insitro

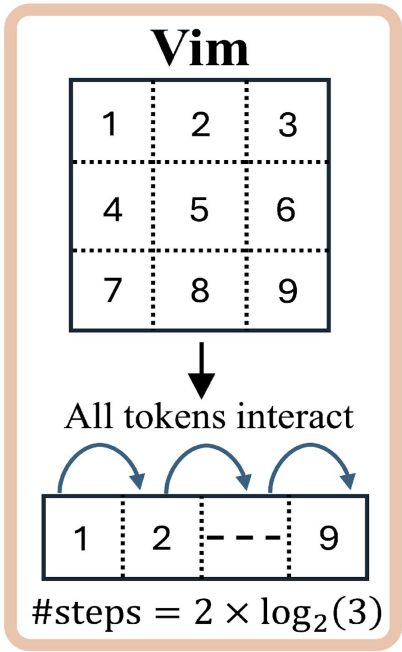
Fast Vision Mamba: Pooling Spatial Dimensions for Accelerated Processing

Saarthak Kapse[1] · Robin Betz[2] · Srinivasan Sivanandan[2]

Oral Presentation, WACV 2026

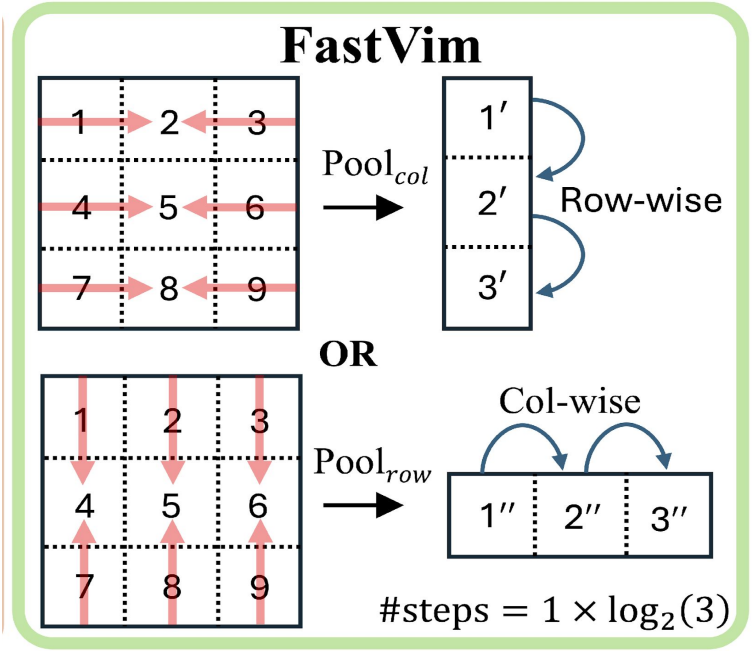
Background

- **Linear Complexity of Mamba:** State Space Models (SSMs) like Mamba achieve linear complexity for token interactions, unlike the quadratic complexity of Vision Transformers.
- **Parallel Efficiency:** Vision Mamba uses a parallel scan algorithm to reduce recurrent steps to a logarithmic scale.
- **Resolution Bottleneck:** The number of tokens in vision tasks scales quadratically with increasing image resolution. This quadratic token scaling translates to a 2x increase in the number of parallel steps, creating a throughput bottleneck for high-resolution images

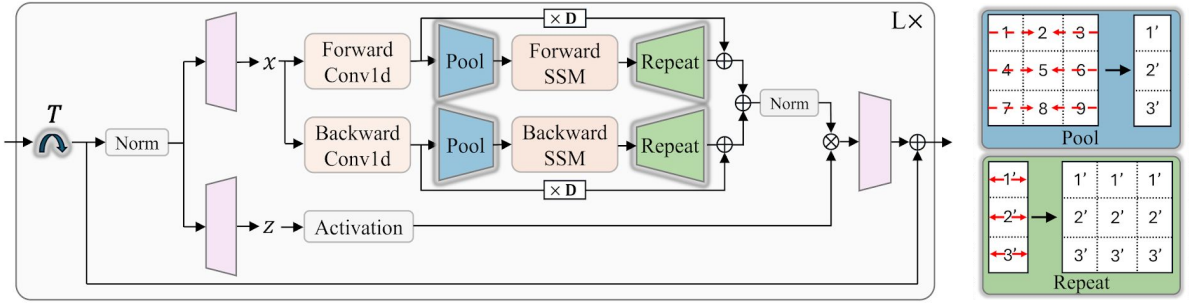


Introducing Fast Vision Mamba

- FastVim reduces the number of recurrent steps in Vision Mamba linearly with image resolution.
- For a square token grid, pooling reduces the sequence length from the square of the dimension to just the dimension itself.
- The architecture applies average pooling across one spatial dimension of the 2D token grid right before the recurrent SSM block.
- This token reduction halves the number of parallel steps required in the SSM block.



FastVim Block Architecture



- The input image tokens pass through a 1D convolution layer & Mean pooling compresses the tokens into a one-dimensional grid.
- Compressed tokens undergo parameter projection and selective scan in the SSM module and the output is repeated to decompress and restore the original token grid size before the skip connection.
- The number of tokens remains unchanged across blocks due to this compress-and-decompress mechanism.

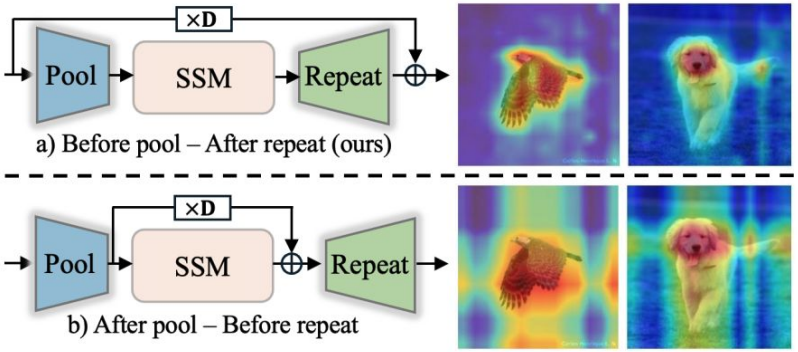
The Necessity of Alternating Pooling

- Pooling tokens exclusively across rows/columns prevents tokens within the same column/row from interacting.
- FastVim solves this by transposing the token grid every block to alternate the pooling dimensions.
- Alternating the pooling dimensions ensures all tokens interact implicitly across multiple stacked blocks. Empirical evidence shows this alternation is strictly necessary for high visual encoding performance.

Model	FastVim-S	Pool_{col}	Pool_{row}
Top-1 (%)	81.1	80.0	79.6

Optimizing Skip Connection Placement

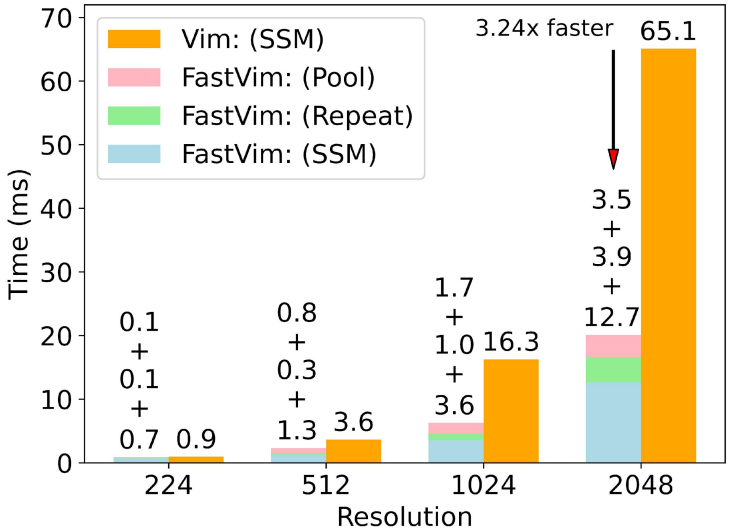
- The standard SSM module includes an inherent skip connection (**D**). Keeping it inside the module (i.e. after-pool / before-repeat in FastVim) causes a significant performance drop.
- Moving the connection entirely outside (before-pool -> after-repeat) bypasses downsampling. This retains fine spatial details and eliminates the repetitive artifacts.



Model	Before **D** (default)	After **D**
Top-1 (%)	81.1	78.7

Computational Efficiency and Speedup

- FastVim reduces FLOPs by up to 38% compared to baseline Vim models at higher resolutions.
- The SSM scan time in FastVim scales linearly with resolutions from 224 to 2048 whereas in baseline Vim, SSM scan time increases by up to 74x when resolution is increased 8x.
- At a resolution of 2048, FastVim processes the Forward and Backward SSM layers 3.24x faster than Vim.



c) Processing time (ms)

Supervised Classification: Imagenet 1k

- FastVim models perform on-par with baseline Vim models on ImageNet-1k classification across all model sizes.
- FastVim-Base achieves 82.6% Top-1 Accuracy on ImageNet-1k.

Model	#Params (M)	Top-1 (%)
Vim-T	7M	76.1
Vim-S	26M	80.3
Vim-B	98M	81.9
Vim-B w/ LN	98M	82.6
FastVim-T	7M	75.4
FastVim-S	26M	81.1
FastVim-B	98M	82.6

Semantic Segmentation & Object Detection

- **Semantic Segmentation (ADE20K):** We conducted experiments using UperNet as our segmentation framework. Our FastVim models consistently outperform DeiT while achieving performance on par with the baseline Vim

- **Object Detection and Instance Segmentation (MSCOCO):** We evaluated our backbones using Cascade Mask R-CNN.

Backbone	mIoU
DeiT-T	39.2
DeiT-S + MLN	43.8
DeiT-B + MLN	45.5
Vim-T	41.0
Vim-S	44.9
FastVim-T	41.8
FastVim-S	44.9
FastVim-B	47.8

Semantic Segmentation

Backbone	AP^{box}	AP^{box}_{50}	AP^{box}_{75}	AP^{mask}	AP^{mask}_{50}	AP^{mask}_{75}
DeiT-T	44.4	63.0	47.8	38.1	59.9	40.5
Vim-T	45.7	63.9	49.6	39.2	60.9	41.7
Vim-S*	47.1	65.8	50.7	40.6	62.9	43.5
FastVim-T	45.1	63.7	48.5	39.0	60.8	41.6
FastVim-S	48.4	67.2	52.2	41.8	64.3	44.7
FastVim-B	50.0	68.7	54.2	43.2	66.0	46.6

Object Detection & Instance Segmentation

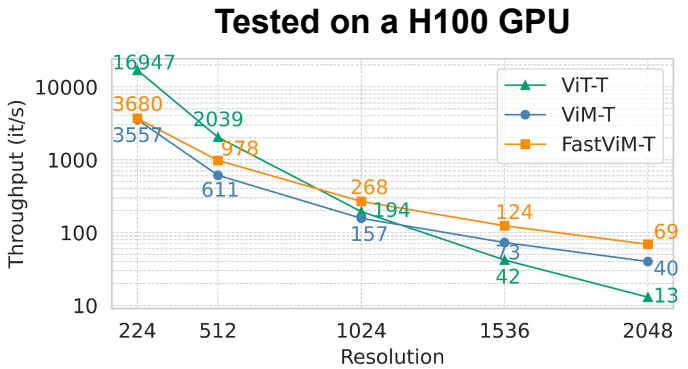
FastChannelVim for Multi-Channel Imaging

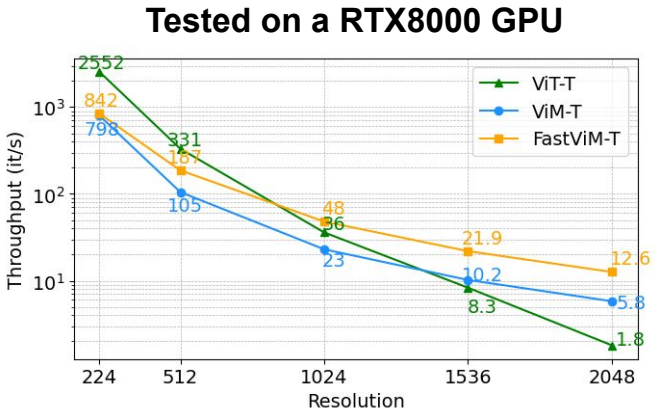
- Standard tokenization is inadequate for domains like microscopy cell imaging, where per-channel information is highly complementary
- We found that when using large token sequences (with a patch size of 8), our full-contextualization ChannelVim significantly surpasses ChannelViT by approximately 8%
- FastChannelVim-S maintains this similar high performance to ChannelVim-S while delivering a 62.3% throughput speedup.

Method	Token Grid	Top-1 (%)
ViT-S/16	14^2	58.9
Vim-S/16	14^2	61.0
ChannelViT-S/16	$14^2 \times 8$	68.6
ChannelVim-S/16	$14^2 \times 8$	73.5
FastCha.Vim-S/16	$14^2 \times 8$	73.6
ViT-S/8	28^2	67.6
Vim-S/8	28^2	66.4
ChannelViT-S/8	$28^2 \times 8$	74.8
ChannelVim-S/8	$28^2 \times 8$	83.0
FastCha.Vim-S/8	$28^2 \times 8$	83.1

Inference Throughput

- The throughput advantage of FastVim relative to Vim continually improves and widens as the image resolution increases
- At resolutions of 1024 and higher, FastVim successfully outperforms ViT in terms of pure processing speed.
- This efficiency ultimately translates to nearly a 72.5% speedup in the overall model framework for high-resolution images (2048x2048).





Conclusion

- FastVim provides a purely parameter-free technique to dramatically accelerate contextualization in SSM scans.
- Throughput speedups widen significantly as image resolutions increase.
- The method retains strong predictive power and matches or exceeds baseline performance in classification, segmentation, and object detection.

https://github.com/insitro/FastVim