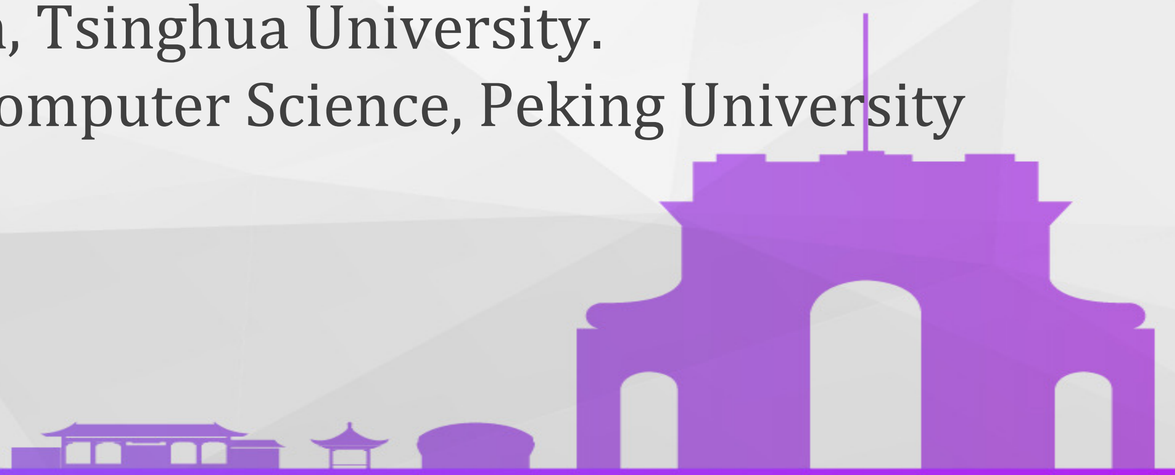
Latency-aware Spatial-wise Dynamic Networks

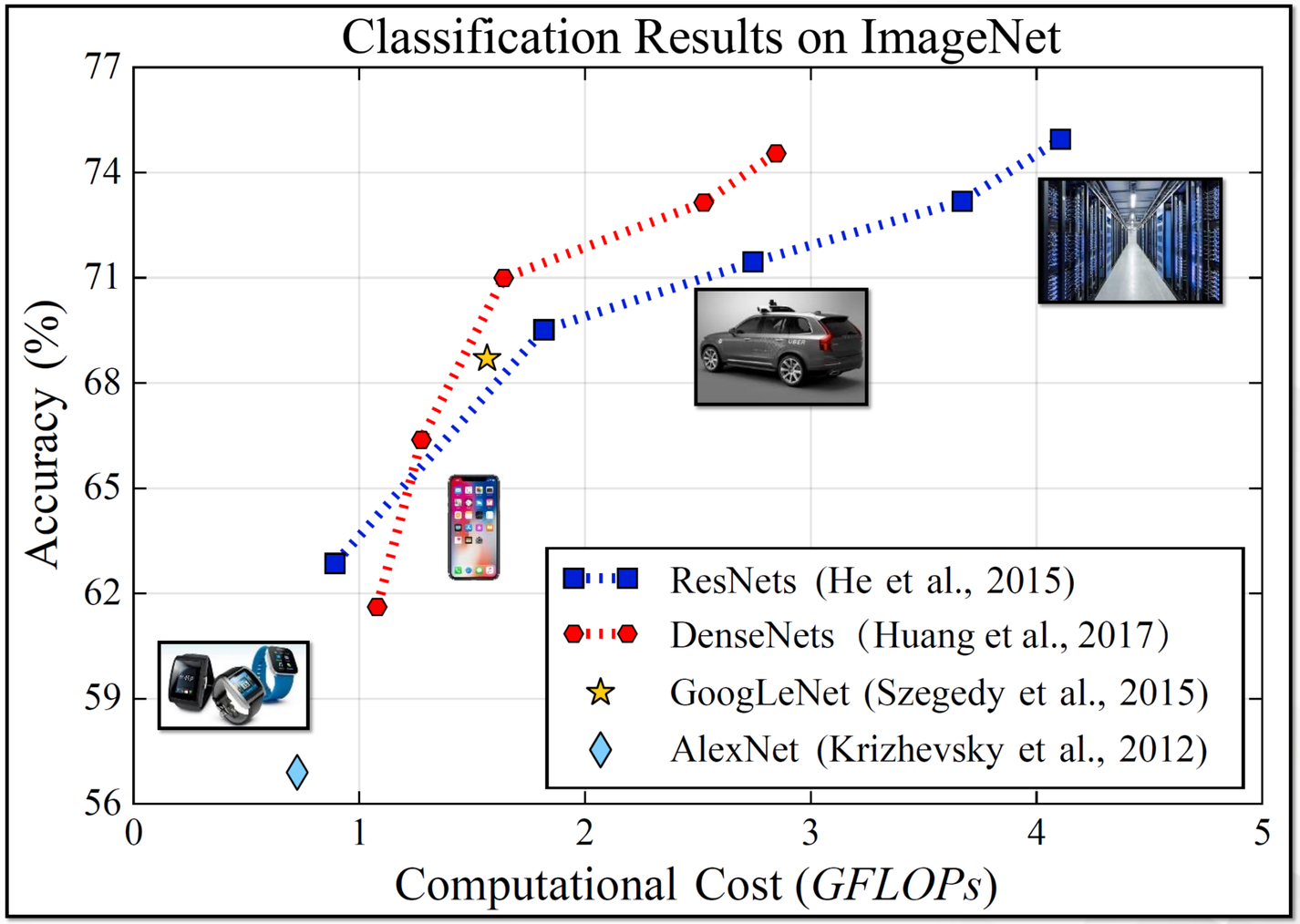
Yizeng Han[*1] , Zhihang Yuan[*2], Yifan Pu[*1] , Chenhao Xue[2] ,
Shiji Song[1] , Guangyu Sun[2], and Gao Huang[1]

[*] Equal contribution

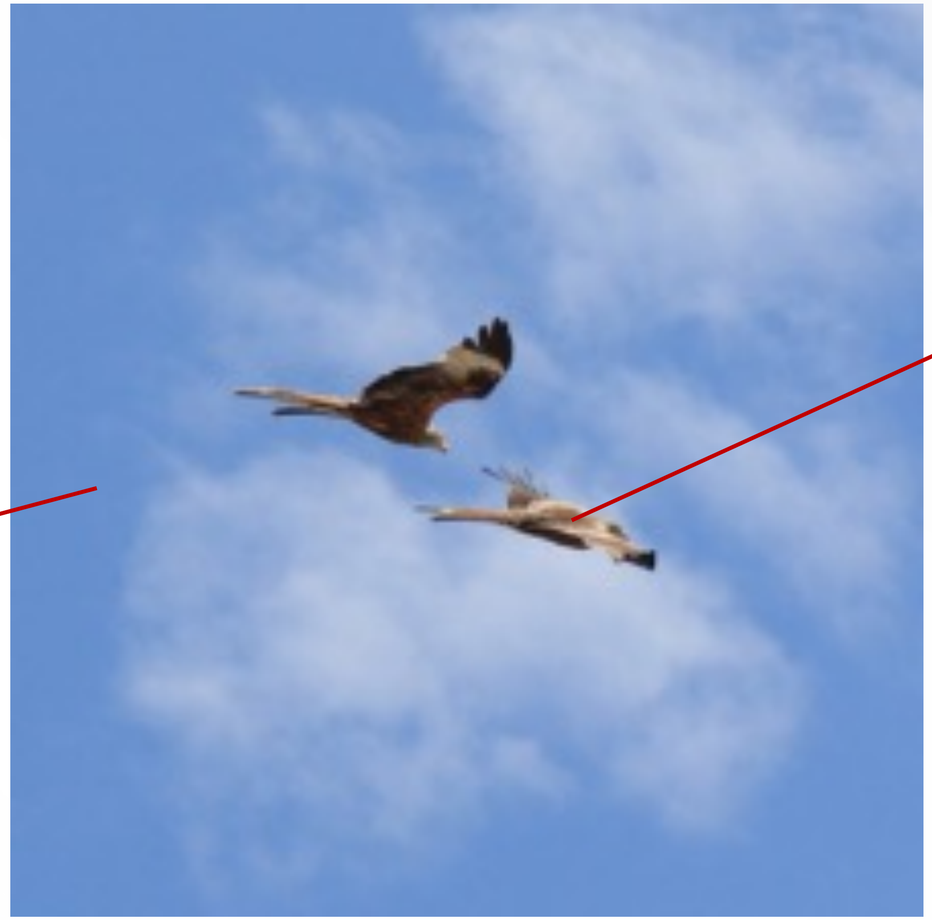
[1]Department of Automation, Tsinghua University.
[2]School of Electronics Engineering and Computer Science, Peking University



Accuracy-Efficiency Tradeoff



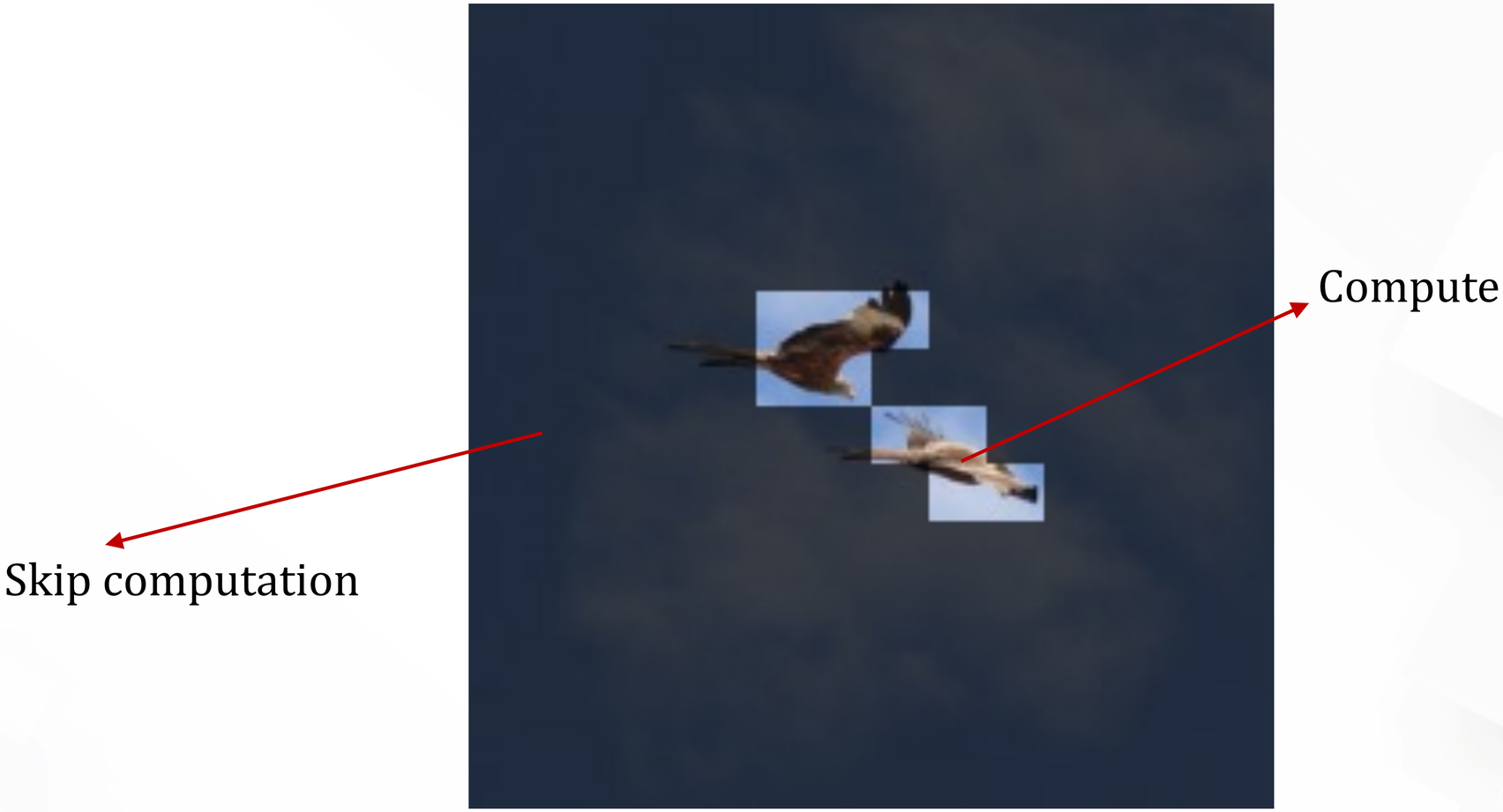
Spatial redundancy in images



Foreground (important) areas

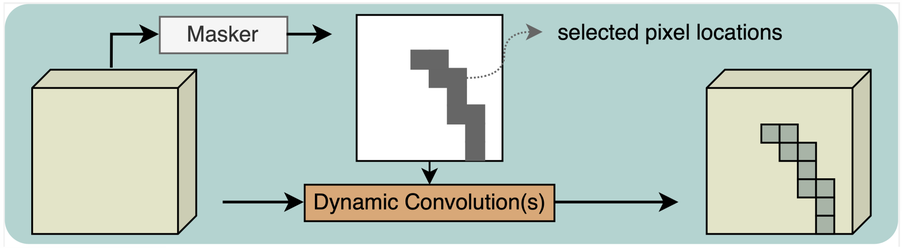
Background (unimportant) areas

Spatially adaptive computation



Spatial-wise dynamic networks

Algorithm design

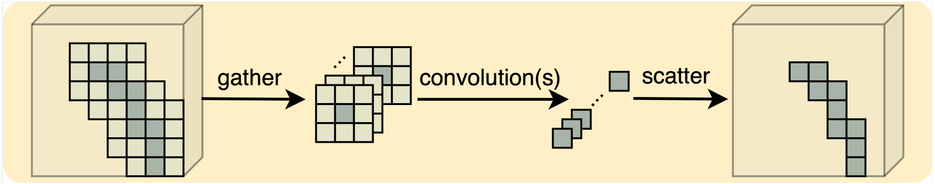


Activation rate $r = \frac{\sum_{i,j} \mathbf{M}_{i,j}}{H \times W} \in [0,1]$

Reduced **Theoretical** Computation

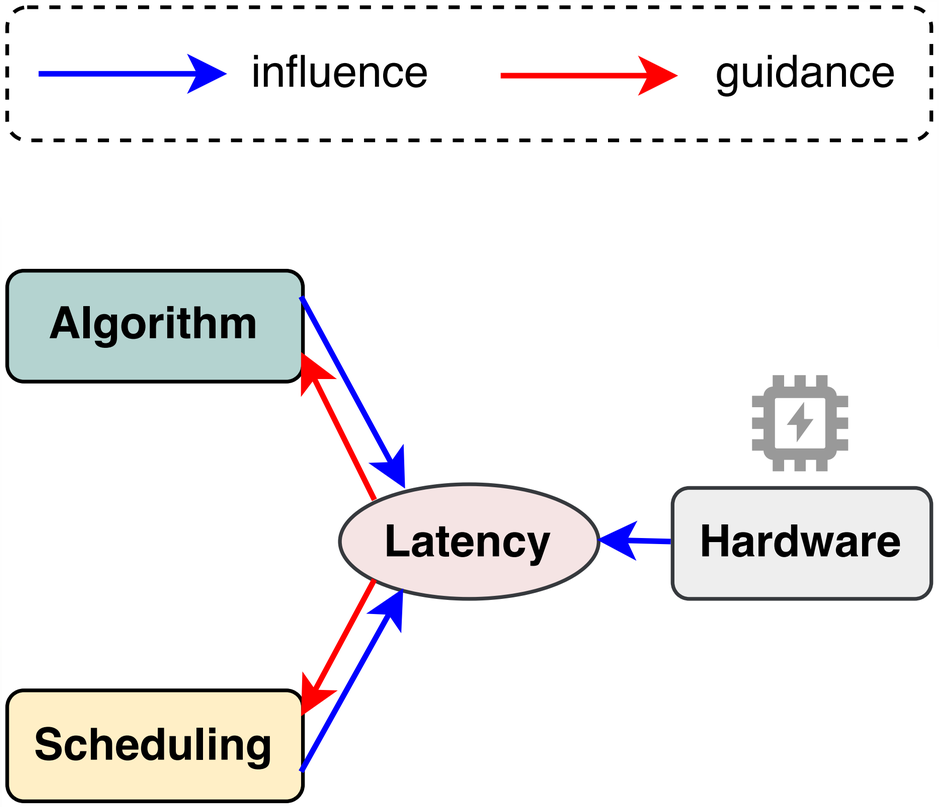
Gap

Practical implementation (**scheduling**)



Low **Practical** Efficiency

Inference latency is influenced by 3 key factors



Given a **hardware** device, we directly use the **latency** to guide:

1. **algorithm design**
2. **scheduling optimization.**

Challenges of achieving realistic speedup

Challenge 1 (algorithm design): pixel-level spatially adaptive computation

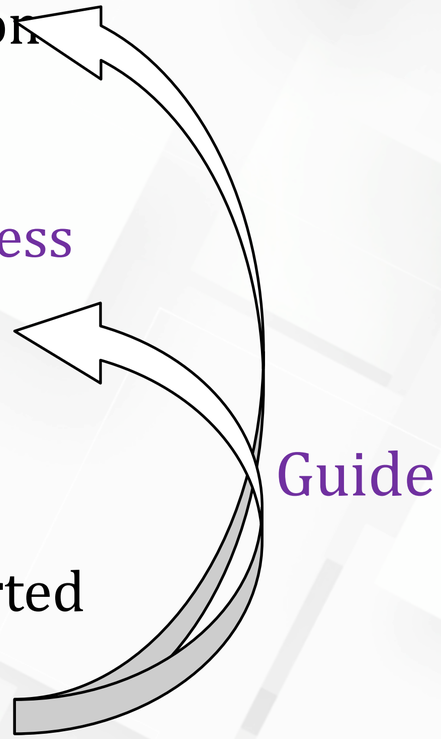
Solution: Coarse-grained (patch-level) spatially adaptive computation

Challenge 2 (scheduling strategies): considerable cost on memory access

Solution: Operator fusion

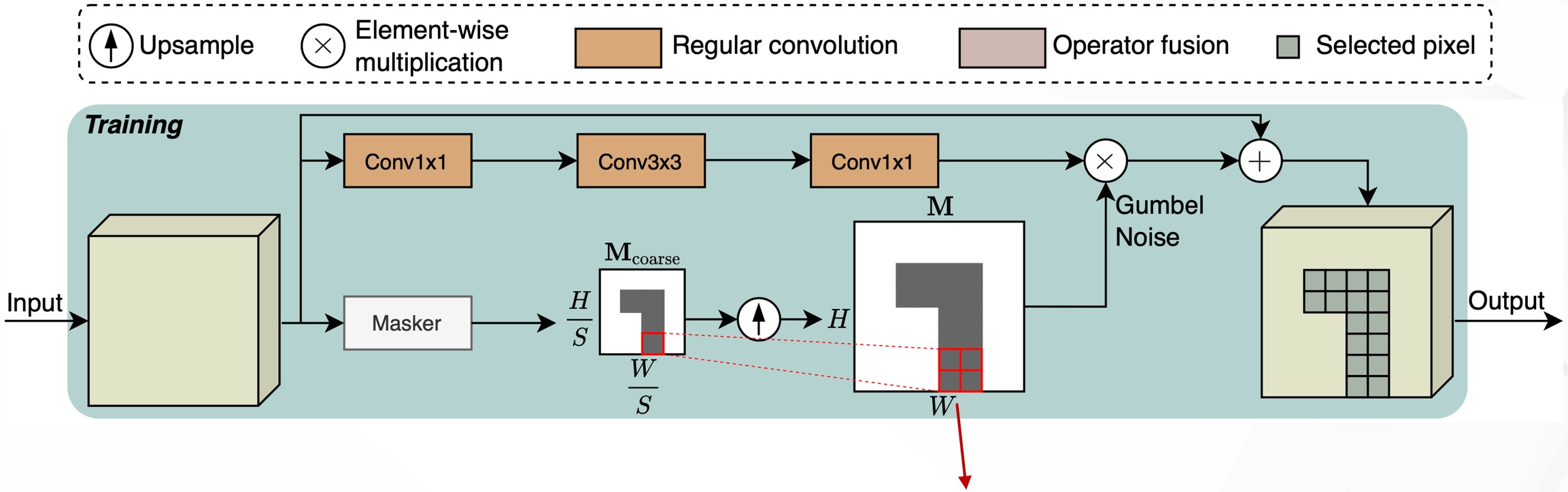
Challenge 3 (awareness of latency): dynamic operators are not supported

Solution: Latency prediction model



1) Algorithm design：

Coarse-grained spatially adaptive computation

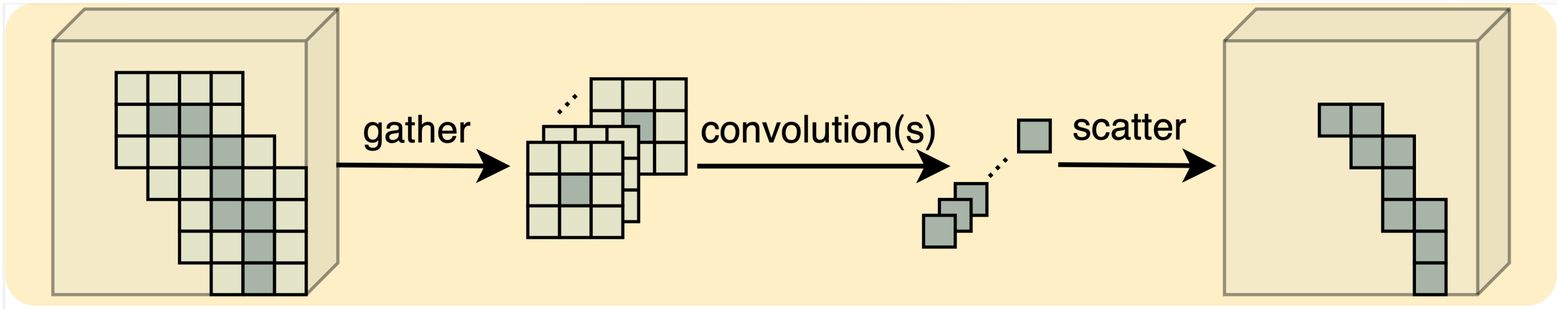


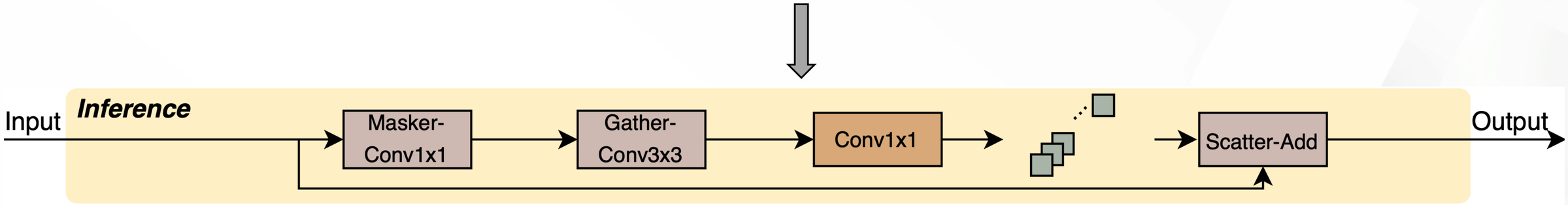
One element on the mask determines the computation of a **patch** rather than a **pixel**

- Larger S: **more contiguous memory access** & **less flexibility**
- An appropriate setting of S needs the guidance of accuracy/latency

2) Scheduling optimization：

Operator fusion



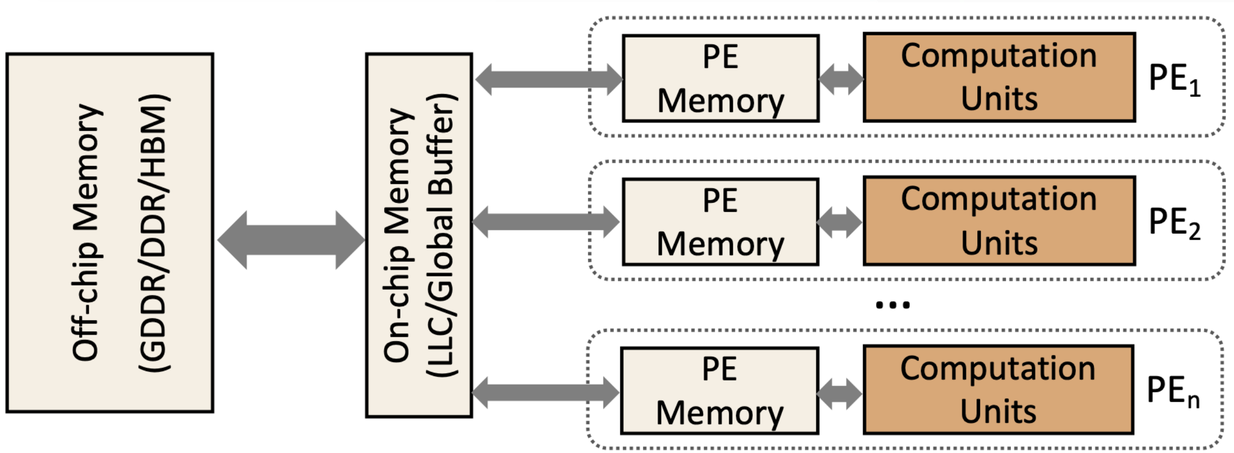


Fusing some operators can significantly reduce the cost on **memory access**.

Masker-Conv1x1	Gather-Conv3x3	Scatter-Add	Latency (μs)
✗	✗	✗	163.2
✓	✗	✗	90.1
✓	✓	✗	86.7
✓	✓	✓	**71.4**

3) Latency prediction model

- Hardware **modeling** (3-level structure)

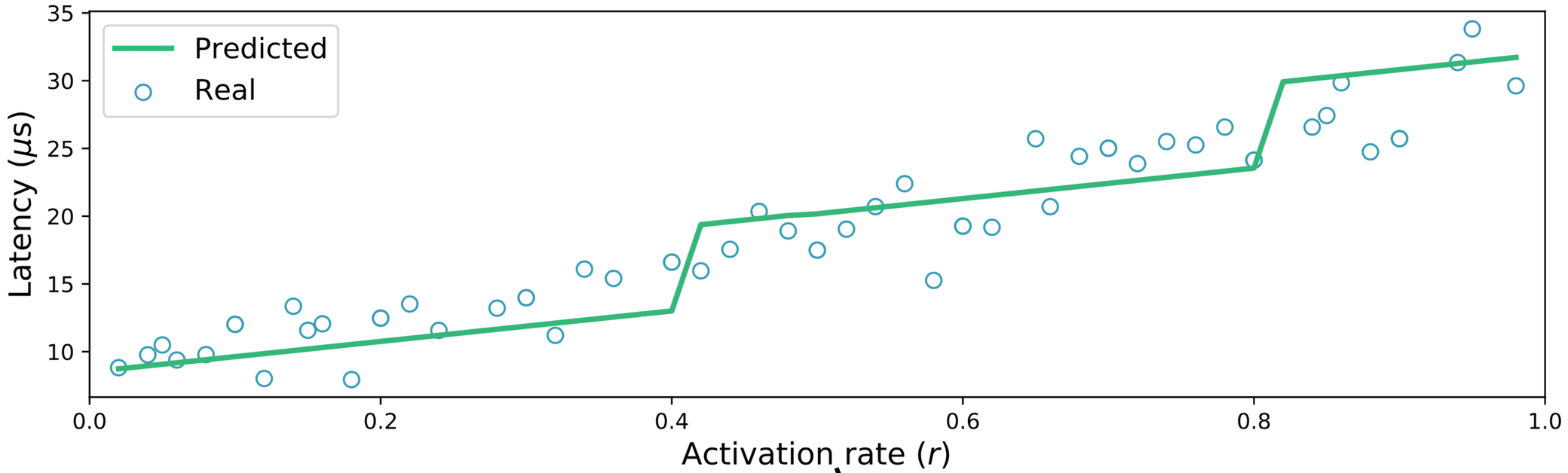


- Latency prediction model:
  - Input: hardware properties $\mathbf{H}$, layer parameters $\mathbf{P}$, spatial granularity S, and activation rate r
  - Output: latency $\ell = g(\mathbf{H}, \mathbf{P}, S, r)$

$$\ell = \ell_{\text{data}} + \ell_{\text{computation}}$$

A larger S leads to
- More **contiguous memory access** (lower ℓ_{data})
- Higher **parallelism of computation** (lower $\ell_{\text{computation}}$)

Performance of the latency prediction model

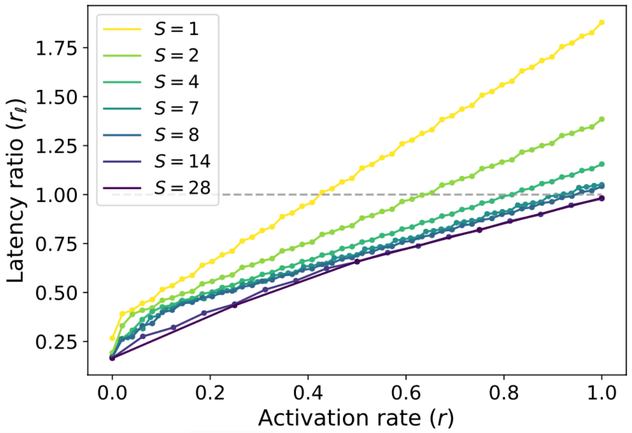


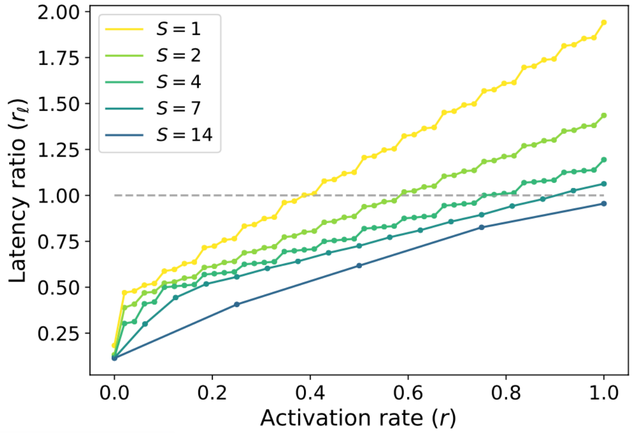
The ratio of **computed** regions

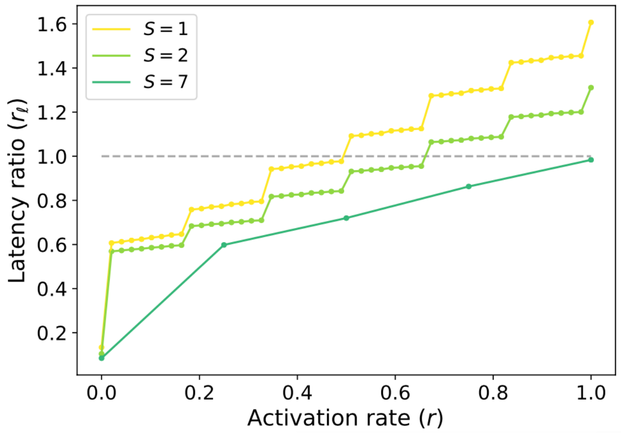
Our latency predictor can **accurately** predict the **practical latency** on a given hardware device.

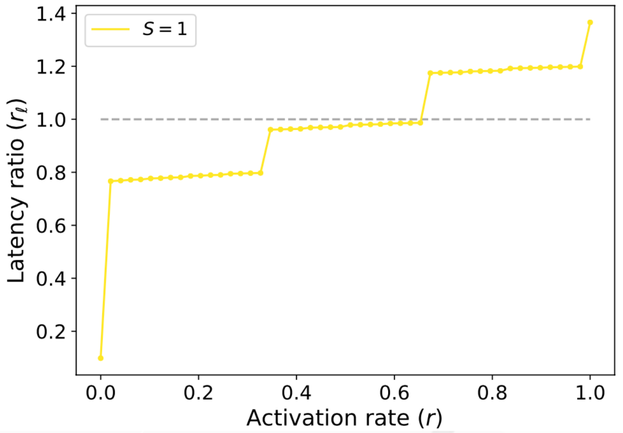
Effect of S on the practical latency

- Activation rate $r \in [0,1]$: theoretical computation ratio
- Latency ratio $r_\ell = \frac{\ell_{\text{dynamic}}}{\ell_{\text{static}}}$
- $r_\ell < 1$: **practical speedup**





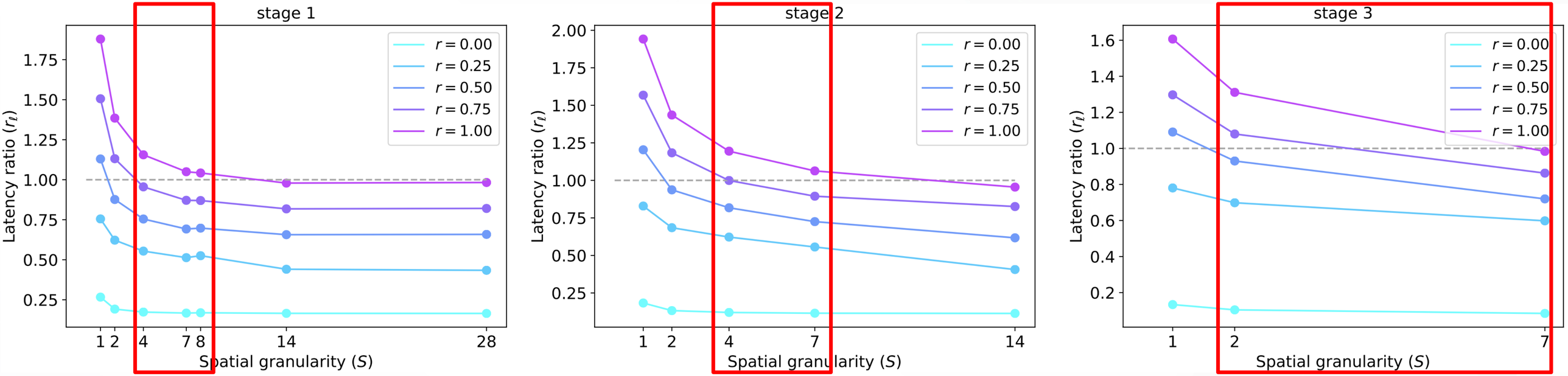




- $S = 1$ cannot always improve the practical efficiency;
- $S > 1$ can alleviate this problem

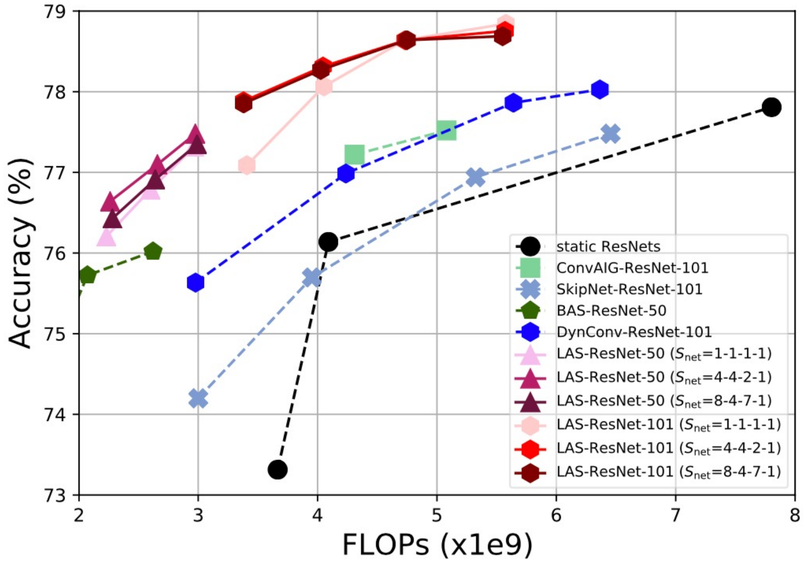
Selection of S

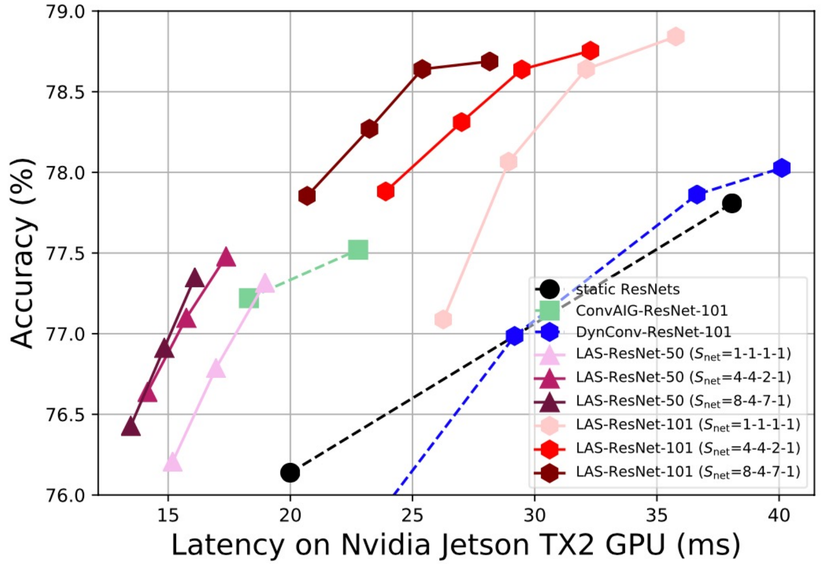
- Relationship between r_ℓ and S

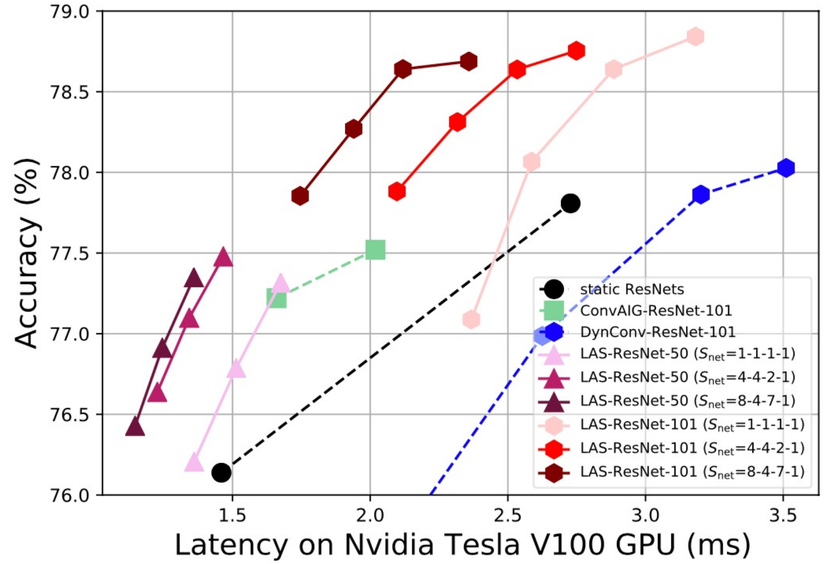


- **Sweet spots balancing efficiency and flexibility:**
  - Stage 1 (feature resolution = 56×56): $S = 4,7,8$
  - Stage 2 (feature resolution = 28×28): $S = 4,7$
  - Stage 3 (feature resolution = 14×14): $S = 2,7$
  - Stage 4: $S = 1$

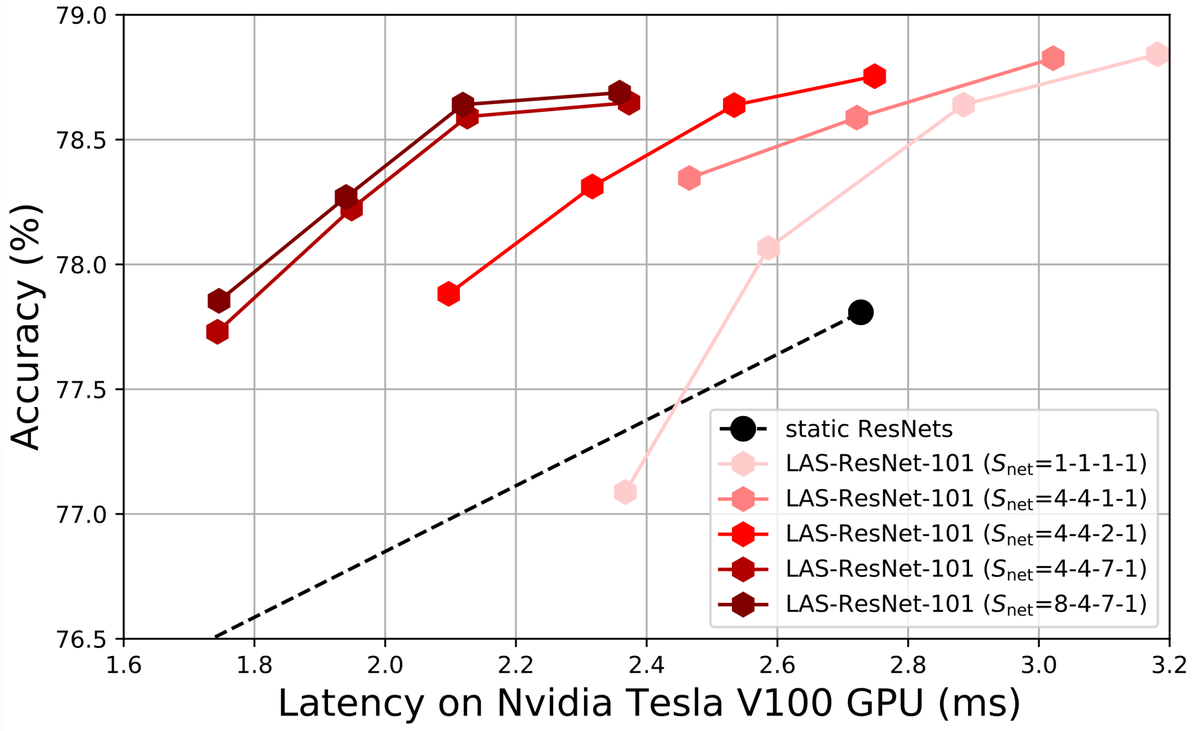
Main results with ResNets

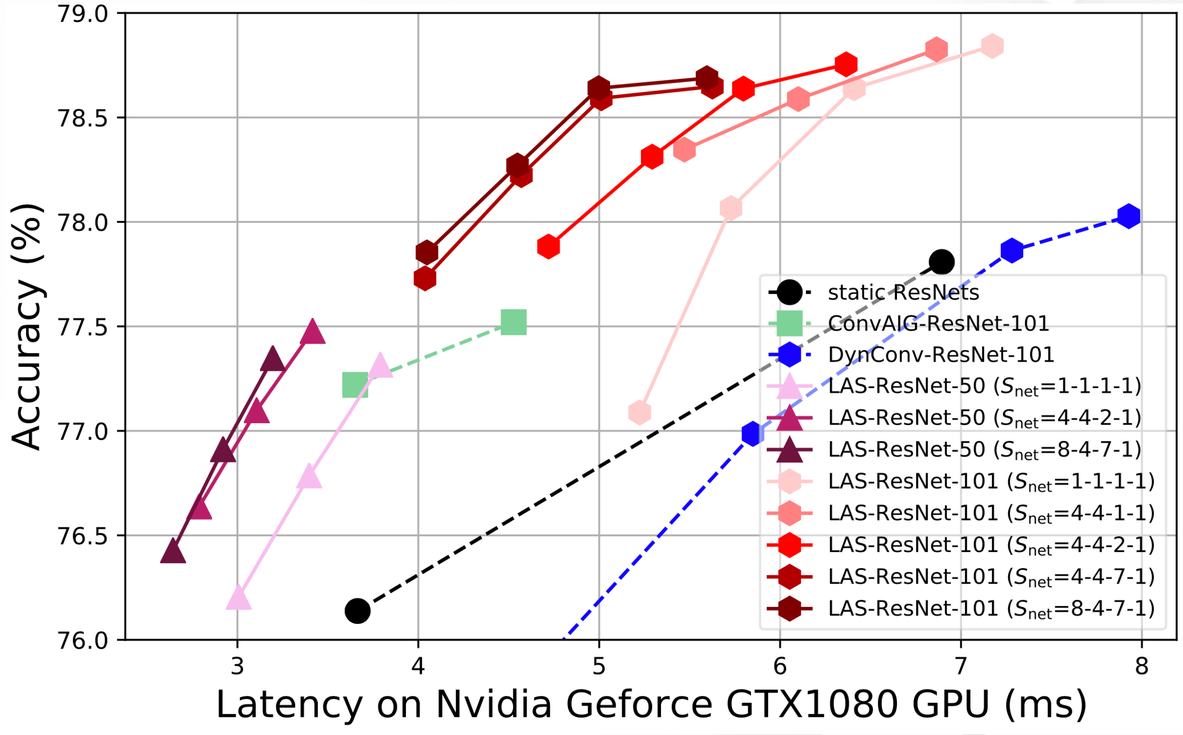






Ablation study of spatial granularity *S*





Visualization results

Results on downstream tasks

Table 2: Object detection results on the COCO dataset.

Detection Framework	Backbone	Backbone FLOPs (G)	Backbone Latency (ms) V100	GTX1080	TX2	mAP (%)
Faster R-CNN	ResNet-101 (Baseline)	141.2	39.5	119.3	729.4	39.4
	LAS-ResNet-101 (S_{net}=4-4-2-1, t=0.6)	90.7	33.8	90.6	524.9	**40.3**
	LAS-ResNet-101 (S_{net}=4-4-2-1, t=0.5)	79.3	30.7	82.5	477.1	39.8
	LAS-ResNet-101 (S_{net}=4-4-7-1, t=0.5)	79.5	29.0	79.9	464.8	40.0
	LAS-ResNet-101 (S_{net}=4-4-7-1, t=0.4)	**67.9**	**25.3**	**69.1**	**401.3**	39.5
RetinaNet	ResNet-101 (Baseline)	141.2	39.5	119.3	729.4	38.5
	LAS-ResNet-101 (S_{net}=4-4-2-1, t=0.5)	77.8	30.4	81.8	472.6	**39.3**
	LAS-ResNet-101 (S_{net}=4-4-7-1, t=0.5)	79.4	28.9	79.9	464.8	**39.3**
	LAS-ResNet-101 (S_{net}=4-4-7-1, t=0.4)	**66.4**	**25.3**	**69.1**	**401.3**	38.9

Results on downstream tasks

Table 3: Instance Segmentation results on the COCO dataset.

Segmentation Framework	Backbone	Backbone FLOPs (G)	Backbone Latency (ms) V100	GTX1080	TX2	AP^{mask} (%)	AP^{box} (%)
Mask R-CNN	ResNet-101 (Baseline)	141.2	39.5	119.3	729.4	36.1	40.0
	LAS-ResNet-101 (S_{net}=4-4-2-1, t=0.5)	80.5	31.1	83.3	481.9	**37.0**	**41.0**
	LAS-ResNet-101 (S_{net}=4-4-2-1, t=0.4)	69.2	27.9	74.8	431.6	36.1	40.0
	LAS-ResNet-101 (S_{net}=4-4-7-1, t=0.4)	**68.8**	**25.8**	**70.9**	**411.8**	36.2	40.0

Conclusion (takeaway messages)

- **Observation**
  - Despite its **theoretical efficiency**, traditional spatially adaptive inference can hardly achieve **realistic speedup** on hardware devices.

- **Challenges & Solutions**
  - Lacking the direct guidance from latency
    - **Latency prediction model**
  - Sub-optimal algorithm
    - **Coarse-grained (patch-level) spatially adaptive computation**
  - Sub-optimal scheduling strategy
    - **Operator fusion**

Thank You!
Welcome to our poster

Paper

Code