLR 2022]

Power Meter

Where to optimize?

- ✓Model Architecture
- ✓Model Optimization
- ○Frameworks and Libraries
- ✓Model format and compilation

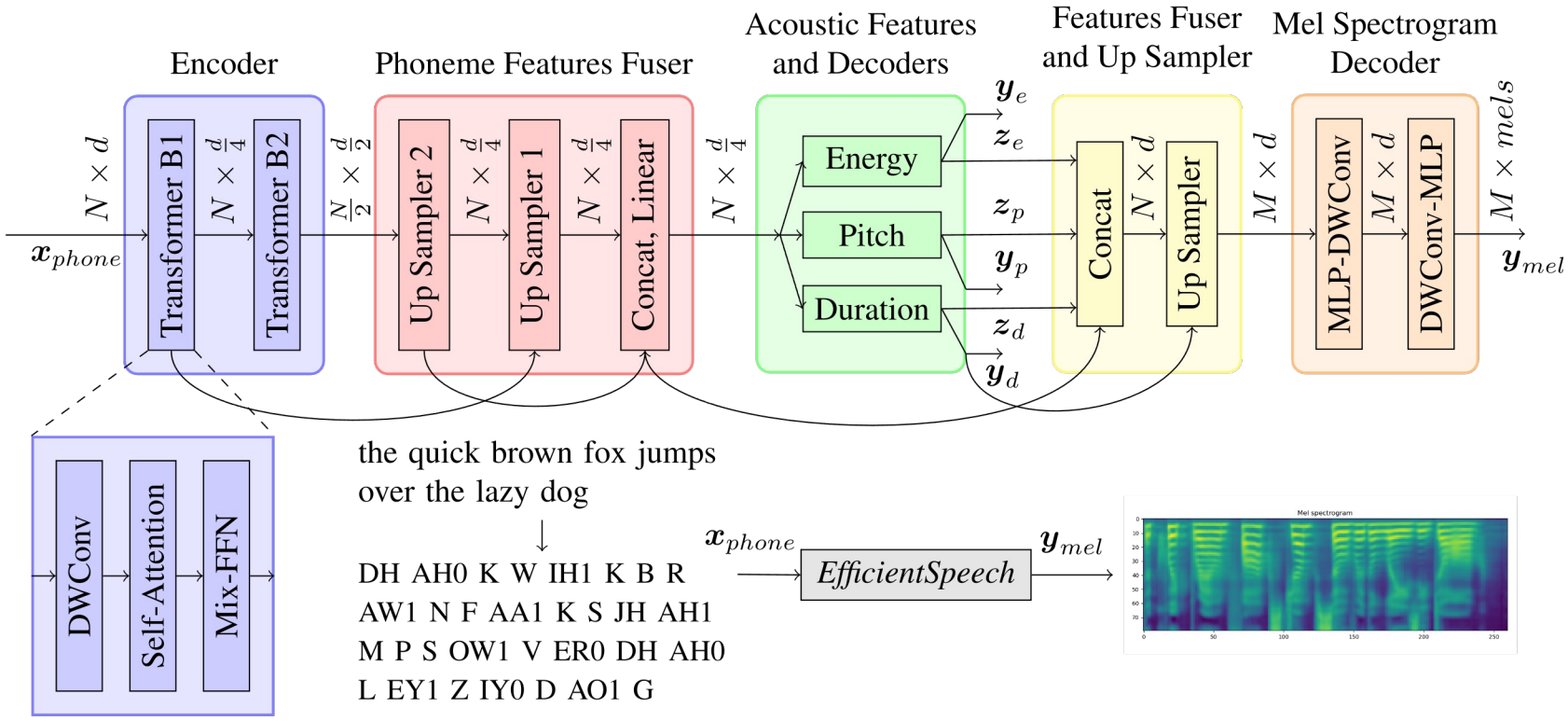




Where to optimize?

Model Architecture

EfficientSpeech: Use of Shallow Pyramid Style Transformer



Model Architecture - Pruning

Depth Pruning with Auxiliary Networks for Tinyml.

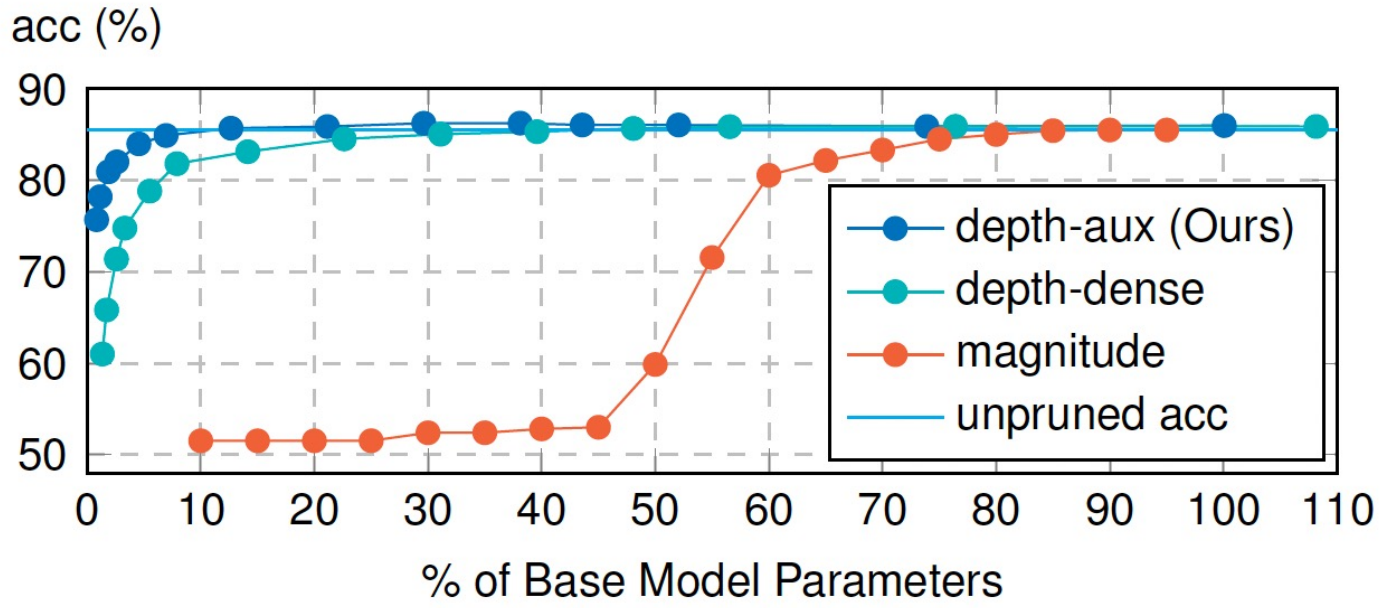
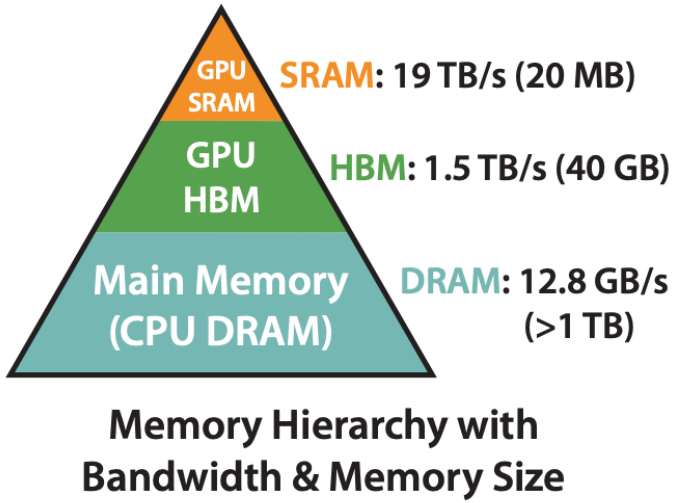


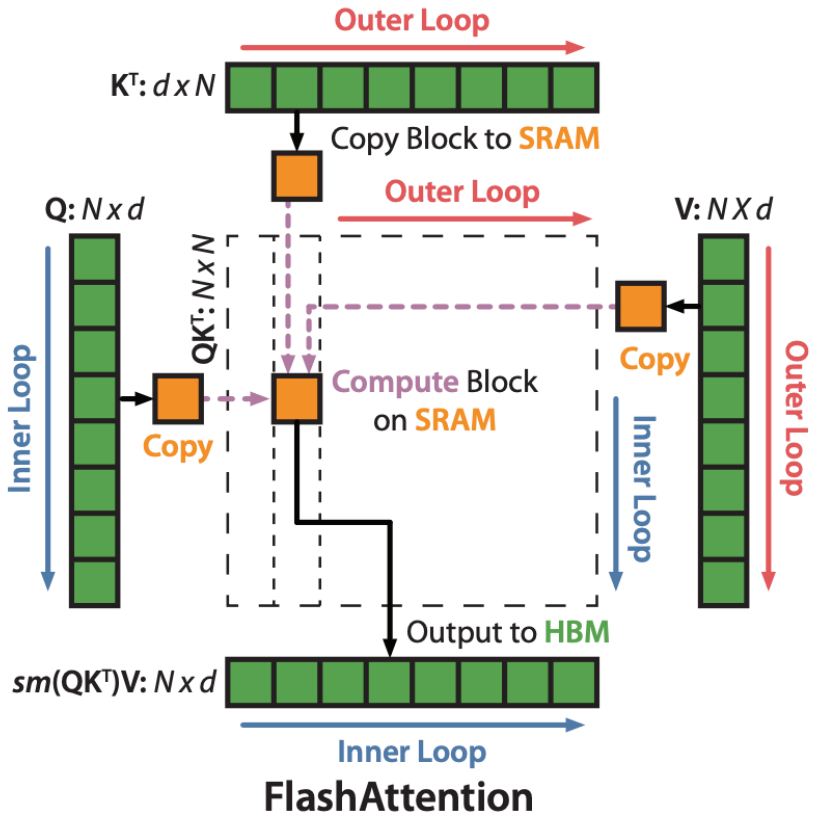
Fig. 1. Pruning Method Comparison on MobilenetV1 VWW task. Parameters are reduced by 93% for our method (frozen tail), 68.9% for depth pruning (dense), and 20% for magnitude pruning if maximum accuracy drop is set to 0.65%.

Model Optimization - Limit Data Movement

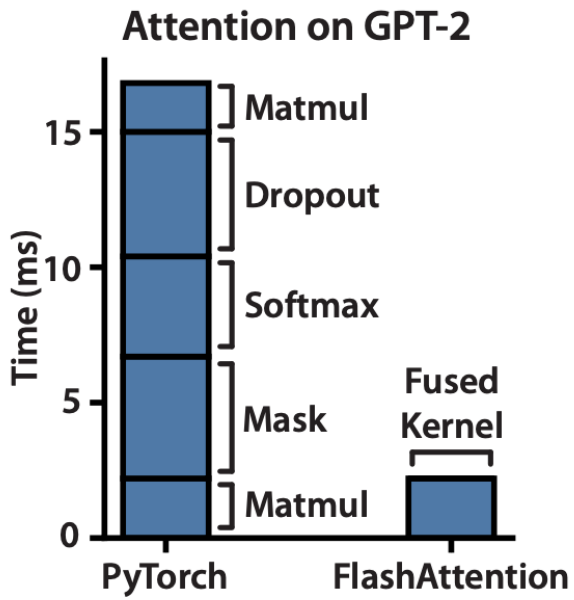
Dao, Tri, et al. "Flashattention: Fast and memory-efficient exact attention with io-awareness." *Advances in Neural Information Processing Systems* 35 (2022): 16344-16359.



Memory Hierarchy with Bandwidth & Memory Size



FlashAttention



Attention Block is HBM intensive

Algorithm 0 Standard Attention Implementation

Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM.

1: Load $\mathbf{Q}, \mathbf{K}$ by blocks from HBM, compute $\mathbf{S} = \mathbf{Q}\mathbf{K}^\top$, write $\mathbf{S}$ to HBM.
2: Read $\mathbf{S}$ from HBM, compute $\mathbf{P} = \mathrm{softmax}(\mathbf{S})$, write $\mathbf{P}$ to HBM.
3: Load $\mathbf{P}$ and $\mathbf{V}$ by blocks from HBM, compute $\mathbf{O} = \mathbf{P}\mathbf{V}$, write $\mathbf{O}$ to HBM.
4: Return $\mathbf{O}$.

3X Speed up on GPT2

FMA Instruction – Fused Multiply Add

- ✓Intel and AMD
- ✓NVIDIA CUDA Core
- ✓ARM (Cortex-A5 and Cortex-M4 processors)

$$y = wx + b$$

$$(Single\ FMA\ instruction)$$

FMA in NVIDIA Tensor Core

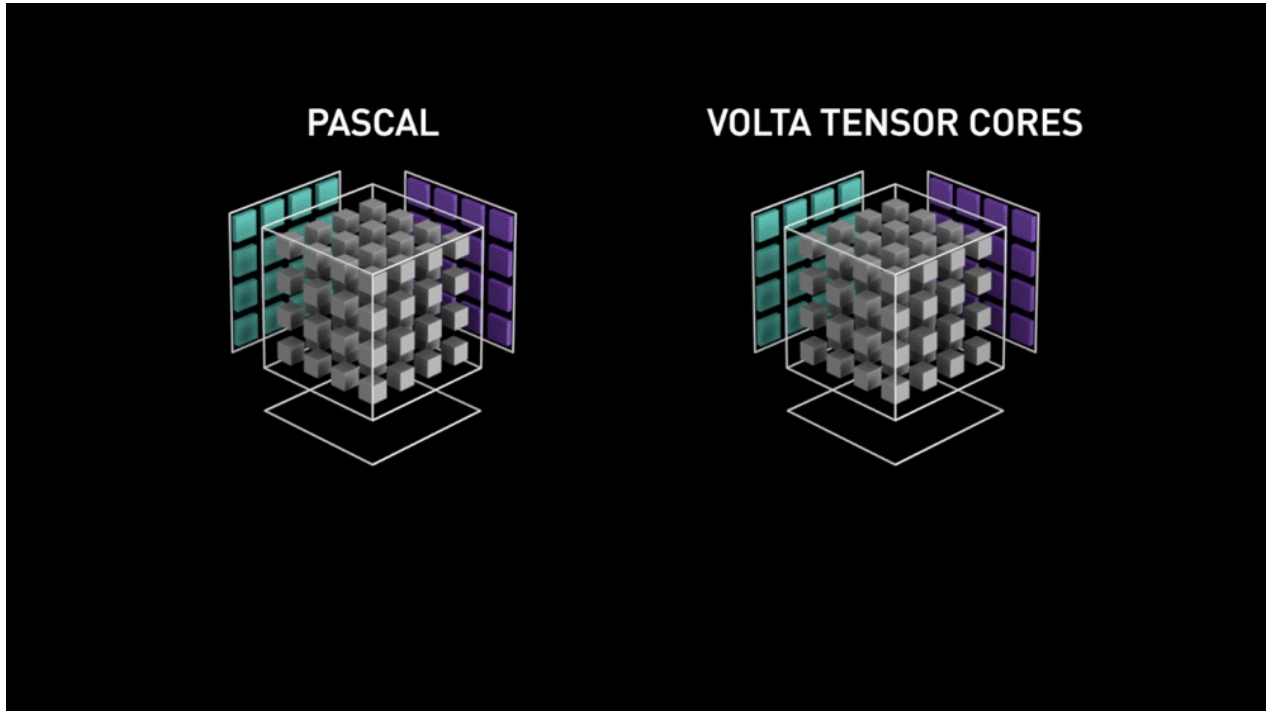
For 1 Tensor Core, only 1 GPU cycle to compute:

$$Y_{4\times4} = W_{4\times4}X_{4\times4} + B_{4\times4}$$

Tensor Core can do 64 FMAs in a GPU cycle

V100 GPU has 640 Tensor Cores or 40,960 FMAs

32.5 MFLOPS can be done in 784 GPU cycles



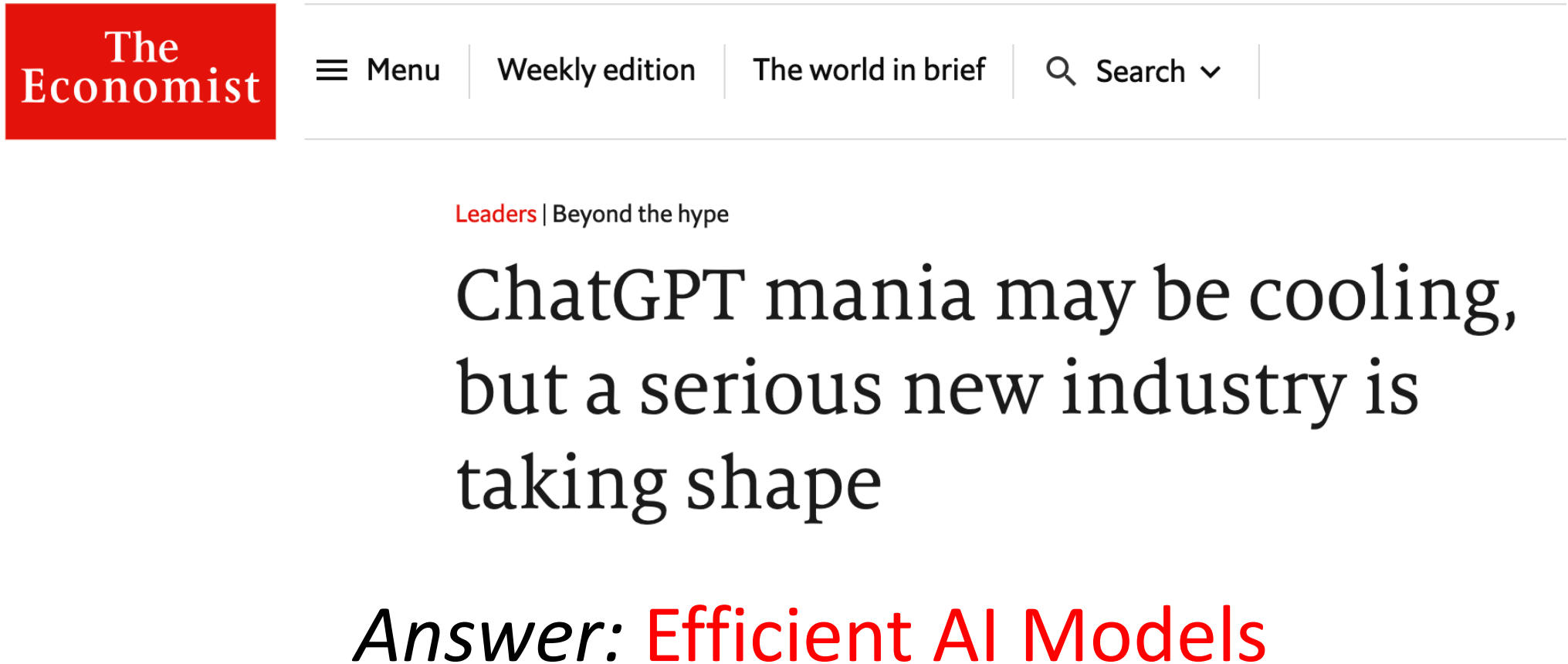
https://developer.nvidia.com/blog/programming-tensor-cores-cuda-9/

ONNX & TensorRT:
1.8GFLOPS, 11.7M Parameters 1 RTX6000 GPU, AMD CPU

ResNet18	Latency μsec	Speed up
CPU	8,550	1
CPU ONNX	3,830	2
GPU	1,982	4
GPU ONNX	1,218	7
GPU ONNX TensorRT	917	9

Both ONNX and TensorRT accelerate inference time

Lesson

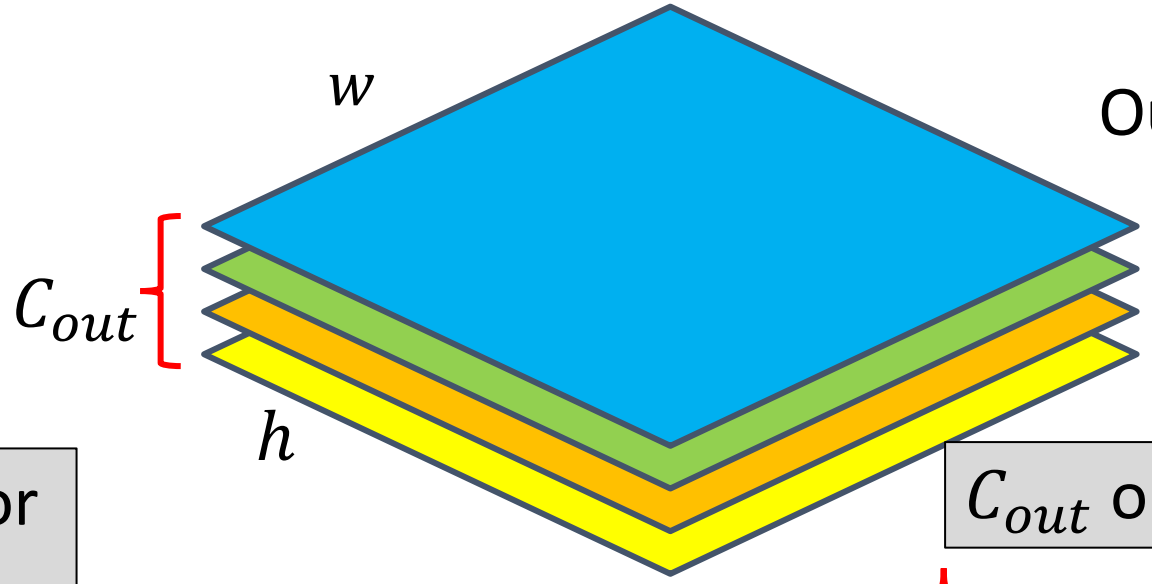


Answer: Efficient AI Models

Model Architecture

Convolutional Neural Network (CNN)

CNN

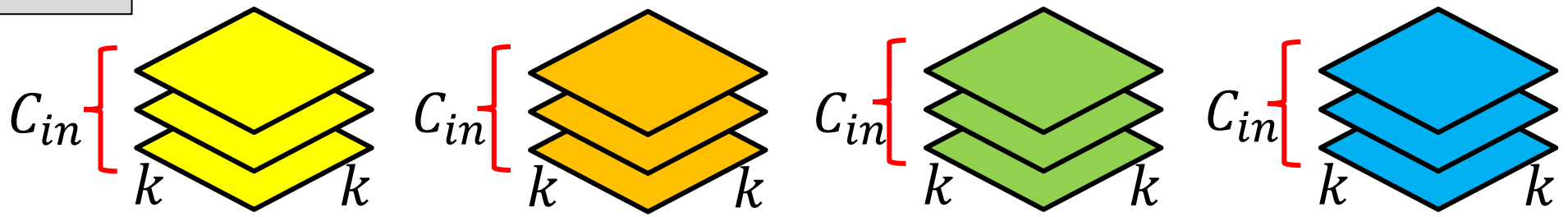


Output feature maps

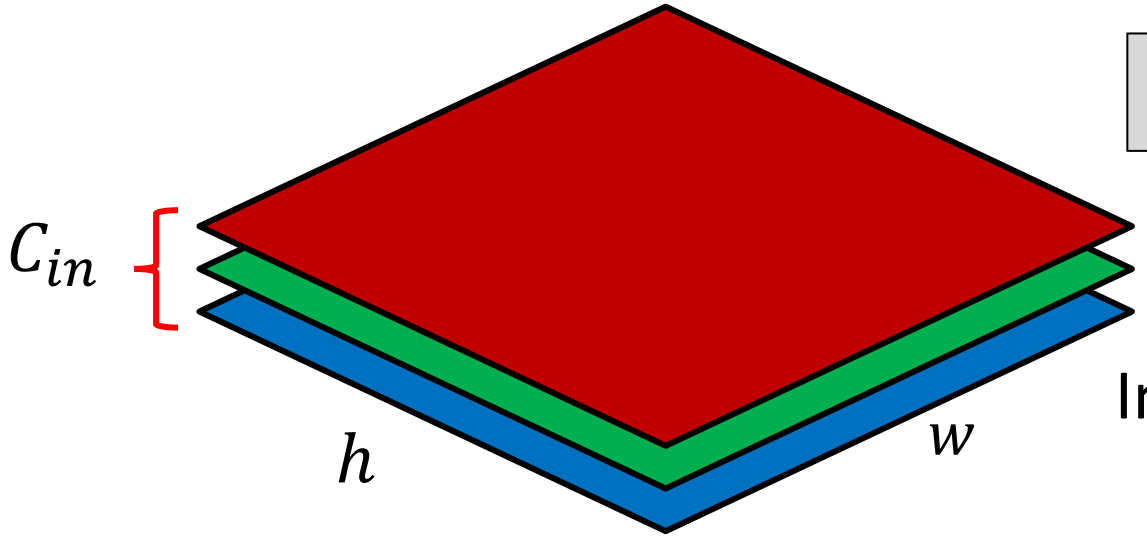
$$output = w \times h \times C_{out}$$

C_{out} output filters or channels

C_{in} input filters or channels



$$kernel = k \times k$$



$$input = w \times h \times C_{in}$$

Input feature maps

convolution with padding

CNN Number of Parameters

$Input\colon\ x \in \mathbb{R}^{H_{in} \times W_{in} \times C_{in}}$

$Kernel\colon k \in \mathbb{R}^{k \times k \times C_{in} \times C_{out}}$

$Output\colon\ y \in \mathbb{R}^{H_{out} \times W_{out} \times C_{out}}$

Number of parameters:

$$N_\theta = \underbrace{k^2 C_{in} C_{out}}_{\text{weights}} + \underbrace{C_{out}}_{\text{bias}}$$

CNN Number of Parameters

Example: 3×3 kernel, 224×224×3 input image, 24 output channels

$$N_\theta = k^2 C_{in} C_{out} + C_{out} = 3 * 3 * 3 * 24 + 24 = 672$$

Multiply and Add Operations

1st Output Channel:

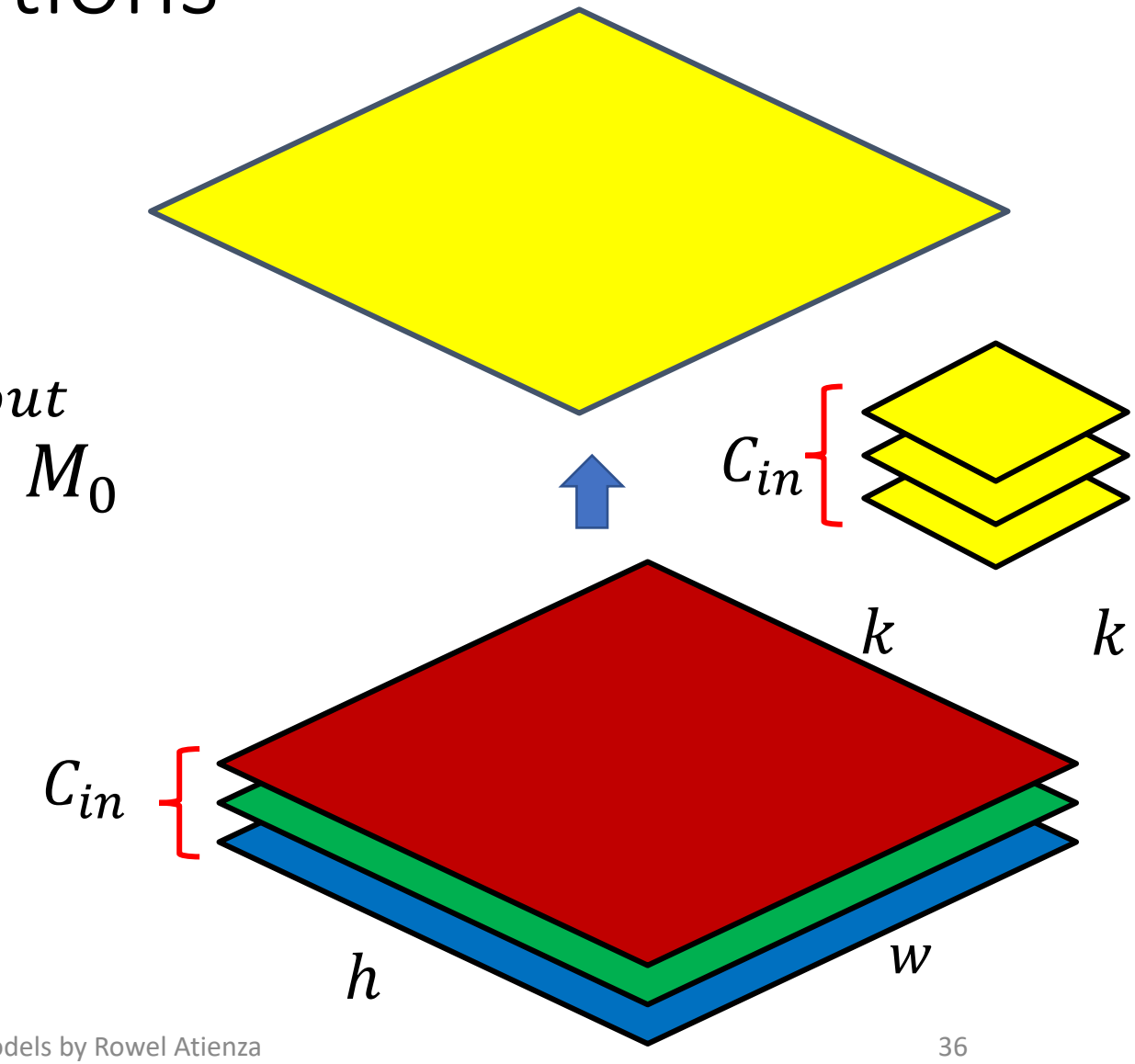
Multiply: $M_0 = k^2 C_{in} W_{out} H_{out}$

Add: $A_0 \approx k^2 C_{in} W_{out} H_{out} = M_0$

All Output Channels:

Multiply: $M = M_0 C_{out}$

Add: $A \approx M$



FMA Instruction – Fused Multiply Add

- ✓Intel and AMD
- ✓NVIDIA CUDA Core
- ✓ARM (Cortex-A5 and Cortex-M4 processors)

$$y = wx + b$$

$$(Single\ FMA\ instruction)$$

FLOPS is a measure of number of FMA

Convolution:

$$FLOPS = FMAs = k^2 C_{in} W_{out} H_{out} C_{out}$$

Example: 3×3 kernel, 224×224×3 input feature, 224×224×24 output feature

$$FLOPS = 3 * 3 * 3 * 224 * 224 * 24 = 32.514M$$

FMA in NVIDIA Tensor Core

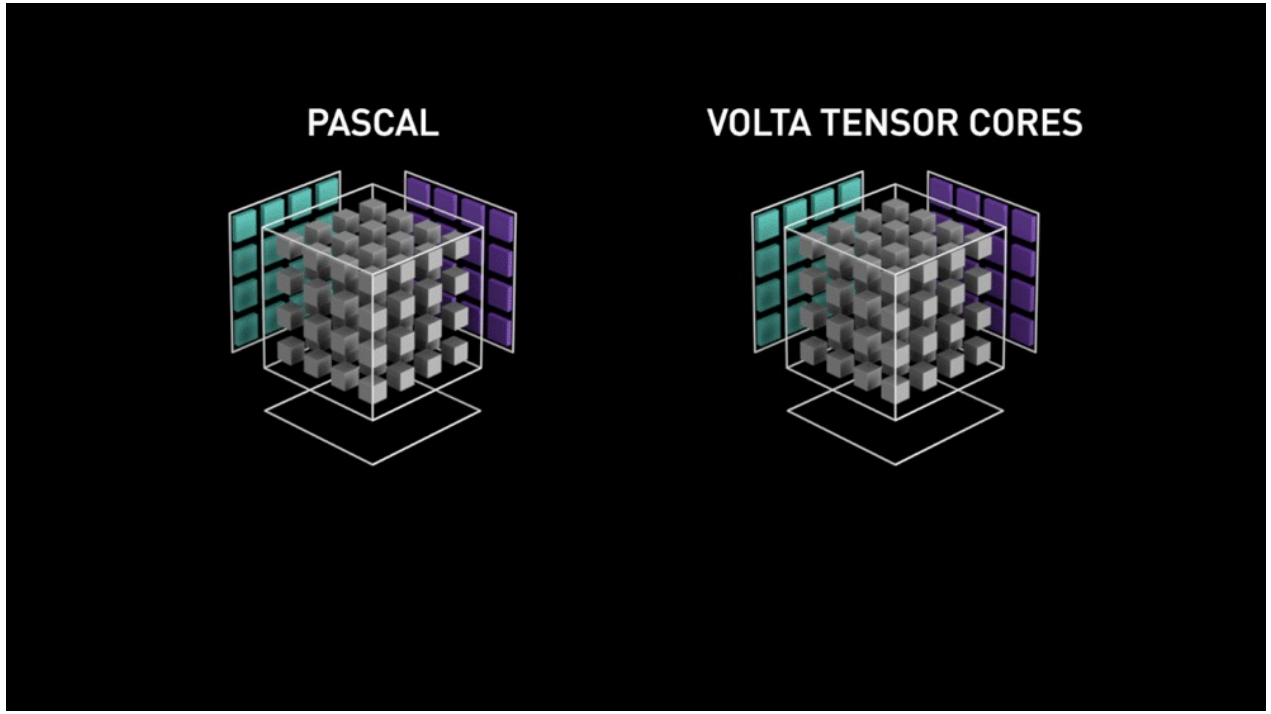
For 1 Tensor Core, only 1 GPU cycle to compute:

$$Y_{4\times4} = W_{4\times4}X_{4\times4} + B_{4\times4}$$

Tensor Core can do 64 FMAs in a GPU cycle

V100 GPU has 640 Tensor Cores or 40,960 FMAs

32.5 MFLOPS can be done in 784 GPU cycles



https://developer.nvidia.com/blog/programming-tensor-cores-cuda-9/

Maximizing Use of Tensor Cores

Satisfy Tensor Core shape constraints

Increase arithmetic intensity

- Depth-wise separable convolutions have lower arithmetic intensity than vanilla convolutions

Decrease fraction of work in non-Tensor Core operations

Satisfying Tensor Core Shape Constraints

Due to their design, Tensor Cores have shape constraints on their inputs.

For matrix multiplication:

- On FP16 inputs, all three dimensions (M, N, K) must be multiples of 8.
- On INT8 inputs (Turing only), all three dimensions must be multiples of 16.

For convolution:

- On FP16 inputs, input and output channels must be multiples of 8.
- On INT8 inputs (Turing only), input and output channels must be multiples of 16.

https://docs.nvidia.com/deeplearning/performance/mixed-precision-training/index.html#tensor-core-shape

FLOPS and Number of Parameters Instrumentation

✓`fvcore` : https://github.com/facebookresearch/fvcore

`thop`: https://github.com/Lyken17/pytorch-OpCounter

`OpenMMLab`: https://github.com/open-mmlab/mmcv/blob/master/mmcv/cnn/utils/flops_counter.py

https://github.com/roatienza/benchmark

`fvcore` – A tool for measuring FLOPS & number of parameters

```
pip install -U fvcore
```

```
class SimpleCNN(nn.Module):
    def __init__(self, in_channel=3, out_channel=24,
                 kernel_size=3, group=False, separable=False, residual=False):
        super(SimpleCNN, self).__init__()
        self.separable = separable
        self.residual = residual
        groups = in_channel if group or separable else 1
        assert out_channel % groups == 0
        if separable:
            self.conv1 = nn.Conv2d(in_channel, in_channel, padding=kernel_size//2, \
                                   kernel_size=kernel_size, groups=groups)
            self.conv2 = nn.Conv2d(in_channel, out_channel, \
                                   kernel_size=1.)
        else:
            self.conv1 = nn.Conv2d(in_channel, out_channel, padding=kernel_size//2, \
                                   kernel_size=kernel_size, groups=groups)
```

`fvcore` FLOPS & number of parameters

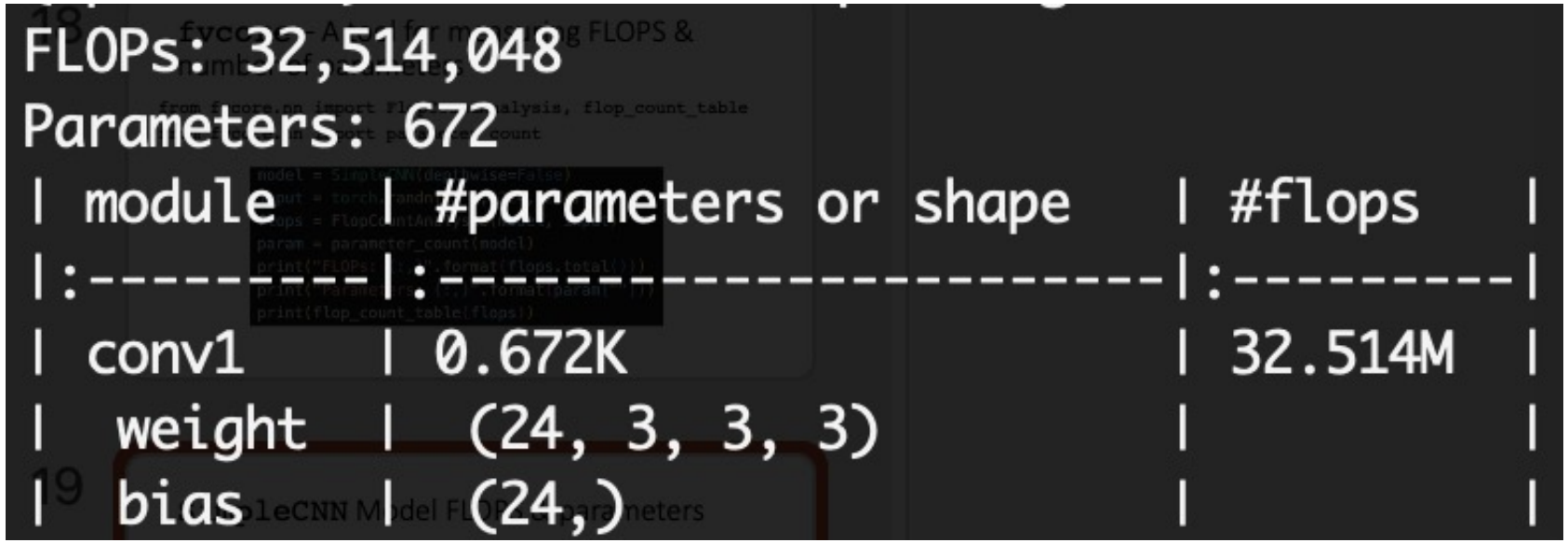
```
from fvcore.nn import FlopCountAnalysis
from fvcore.nn import parameter_count

model = SimpleCNN().to(device)
dummy_input = torch.randn(1, 3, 224, 224).to(device)
flops = FlopCountAnalysis(model, dummy_input)
param = parameter_count(model)
```

SimpleCNN Model FLOPS & parameters

$$FLOPS = k^2 C_{in} W_{out} H_{out} C_{out} = 3 * 3 * 3 * 224 * 224 * 24 = 32{,}514{,}048$$

$$N_\theta = k^2 C_{in} C_{out} + C_{out} = 3 * 3 * 3 * 24 + 24 = 672$$



```
FLOPs: 32,514,048
Parameters: 672
| module    | #parameters or shape   | #flops    |
|:----------|:-----------------------|:----------|
| conv1     | 0.672K                 | 32.514M   |
|  weight   |  (24, 3, 3, 3)         |           |
|  bias     |  (24,)                 |           |
```

Using benchmark

```
python3 benchmark.py --verbose
```

Scaling Convolution – How to make it efficient?

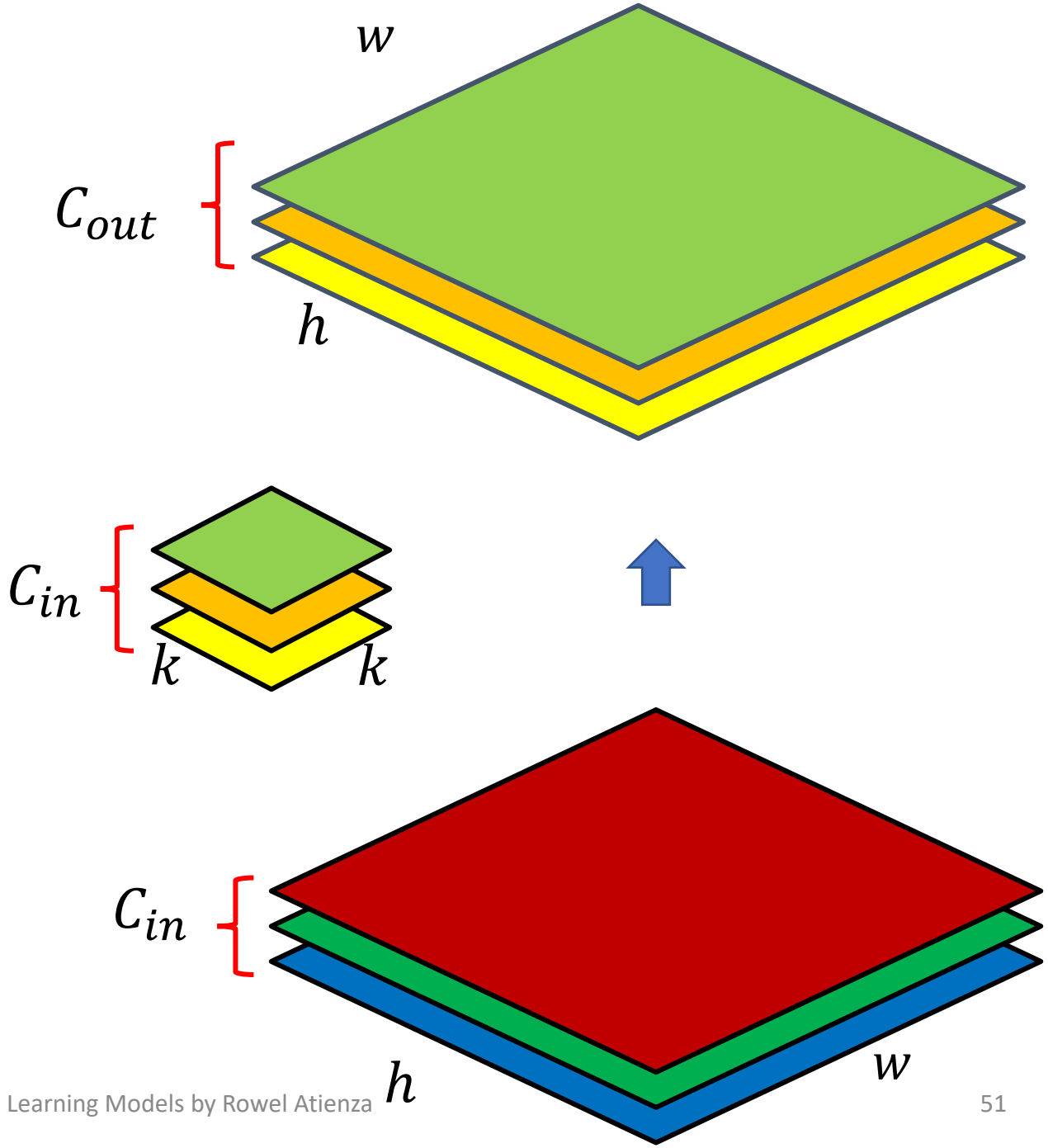
Reducing the number of parameters and FLOPS

Scaling Convolution

Group convolution

Depth-wise separable convolution

Group Convolution [ResNext, ShuffleNet]



Standard vs Group Convolution

Type	Number of Parameters, N_θ	FLOPS
Standard	$k^2 C_{in} C_{out} + C_{out}$	$k^2 C_{in} W_{out} H_{out} C_{out}$
Group	$k^2 C_{out} + C_{out}$	$k^2 W_{out} H_{out} C_{out}$

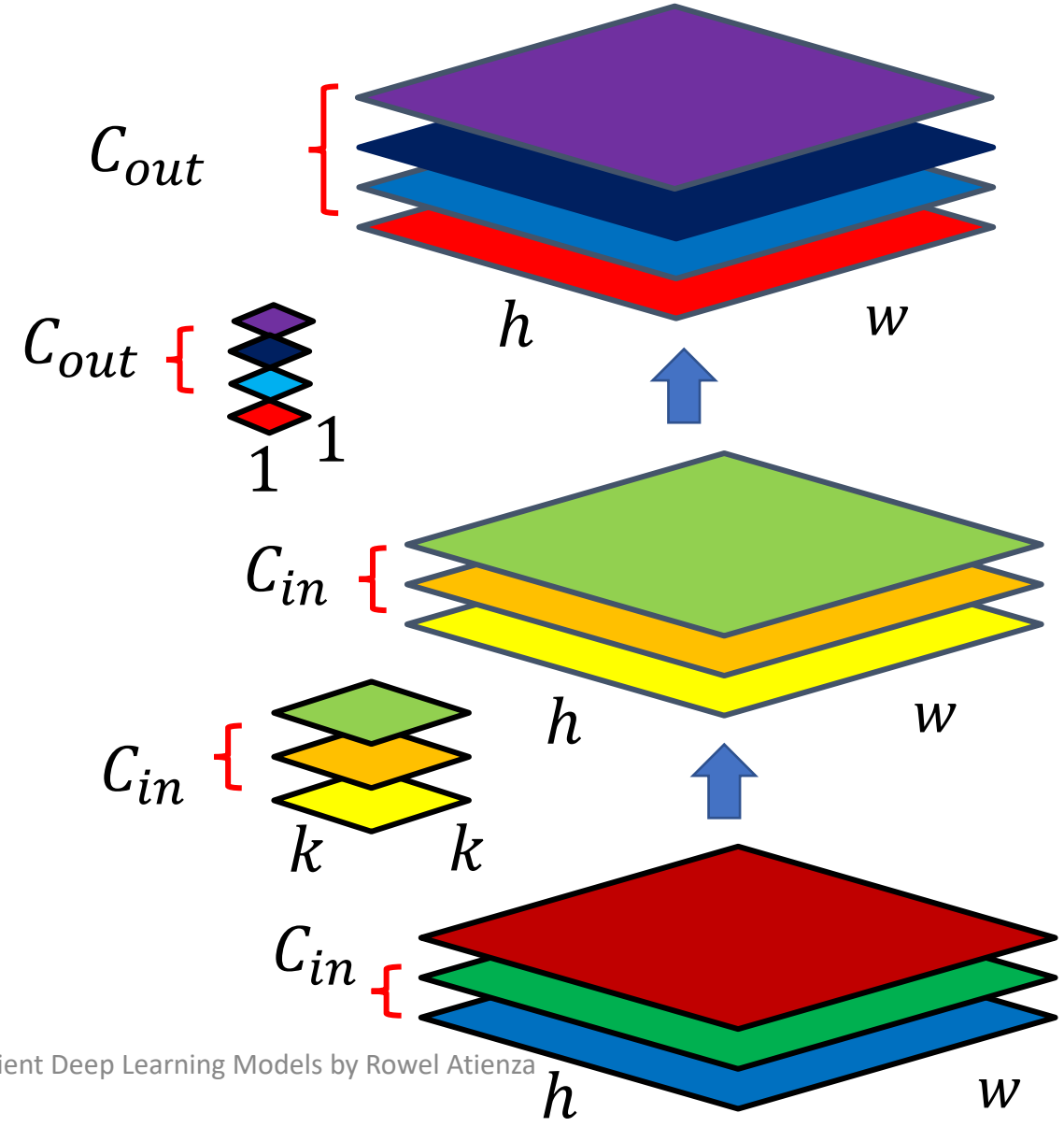
Group convolution reduces the number of parameters and FLOPS by C_{in}

`SimpleCNN` Standard vs Group Convolution

Type	Number of Parameters, N_θ	FLOPS
Standard `SimpleCNN(group=False)`	672	32.5M
Group `SimpleCNN(group=True)`	240	10.8M

Group convolution reduces the number of parameters and FLOPS by $C_{in} = 3$

Depthwise Separable Convolution [MobileNet]



Standard vs Group vs Depthwise Convolution

Type	Number of Parameters, N_θ	FLOPS
Standard	$k^2 C_{in} C_{out} + C_{out}$	$k^2 C_{in} W_{out} H_{out} C_{out}$
Group	$k^2 C_{out} + C_{out}$	$k^2 W_{out} H_{out} C_{out}$
Depthwise	$k^2 C_{in} + C_{in} + C_{in} C_{out} + C_{out}$	$k^2 W_{out} H_{out} C_{in} + C_{in} W_{out} H_{out} C_{out}$

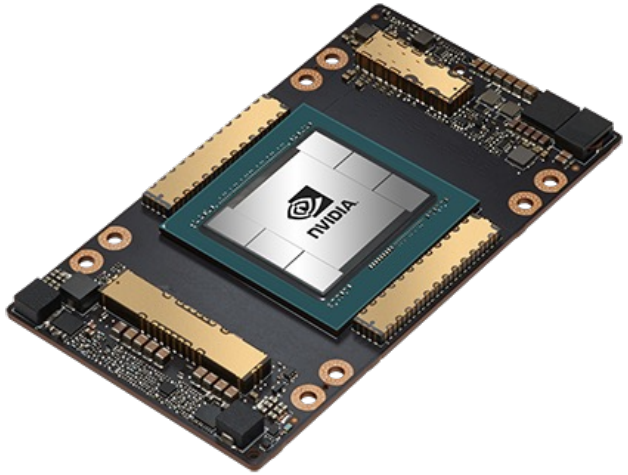
`SimpleCNN` Standard vs Depthwise Convolution vs Depthwise Separable

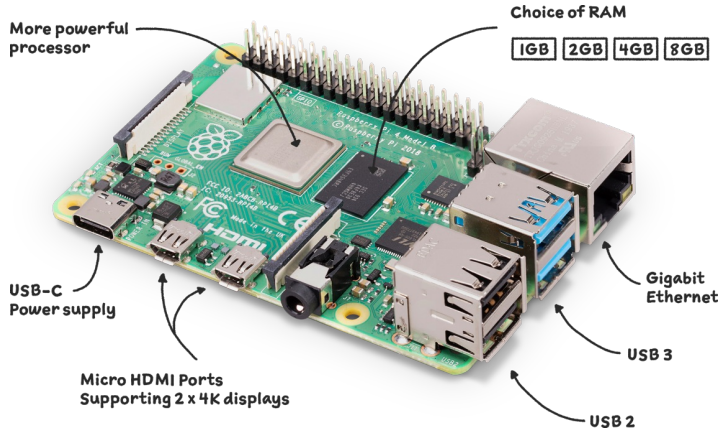
Type	Parameters, N_θ	FLOPS (M)
Standard `SimpleCNN(group=False)`	672	32.5
Group `SimpleCNN(group=True)`	240	10.8
Depthwise Separable `SimpleCNN(separable=True)`	<u>126</u>	<u>5.0</u>

Group conv reduction by 3, Depthwise separable reduction by 6,

Latency

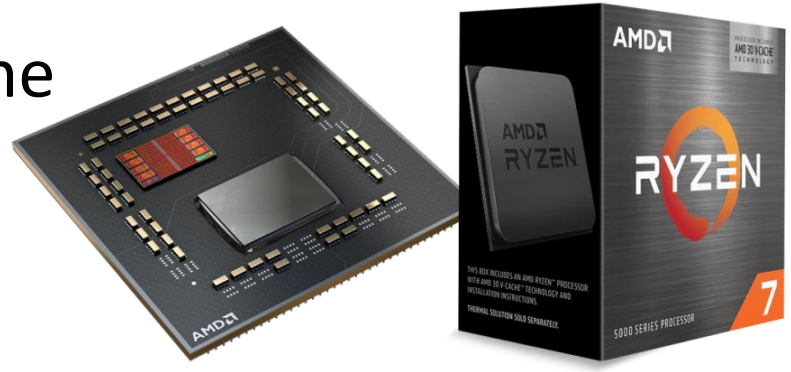
Model Inference Latency





$$GPU_{latency} < CPU_{latency}$$

Generally but not all the time



nvidia.com, amd.com, raspberrypi.com

Factors affecting latency

- ✓Graph Parallelization
  - Neural network architecture
  - GPU, CPU cores
- Special tensor-level operation instruction sets
  - ✓Operators fusion
- ✓Memory access cost (MAC)
  - Memory cache
  - Speed of interconnect (PCIe vs NVLink)

Graph parallelization

CNN

- Kernel level dot product

Transformer/MLP

- Tensor multiplication and addition

~~RNN~~

Memory Access Cost (MAC) vs Speed

- Reduce memory access cost, increase the speed [MobileNetV2]
- For the same FLOPS count, when $\frac{C_{in}}{C_{out}} \rightarrow 1$, the inference speed is optimal [ShuffleNetV2]

Element-wise Operations are MAC-heavy though FLOPS-light

Element-wise ops: ReLU, Add Bias [ShuffleNetV2]

Activation Function Latency [MobileOne]

Activation Function	Latency (ms)
ReLU [42]	1.53
GELU [43]	1.63
SE-ReLU [38]	2.10
SiLU [44]	2.54
Dynamic Shift-Max [40]	57.04
DynamicReLU-A [41]	273.49
DynamicReLU-B [41]	242.14

Table 3: Comparison of latency on mobile device of different activation functions in a 30-layer convolutional neural network.

Other latency factors

Skip connection speeds up training but is MAC heavy

- Remove skips altogether (RepVGG 2021)

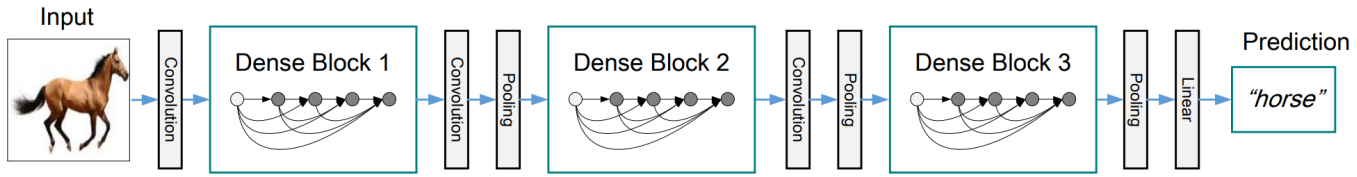


Figure 2: A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature-map sizes via convolution and pooling.

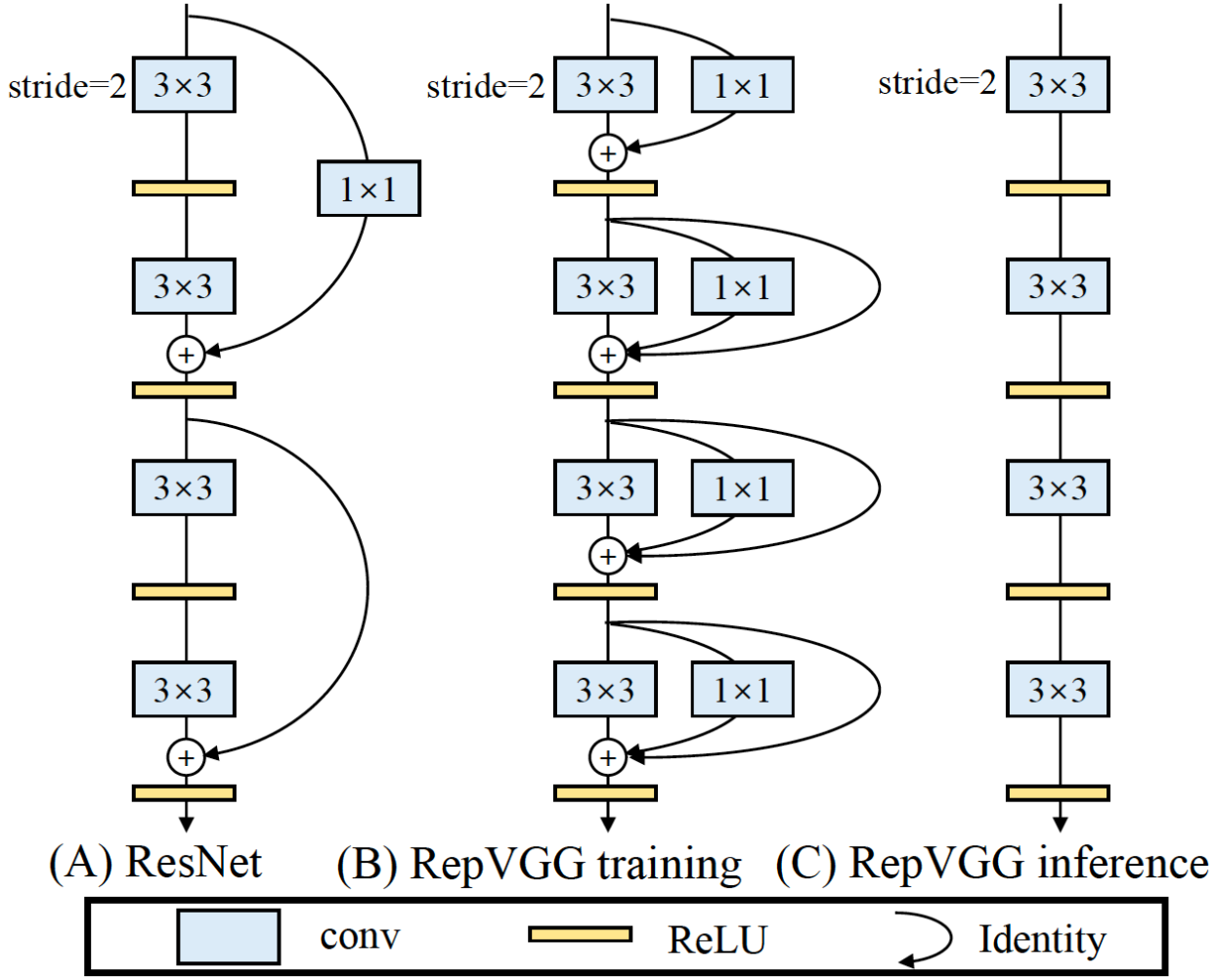
Multi-branch architectures increases MAC significantly [RepVGG]

Delay due to waiting/sync

- Could use Squeeze/Excitation Net to force sync
- Fold/fuse batch norm into conv [RepVGG, MobileOne]

RepConv by Reparameterization

Combines 3x3, 1x1 and residual into 1 [RepVGG] using *Operator Folding*.



32-bit fp vs 16-bit fp

- 16-bit requires half memory access thus faster
- Tensor cores only support 16-bit mult (16/32-bit add)
- Use of AMP (Automatic Mixed Precision) training

Other speed up techniques

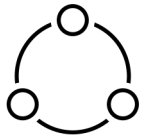
Pruning

Quantization

De Leon, Josen Daniel, and Rowel Atienza. "Depth Pruning with Auxiliary Networks for Tinyml." *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2022.

Inference Latency Instrumentation

Algorithm

1. Model in evaluation mode
2. Model and data in AI Accelerator (GPU or CPU)
3. Disable gradients
4. GPU warmup
5. Start and end event timers – Loop 

Inference Latency Measurement

```
# log elapsed times
with torch.no_grad():
    for rep in range(repetitions):
        start_time = time.time()
        if onnx_model is not None:
            y = session.run(None, dummy_input)
            y = y[0]
        else:
            y = torch_model(dummy_input)
            y = y.detach().cpu().numpy()
        timings[rep] = (time.time() - start_time)  * 1e6

mean_syn = np.sum(timings) / repetitions
std_syn = np.std(timings)
print("(CPU Timer) Ave infer time: {0:.1f} usec".format(mean_syn))
print("(CPU Timer) Std infer time: {0:.1f} usec".format(std_syn))
```

`SimpleCNN` Efficiency

Type	Parameters N_θ	FLOPS (M)	GPU Latency μsec	CPU Latency μsec
Standard	672	32.5	80.0	315
Group	240	10.8	<u>58.8</u>	<u>170</u>
Depthwise Separable	<u>126</u>	<u>5.0</u>	94.0	440

Small number of parameters and FLOPS does not directly translate to speed

`SimpleCNN` Standard vs Group Convolution vs Depthwise Separable with Skip

Type	Parameters N_θ	FLOPS (M)	Latency μsec	Latency (Res) μsec
Standard	672	32.5	80.0	100.5
Group	240	10.8	58.8	80.2
Depthwise Separable	126	5.0	94.0	109.6

Residual connection adds latency of about 20usec

What about Neural Architectural Search (NAS)

Search is very expensive

No guarantee that the resulting network is hardware friendly

Transformers

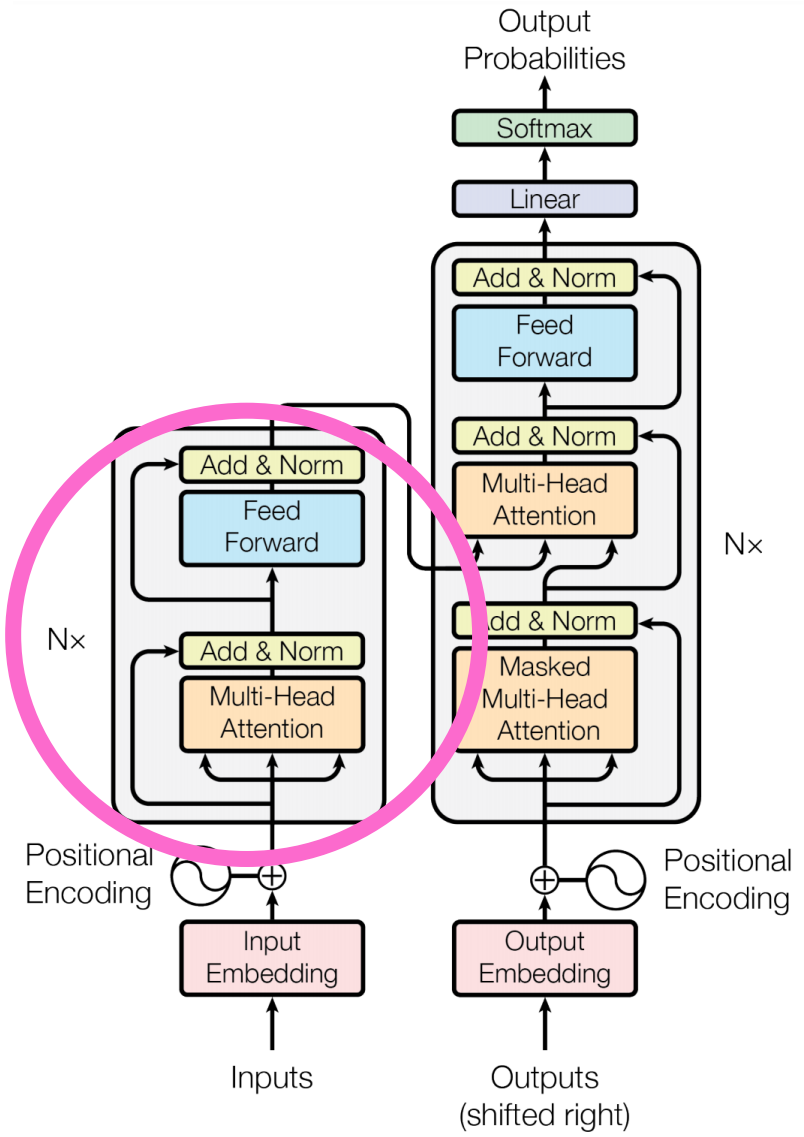
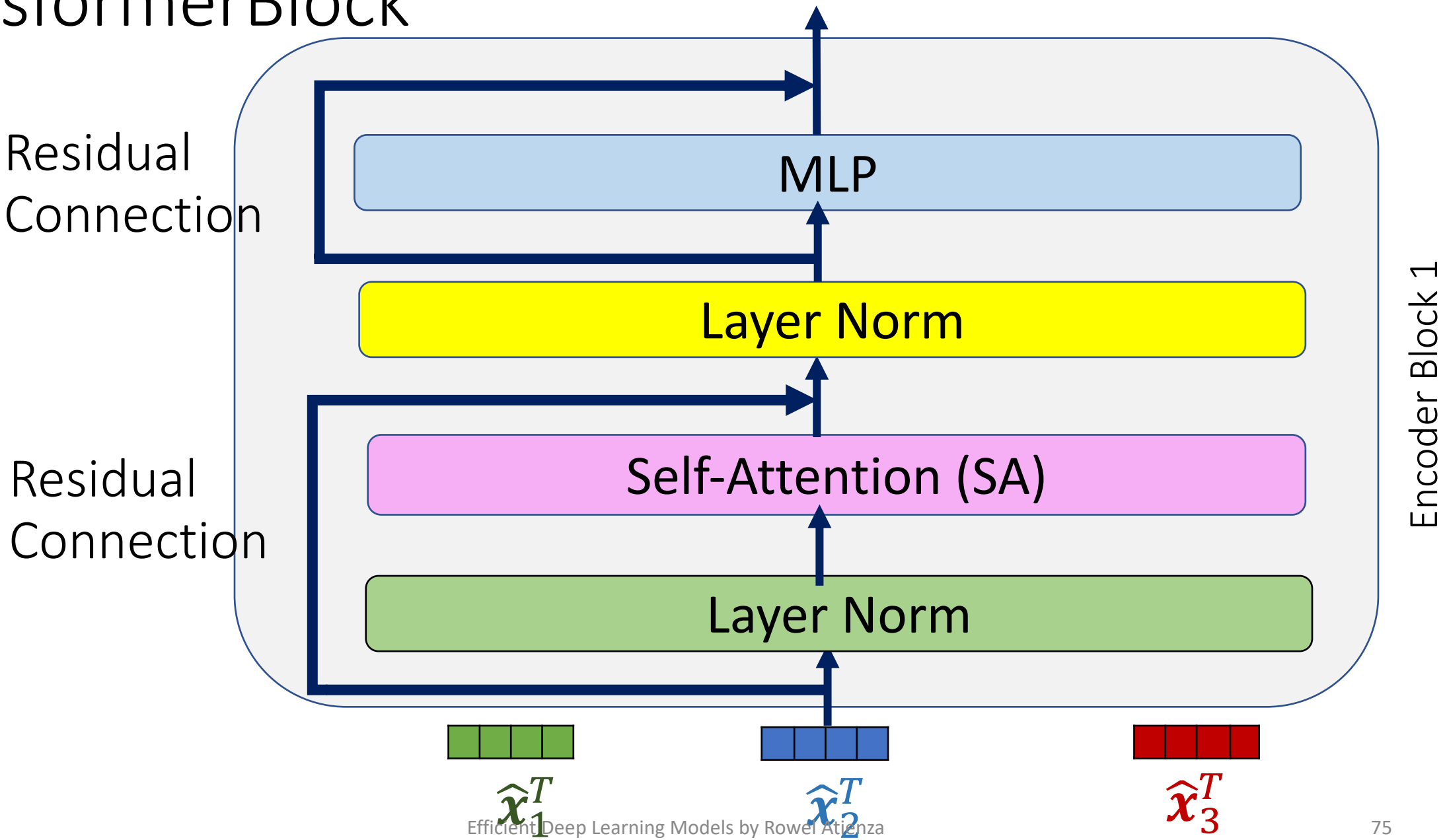


Figure 1: The Transformer - model architecture.

TransformerBlock



$Query$ Key $Value$

$$z = Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d}}\right)V$$

$$Q, K, V \in \mathbb{R}^{N \times d}$$

$$Q = XW^Q, K = XW^K, V = XW^V, W^* \in \mathbb{R}^{d \times d}$$

N is the sequence length (e.g. 196)

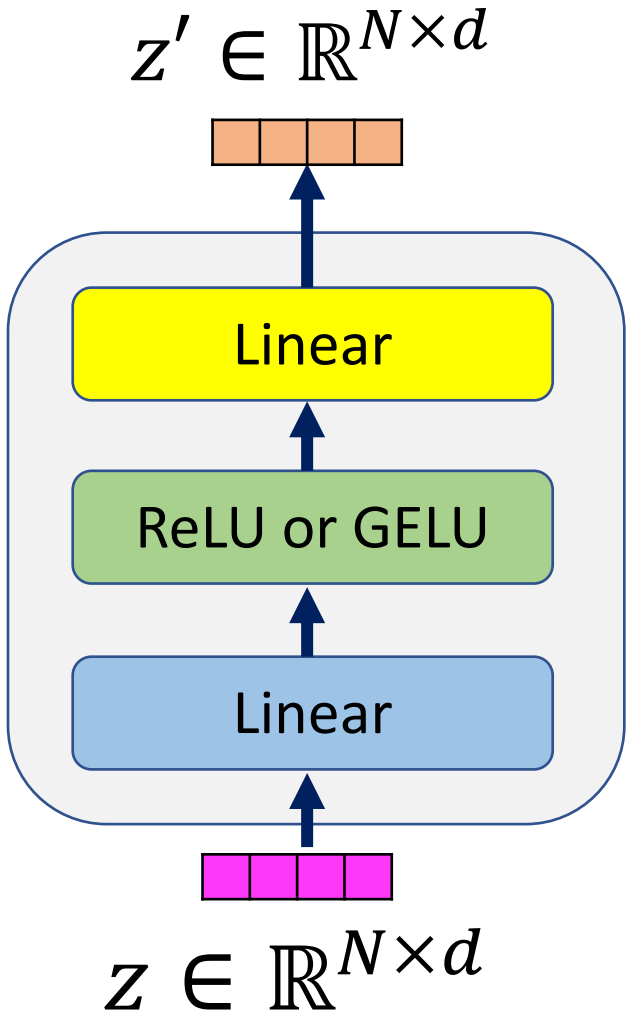
d is the feature dim (e.g. 192)

$$FLOPS = 2N^2d + 3Nd^2$$

$$Parameters = 3d^2$$

Note: `fvcore` does not include $2N^2d$ in FLOPS

MLP



N is the sequence length (e.g. 196)

d is the feature dim (e.g. 192)

$$FLOPS = 2Nd^2$$

$$Parameters = 2d^2$$

Note: all biases not included in the parameter count estimate

TransformerBlock

$$Parameters = 3d^2 + 2d^2 = 5 * 192 * 192 = 185k$$

$$FLOPS = 3Nd^2 + 2Nd^2 = 5 * 196 * 192 * 192 = 36M$$

Type	Param $N_\theta(M)$	FLOPS (M)	Latency μsec	Latency, μsec	
				SA	MLP
Standard	0.19	36	240	172	88

Note: Linear projection in the self-attention module not included for simplicity

TransformerBlock

Type	Param $N_\theta(M)$	FLOPS (M)	Theoretical FLOPS (M)	Latency μsec
Standard	0.19	36	50.8	240
$^1/_4$ Sequence Len	0.19	9	9.9	210
$^1/_2$ feature dim	0.05	9	16.4	220

Transformer is inherently slow even under reduction in seqlen or feature dim

Transformers

Reducing sequence length or feature dimension reduces FLOPS

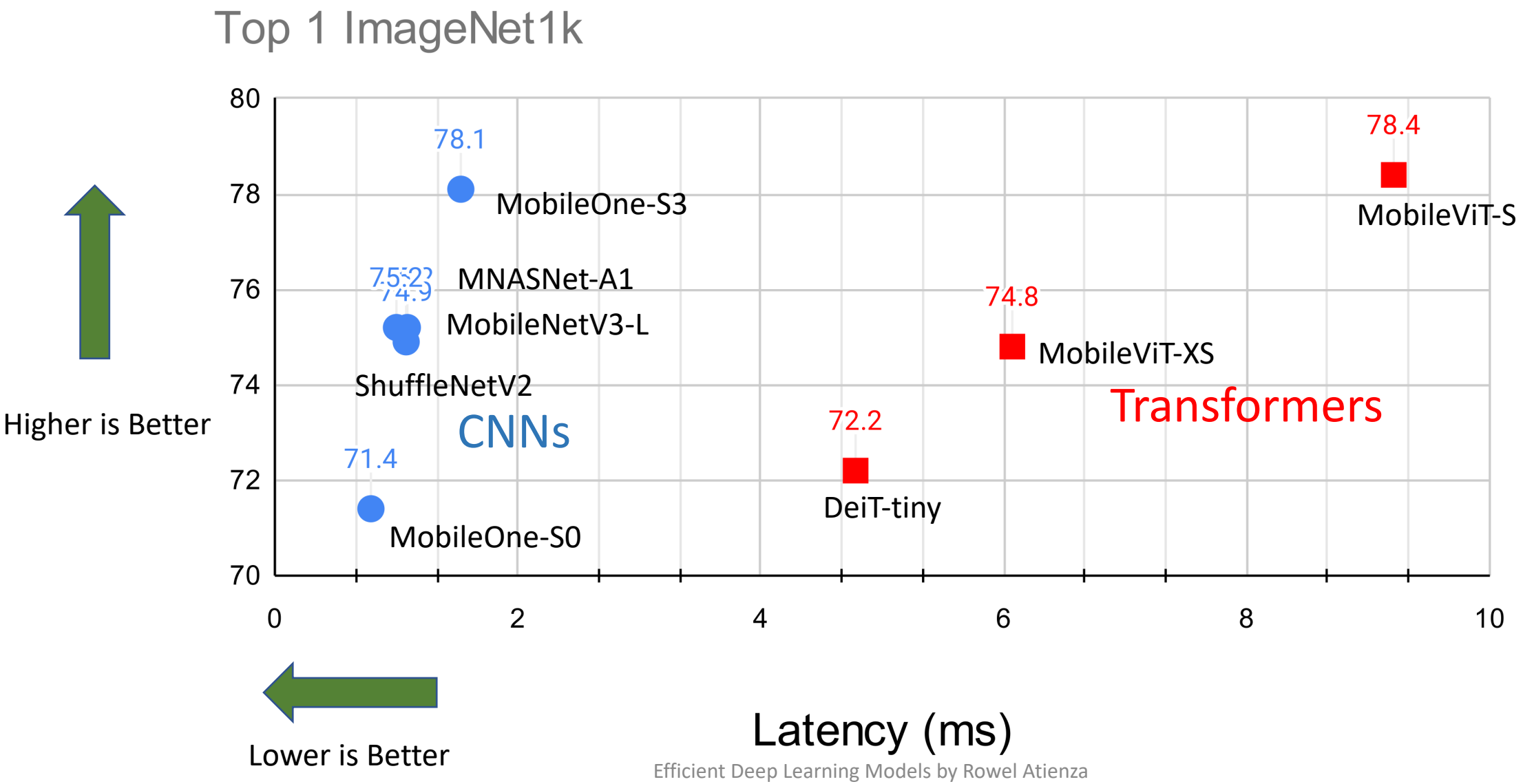
Reducing feature dimension reduces parameter count

How to reduce latency?

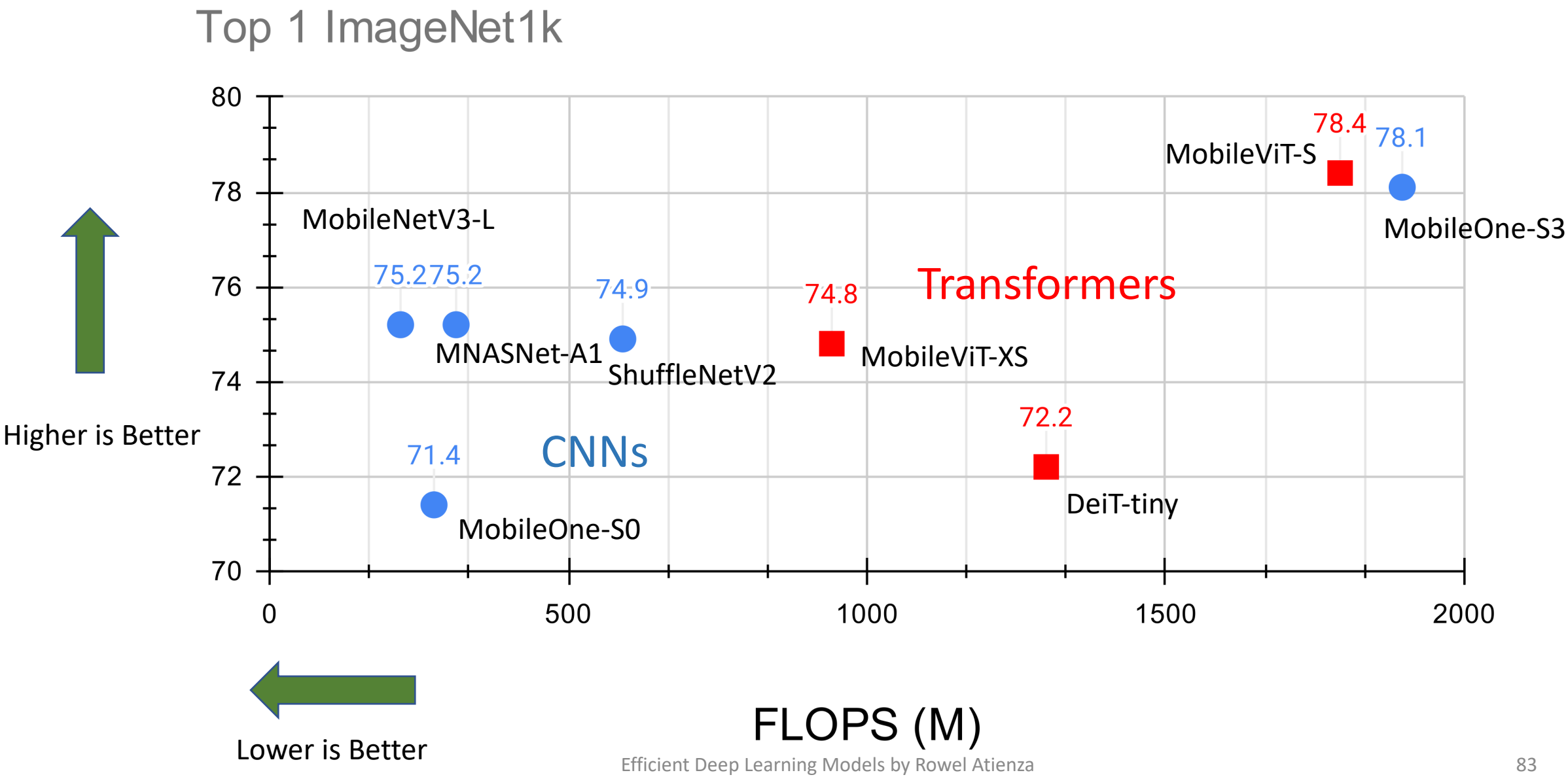
CNN vs Transformer Models

Resource vs ImageNet1k Accuracy on Efficient Models Regime

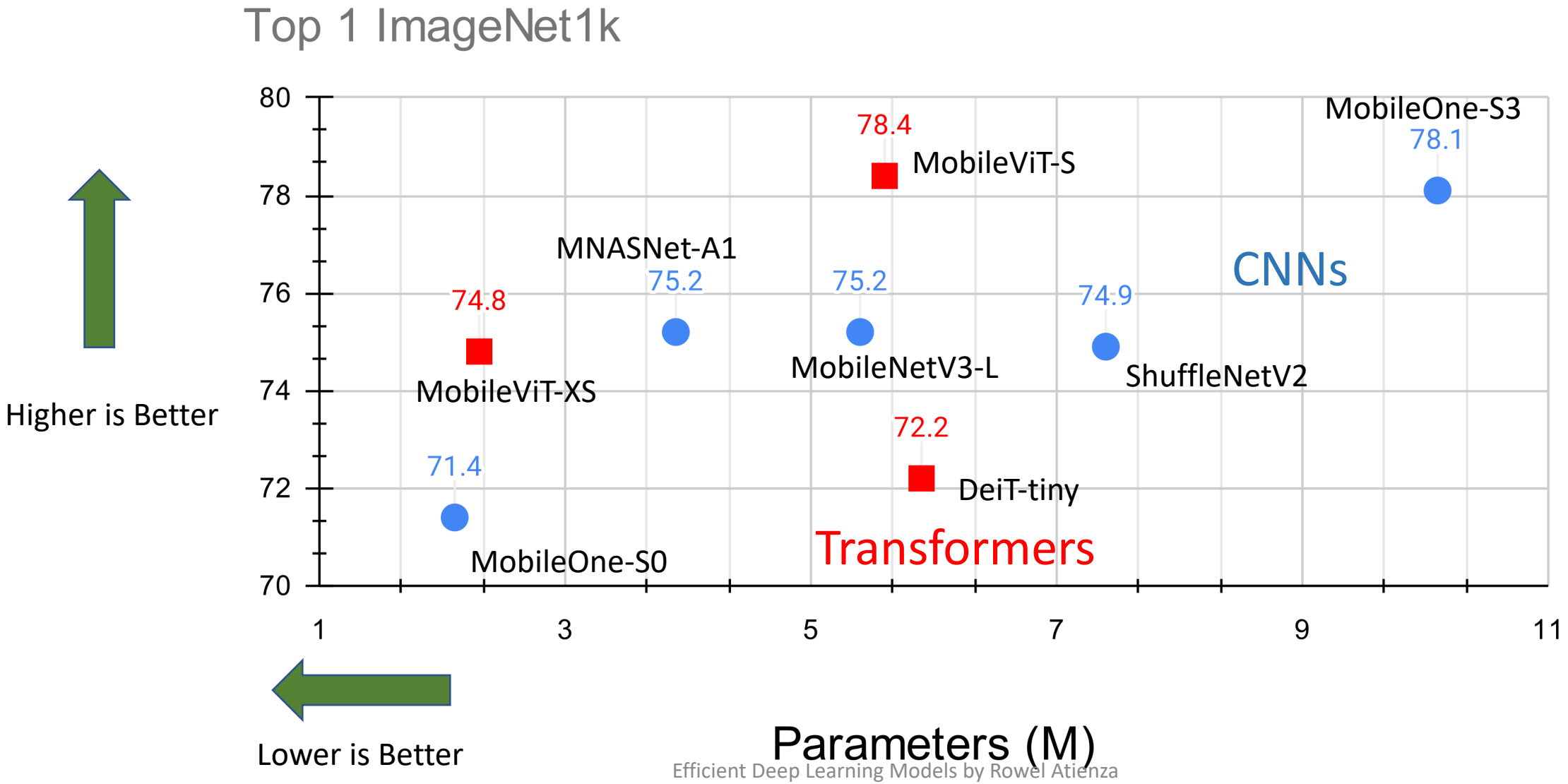
Transformers Slower than CNNs



Transformers More Computer Intensive than CNNs



Transformers more Parameter Efficient than CNNs



Model Format for Latency Reduction

ONNX and TensorRT

Open Neural Network Exchange

Model format for framework interoperability

- PyTorch to ONNX: `torch.onnx.export()`

Graph optimization

Operator/constant/node folding

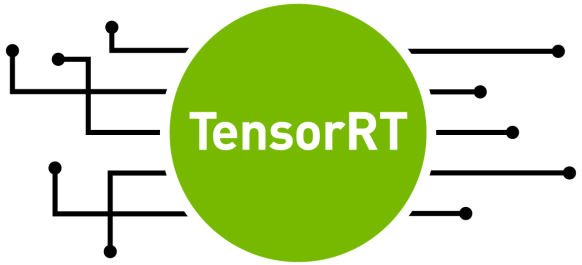
Redundant node elimination

Activation function approximation (eg GELU)

Data layout optimization

https://onnxruntime.ai/docs/performance/graph-optimizations.html

TensorRT



Optimizes model execution on CUDA and Tensor cores

Layer and tensor fusions

Kernel autotuning

Dynamic tensor memory

A100 has support for sparse tensor cores

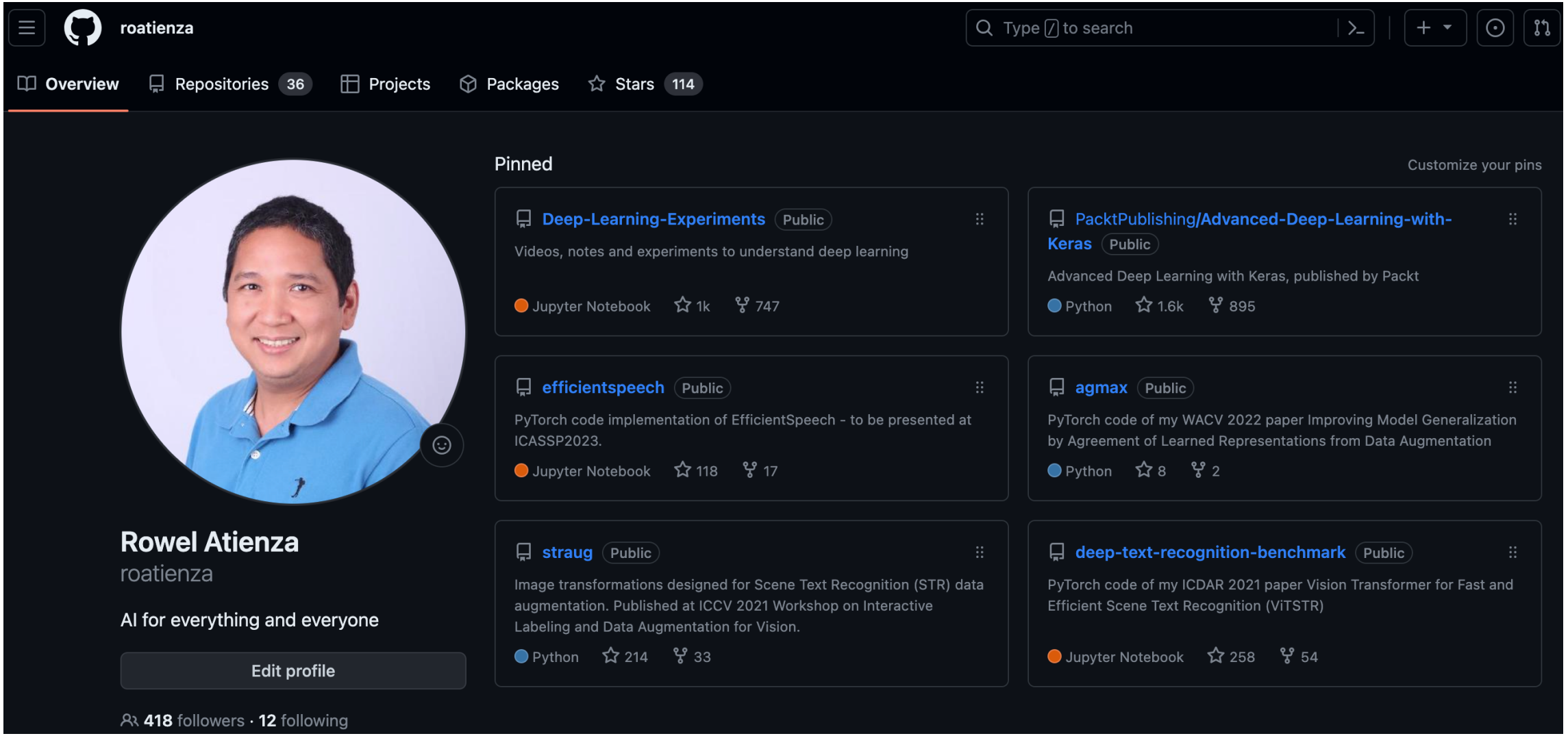
Mixed precision inference

ONNX & TensorRT:
1.8GFLOPS, 11.7M Parameters 1 RTX6000 GPU, AMD CPU

ResNet18	Latency μsec	Speed up
CPU	8,550	1
CPU ONNX	3,830	2
GPU	1,982	4
GPU ONNX	1,218	7
GPU ONNX TensorRT	917	9

Both ONNX and TensorRT accelerate inference time

Thank you!


roatienza
Type / to search
Overview
Repositories 36
Projects
Packages
Stars 114
Pinned
Customize your pins
Deep-Learning-Experiments Public
Videos, notes and experiments to understand deep learning
Jupyter Notebook 1k 747
PacktPublishing/Advanced-Deep-Learning-with-Keras Public
Advanced Deep Learning with Keras, published by Packt
Python 1.6k 895
efficientspeech Public
PyTorch code implementation of EfficientSpeech - to be presented at ICASSP2023.
Jupyter Notebook 118 17
agmax Public
PyTorch code of my WACV 2022 paper Improving Model Generalization by Agreement of Learned Representations from Data Augmentation
Python 8 2
straug Public
Image transformations designed for Scene Text Recognition (STR) data augmentation. Published at ICCV 2021 Workshop on Interactive Labeling and Data Augmentation for Vision.
Python 214 33
deep-text-recognition-benchmark Public
PyTorch code of my ICDAR 2021 paper Vision Transformer for Fast and Efficient Scene Text Recognition (ViTSTR)
Jupyter Notebook 258 54
Rowel Atienza
roatienza
AI for everything and everyone
Edit profile
418 followers · 12 following