# SparTA: Deep-Learning Model Sparsity via Tensor-with-Sparsity-Attribute

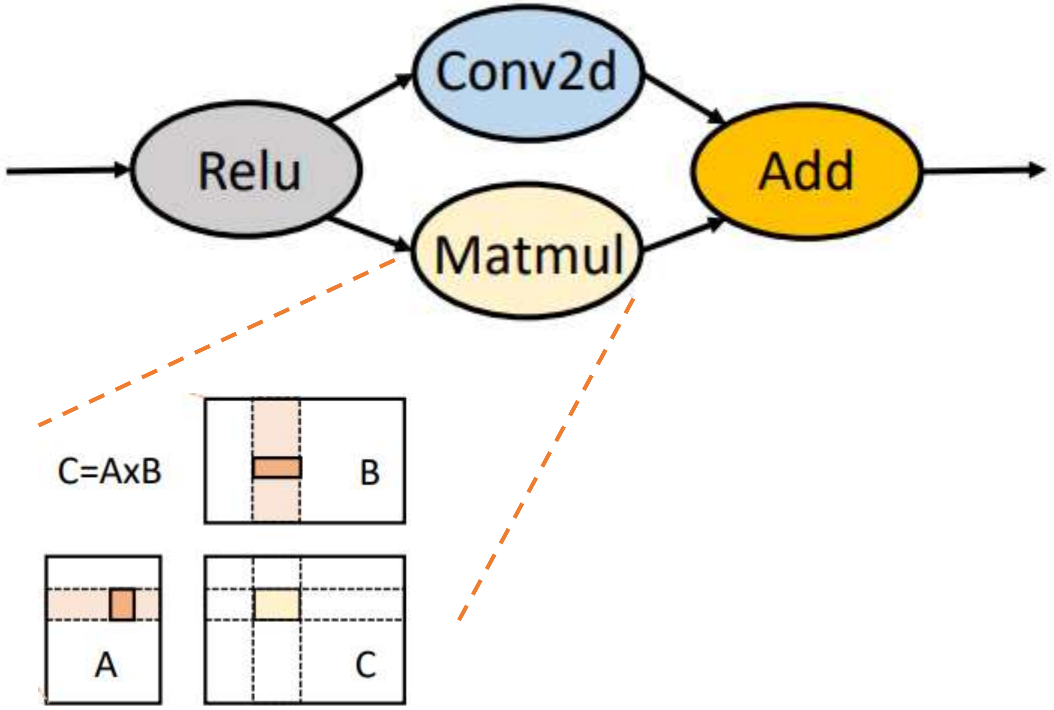
Ningxin Zheng, Microsoft Research Asia; Bin Lin, Tsinghua University; Quanlu Zhang, Lingxiao Ma, Yuqing Yang, Fan Yang, Yang Wang, Mao Yang, and Lidong Zhou, Microsoft Research Asia

*Presented by Guanbin Xu*
*@USTC_ADSL_ReadingGroup*

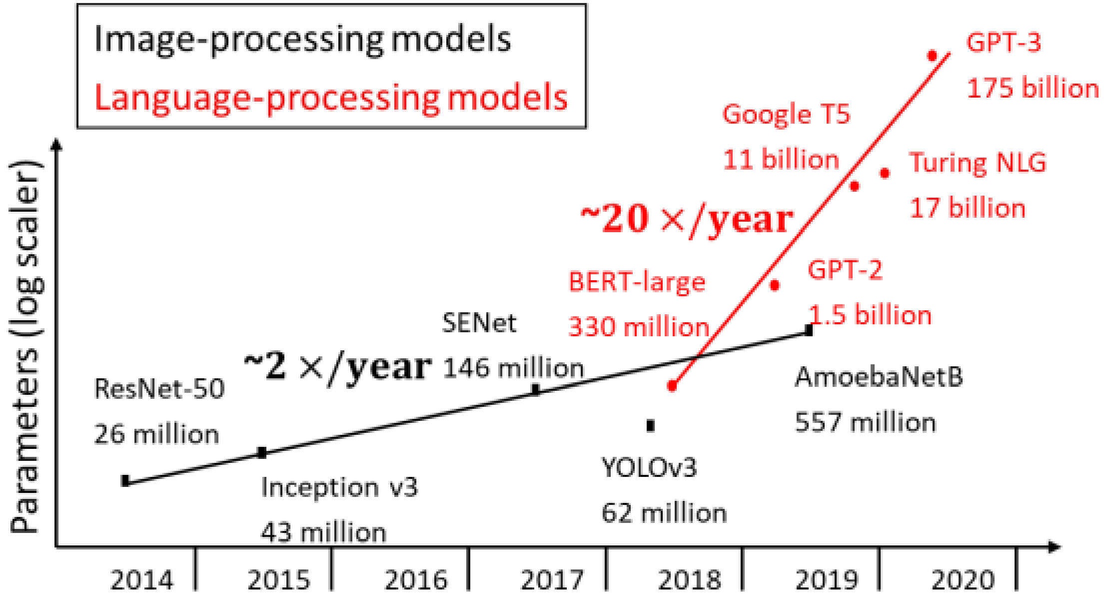
# Outline

- Motivation
  - DNN models become large and complex
  - Various forms of sparsity
  - The myth of FLOPS
  - The diminishing end2end return
  - Across-stack sparsity innovations in silos
- Goals
- Design
- Evaluation
- Related works

# DNN models become large and complex




Dataflow Graph for DNN model



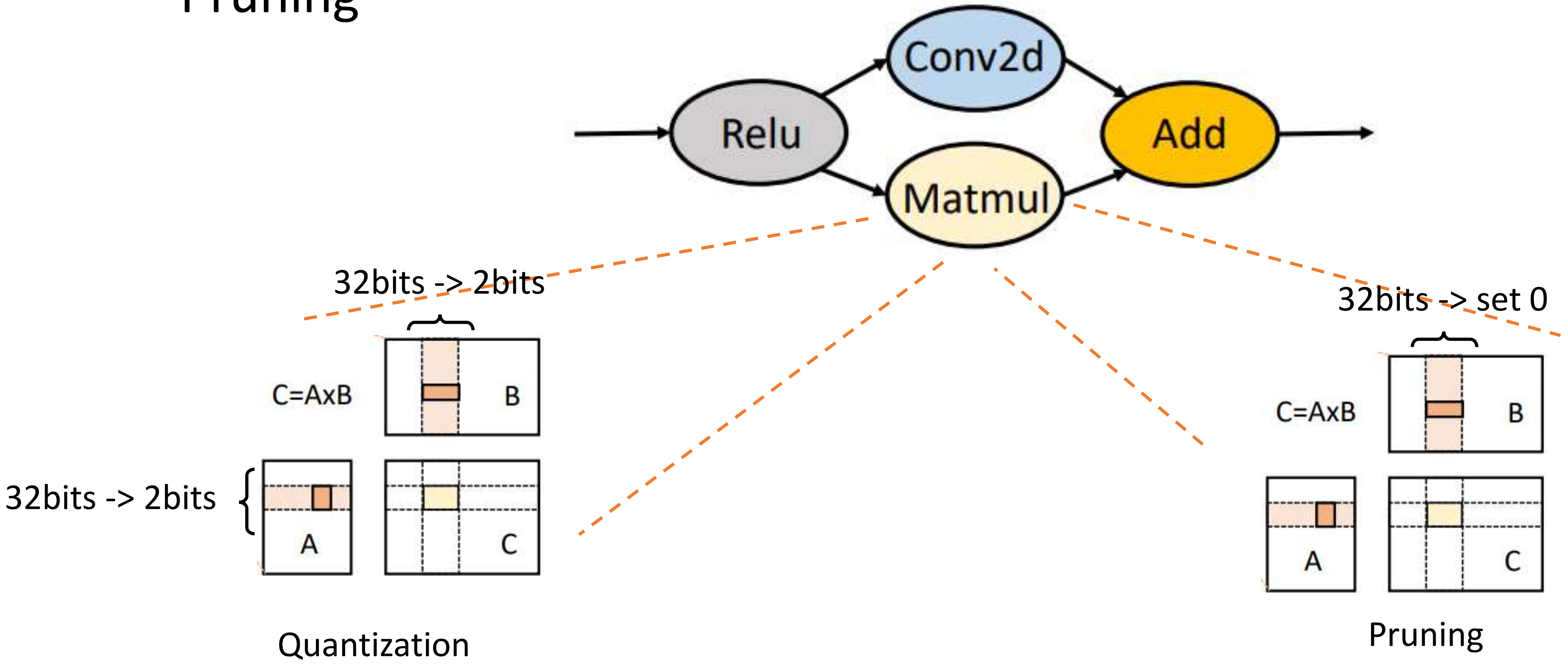

Model size trends [1]

[1] https://openai.com/blog/ai-and-compute/

**How to reduce inference latency?**

# Various forms of sparsity

- Quantization
- Pruning



# Various forms of sparsity (Cont.)

- Quantization
  - Binarized models[20], 8-bit models[33, 68]
  - Mixed precision[24, 38, 55, 47, 62]
- Pruning
  - Fine-grained[29, 35, 36]
  - Block sparsity[37, 40, 42, 44]
- Combination with quantization and pruning[28, 53, 54, 57, 61, 66]

# Various forms of sparsity (Cont.)

- Quantization
  - Binarized models[20], 8-bit models[33, 68]
  - Mixed precision[24, 38, 55, 47, 62]
- Pruning

**Active & Extensively!**

  - Fine-grained[29, 35, 36]
  - Block sparsity[37, 40, 42, 44]
- Combination with quantization and pruning[28, 53, 54, 57, 61, 66]

# Various forms of sparsity (Cont.)

- Quantization
  - Binarized models[20], 8-bit models[33, 68]
  - Mixed precision[24, 38, 55, 47, 62]
- Pruning

**Active & Extensively!**

  - Fine-grained[29, 35, 36]
  - Block sparsity[37, 40, 42, 44]
- Combination with quantization and pruning[28, 53, 54, 57, 61, 66]

**DNN operators customized for the sparsity patterns, the resulting model will, *hopefully*, come with a lower inference latency.**

# The myth of FLOPS

***Unfortunately***, **model sparsity does not translate directly into performance benefits.**

# The myth of FLOPS (Cont.)

We use the prediction accuracy of several CNN models on SVHN dataset to evaluate the efficacy of configurations. Model A is a CNN that costs about 80 FLOPs for one 40x40 image, and it consists of seven convolutional layers and one fully-connected layer.

ameters, $T^{\mathrm{frm}}_{\mathrm{calc}} \sim \frac{M}{\mathrm{FLOPS}}$, $T_{\mathrm{calc}}$ and $T_{\mathrm{calc}}$ $\sim \frac{\ }{r}$ with RAM bandwidth $r$, and $T^{\mathrm{float}}_{\mathrm{comm}} \sim$ $M$ with peer to peer bandwidth $b$. Importantly, unlike $T^{\mathrm{frm}}$

By only requiring 1/4 number of the FLOPS they still manage to achieve a 2.7% increase in accuracy for MobileNet-V1. This also corresponds to a 1.53 times speed up on a Titan Xp GPU and 1.95 times

From the left of Figure 1, we see that in general, larger overparameterized CNN networks generalize better for ImageNet (a large image classification benchmark dataset). However, recent architectures that aim to reduce the number of floating point operations (FLOPs) and improve training efficiency with less parameters have also shown impressive performance e.g EfficientNet [173].

ation of compression techniques is performance metric (e.g accuracy) vs model size. When evaluating for speedups obtained from the model compression, the number of floating point operations (FLOPs) is a commonly used metric. When claims of storage improvements are made, this can be demonstrated by reporting the run-time memory footprint which is essentially the ratio of the space for storing hidden layer features during run time when compared to the original network.

**The proxy metric(FLOP per sec) is flawed and leads to inaccurate results!**

# The myth of FLOPS

**Table 1.** Speed of matrix multiplication (1024*1024*1024) in cuSPARSE and cuBLAS (unit: us).

| Sparsity Ratio | 50% | 90% | 95% | 99% |
|---|---|---|---|---|
| cuSPARSE | 1652.5 | 633.9 | 463.0 | 181.7 |
| cuBLAS | 208.3 | 208.3 | 208.3 | 208.3 |

**The Gap between FLOPS and implementation**

Q: distribution of sparsity in real workloads?

# The diminishing end2end return

**Current optimizations focus on certain operators, ignoring the propagation across the whole model .**

# The diminishing end2end return

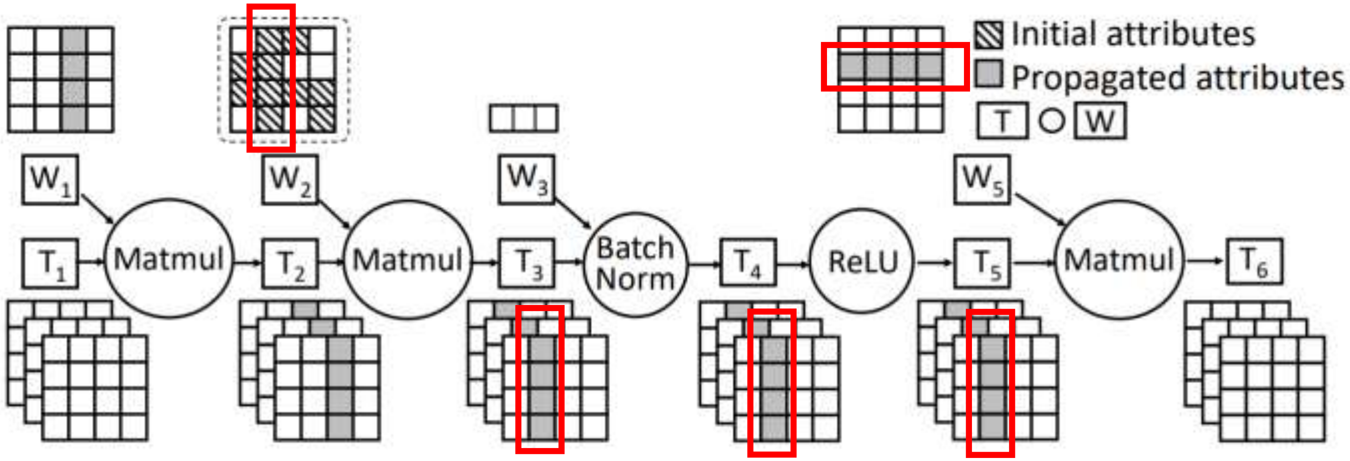



**Figure 1.** The sparsity attribute of one tensor can be propagated along the deep learning network.

Forward propagation opportunities

# The diminishing end2end return

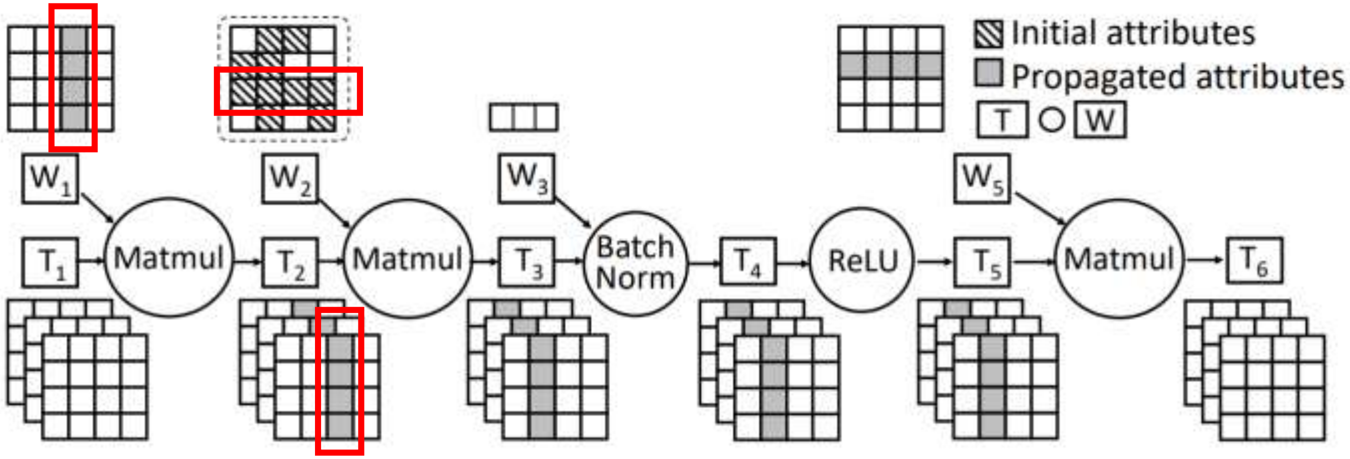



**Figure 1.** The sparsity attribute of one tensor can be propagated along the deep learning network.

Backward propagation opportunities

# Across-stack sparsity innovations in silos

Quantization

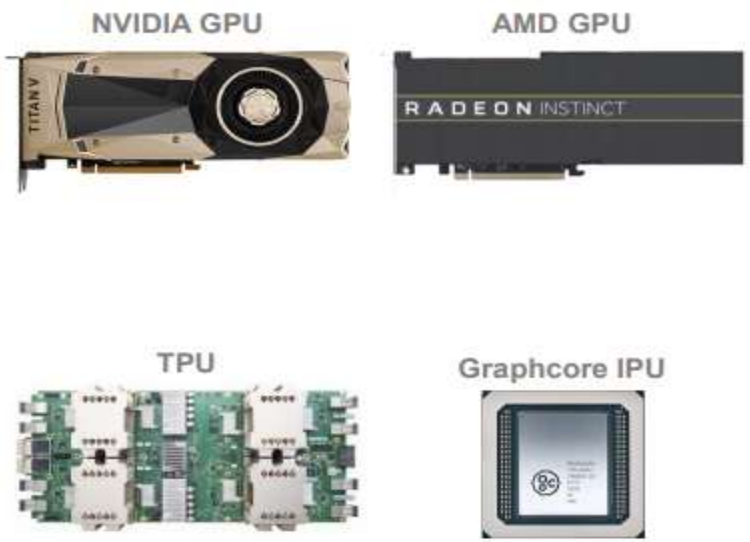
- Binarized models[20], 8-bit models[33, 68]
- Mixed precision[24, 38, 55, 47, 62]

Pruning **Active & Extensively!**

- Fine-grained[29, 35, 36]
- Block sparsity[37, 40, 42, 44]

Combination with quantization and pruning[28, 53, 54, 57


Modern Accelerators




**Machine learning practitioners often have to implement their sparsity algorithms end-to-end manually.**

# Goals

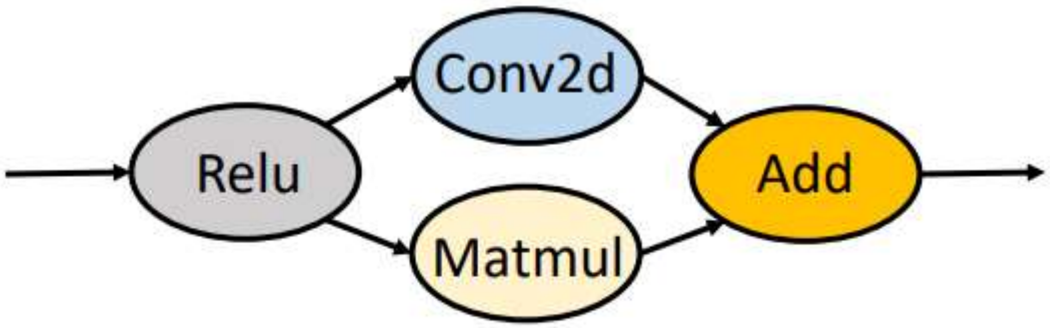
- Problems
  - Generic sparse kernels remain far from optimal.
  - Local optimizations miss the global gains.
  - The support for sparsity innovations is insufficient.
- Goals
  - Extreme performance and applicability
  - End-to-end optimization
  - Customizable and extensible to new sparsity innovations
  - Covering the whole-stack

# Outline

- Motivation
- Goals
- Design
  - Design overview
    - TeSA, propagation, code generation workflow
  - Design meets goals
- Evaluation
- Related works

# Design overview

- TeSA: Tensor-with-Sparsity-Attribute
- Sparsity attribute propagation
- Generating efficient code



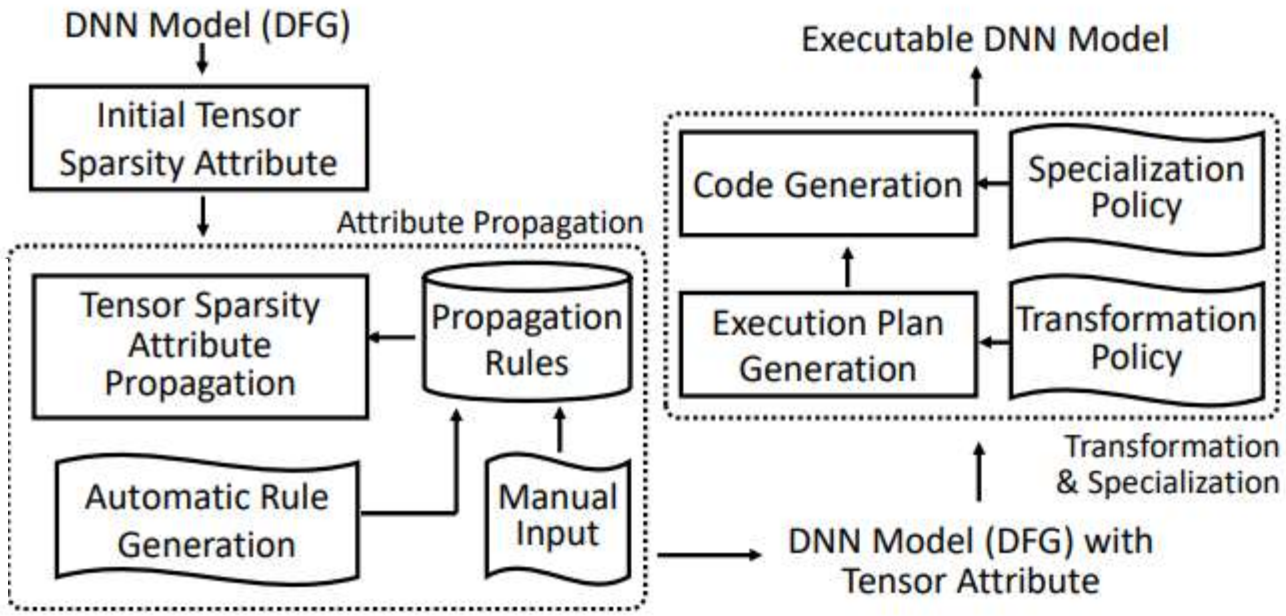
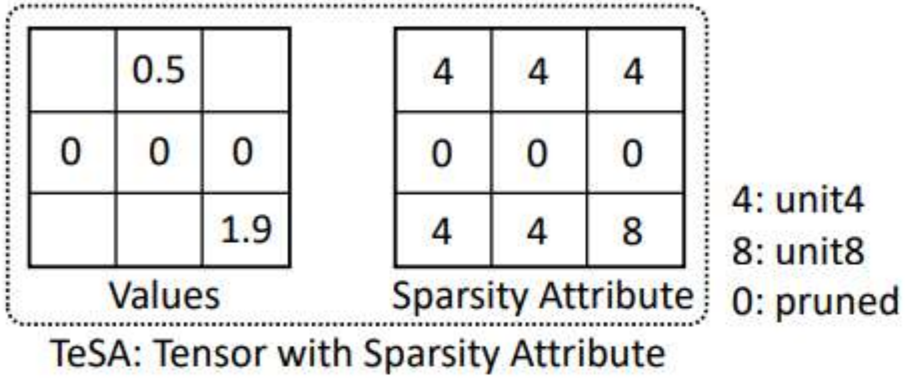




**Figure 2.** The system architecture of SparGen.

# Design - TeSA

- TeSA: Tensor-with-Sparsity-Attribute
  - Initialized by users
  - Updated by propagations




***A irregular sparsity pattern case***

**Figure 3.** An example of TeSA abstraction. Sparsity Attribute denotes the quantization scheme, 4 means `uint4`, 8 means `uint8`, and 0 means the element is pruned.

Q: how to support dynamic sparsity pattern?

# Design - propagation

- Sparsity attribute propagation
  - Propagation rules

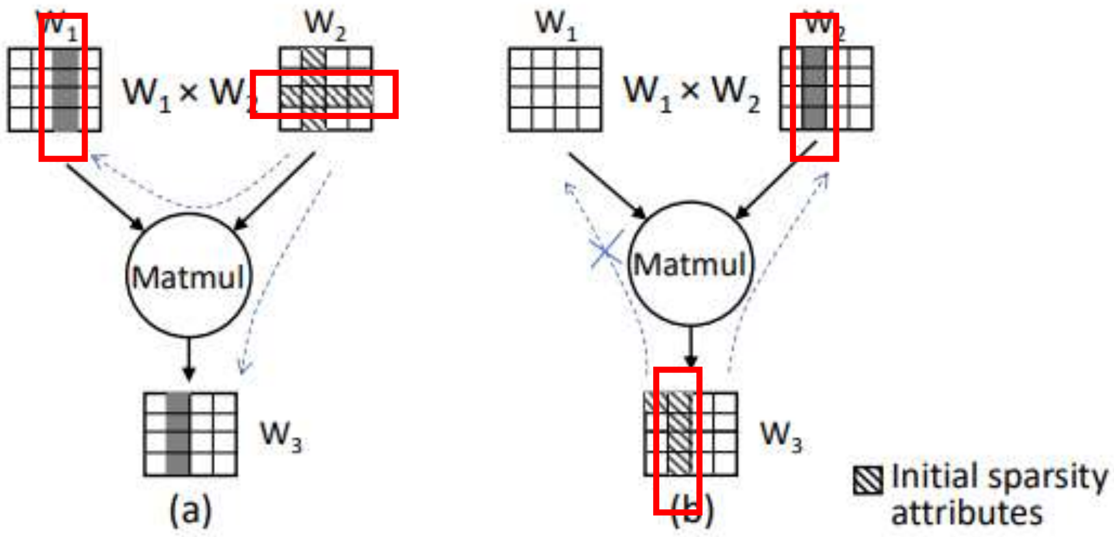



**Figure 4.** The propagation of sparsity attribute. The gray blocks are propagated sparsity attributes.

# Design – propagation (cont.)

- Sparsity attribute propagation
  - Propagation rules and conflict resolution

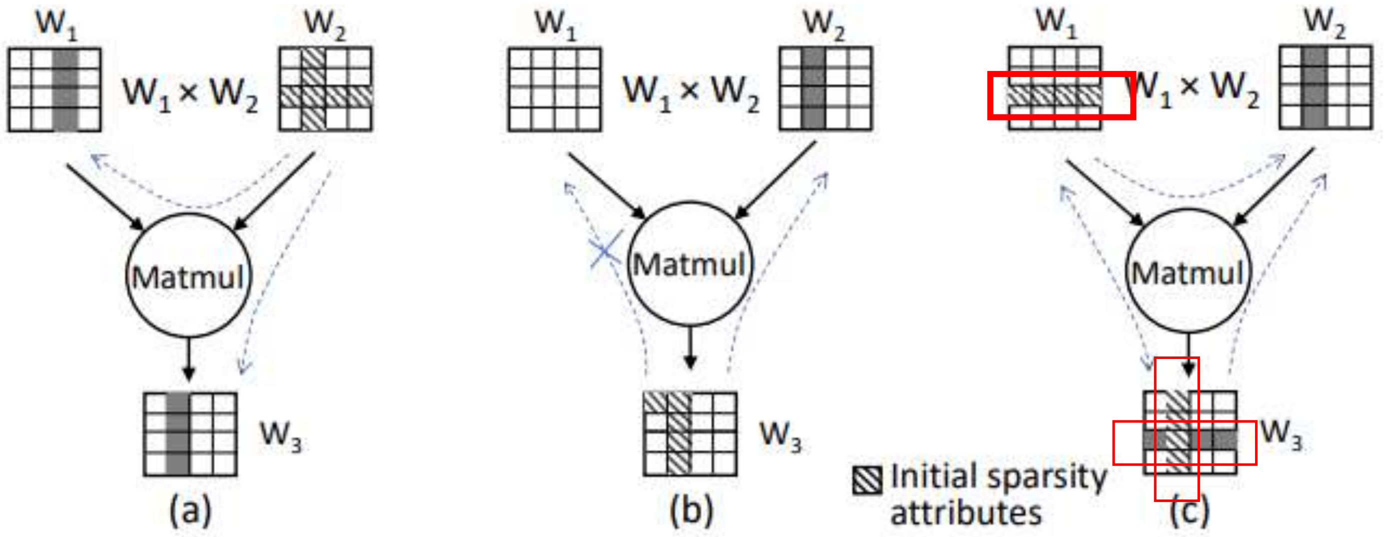
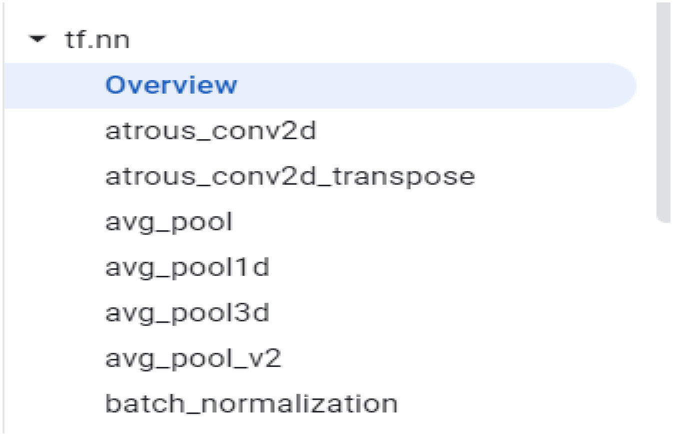



**Figure 4.** The propagation of sparsity attribute. The gray blocks are propagated sparsity attributes.

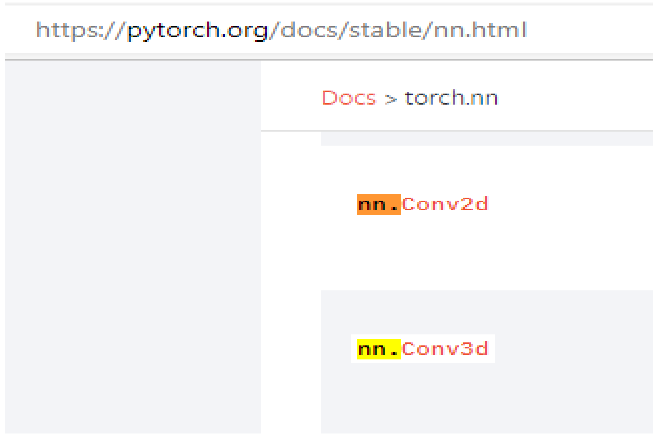
Conflict resolution：
- Pruning: the union of the pruned elements
- Low-precision: the lower precision

# Design – propagation (cont.)

- Sparsity attribute propagation
  - Propagation rules and conflict resolution
  - Rules: manual input & automatic generation




Tensorflow: 108+ operators

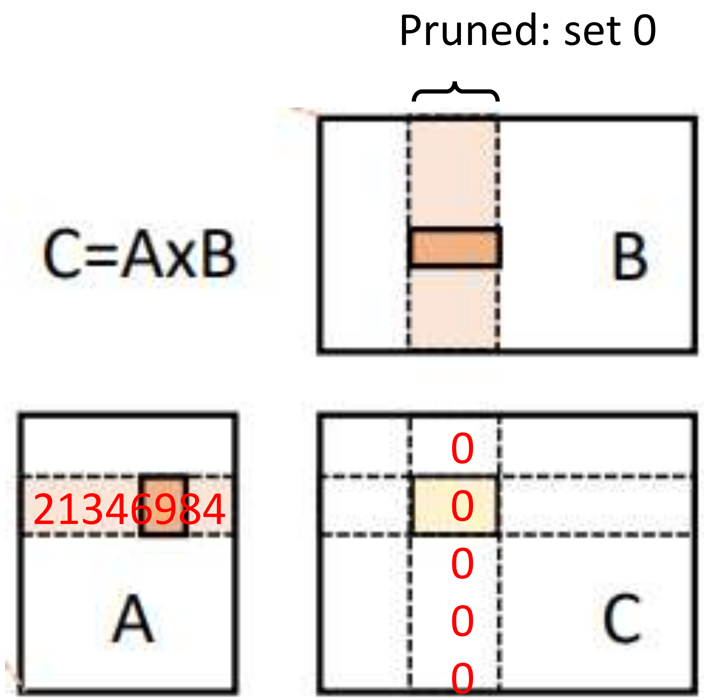



Pytorch: 174+ operators

**It is a burden to define propagation rules for every operator.**

# Design – propagation (cont.)

- Sparsity attribute propagation
  - Propagation rules and conflict resolution
  - Rules: manual input & automatic generation




**Tensor scrambling detects the invariant elements of a tensor by scrambling the values of other related tensors.**

# Design – code generation workflow

- Generating efficient code
  - Decomposition of TeSAs and operators
  - Dead code elimination
  - Hardware-supported low-precision instructions

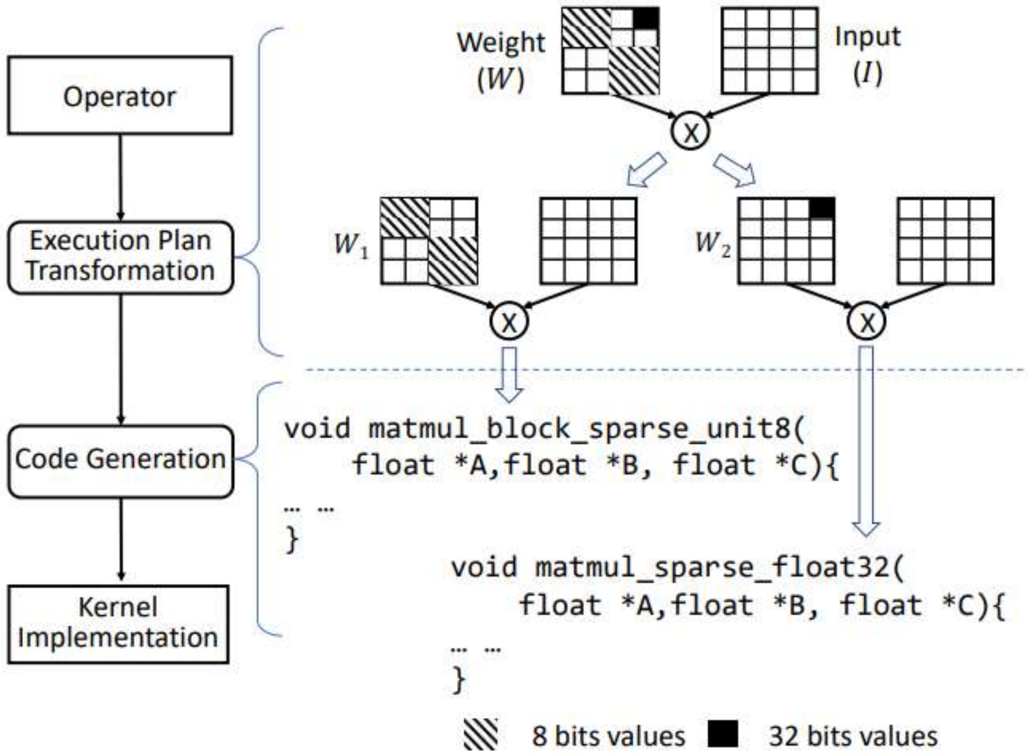



**Figure 6.** The two-pass compilation process to generate an efficient kernel implementation for an operator.

**From a generic kernel for dynamic workloads to generated kernels for specific workloads.**

# Design – code generation (cont.)

- Generating efficient code
  - Decomposition of TeSAs and operators for ***irregular sparsity patterns***
    - Transformation policy

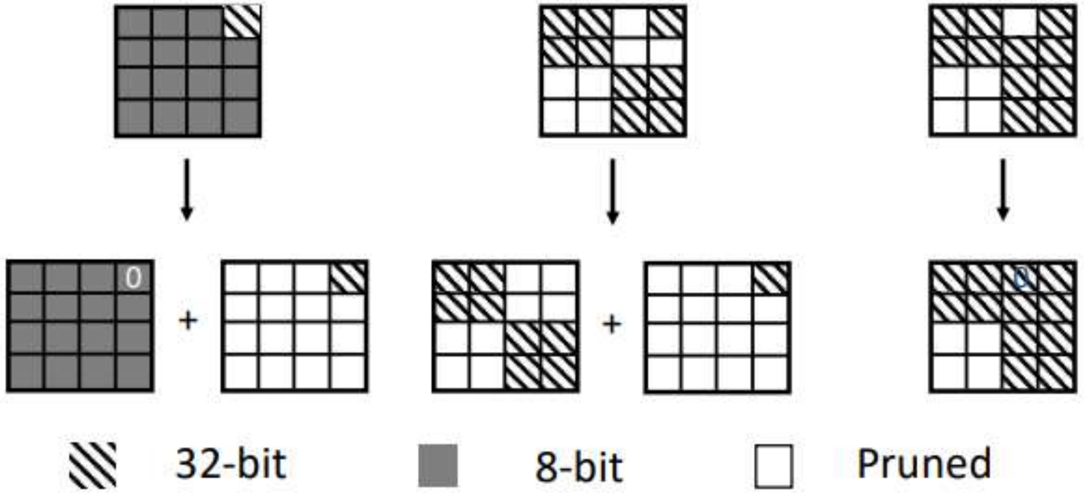



Figure 7. Examples of transformation policies.

# Design – code generation (cont.)

- Generating efficient code
  - Decomposition of TeSAs and operators for ***irregular sparsity patterns***
    - Transformation policy
  - Dead code elimination for ***regular patterns***
  - Hardware-supported low-precision instructions
    - Specialization policy

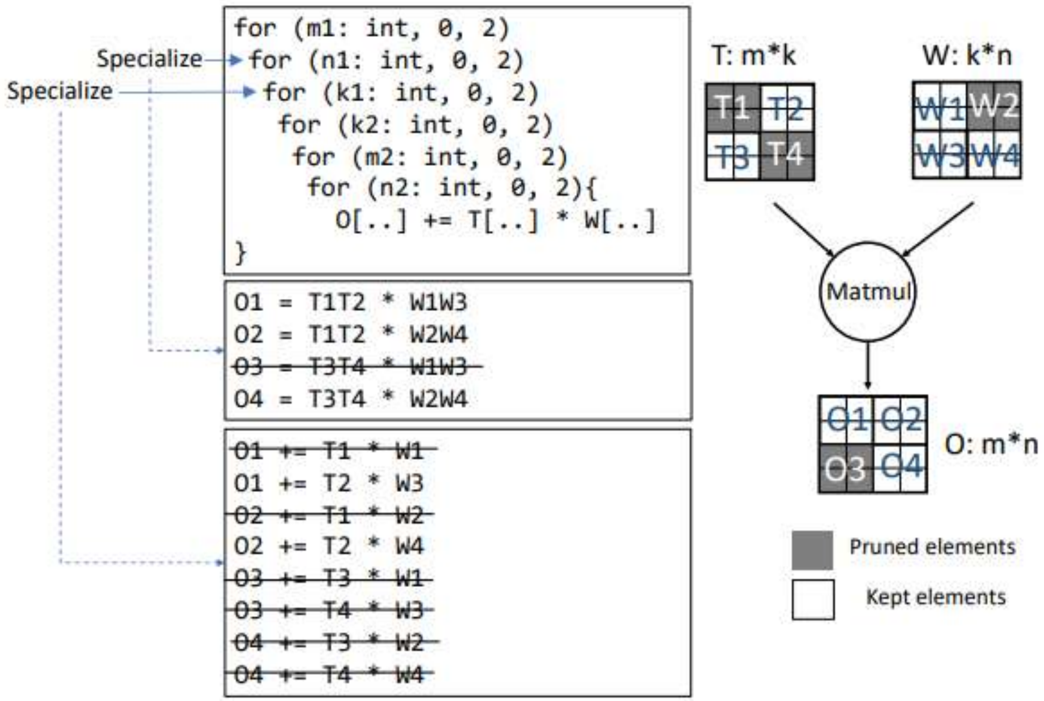



**Figure 9.** Sparsity-aware code specialization: loop unrolling and dead code elimination.

# Design – code generation (cont.)

- Generating efficient code
  - Decomposition of TeSAs and operators
    - Transformation policy
  - Dead code elimination
  - Hardware-supported low-precision instructions
    - Specialization policy: mma_sync, DP4A

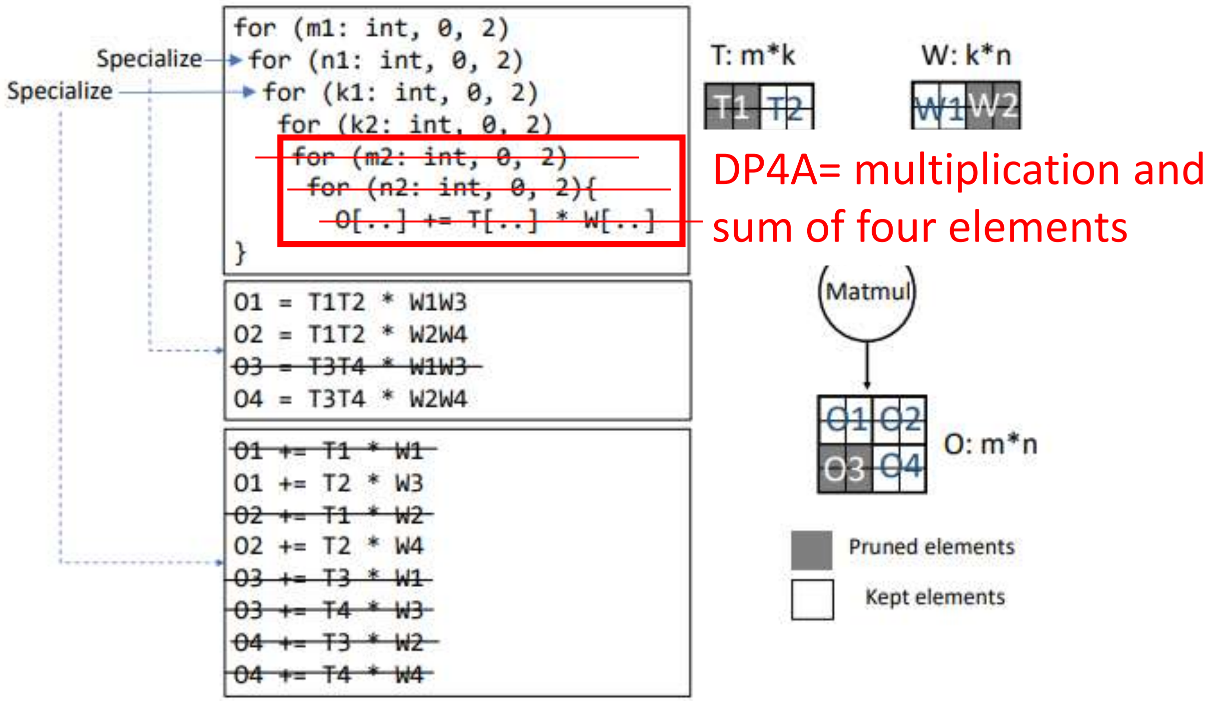



**Figure 9.** Sparsity-aware code specialization: loop unrolling and dead code elimination.

# Design highlights

- TeSA: Tensor-with-Sparsity-Attribute
  - Initialized by  users
- Sparsity attribute propagation
  - Propagation rules and conflict resolution
  - Rules: manual input & automatic generation
- Generating efficient code
  - Decomposition of TeSAs and operators
    - Transformation policy
  - Dead code elimination
  - Hardware-supported low-precision instructions
    - Specialization policy: mma_sync, DP4A

# Design meets goals

- TeSA: Tensor-with-Sparsity-Attribute
  - Initialized by users
- Sparsity attribute propagation
  - Propagation rules and conflict resolution
  - Rules: manual input & automatic generation
- Generating efficient code
  - Decomposition of TeSAs and operators
    - Transformation policy
  - Dead code elimination
  - Hardware-supported low-precision instructions
    - Specialization policy: mma_sync, DP4A

Covering the whole-stack

# Design meets goals

- TeSA: Tensor-with-Sparsity-Attribute
  - Initialized by  users
- Sparsity attribute propagation
  - Propagation rules and conflict resolution
  - Rules: manual input & automatic generation
- Generating efficient code
  - Decomposition of TeSAs and operators
    - Transformation policy
  - Dead code elimination
  - Hardware-supported low-precision instructions
    - Specialization policy: mma_sync, DP4A

Customizable and extensible to new sparsity innovations

# Design meets goals

- TeSA: Tensor-with-Sparsity-Attribute
  - Initialized by  users
- Sparsity attribute propagation
  - Propagation rules and conflict resolution
  - Rules: manual input & automatic generation
- Generating efficient code
  - Decomposition of TeSAs and operators
    - Transformation policy
  - Dead code elimination
  - Hardware-supported low-precision instructions
    - Specialization policy: mma_sync, DP4A

End-to-end optimization
& Extreme performance

# Outline

- Motivation
- Goals
- Design
- Evaluation
  - Performance
  - Effectiveness of designs
  - Facilitating exploration of model sparsity
- Related works

# Evaluation - performance

- 5 pages in 12 pages
- If I were the author, …

# Evaluation – perf.

- TeSA: Tensor-with-Sparsity-Attribute
  - Initialized by users
- Sparsity attribute propagation
  - Propagation rules and conflict resolution
  - Rules: manual input & automatic generation
- Generating efficient code
  - Decomposition of TeSAs and operators
    - Transformation policy
  - Dead code elimination
  - Hardware-supported low-precision instructions
    - Specialization policy: mma_sync, DP4A

End-to-end optimization
& Extreme performance

- 5 pages in 12 pages
- If I were the author, …
  - End2end performance
    - What models? * what sparsity algorithms? * what baselines? * what testbeds?
  - Breakdown
    - Pruning(dead code elimination), low-precision(hardware instruction), propagation(more pruning and low-precision elements)

# Evaluation – perf.

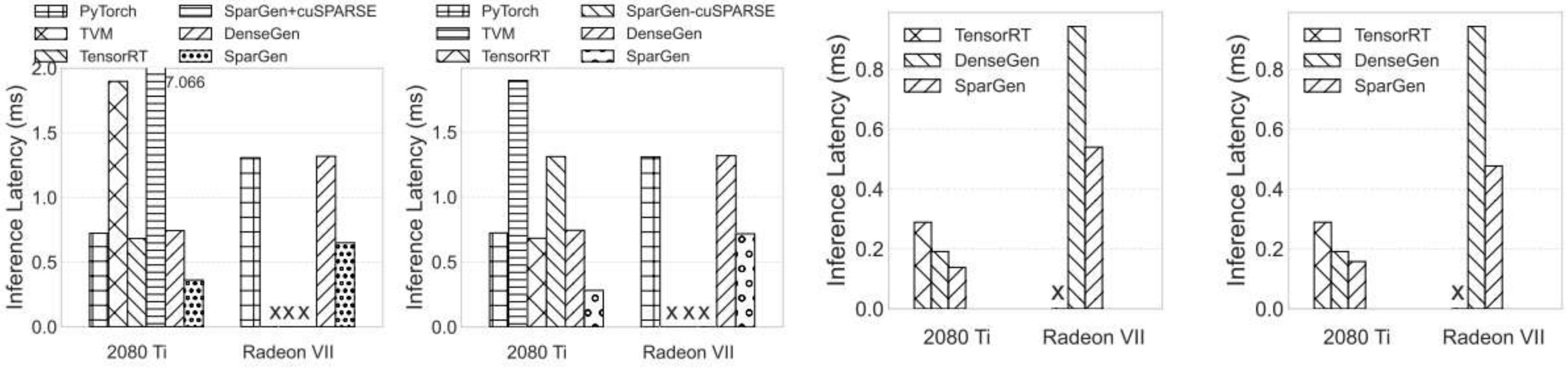
- TeSA: Tensor-with-Sparsity-Attribute
  - Initialized by users
- Sparsity attribute propagation
  - Propagation rules and conflict resolution
  - Rules: manual input & automatic generation
- Generating efficient code
  - Decomposition of TeSAs and operators
    - Transformation policy
  - Dead code elimination
  - Hardware-supported low-precision instructions
    - Specialization policy: mma_sync, DP4A

End-to-end optimization & Extreme performance

- 5 pages in 12 pages
- The author, …
  - End2end performance(3*4*6*2)
    - What models: MLP, MobileNet, BERT
    - What sparsity algorithms: coarse-grained pruning, fine-grained pruning, coarse-grained pruning+8bit, block pruning+8bit
    - What baselines: Pytorch, TVM, TensorRT, SparGen-cuSPARRSE, DenseGen, SparGen
    - What testbed: Nvidia GeForce RTX 2080Ti, AMD Radeon VII
  - Breakdown
    - Pruning(dead code elimination), low-precision(hardware instruction), propagation(more pruning and low-precision elements)
  - For one kernel:
    - Workload: matrix multiplication, problem size (1024*1024*1024)
    - cuSPARSE, cublas, TACO, Sparse GPU kernels, SparseRT, SparGen

Algorithms: pruning (coarse-grained, fine-grained), low-precision, pruning + precision
*No low-precision*: the improvement is small since the baselines with quantization are good.

# End2end performance analysis




(a) Coarse-grained sparsity (b) Fine-grained sparsity (c) Coarse-grained sparsity + 8-bit (d) Block sparsity + 8-bit

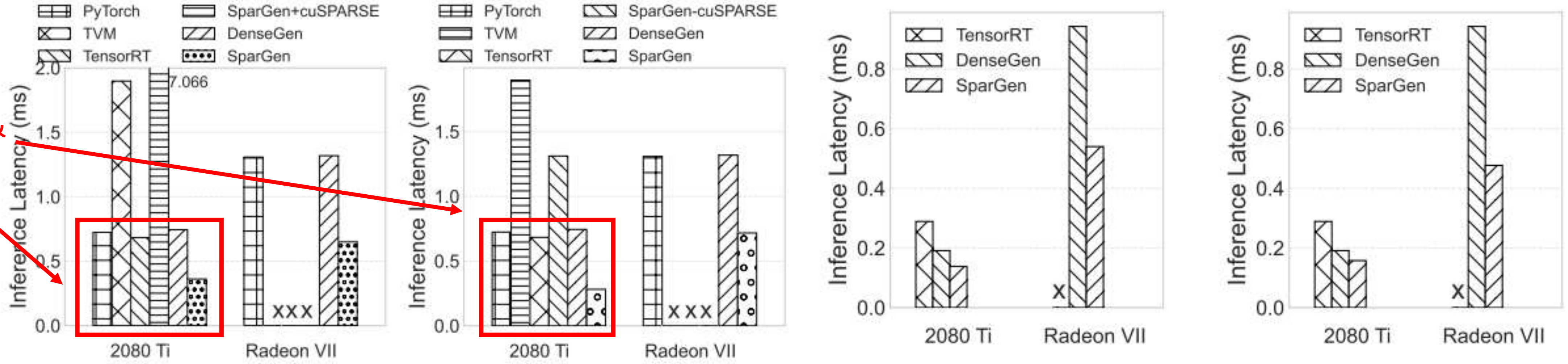
**Figure 10.** The end-to-end inference latency of MLP with four different sparsity patterns.

SparGen performs the best: 2.4x - 6.8x speedup: propagation + hardware instruction

In-consistent legends

# End2end performance analysis

Propagation & instruction




(a) Coarse-grained sparsity (b) Fine-grained sparsity (c) Coarse-grained sparsity + 8-bit (d) Block sparsity + 8-bit

**Figure 10.** The end-to-end inference latency of MLP with four different sparsity patterns.

SparGen performs the best: 2.4x - 6.8x speedup: propagation + hardware instruction

# End2end performance analysis

cuSPARSE latency

TVM tuned kernel

Propagation & instruction

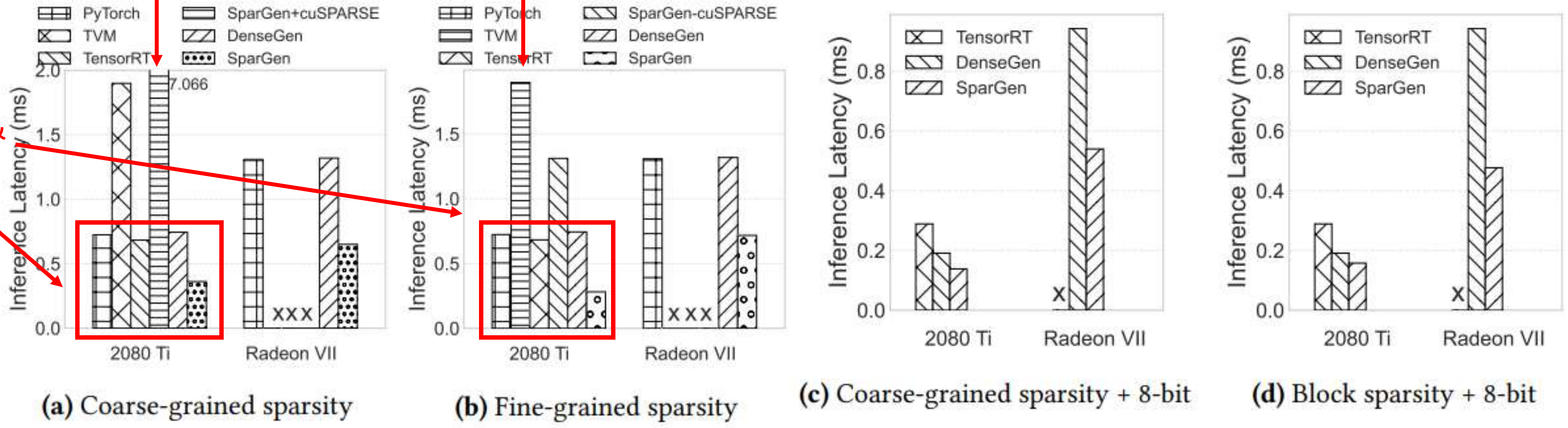



**Figure 10.** The end-to-end inference latency of MLP with four different sparsity patterns.

SparGen performs the best: 2.4x - 6.8x speedup: propagation + hardware instruction

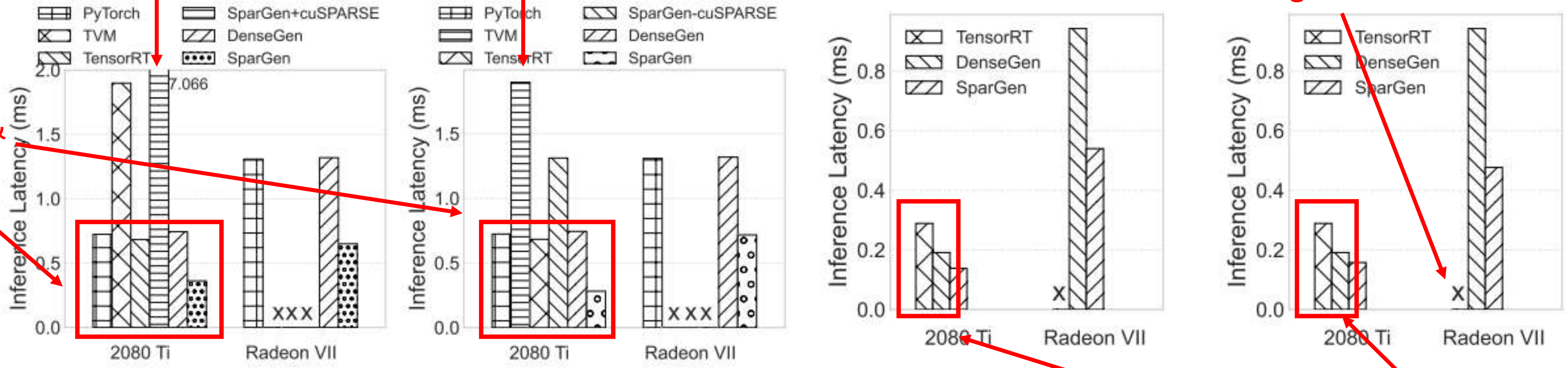
# End2end performance analysis

cuSPARSE latency

TVM tuned kernel

Unsupported v.s. general

Propagation & instruction




**(a)** Coarse-grained sparsity **(b)** Fine-grained sparsity **(c)** Coarse-grained sparsity + 8-bit **(d)** Block sparsity + 8-bit

**Figure 10.** The end-to-end inference latency of MLP with four different sparsity patterns.

DenseGen has good kernels

SparGen performs the best: 2.4x - 6.8x speedup: propagation + hardware instruction

# End2end performance analysis (cont.)

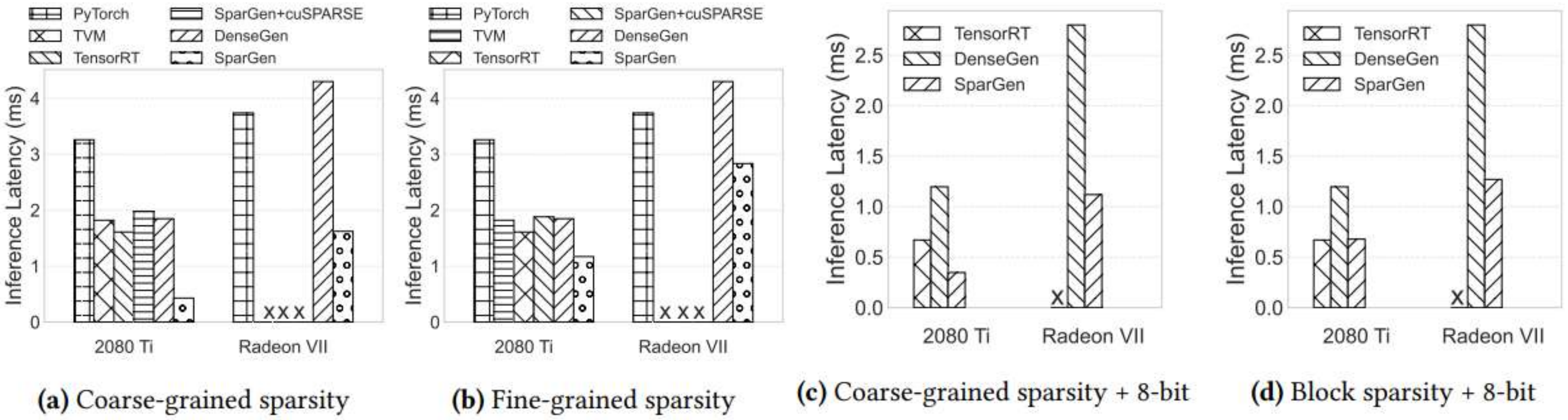



**Figure 11.** The end-to-end inference latency of MobileNet with four different sparsity patterns.

SparGen performs the best: 3.7x - 7.8x speedup: propagation + hardware instruction

In-consistent results: TVM, cuSPARSE

# End2end performance analysis (cont.)

With 60% sparsity initialization, get 4.3x than DenseGen: propagation

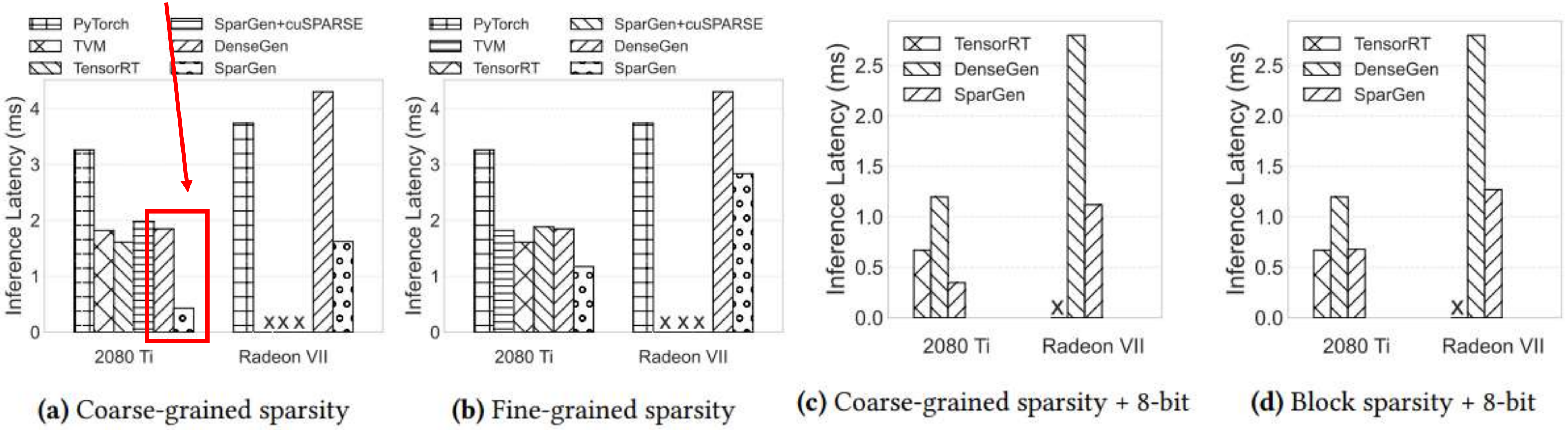



**Figure 11.** The end-to-end inference latency of MobileNet with four different sparsity patterns.

SparGen performs the best: 3.7x - 7.8x speedup: propagation + hardware instruction

# End2end performance analysis (cont.)

60% sparsity, get 4.3x: propagation

Use cuSPARSE only for one linear layer. Conv is unsupported.

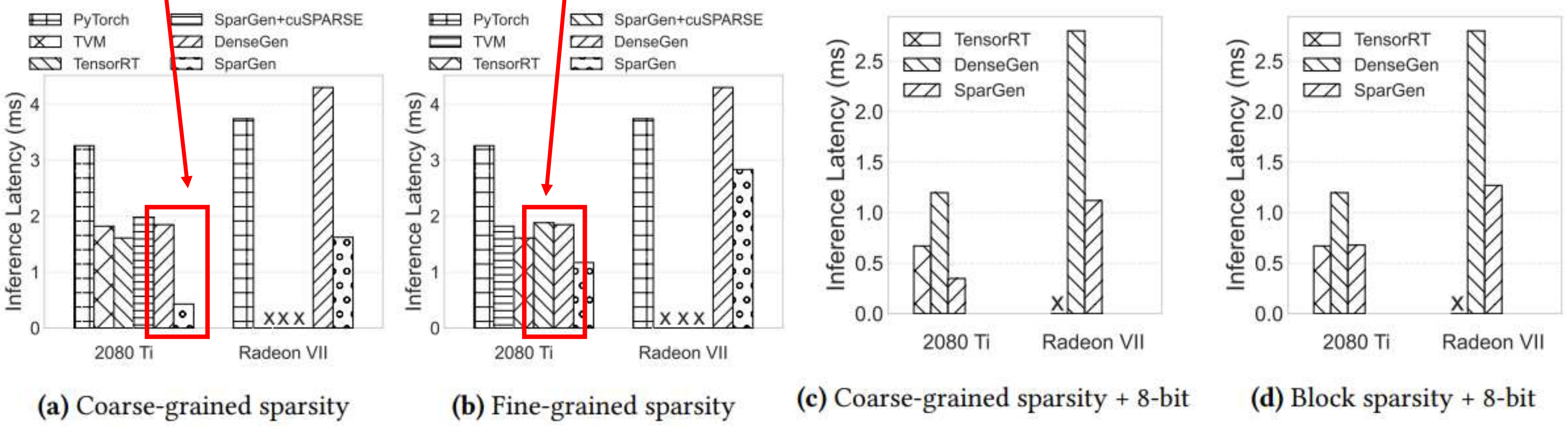



**Figure 11.** The end-to-end inference latency of MobileNet with four different sparsity patterns.

SparGen performs the best: 3.7x - 7.8x speedup: propagation + hardware instruction

# End2end performance analysis (cont.)

60% sparsity, get 4.3x: propagation

Use cuSPARSE only for one linear layer. Conv is unsupported.

TensorRT optimize MobileNet with 8-bits

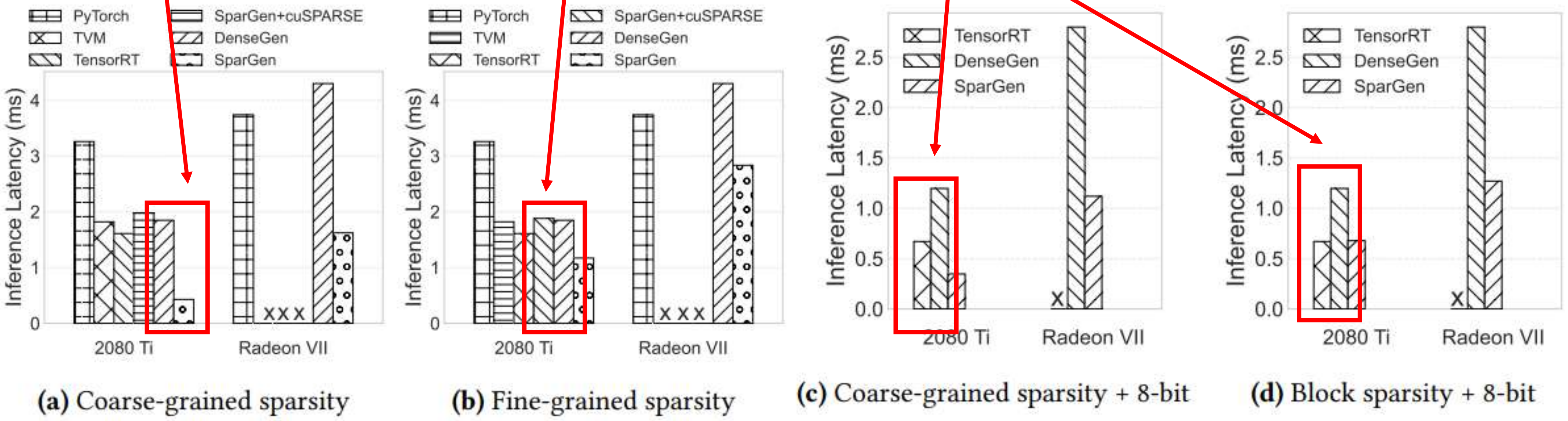



**Figure 11.** The end-to-end inference latency of MobileNet with four different sparsity patterns.

SparGen performs the best: 3.7x - 7.8x speedup: propagation + hardware instruction

In-consistent results: TensorRT + MLP and +MobileNet/BERT
DenseGen has good kernels?

# Performance Breakdown

Due to w/o 8-bit

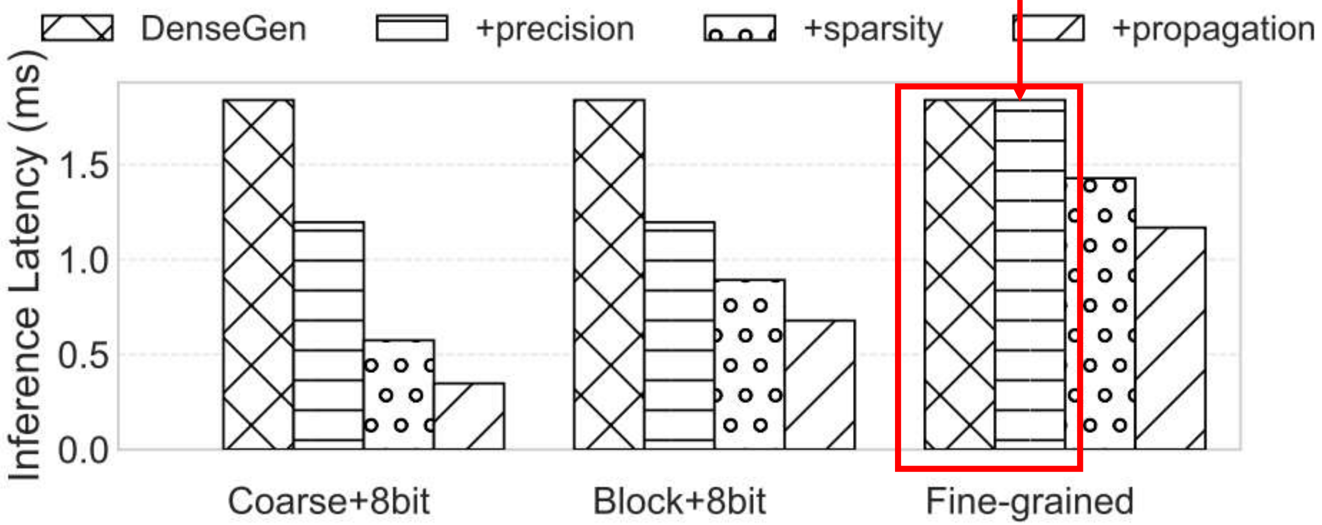
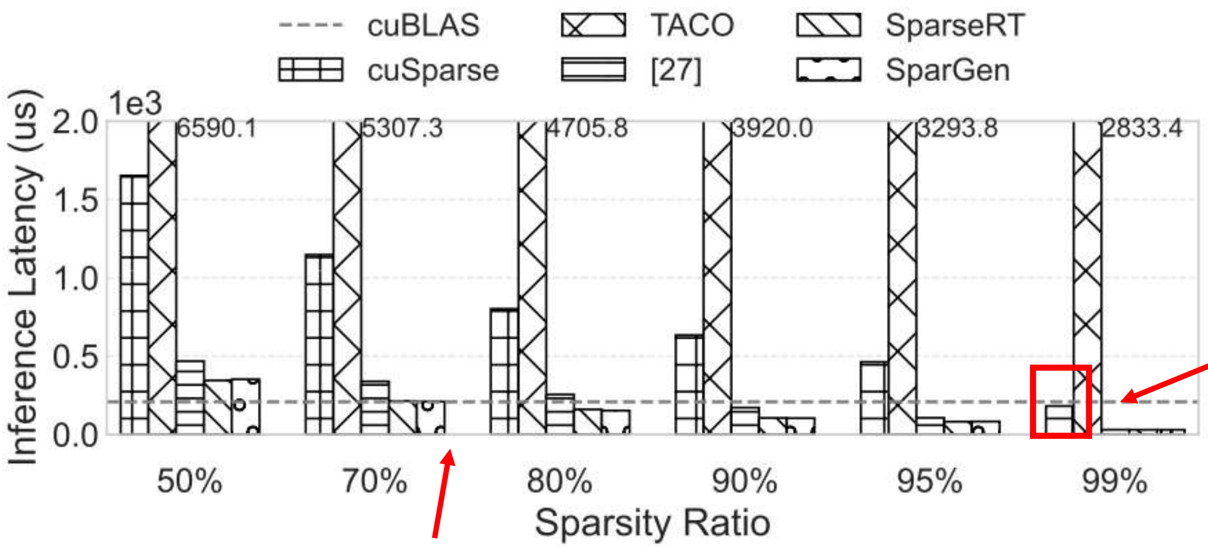



**Figure 13.** Performance breakdown of SparGen for different sparsity patterns of MobileNet on 2080 Ti. Each bar shows the result of applying the additional optimization labeled on this bar from the previous one.

+precision gets more 0? And get more sparsity optimization space?

# Performance comparison of one kernel




cuSPARSE outperforms cublas only here.

SparGen outperforms cublas from here.

**Figure 14.** Comparison of cuSPARSE, TACO, and SparGen on matrix multiplication (1024x1024x1024) with fine-grained sparsity under different sparsity ratios. The sparsity is applied on $B$ for $A * B$.

# Evaluation – effecti.

- TeSA: Tensor-with-Sparsity-Attribute
  - Initialized by users
- Sparsity attribute propagation
  - Propagation rules and conflict resolution
  - Rules: manual input & automatic generation
- Generating efficient code
  - Decomposition of TeSAs and operators
    - Transformation policy
  - Dead code elimination
  - Hardware-supported low-precision instructions
    - Specialization policy: mma_sync, DP4A

Customizable and extensible to new sparsity innovations

- 5 pages in 12 pages
- If I were the author, …
  - Micro-benchmarks
    - Propagation rules: sparsity of layers with or without propagation
      - What models? * what sparsity algo.?
    - Transformation policy: generating kernels for mixed various sparsity algo.
      - What kernel? * what sparsity algo.
    - Specialization policy: new hardware

# Evaluation – effecti.

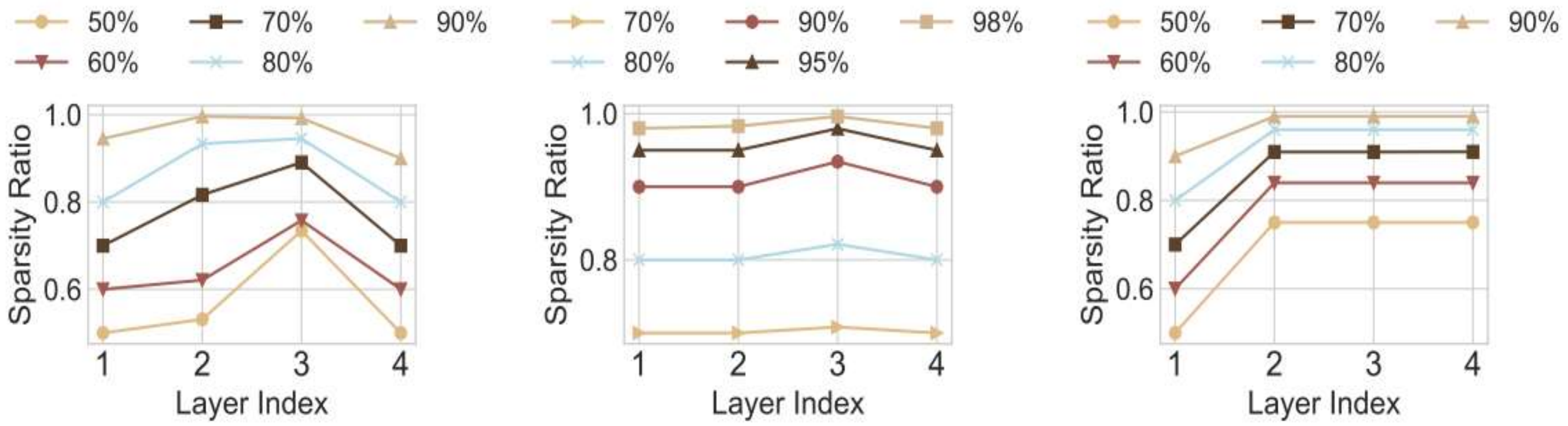
- TeSA: Tensor-with-Sparsity-Attribute
  - Initialized by users
- Sparsity attribute propagation
  - Propagation rules and conflict resolution
  - Rules: manual input & automatic generation
- Generating efficient code
  - Decomposition of TeSAs and operators
    - Transformation policy
  - Dead code elimination
  - Hardware-supported low-precision instructions
    - Specialization policy: mma_sync, DP4A

Customizable and extensible to new sparsity innovations

- 5 pages in 12 pages
- The author, …
  - Micro-benchmarks
    - Propagation rules: sparsity of layers with or without propagation
      - What models (MLP) * what sparsity algo. (Block pruning, fine-grained pruning, coarse-grained pruning)
      - What models (MobileNet) * what sparsity algo. (Quantization)
    - Transformation policy: generating kernels for mixed various sparsity algo.
      - What kernel (matrix multiplication, 1024*1024*1024)
      - What sparsity algo.: mixed precision(float32 for 0-5% elements+int8), mix of block pruning for 70%-90% elements and fine-grained pruning for 1% elements
    - Specialization policy: new hardware
      - Nvidia GeForce RTX 2080Ti, AMD Radeon VII

# Effectiveness on sparsity attribute propagation

Initial sparsity:



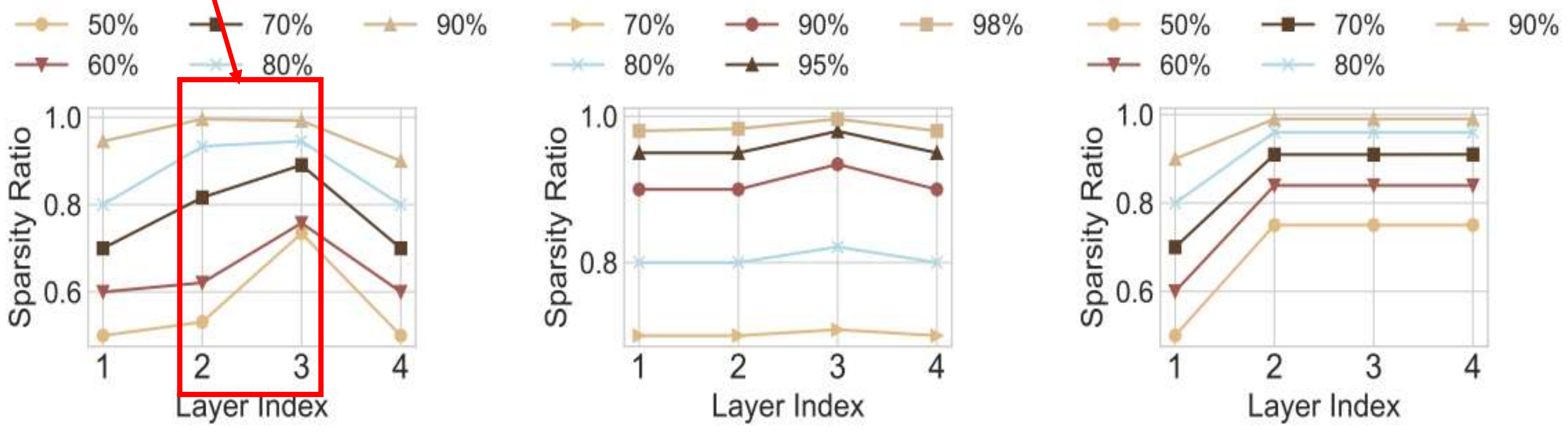

Sparsity pattern: **(a)** Block sparsity **(b)** Fine-grained sparsity **(c)** Coarse-grained sparsity

**Figure 16.** Propagated sparsity across the layers for different sparsity patterns on the MLP model.

# Effectiveness on sparsity attribute propagation

Receive more propagations

Initial sparsity:



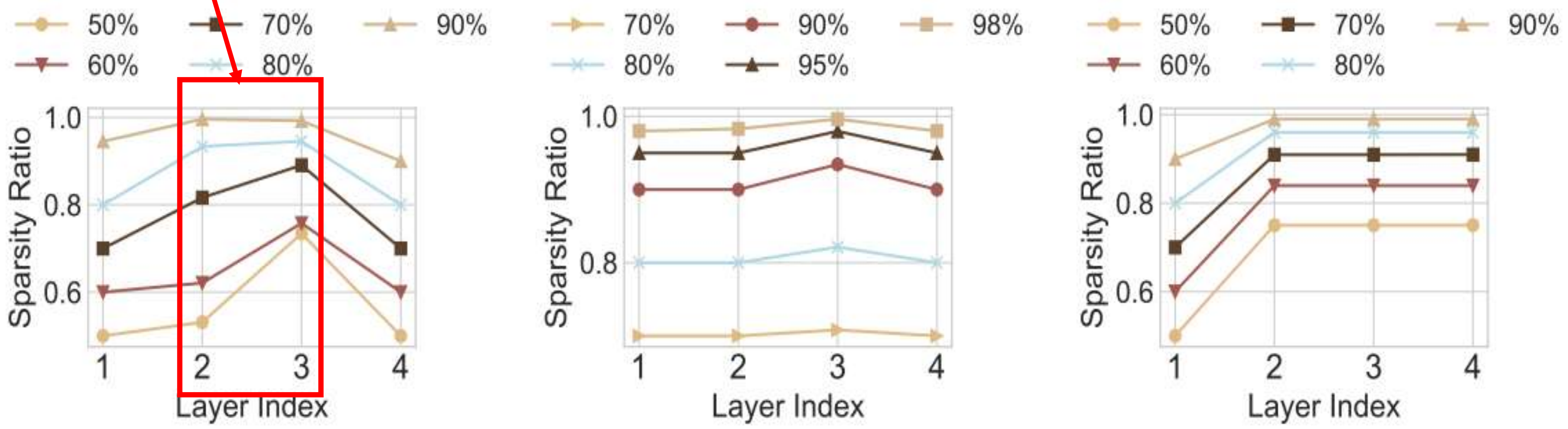

Sparsity pattern: **(a)** Block sparsity **(b)** Fine-grained sparsity **(c)** Coarse-grained sparsity

**Figure 16.** Propagated sparsity across the layers for different sparsity patterns on the MLP model.

# Effectiveness on sparsity attribute propagation

Receive more propagations

Initial sparsity:




Sparsity pattern: (a) Block sparsity (b) Fine-grained sparsity (c) Coarse-grained sparsity

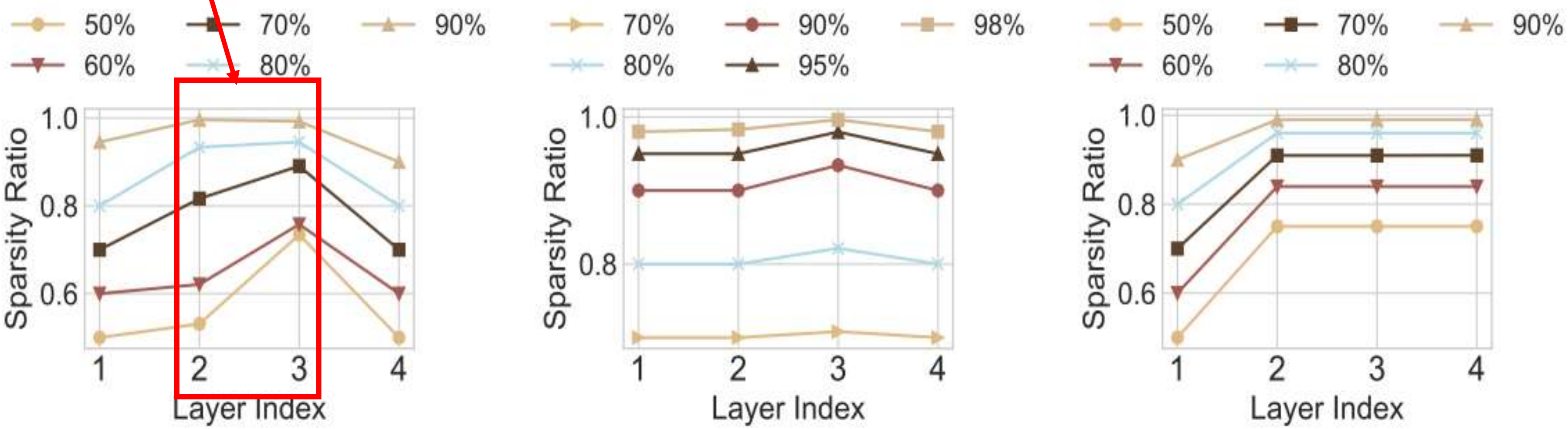
**Figure 16.** Propagated sparsity across the layers for different sparsity patterns on the MLP model.

The probability that an entire column or row is pruned is much lower

# Effectiveness on sparsity attribute propagation

Receive more propagations

Initial sparsity:




Sparsity pattern:

**(a)** Block sparsity

**(b)** Fine-grained sparsity

**(c)** Coarse-grained sparsity

**Figure 16.** Propagated sparsity across the layers for different sparsity patterns on the MLP model.

The probability that an entire column or row is pruned is much lower

More forward propagation due to entire row pruning in this experiment.

# Effectiveness on sparsity attribute propagation

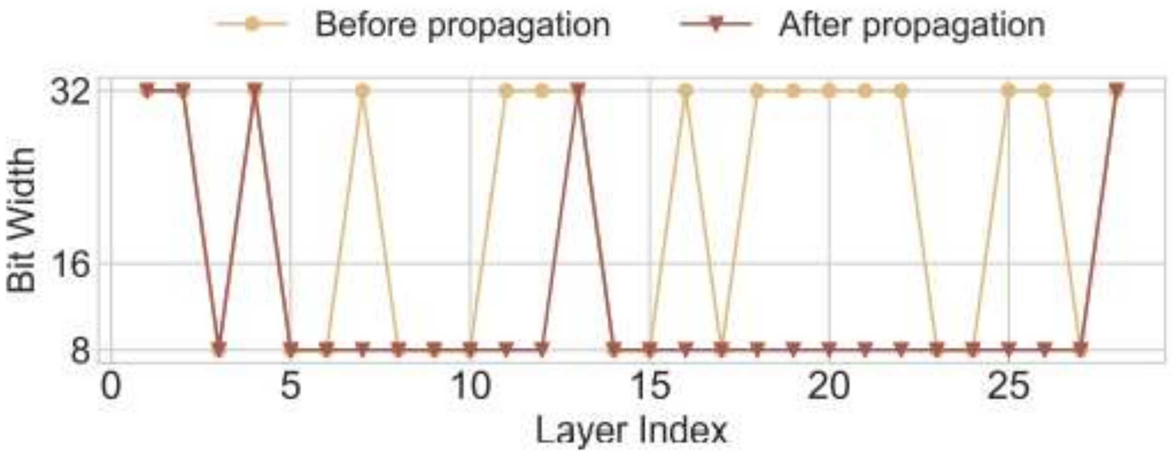
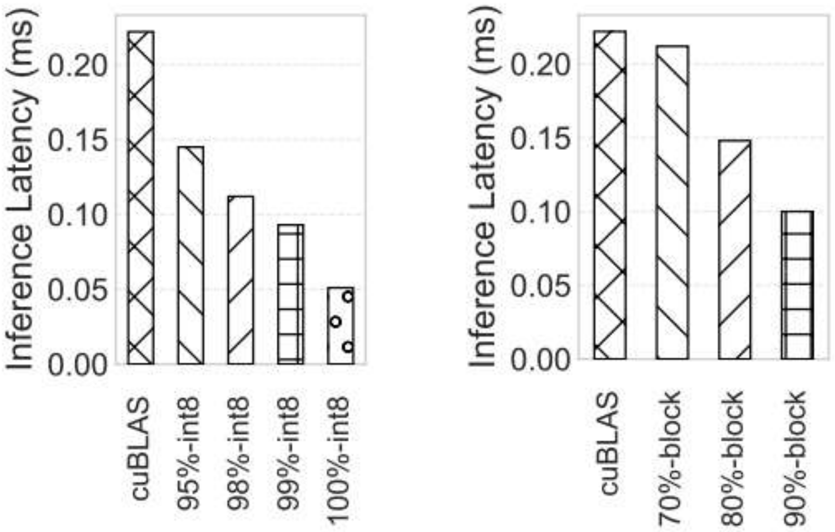



**Figure 17.** The quantization (bit width) of each layer in MobileNet before and after propagation. They have the same accuracy.

# Effectiveness of execution plan transformation




float32 for 0-5% elements+int8

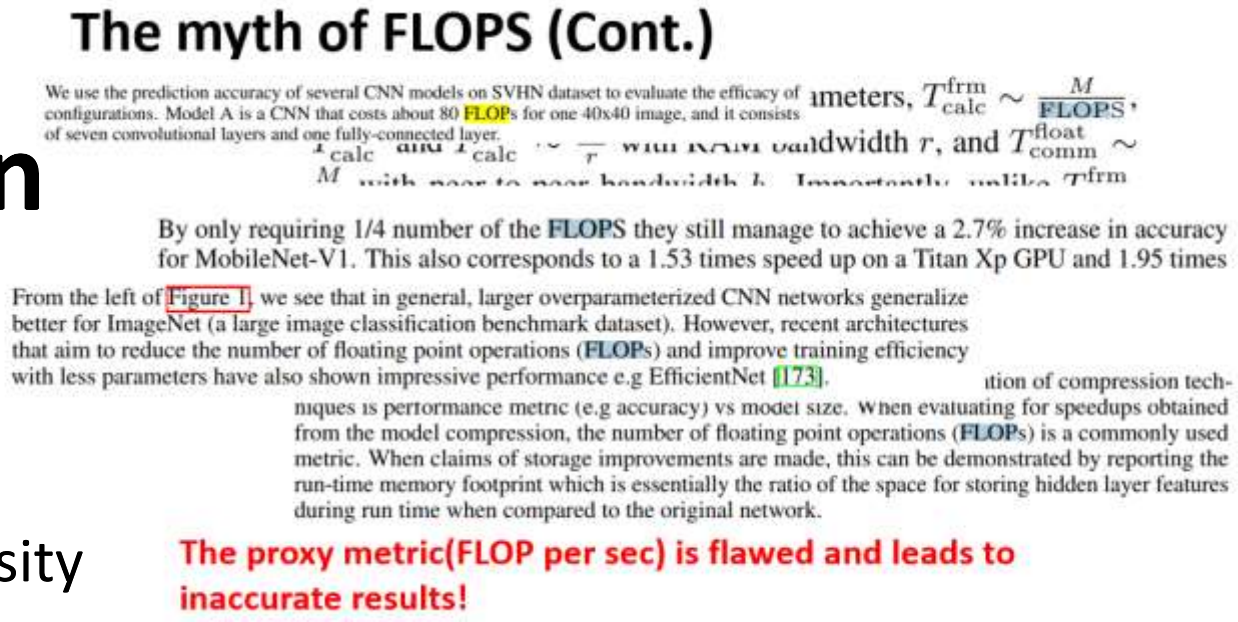
**(a)** Mixed precisions **(b)** Mixed sparsity patterns

70%-90% elements and fine-grained pruning for 1% elements

**Figure 15.** The performance of the execution plan transformation in SparGen for mixed precision and sparsity patterns. $B$ is sparsified for the matrix multiplication $A * B$ (1024x1024x1024). "X%-block" means X% block sparsity mixed with 1% fine-grained sparsity.

# Evaluation - exploration

- 5 pages in 12 pages
- If I were the author, …
    - Facilitating exploration of model sparsity
        - Better
        - Faster



# Evaluation - exploration

- 5 pages in 12 pages
- The author, …
    - Facilitating exploration of model sparsity
        - Profiling valuable feedback other than proxy metrics.
        - Better: propagation aware sparsity exploration gets higher accuracy.
        - Faster: speeding up sparsity exploration

## The myth of FLOPS (Cont.)

We use the prediction accuracy of several CNN models on SVHN dataset to evaluate the efficacy of configurations. Model A is a CNN that costs about 80 FLOPs for one 40x40 image, and it consists of seven convolutional layers and one fully-connected layer.

By only requiring 1/4 number of the FLOPS they still manage to achieve a 2.7% increase in accuracy for MobileNet-V1. This also corresponds to a 1.53 times speed up on a Titan Xp GPU and 1.95 times

From the left of Figure 1, we see that in general, larger overparameterized CNN networks generalize better for ImageNet (a large image classification benchmark dataset). However, recent architectures that aim to reduce the number of floating point operations (FLOPs) and improve training efficiency with less parameters have also shown impressive performance e.g EfficientNet [173].

…tion of compression techniques is performance metric (e.g accuracy) vs model size. When evaluating for speedups obtained from the model compression, the number of floating point operations (FLOPs) is a commonly used metric. When claims of storage improvements are made, this can be demonstrated by reporting the run-time memory footprint which is essentially the ratio of the space for storing hidden layer features during run time when compared to the original network.

**The proxy metric(FLOP per sec) is flawed and leads to inaccurate results!**

# Exploration - real and valuable feedback

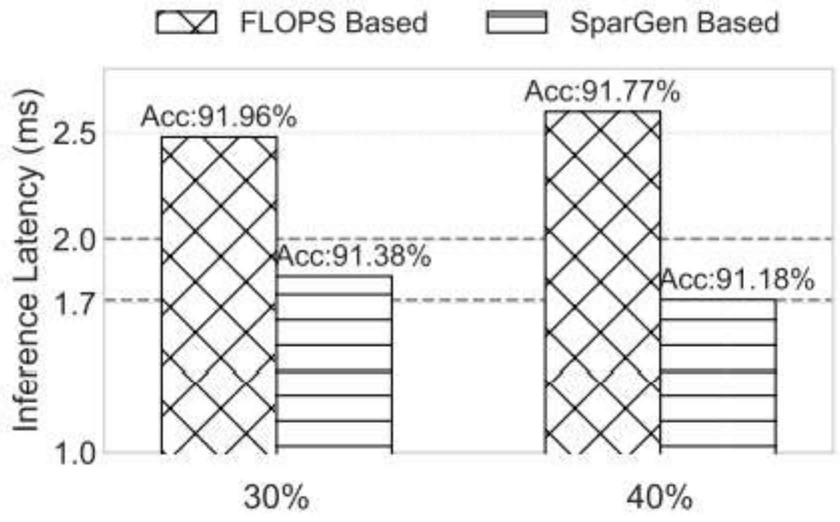
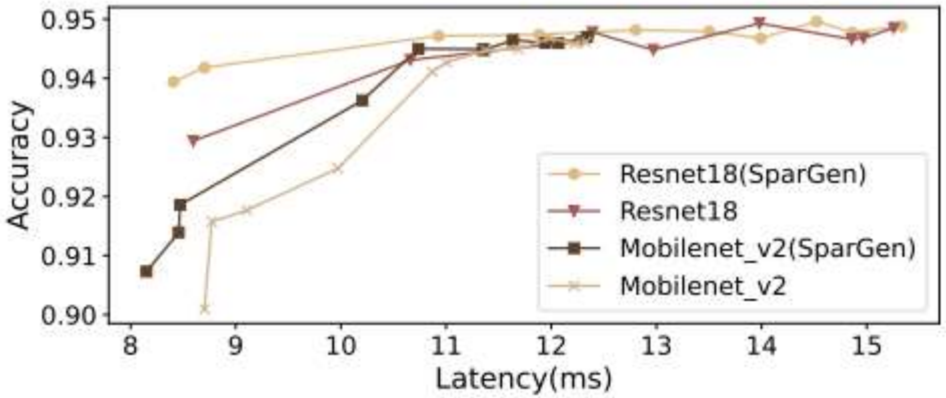



**Figure 18.** The comparison of using real latency or FLOPS as metric to explore sparse models by Simulated Annealing.

# Exploration - better and faster




Prune 50%, then re-codegen

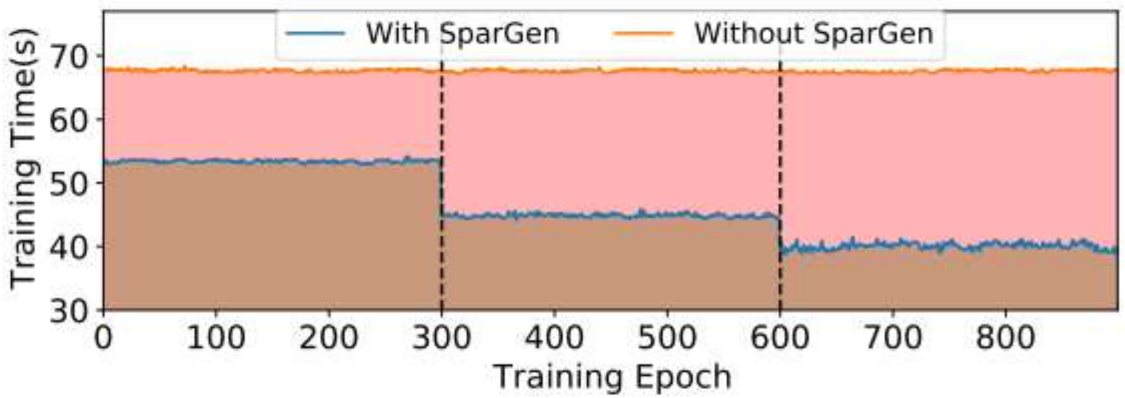



**Figure 19.** The performance comparison of being aware o sparsity propagation and not being aware.

**Figure 20.** The exploration time when using SparGen-accelerated sparse model vs. not using the accelerated model.

# Related works

- Same goal + similar techniques: SparseRT(a special case)
- Same goal + various techniques:
  - PyTorch [48], TensorFlow [13], TVM/Ansor [18, 67], all treat model sparsity as an afterthought
  - optimizations for certain type of hardware
  - data format
- Same goal + orthogonal techniques: classic compiler techniques, new hardware

# Position

- SparGen:
  - SparGen takes a principled system approach to model sparsity in deep learning, centered on the new TeSA abstraction.
  - SparGen is designed to accommodate a rich set of sparsity patterns, work end-to-end and across the stack to support propagation of sparsity patterns and the optimizations that take advantage of those patterns, and leverage compiler technology and hardware support, all in an extensible framework.
  - SparGen can not only contribute to superior sparsity-induced speedup, but also accelerate model sparsity innovations within a unified framework, for the first time

# SparTA: Deep-Learning Model Sparsity via Tensor-with-Sparsity-Attribute

# Q&A

*Presented by Guanbin Xu*

# backup