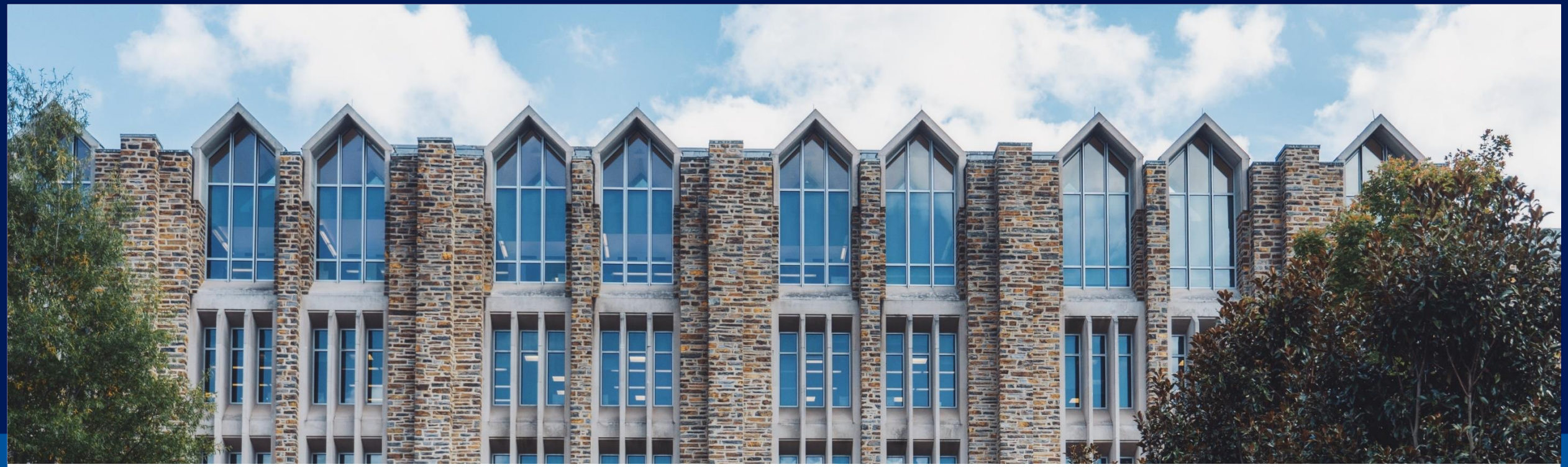


Prosperity: Accelerating Spiking Neural Networks via Product Sparsity

Chiyue Wei, Cong Guo, Feng Cheng, Shiyu Li, Hao “Frank” Yang, Hai “Helen” Li, Yiran Chen

Duke

March 4th, 2025

Acknowledgement

Cong Guo
Duke University

Feng Cheng
Duke University

Shiyu Li
Duke University

Hao “Frank” Yang
Johns Hopkins University

Hai “Helen” Li
Duke University

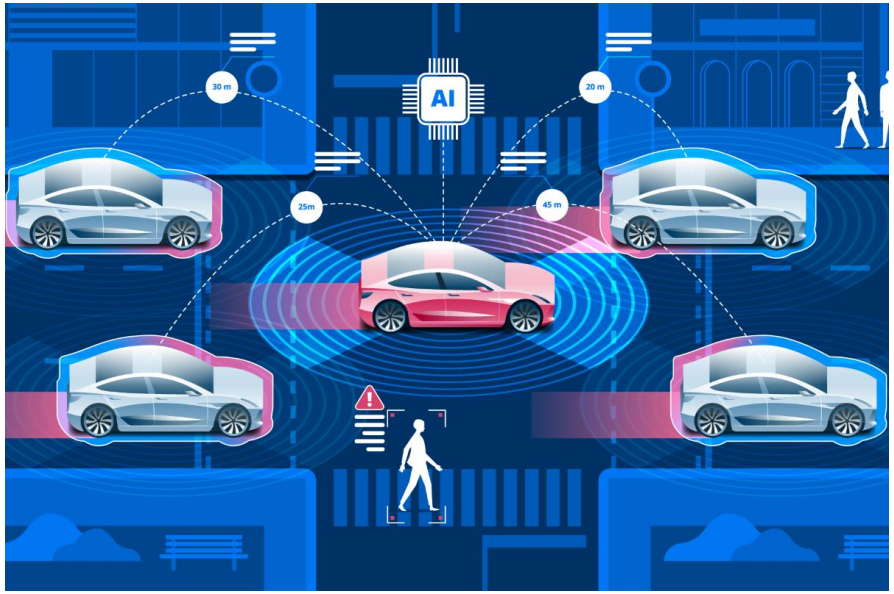
Yiran Chen
Duke University

Contents

Duke

Spiking Neural Networks (SNNs)

- Brain-inspired Neural Networks
- Low energy consumption
- Resource-constrained scenario



Autonomous Driving

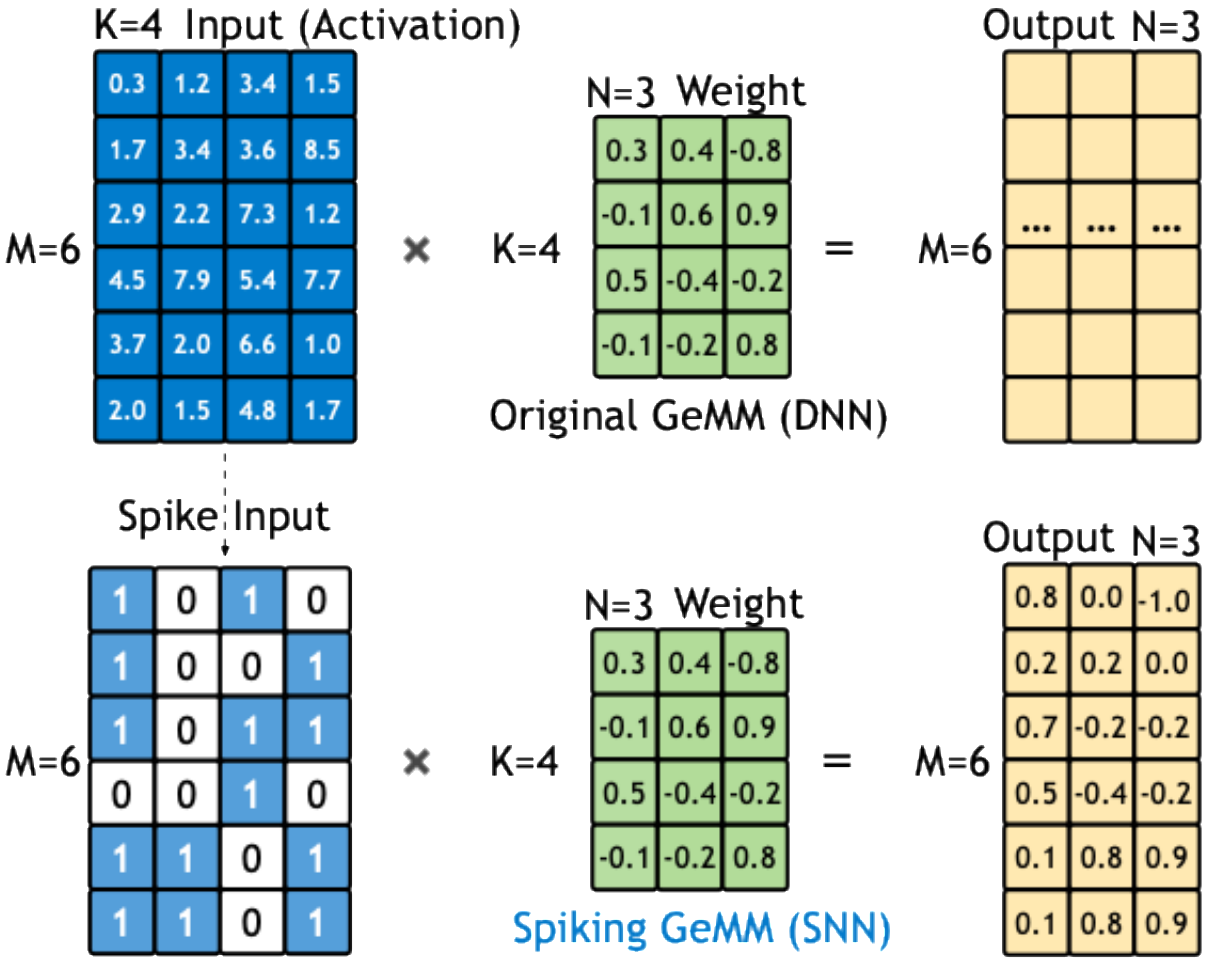
Augmented Reality

Duke

Image credit: https://www.smartcitiesworld.net/opinions/driving-autonomous-vehicles-forward-with-intelligent-infrastructure
https://gazeintelligence.com/eye-tracking-solutions

Computation in SNNs

- Similar to DNN computation
- Difference: Binary Spike Input
  - Inherent sparsity: **BitSparsity**

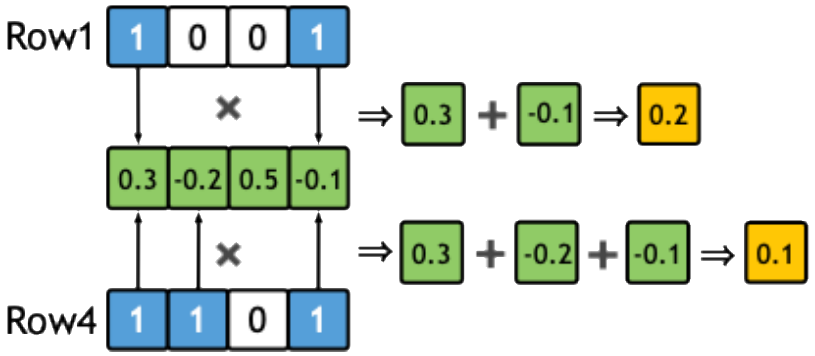


Is BitSparsity Enough?

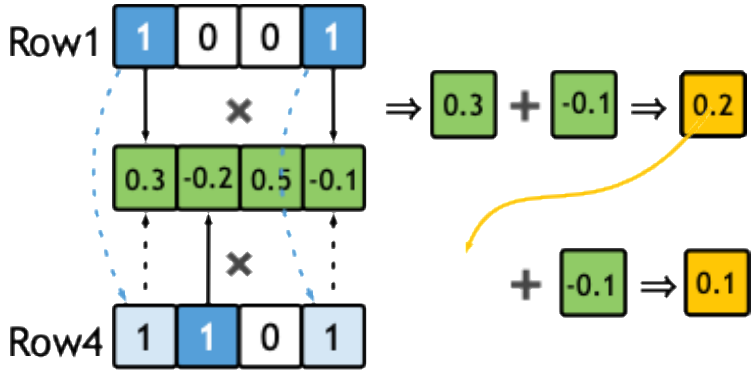
- BitSparsity: Does not fully harness the potential sparsity in SNNs

- New sparsity opportunities: **Product Sparsity** (**ProSparsity**)

BitSparsity in Inner Product
2 Ops + **3** Ops = **5** Ops



ProSparsity in Inner Product
2 Ops + **1** Ops = **3** Ops



Duke

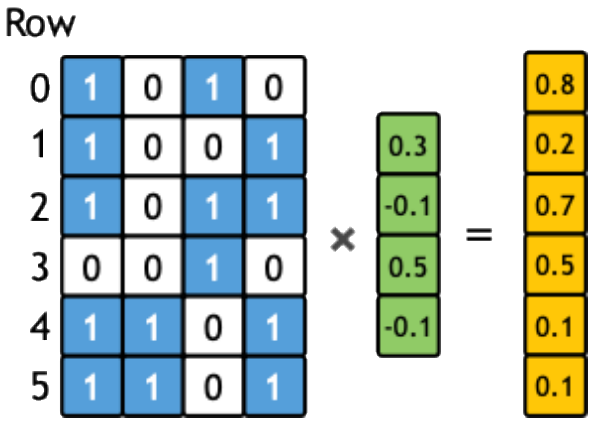
Contents

- Background & Motivation
- **Product Sparsity**
- Prosperity Architecture
- Results

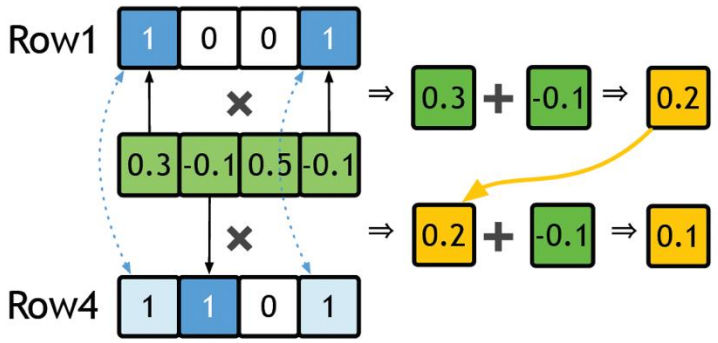
Duke

Product Sparsity (ProSparsity)

- Leverage the similarities between binary rows to skip operations
- Two types of **spatial** relationship between rows



Spiking GeMM



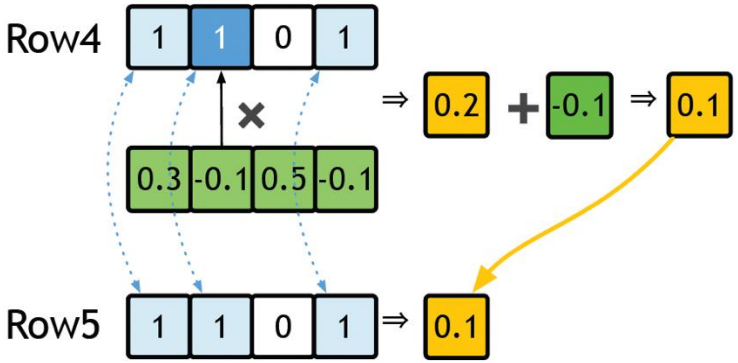
Partial Match (PM)

$S_1 \subset S_4$,

$S_1 = \{0, 3\}, S_4 = \{0, 1, 3\}$

Row4 reuses result of Row1

Save 2 OPs



Exact Match (EM)

$S_4 = S_5$,

$S_1 = \{0, 1, 3\}, S_4 = \{0, 1, 3\}$

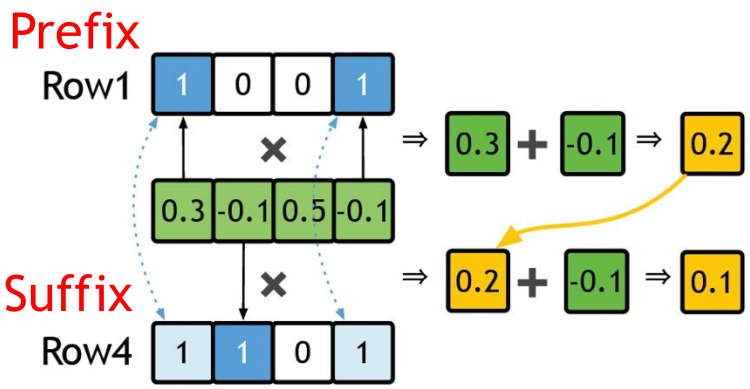
Row5 reuses result of Row4

Save 3 OPs

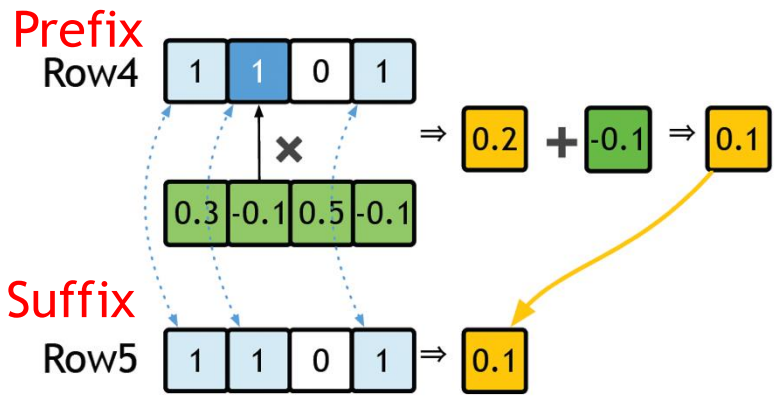
Duke

Temporal order in ProSparsity

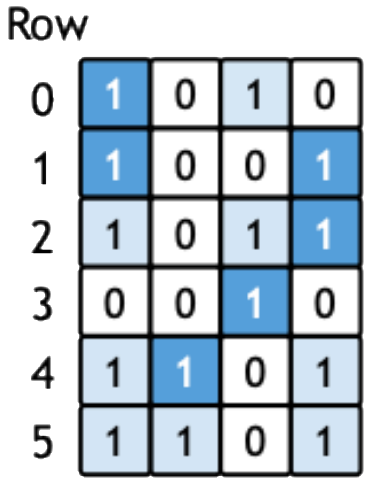
- Define **temporal relationship** within spatial relationships
- Specific **temporal order** has to be followed in execution



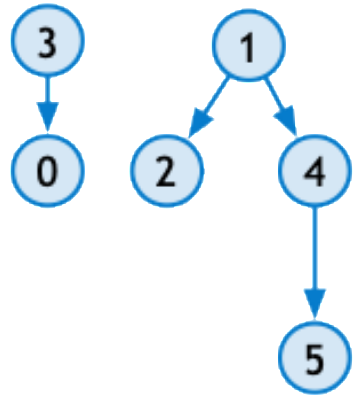
Partial Match (PM)



Exact Match (EM)



Spike Matrix

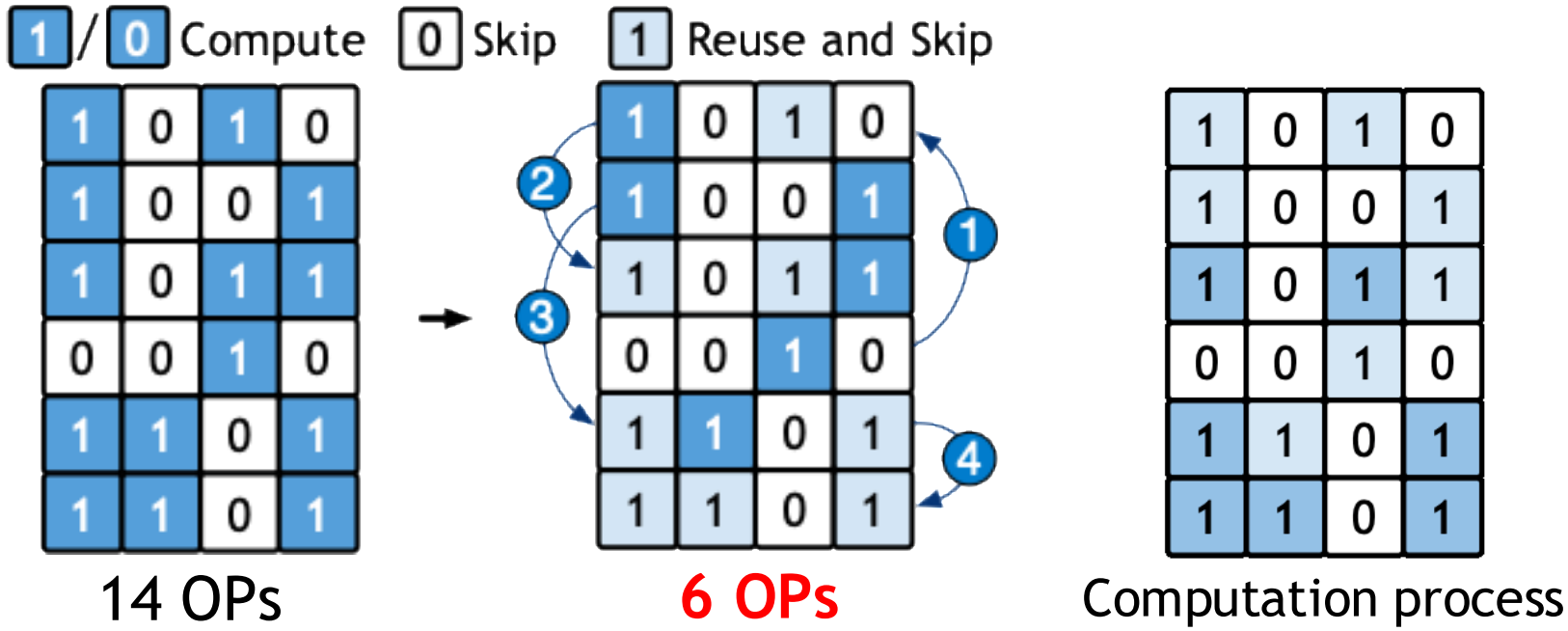


ProSparsity Forest

Duke

Effect of ProSparsity

- Reduce Ops significantly

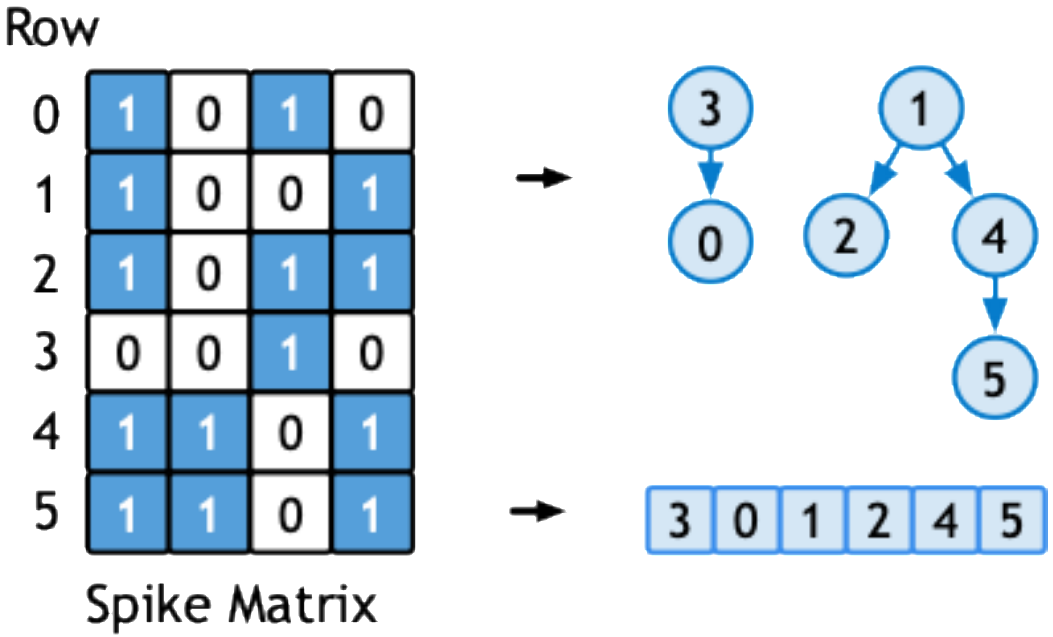


- Num Ops reduced by **3.6x** on Spiking ResNet-18

Duke

Challenges in ProSparsity

- Spike activation is **dynamic**
- Generate ProSparsity **on the fly**



Detect spatial and temporal relationships

Generate temporal order

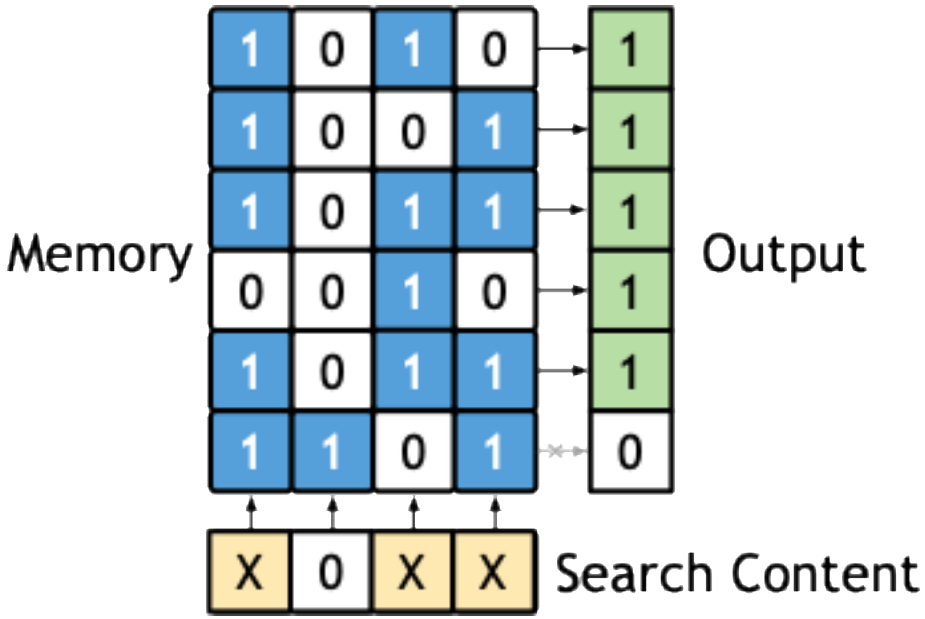
Duke

Solution to Challenges

- Efficiently generates ProSparsity on the fly

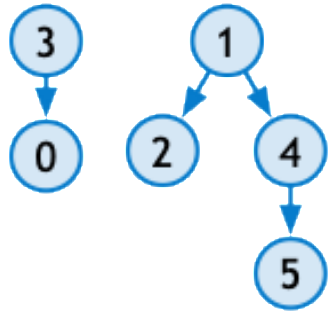
Fast Relationship Detection

TCAM

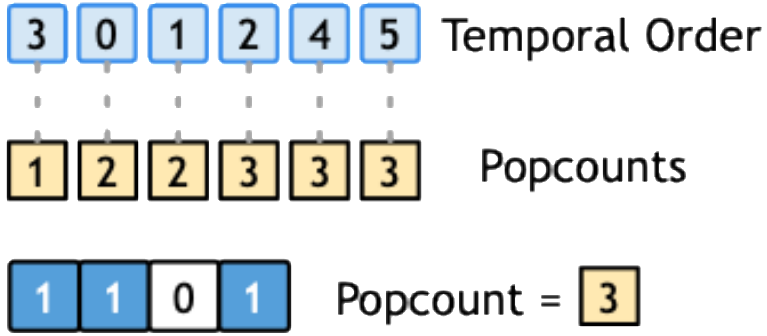


Efficient Temporal Order Generation

Popcounts (number of ones) as temporal order



ProSparsity Forsest



Duke

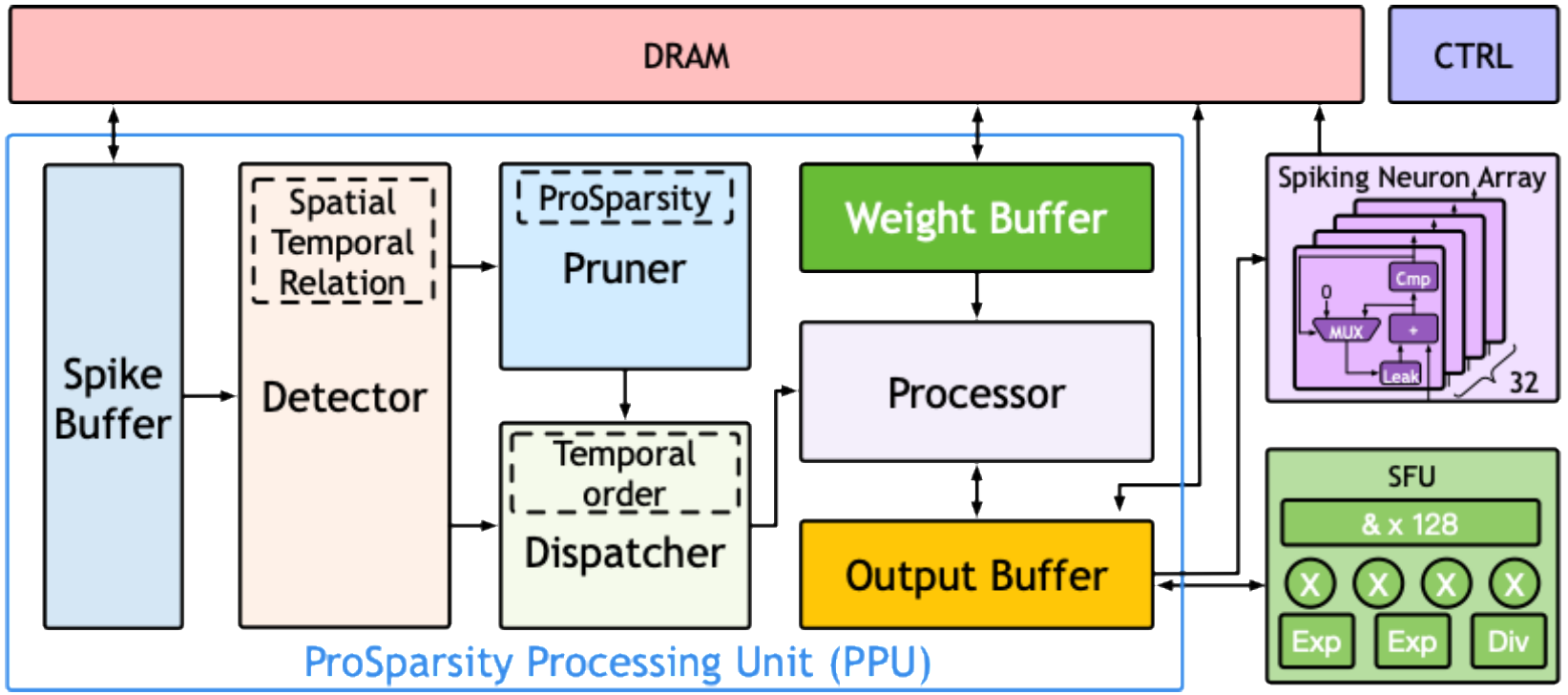
Contents

- Background & Motivation
- Product Sparsity
- **Prosperity Architecture**
- Results

Duke

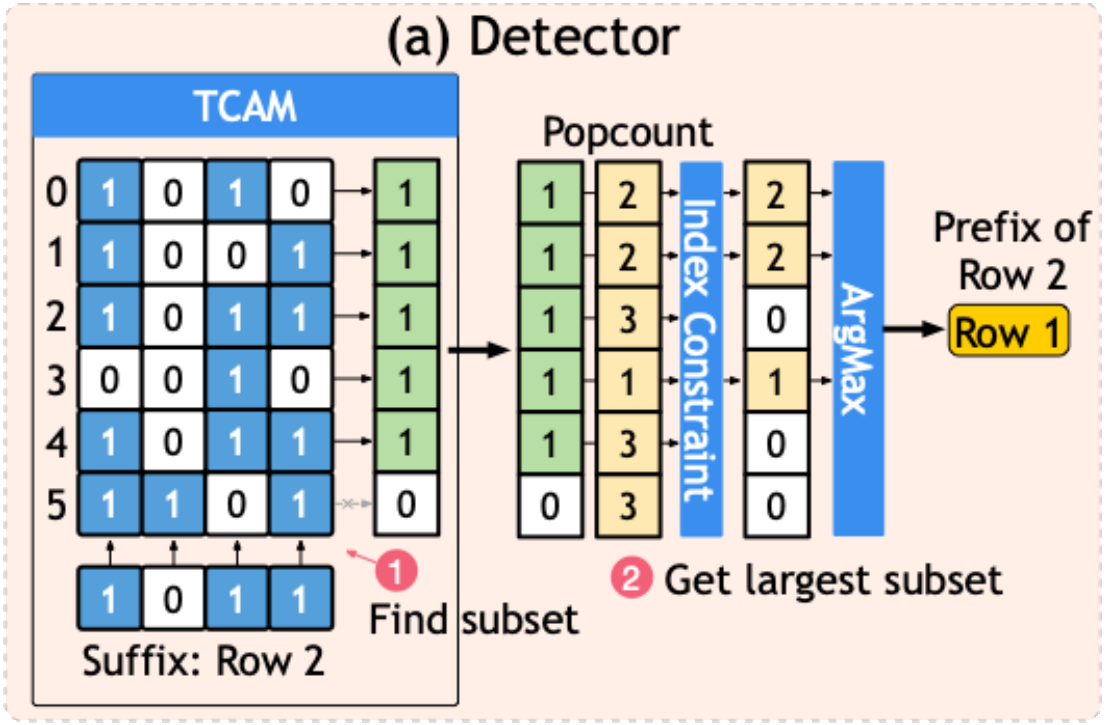
Architecture Overview

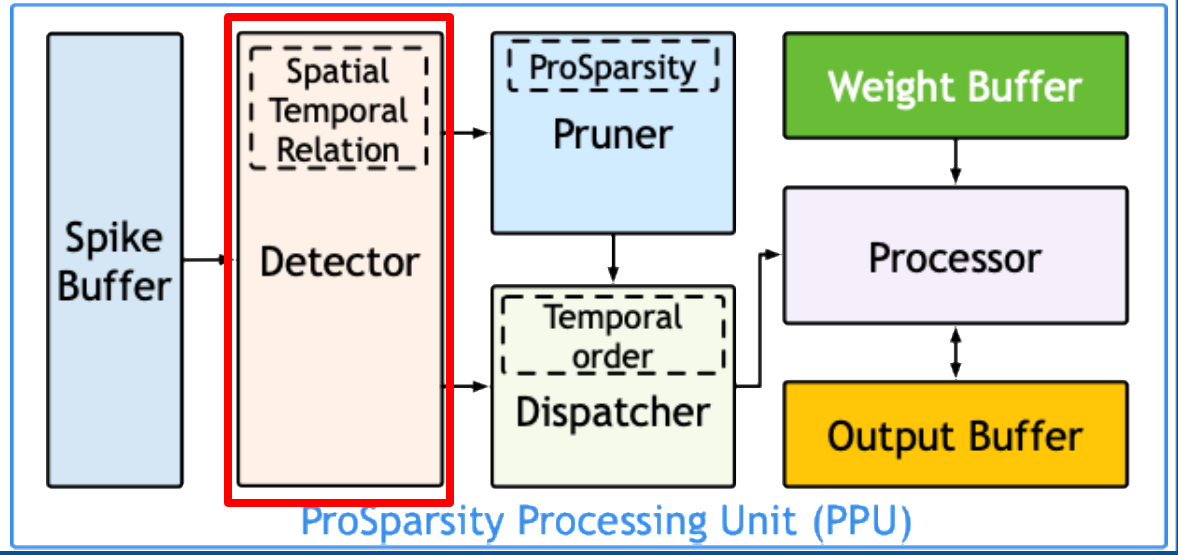
- An architecture generate and utilize ProSparsity
- The main component: ProSparsity Processing Unit (PPU)



Detector

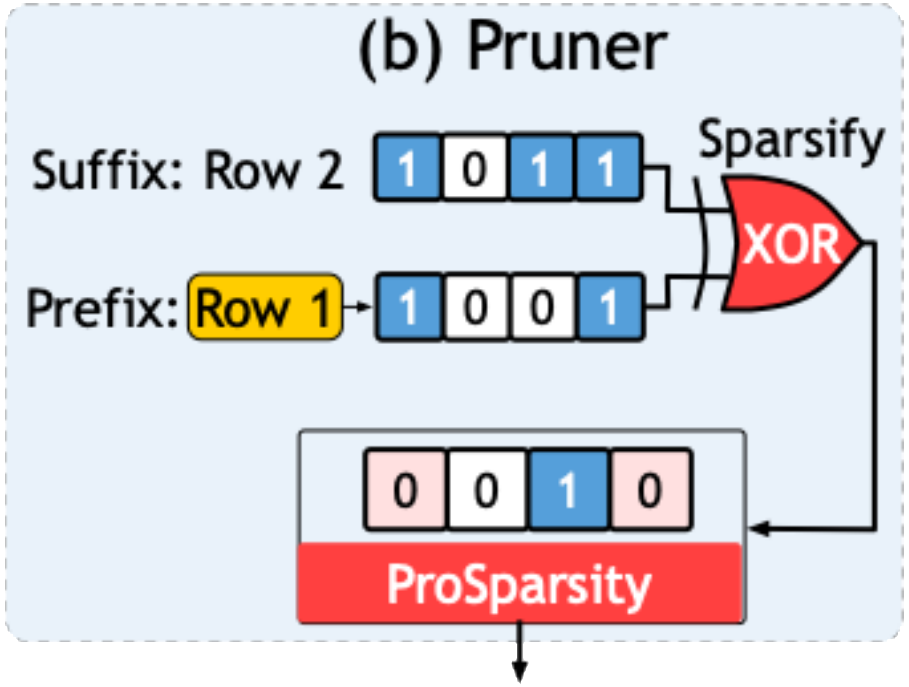
- Detect a prefix for a suffix
  - 1. Find subset of a suffix row
  - 2. Get the largest subset
  - 3. The largest subset as prefix

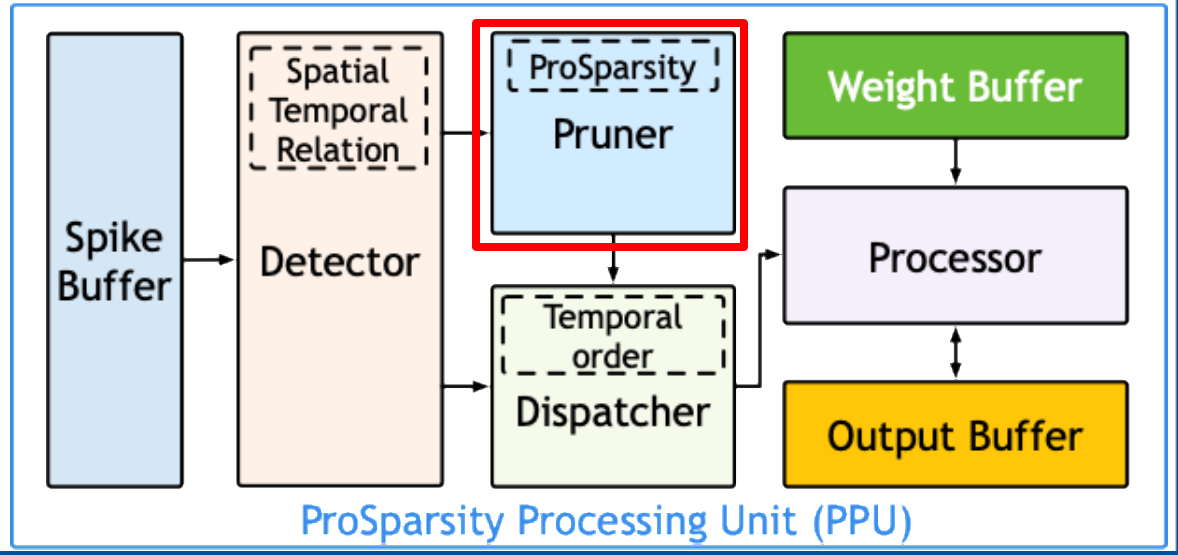




Pruner

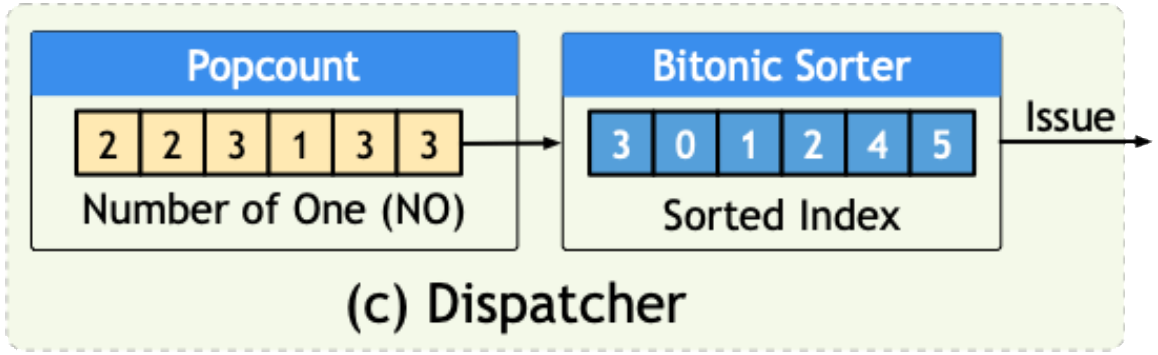
- Generate ProSparsity
  - 1. Get suffix and prefix row
  - 2. Perform bit-wise **XOR**
  - 3. Get ProSparsity for suffix

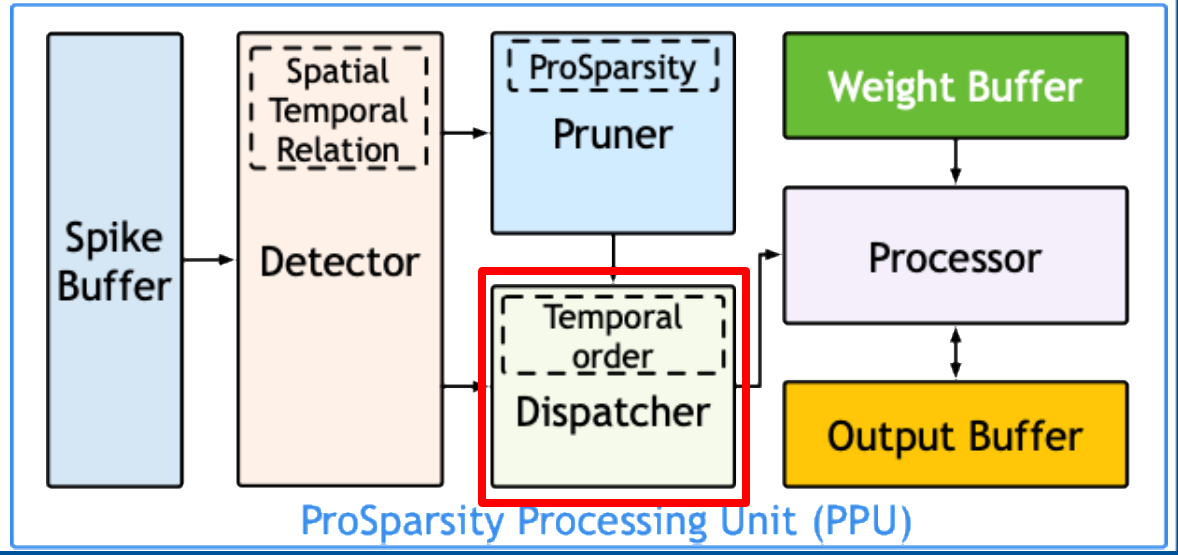




Dispatcher

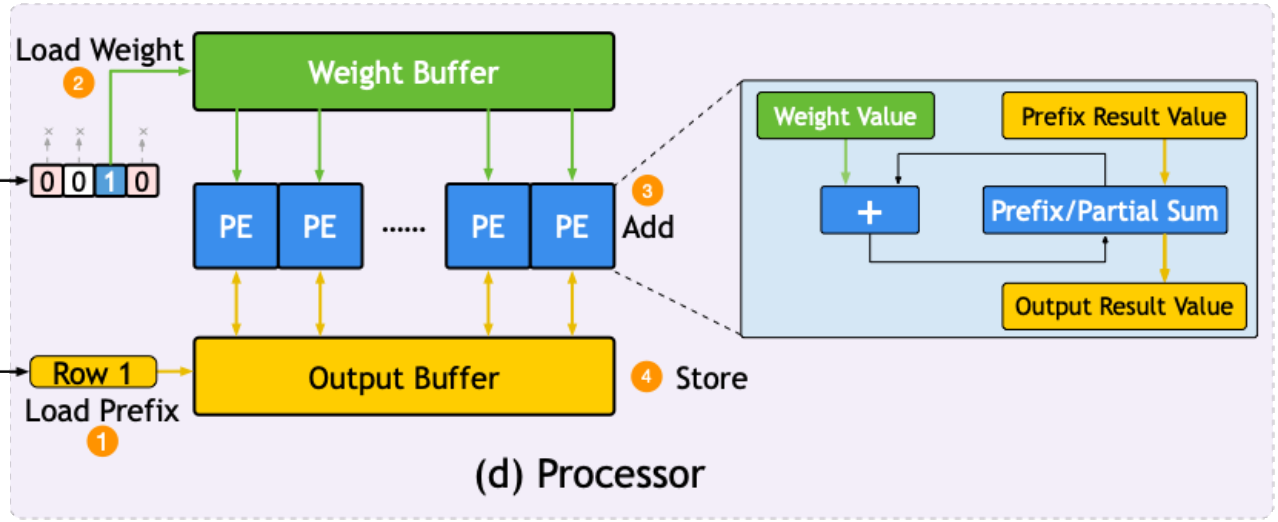
- Generate Temporal Order
  - 1. Get Popcount for each row
  - 2. Sort according to popcount
  - 3. Issue to Processor in sorted order

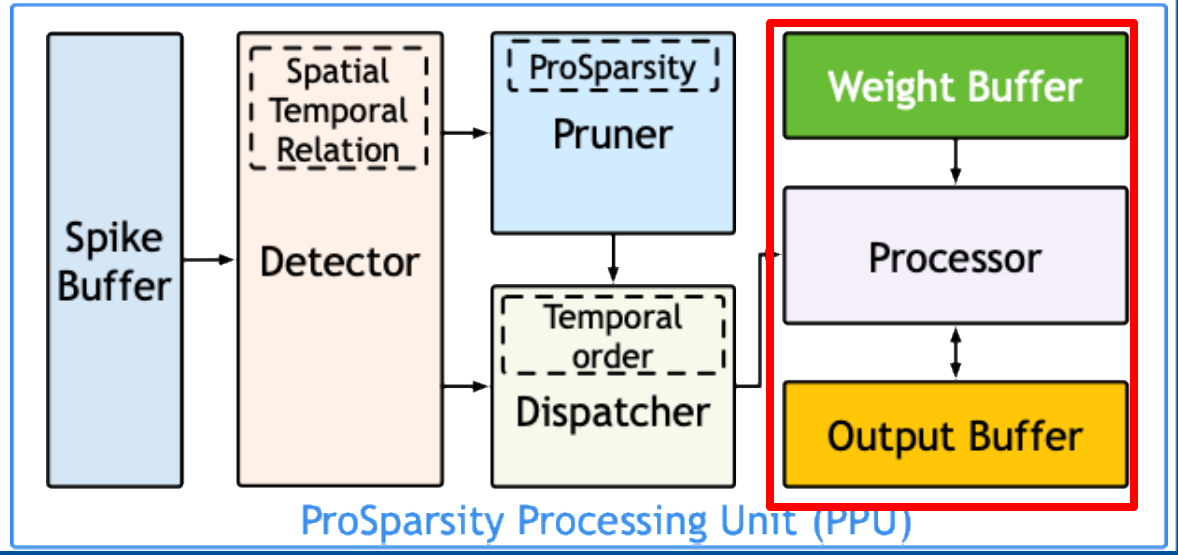




Processor

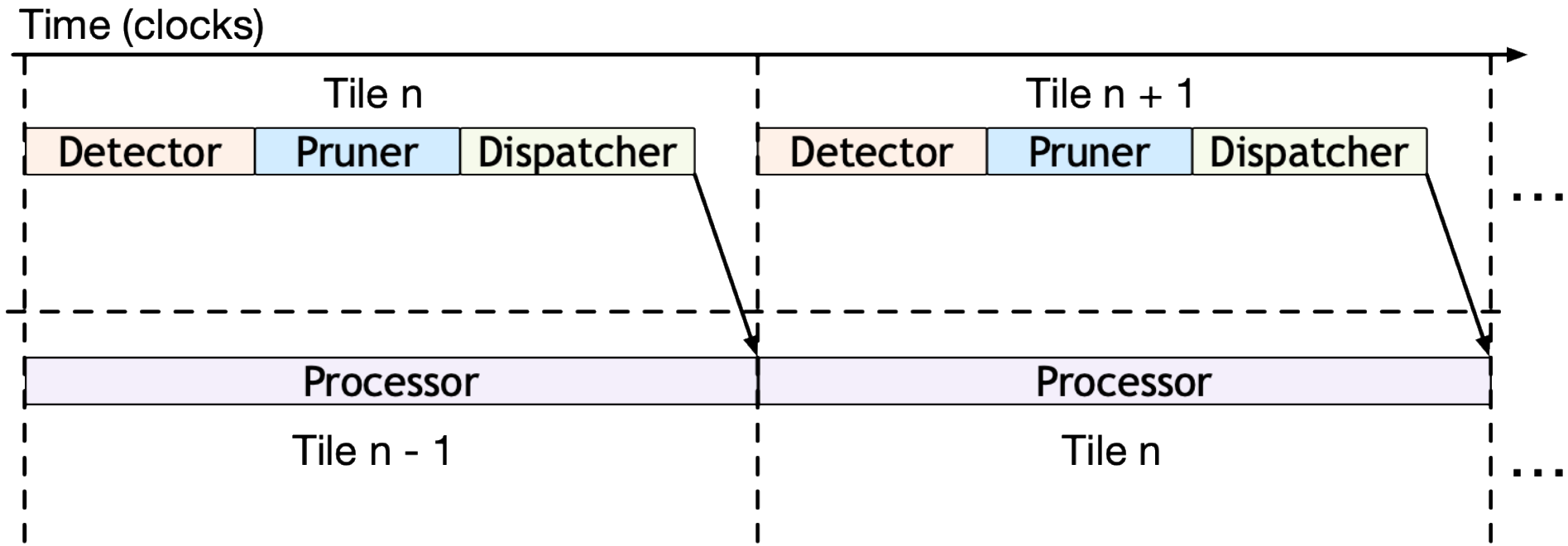
- Perform matrix computation
  - 1. Get prefix result value
  - 2. Load weight according to ProSparsity
  - 3. Accumulate weight
  - 4. Store output value





Pipeline Scheduling

- Tile-wise pipeline to maximize Processor utilization



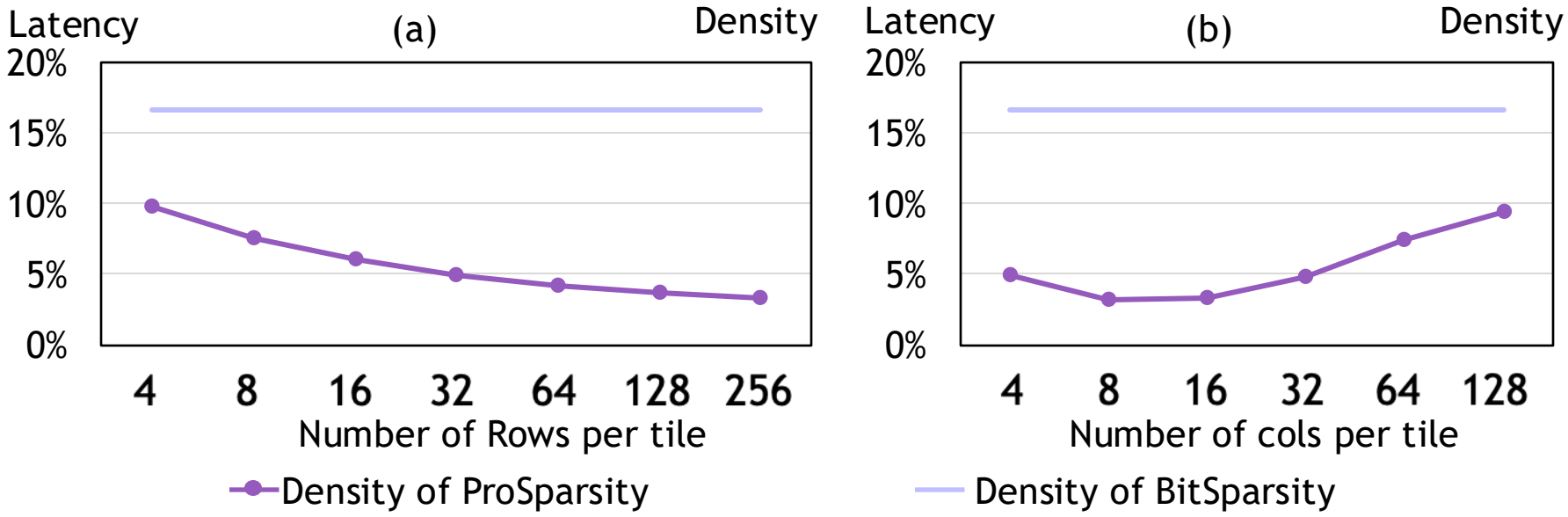
Contents

- Background & Motivation
- Product Sparsity
- Prosperity Architecture
- **Results**

Duke

Tiling Exploration

- More rows per tile make ProSparsity better, but higher overhead
- Number of cols needs to be selected carefully



Duke

Evaluation Baseline and Tools

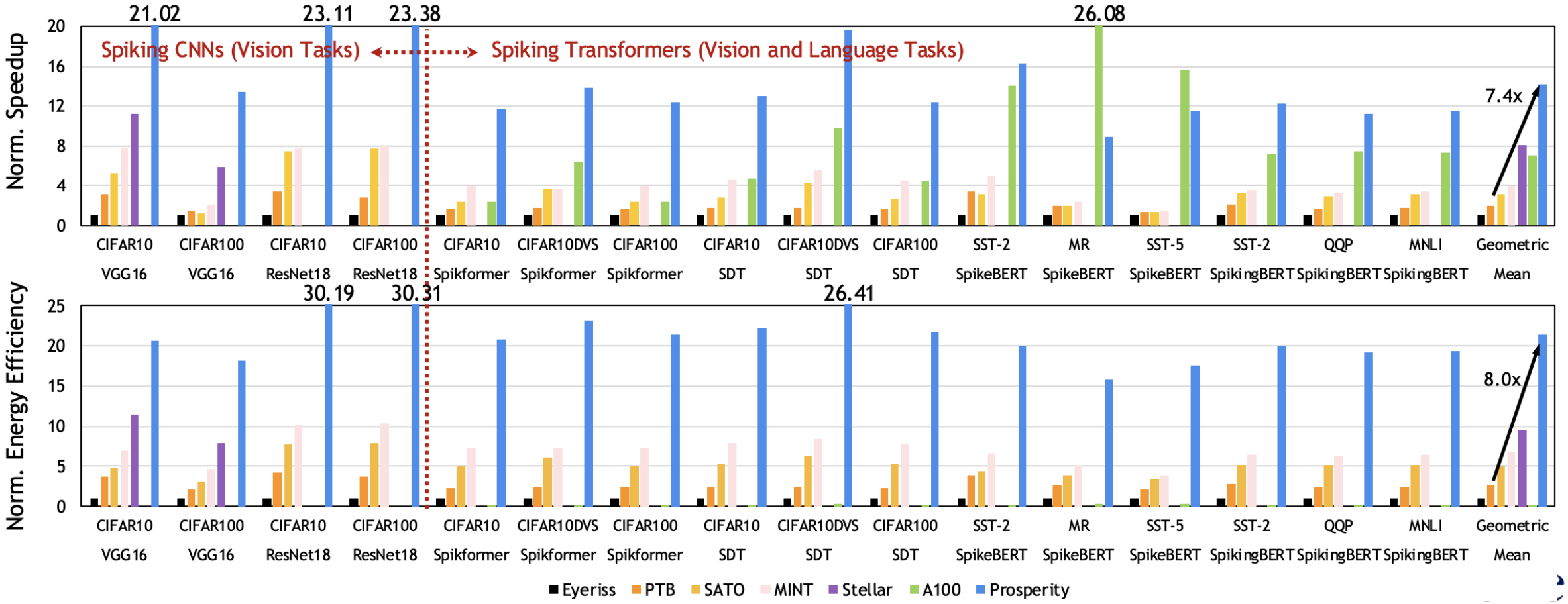
Methods	Type	Sparsity
Eyeriss	DNN accelerator	Dense
A100	GPU	Dense
PTB (HPCA'22)	SNN accelerator	BitSparsity
SATO (DAC'22)	SNN accelerator	BitSparsity
MINT (ASP-DAC'24)	SNN accelerator	BitSparsity
Stellar (HPCA'24)	SNN accelerator	BitSparsity
Prosperity (Ours)	SNN accelerator	**ProSparsity**

Tools	
Synopsys Design Compiler	Logic synthesis
CACTI	Buffer simulation
DRAMsim3	DRAM simulation

Duke

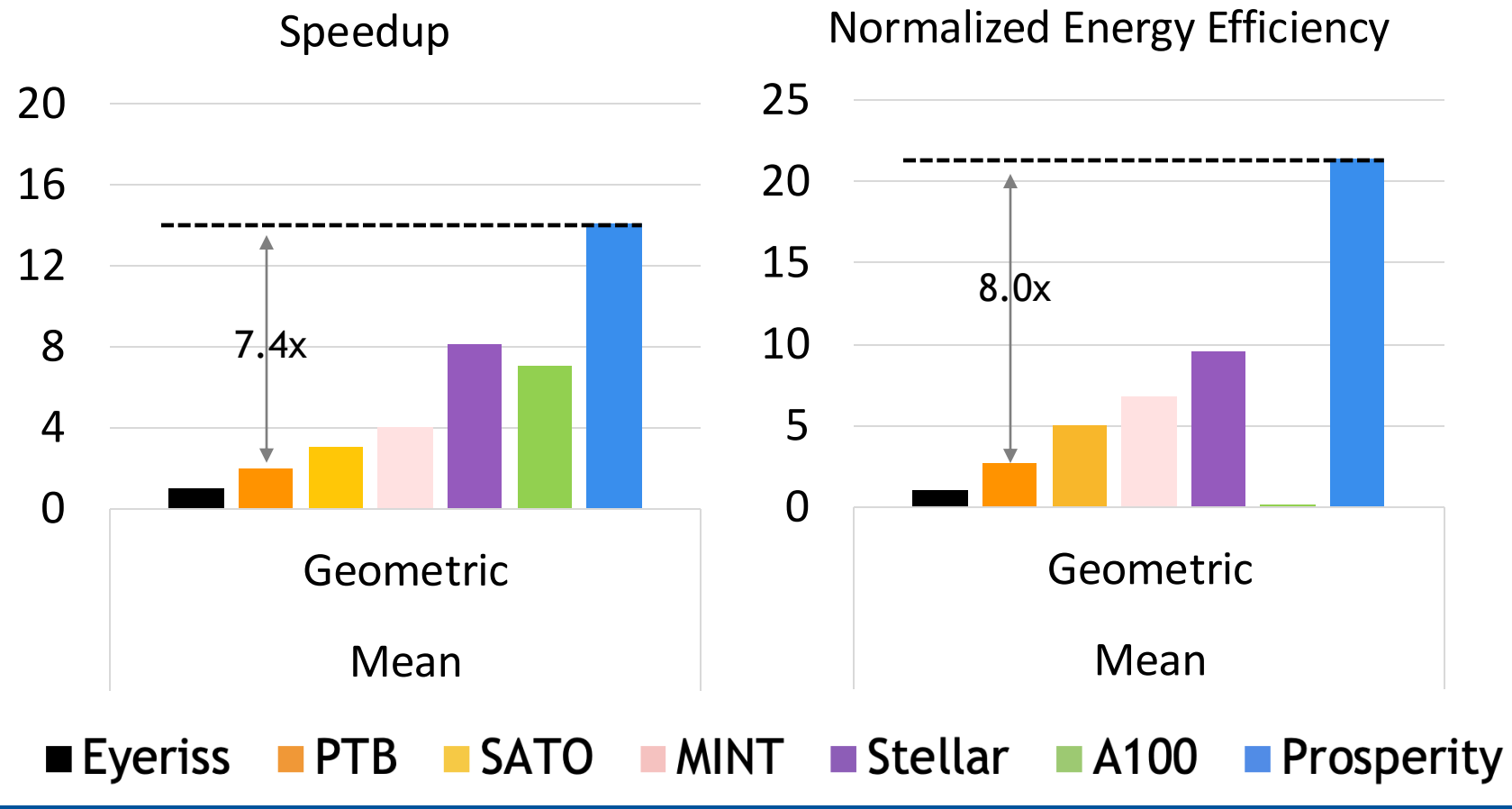
Performance and Energy

- Compared with baselines (DNN accelerator, GPU, SNN accelerators) on various SNN models and datasets



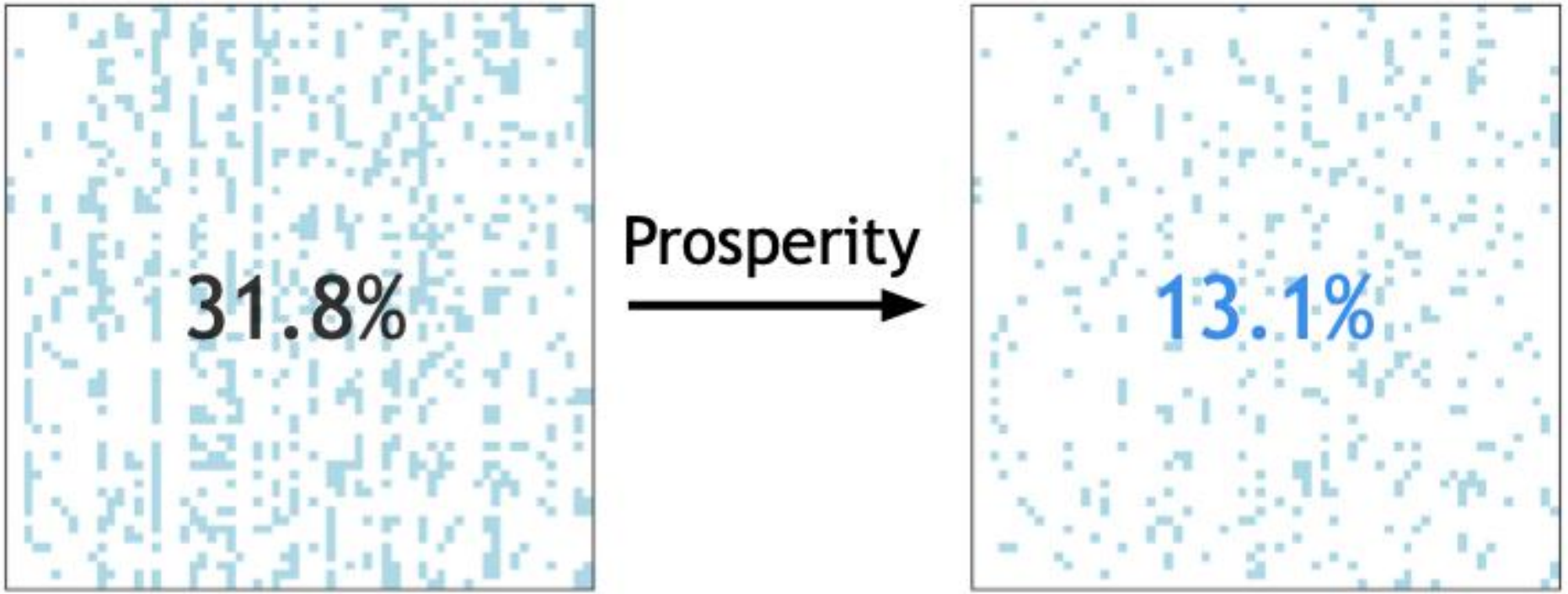
Performance and Energy

- Outperform GPU and all accelerators, 7.4x speedup and 8.0x energy efficiency over PTB (HPCA'22)



Demonstrating ProSparsity

- ProSparsity on a spiking transformer model



Duke

Takeaways

- **ProSparsity**: A sparsity paradigm for SNN that leverage similarities between rows to remove the redundant computations.

- Our proposed architecture: **Prosperity**, addresses the challenges in ProSparsity and achieve significant speedup over state-of-the-art accelerators

Duke

Thanks & QA

Welcome to use our open-source code at

https://github.com/dubcyfor3/Prosperity

Duke