

# LoAS: Fully Temporal-Parallel Dataflow for Dual-Sparse Spiking Neural Networks

---

**Ruokai Yin**<sup>1</sup>, Youngeun Kim<sup>1</sup>,  
Di Wu<sup>2</sup>, and Priyadarshini Panda<sup>1</sup>

---

<sup>1</sup> Department of ECE  
Yale University

<sup>2</sup> Department of ECE  
University of Central Florida

Email: [ruokai.yin@yale.edu](mailto:ruokai.yin@yale.edu)



57<sup>th</sup> MICRO

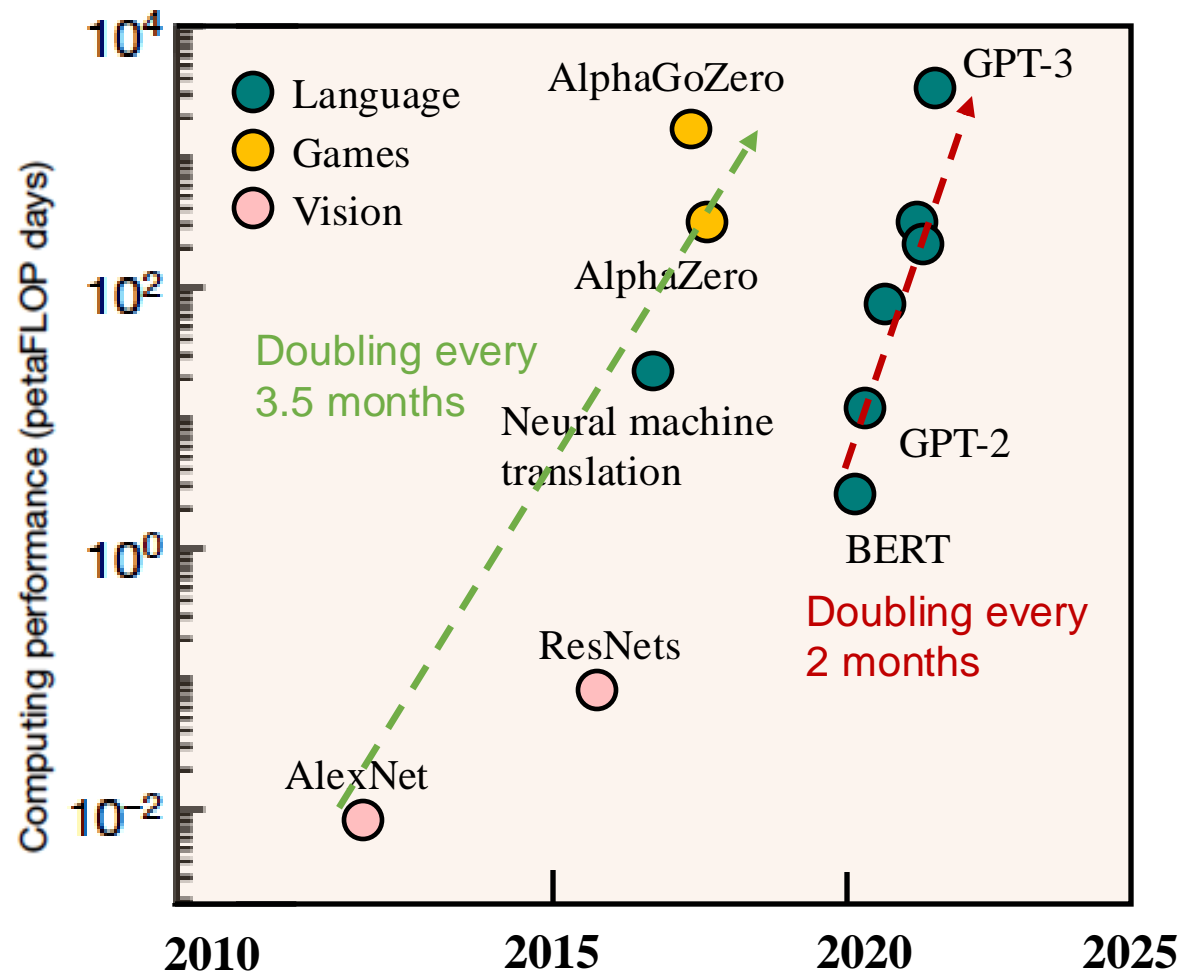
# Contents

---

- **Motivation of accelerating dual-sparse SNNs**
- Challenges of accelerating dual-sparse SNNs
- FTP (fully temporal-parallel) dataflow
- FTP-friendly compression
- LoAS and SNN-friendly inner-joint
- Results

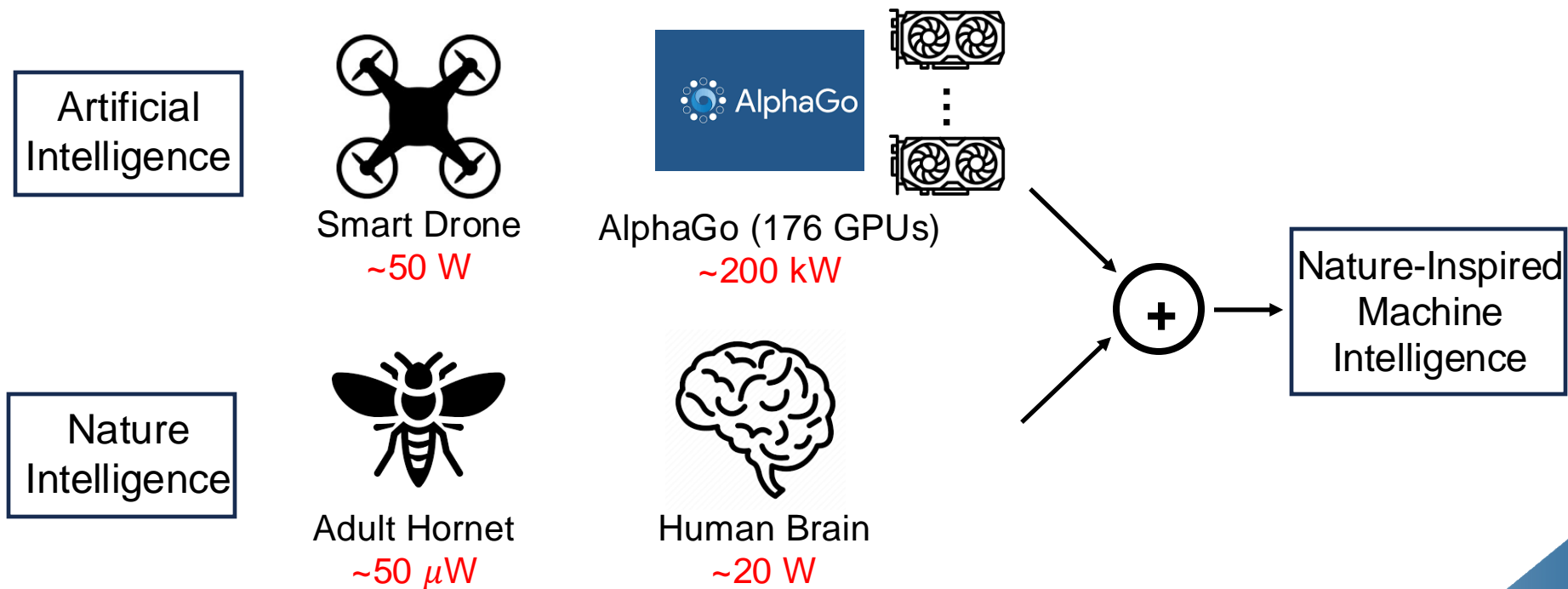


# AI workloads in GPU era



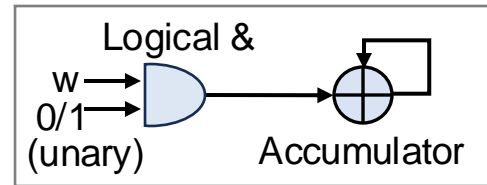
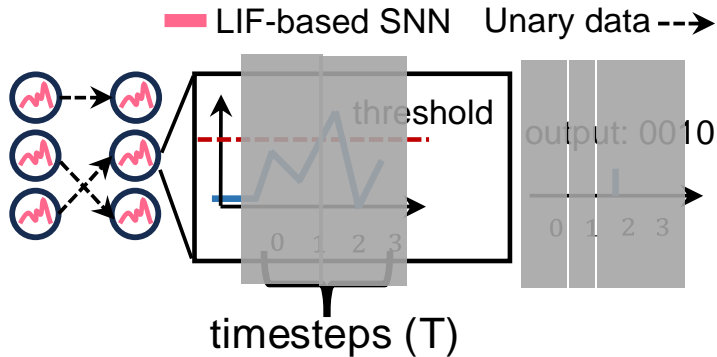
# Power matters!

There exists a huge gap in terms of the power consumption between the artificial intelligence and the nature intelligence.



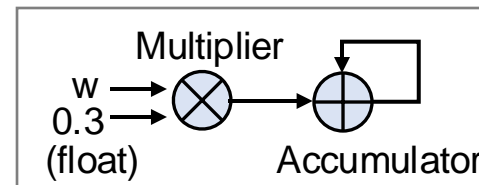
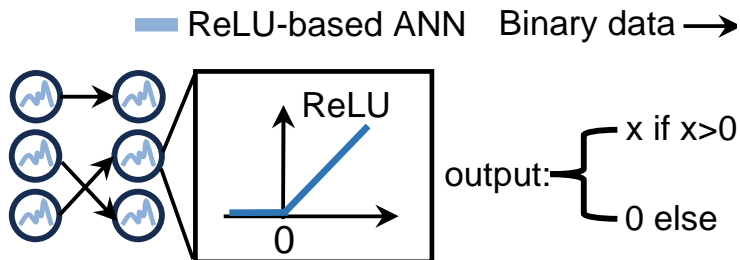
# SNNs as compute-friendly nature machine intelligence

## Spiking Neural Networks (SNNs)

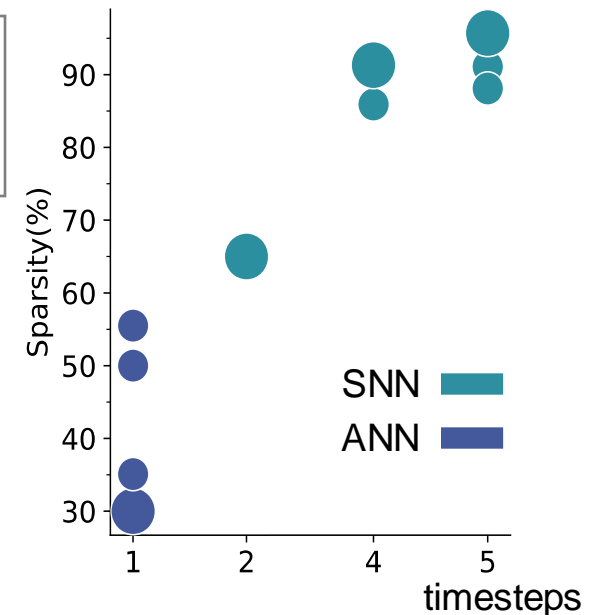


Power Cost: 1x\*

## Artificial Neural Networks (ANNs)



Power Cost: 32x\*

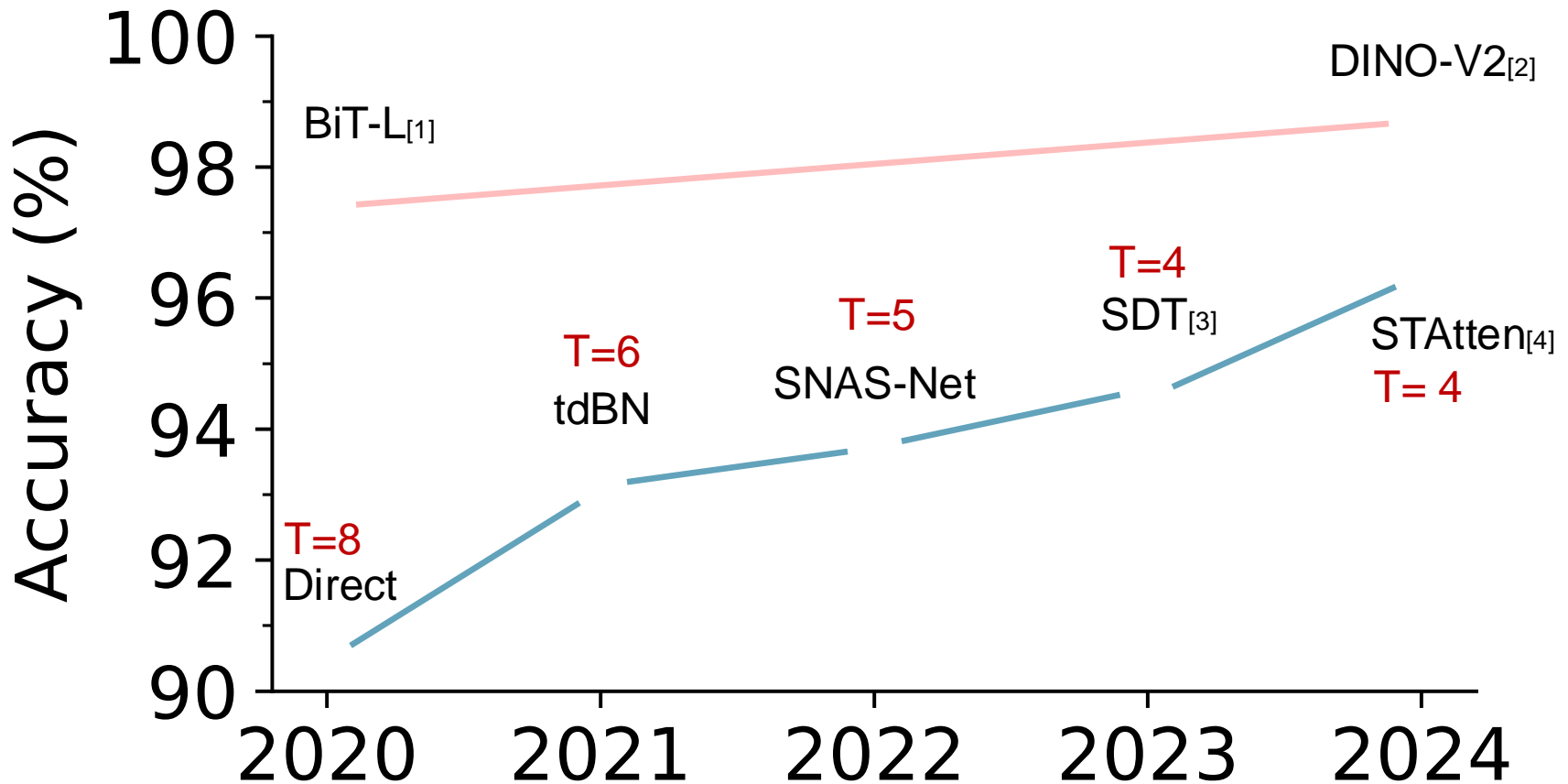


SNN input sparsity: ~90%

ANN input sparsity: ~50%



# Performance Gap: SNNs vs ANNs



[1] Kolesnikov, et al., "Big transfer (bit): General visual representation learning.", ECCV 2020

[2] Oquab, et al., "Dinov2: Learning robust visual features without supervision.", TMLR 2024

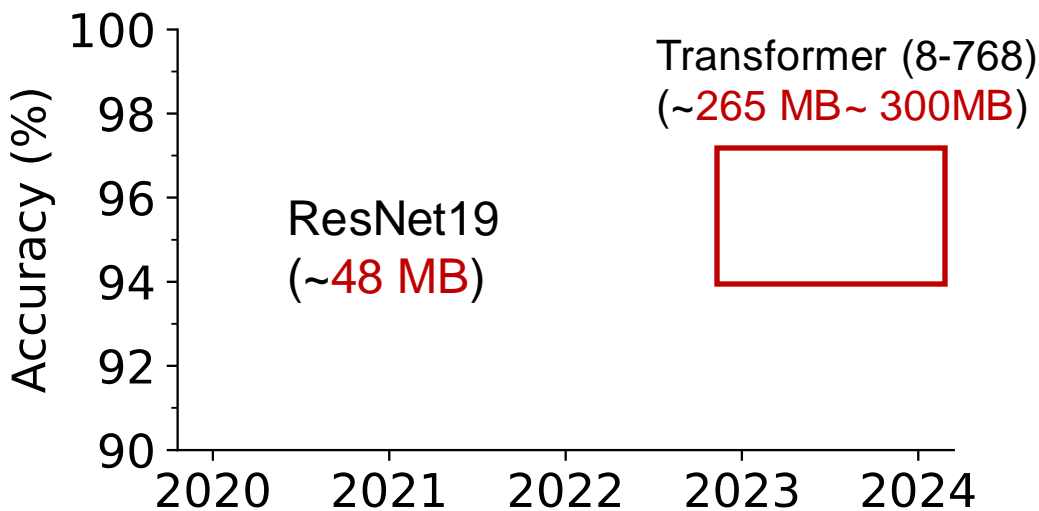
[3] Yao et al., "Spike-driven Transformer", NeurIPS 2024

[4] Lee et al., "Spiking Transformer with Spatial-Temporal Attention", arXiv, 2024



# SNN model sizes also grow

SNN model sizes will keep scaling up ↗

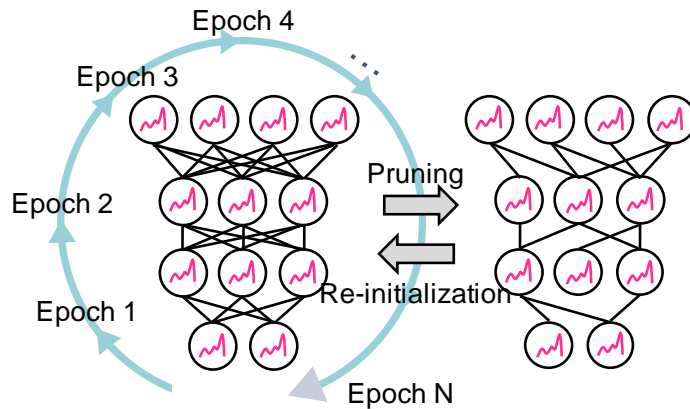


Operation	Precision	Norm Energy*
(SNN) logical & + accumulation	W – FP32 S – INT1	1x
(ANN) Multiplication + accumulation	W – FP32 X – FP32	4.6x
32-bit SRAM Read (1MB)		100x
32-bit DRAM Read		640x

When model sizes grow larger, the data movements can become the hurdle.

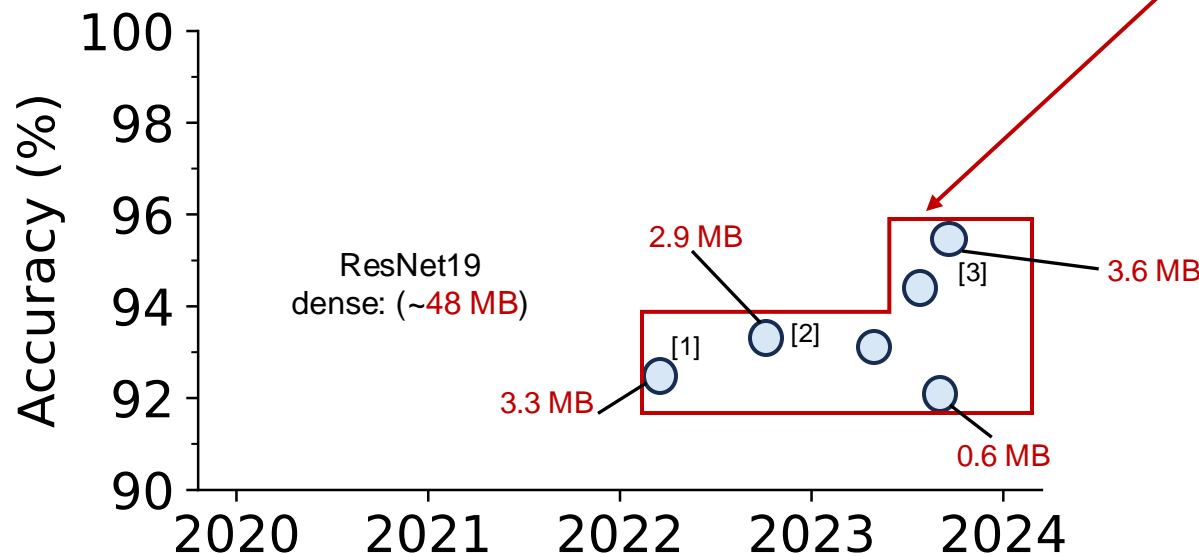


# Compress the SNNs by pruning



Pruning technique removes the redundant synaptic connections during the training.

The resulted weight matrices usually have around **95% sparsity**.



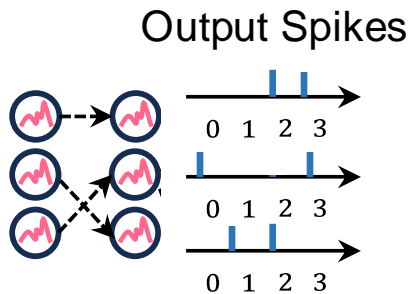
[1] Chen et al., "State transition of dendritic spines improves learning of sparse spiking neural networks.", ICML 2022.

[2] Kim et al., "Exploring lottery ticket hypothesis in spiking neural networks.", ECCV 2022.

[3] Liu et al., "LitE-SNN: Designing Lightweight and Efficient Spiking Neural Network through Spatial-Temporal Compressive Network Search and Joint Optimization.", IJCAI 2024.

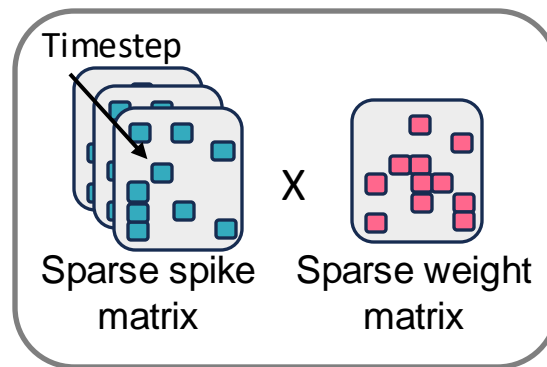


# Accelerating dual-sparse SNNs

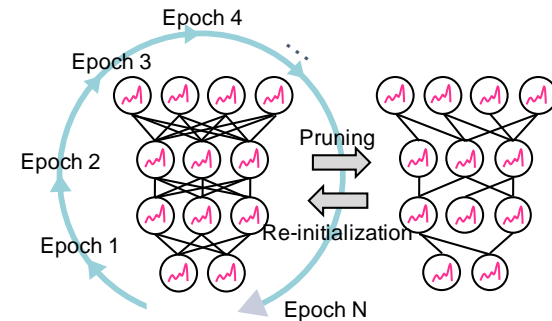


Spikes are sparse due to the LIF-based activation function.

Dual-sparse SNN workloads



Pruned SNN networks



Weights are sparse due to the iterative pruning.

Accelerating the dual-sparse SNNs becomes the acceleration of Sparse-matrix-sparse-matrix (spMspM).

# Contents

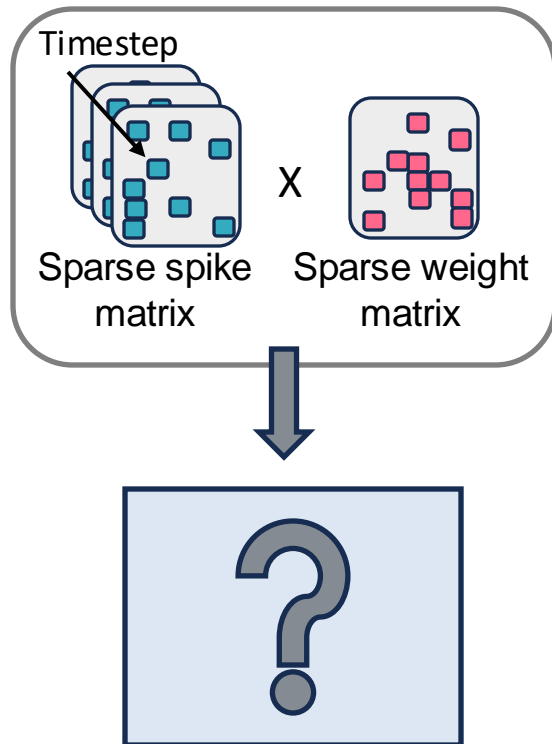
---

- Motivation of accelerating dual-sparse SNNs
- **Challenges of accelerating dual-sparse SNNs**
- FTP (fully temporal-parallel) dataflow
- FTP-friendly compression
- LoAS and SNN-friendly inner-joint
- Results









# Where to deploy dual-sparse SNNs?

## Dual-sparse SNN workloads



Where to deploy?

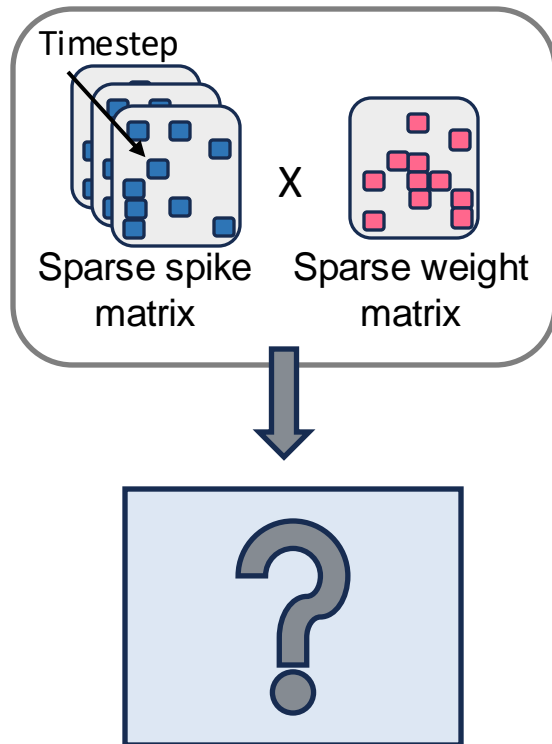
## Prior SNN accelerators

SpinalFlow (ISCA `20)	Spike Sparsity  Weight Sparsity 
PTB (HPCA `22)	Spike Sparsity  Weight Sparsity 
Stellar (HPCA `24)	Spike Sparsity  Weight Sparsity 

Efficient at leveraging the spike sparsity.  
But not support the weight sparsity.

# Where to deploy dual-sparse SNNs?

## Dual-sparse SNN workloads



Where to deploy?

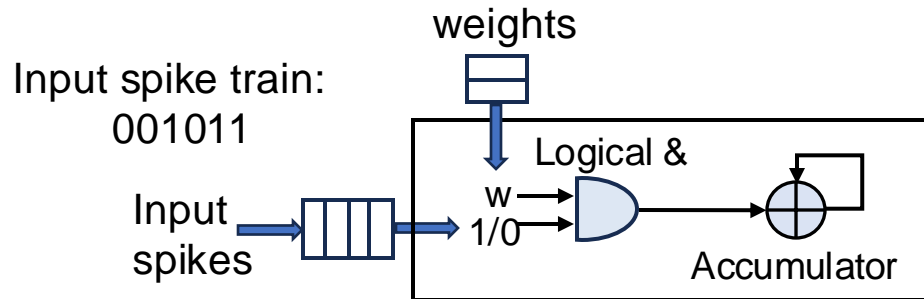
## Prior ANN spMspM accelerators?

SparTen MICRO `19	
GoSPA ISCA `21	Input Sparsity <input checked="" type="checkbox"/>
Gamma ASPLOS `21	Weight Sparsity <input checked="" type="checkbox"/>
...	

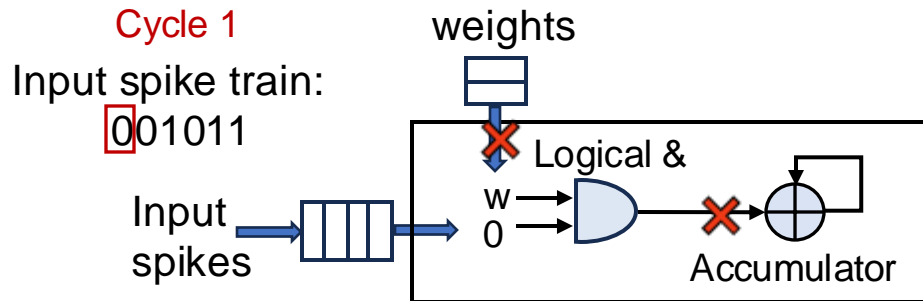
Efficient at leveraging dual-sparsity for ANNs.

However, we observe two **challenges** for deploying dual-sparse SNNs.

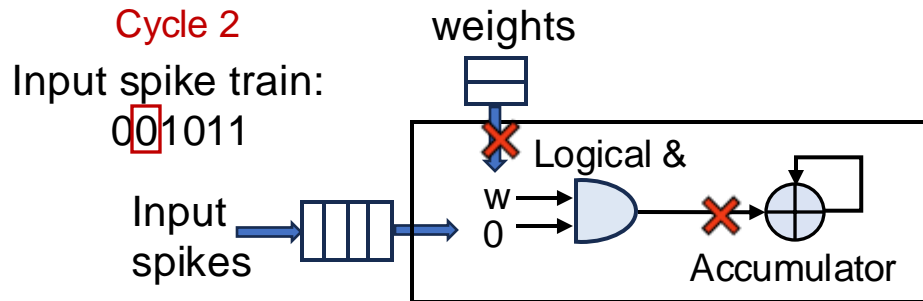
# Challenge 1: compression efficiency



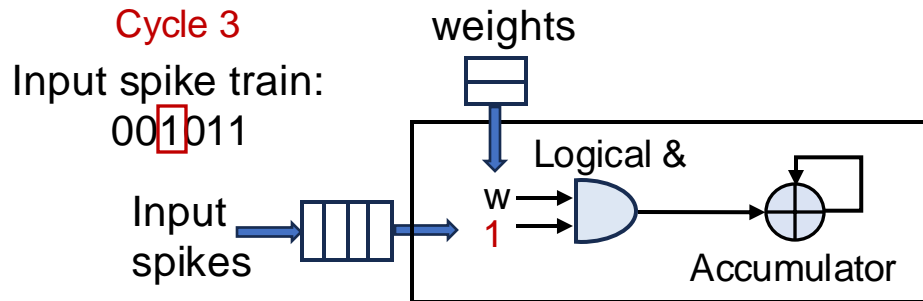
# Challenge 1: compression efficiency



# Challenge 1: compression efficiency

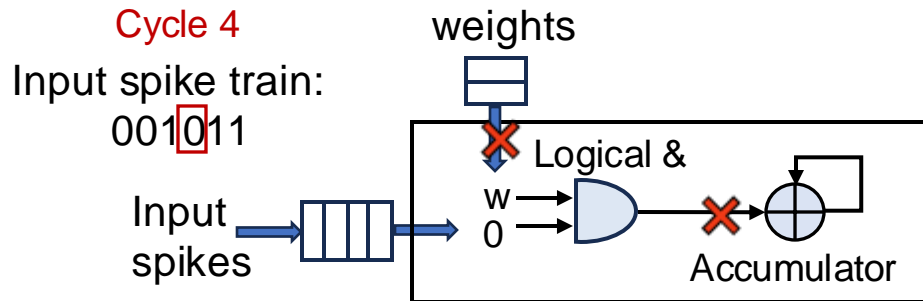


# Challenge 1: compression efficiency

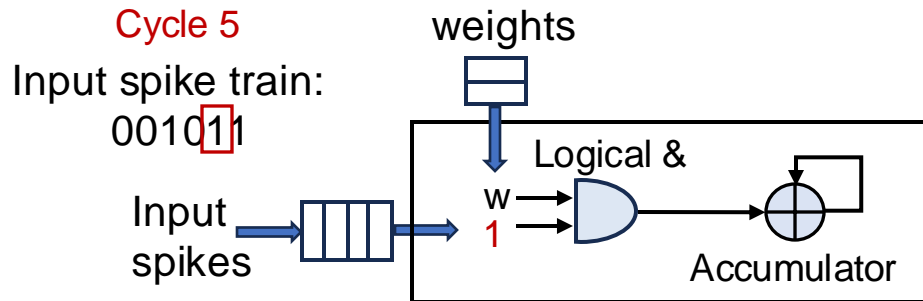




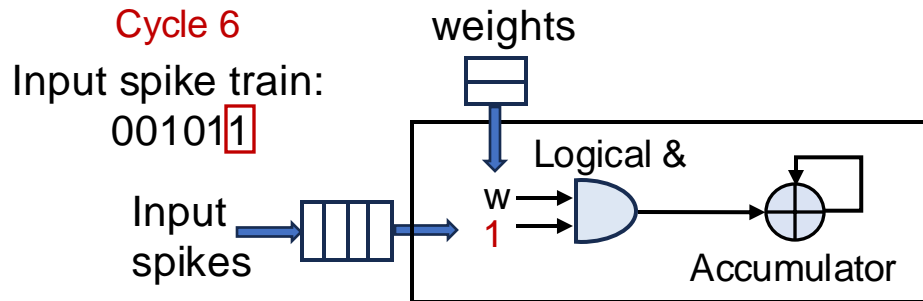
# Challenge 1: compression efficiency



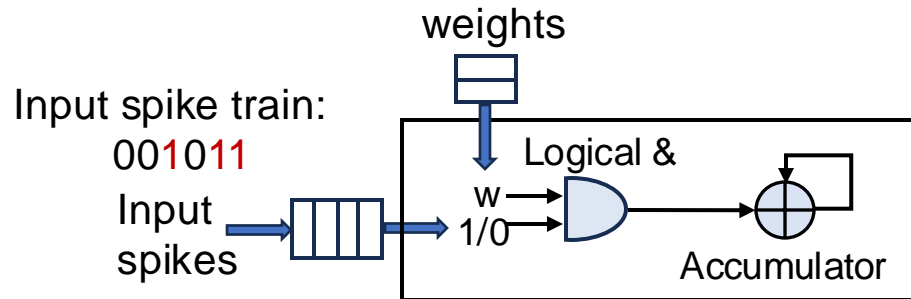
# Challenge 1: compression efficiency



# Challenge 1: compression efficiency



# Challenge 1: compression efficiency



# of compute: 3

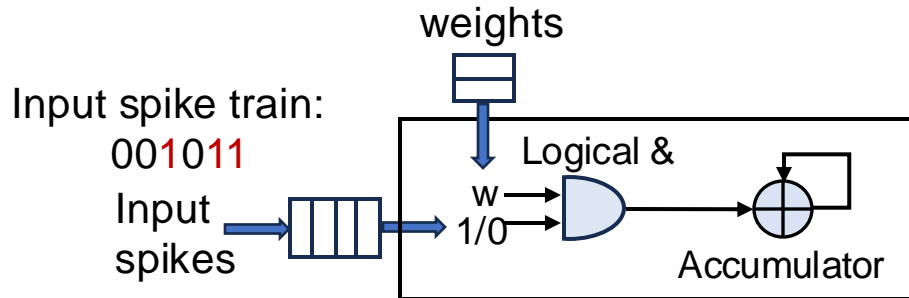
# of weight fetch: 3

# of PE cycles: 6

Not saving PE cycles!

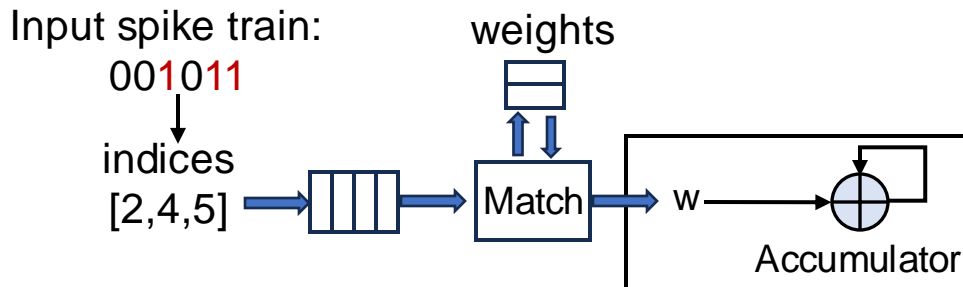


# Challenge 1: Compression Efficiency



# of compute: 3  
# of weight fetch: 3  
# of PE cycles: 6

Not saving PE cycles!



# of compute: 3  
# of weight fetch: 3  
# of PE cycles: 3

We need to compress the spikes to save PE cycles.

**Challenge:** compression efficiency for spikes is low!

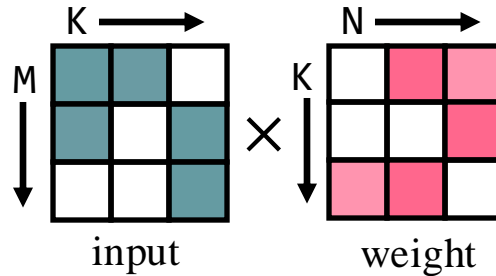
In this example:

3x3-bits CSR to compress 3 bits of data

3/9 -> only 0.33 (<1) compression efficiency

# Challenge 2: Dataflow Design Space

## Dual-sparse ANN spMspM



Different dataflow is the different permutation of the 3-nested for loops.

```
for m in [0,M-1]
  for n in [0,N-1]
    for k in [0,K-1]
```

***Inner-Product***

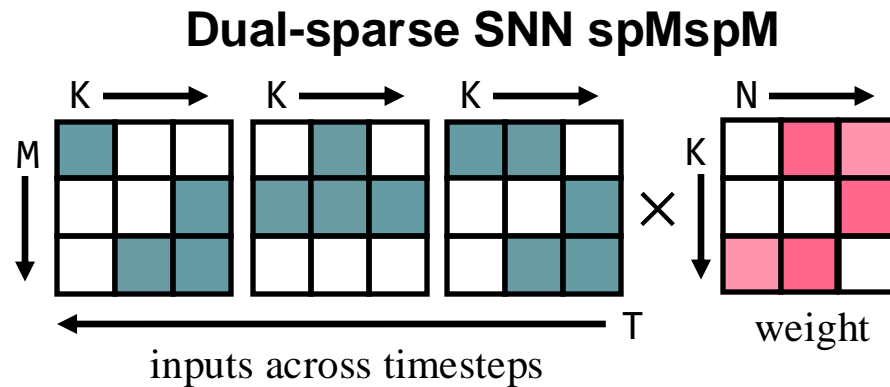
```
for k in [0,K-1]
  for m in [0,M-1]
    for n in [0,N-1]
```

***Outer-Product***

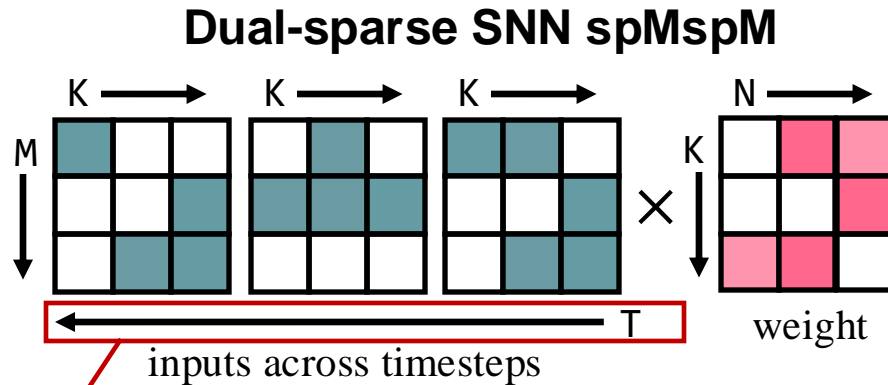
```
for m in [0,K-1]
  for k in [0,M-1]
    for n in [0,N-1]
```

***Gustavson's***

# Challenge 2: Dataflow Design Space



# Challenge 2: Dataflow Design Space



This translates to an extra temporal loop (**t-loop**)

```
for t in [0, T-1]
```

**Challenge:** where to position the **t-loop**?

Original 6 permutations

```
for k in [0, K-1]
  for m in [0, M-1]
    for n in [0, N-1]
```

Now 24 possible permutations

```
for t in [0, T-1]
  for k in [0, K-1]
    for m in [0, M-1]
      for n in [0, N-1]
```



# Contents

---

- Motivation of accelerating dual-sparse SNNs
- Challenges of accelerating dual-sparse SNNs
- **FTP (fully temporal-parallel) dataflow**
- FTP-friendly compression
- LoAS and SNN-friendly inner-joint
- Results



# Observations of SNN spMspM dataflow

**Observation 1: t-loop** increases the data traffic (weights and outputs) of the dimensions below it by **T** times.\*

<code>for t in [0,T-1]</code>	<code>for t in [0,T-1]</code>	<code>for t in [0,T-1]</code> repeat T times
<code>for m in [0,M-1]</code>	<code>for k in [0,K-1]</code>	<code>for m in [0,K-1]</code>
<code>for n in [0,N-1]</code>	<code>for m in [0,M-1]</code>	<code>for k in [0,M-1]</code>
<code>for k in [0,K-1]</code>	<code>for n in [0,N-1]</code>	<code>for n in [0,N-1]</code>
<i>Inner-Product</i>	<i>Outer-Product</i>	<i>Gustavson's</i>



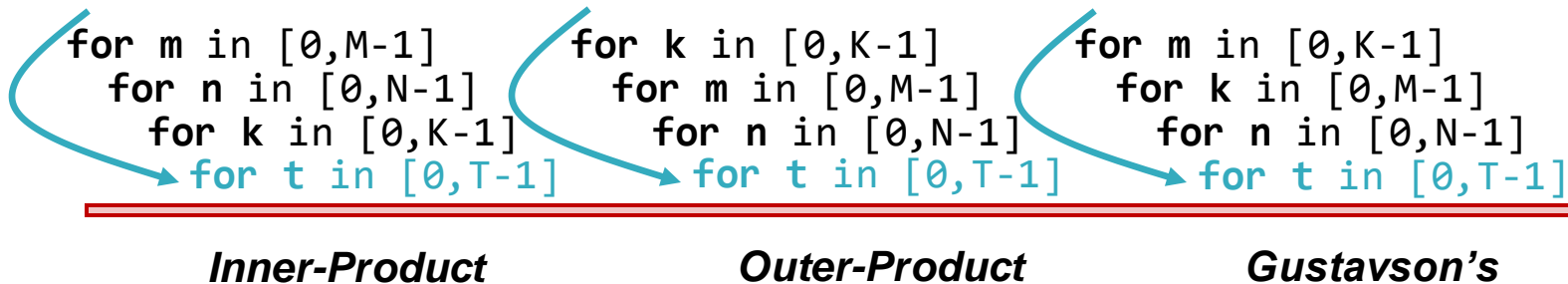
# Observations of SNN spMspM dataflow

**Observation 1: t-loop** increases the data traffic (weights and outputs) of the dimensions below it by **T** times.\*

<pre>for m in [0,M-1]   for t in [0,T-1]     for n in [0,N-1]       for k in [0,K-1]</pre>	<pre>for k in [0,K-1]   for t in [0,T-1]     for m in [0,M-1]       for n in [0,N-1]</pre>	<pre>for m in [0,M-1]   for t in [0,T-1]     for k in [0,M-1]       for n in [0,N-1]</pre>	repeat T times
<i>Inner-Product</i>	<i>Outer-Product</i>	<i>Gustavson's</i>	

# Observations of SNN spMspM dataflow

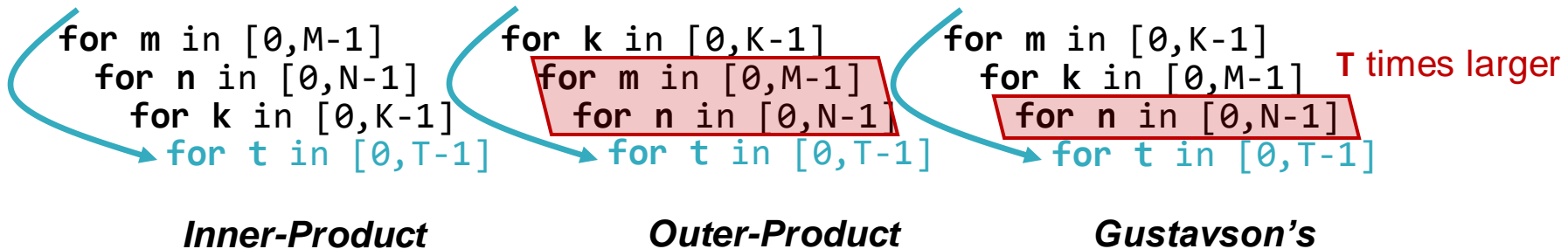
**Observation 1:** **t-loop** increases the data traffic (weights and outputs) of the dimensions below it by **T** times.\*



**Solution 1:** position **t-loop** as the innermost loop.

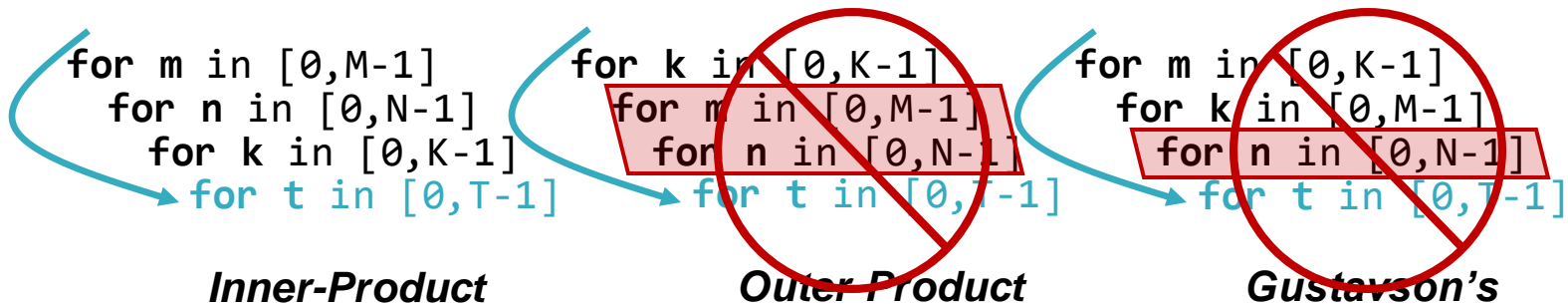
# Observations of SNN spMspM dataflow

**Observation 2: t-loop** expands the partial-sums (rows) in original dataflow by **T** times, regardless of the position.



# Observations of SNN spMspM dataflow

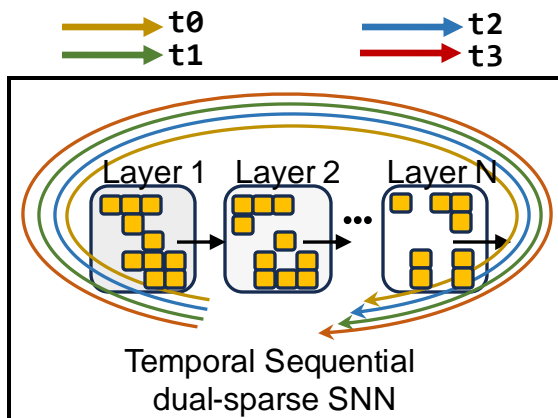
**Observation 2: t-loop** expands the partial-sums (rows) in original dataflow by **T** times, regardless of the position.



**Solution 2:** reduce the partial sums as early as possible (put **k-loop** as second inner).

# FTP dataflow

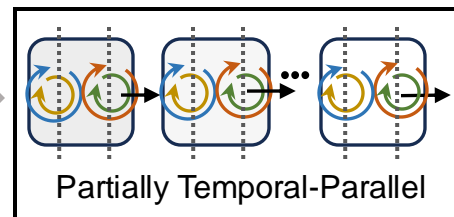
We further **spatially unroll** the **t-loop**. This removes the latency penalty with the minimal hardware instances duplication.



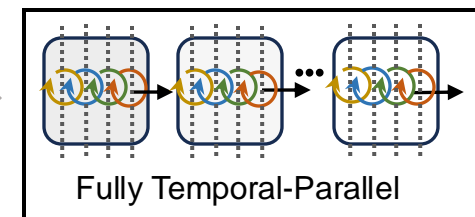
SpinalFlow  
(ISCA '20)

```
for m in [0, M-1]
  for n in [0, N-1]
    for k in [0, K-1]
      parallel-for t in [0, T-1]
```

**Fully temporal-parallel dataflow**



PTB  
(HPCA '22)



Ours

# Contents

---

- Motivation of accelerating dual-sparse SNNs
- Challenges of accelerating dual-sparse SNNs
- FTP (fully temporal-parallel) dataflow
- **FTP-friendly compression**
- LoAS and SNN-friendly inner-joint
- Results

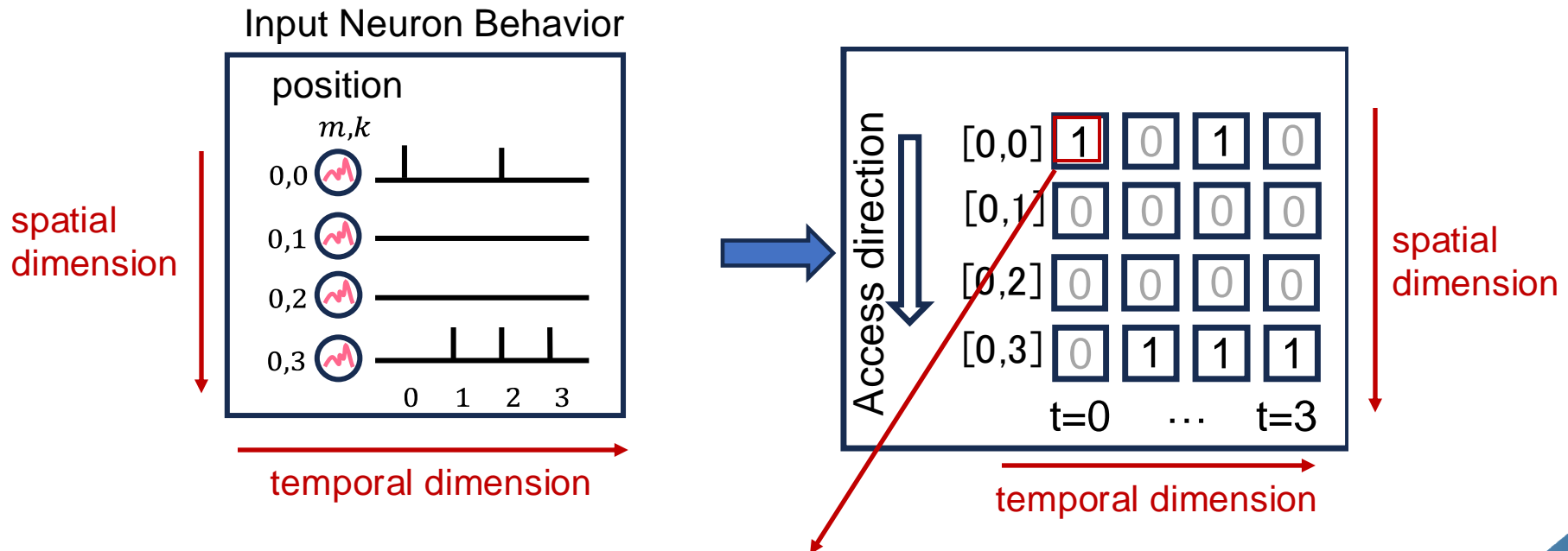




# FTP-friendly compression

Recall from the challenge 1, the compression efficiency of the unary spikes is low.

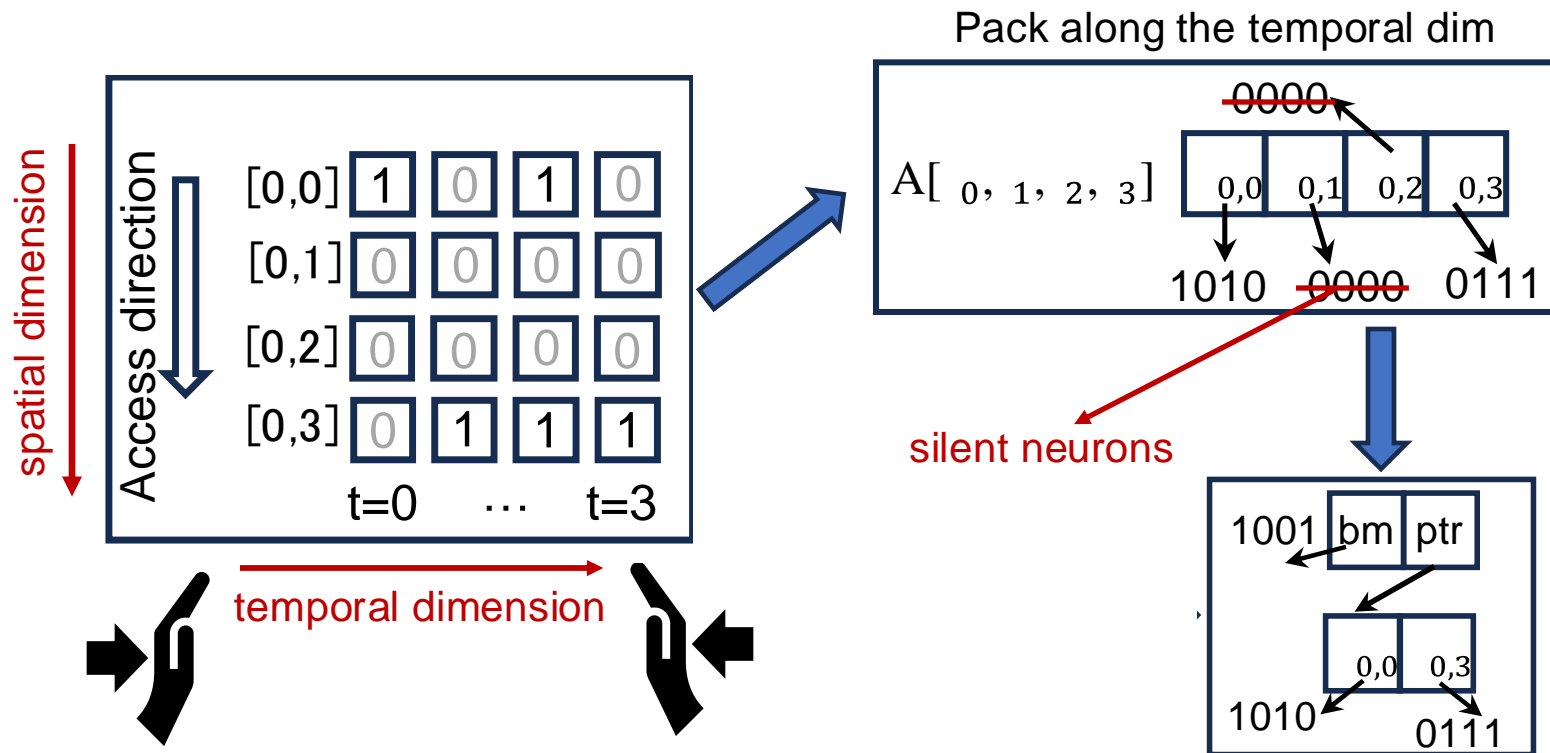
The intuition here is that position index can only encode the data at **spatial location**.



Each **spatial location** only contains **1-bit information** for each timestep.

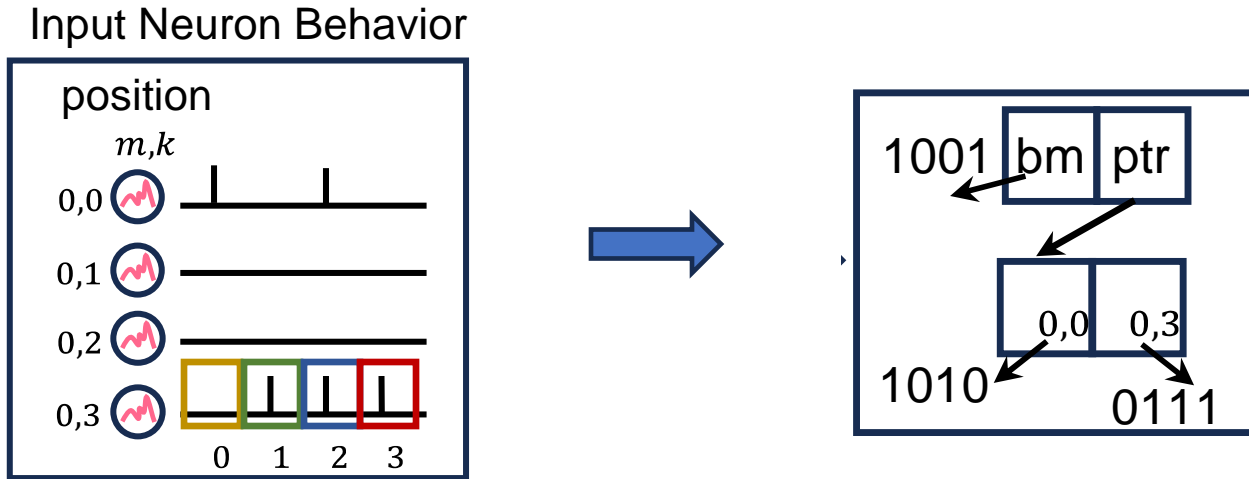
# FTP-friendly compression

Fortunately, data on different temporal dimension shares the same spatial location.



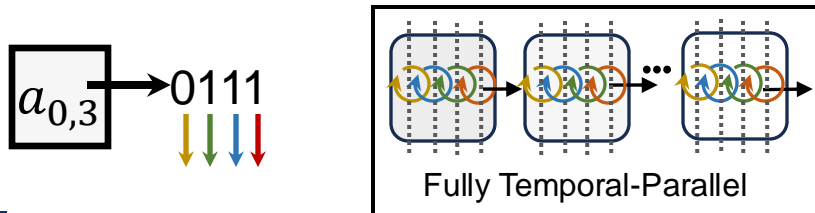
Only store the compressed non-silent neurons

# FTP-friendly compression



Benefit 1: better compression efficiency ( $8/4=2$ )

Benefit 2: Contiguous memory access for FTP dataflow



```

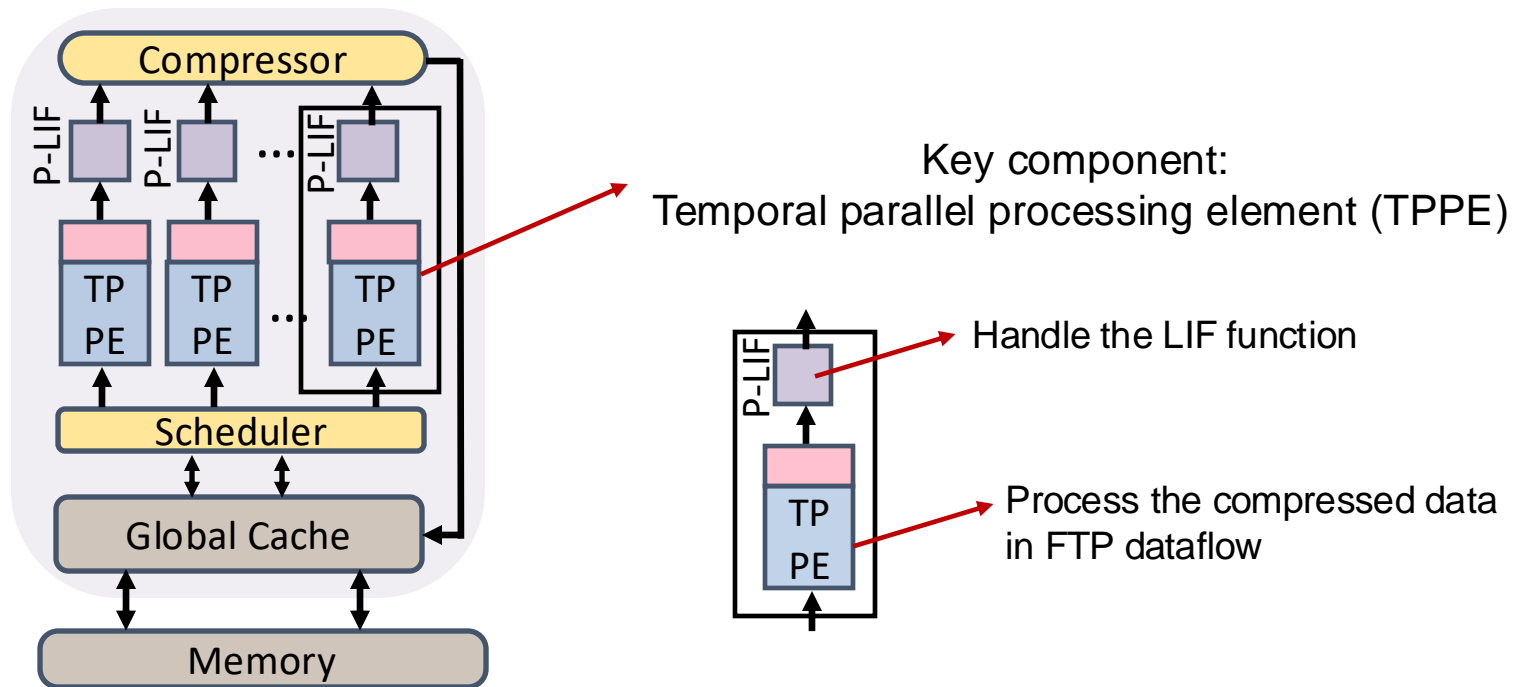
for m in [0,M-1]
  for n in [0,N-1]
    for k in [0,K-1]
      parallel-for t in [0,T-1]
  
```

# Contents

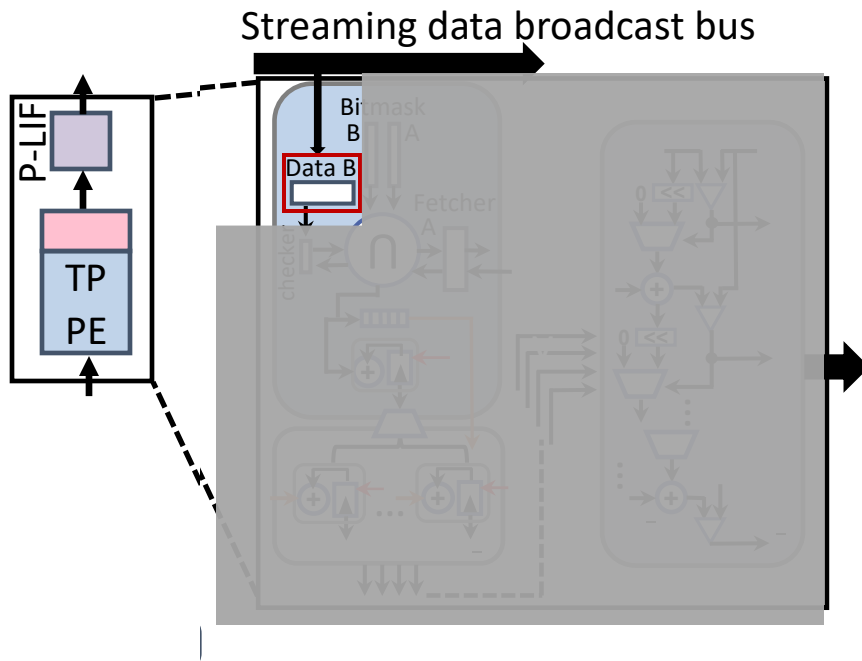
- Motivation of accelerating dual-sparse SNNs
- Challenges of accelerating dual-sparse SNNs
- FTP (fully temporal-parallel) dataflow
- FTP-friendly compression
- **LoAS and SNN-friendly inner-joint**
- Results

# LoAS

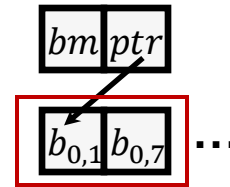
To encapsulate both the FTP dataflow and FTP-friendly compression, we design LoAS (**L**ow-latency inference **A**ccelerator for dual-sparse **S**NNs.)



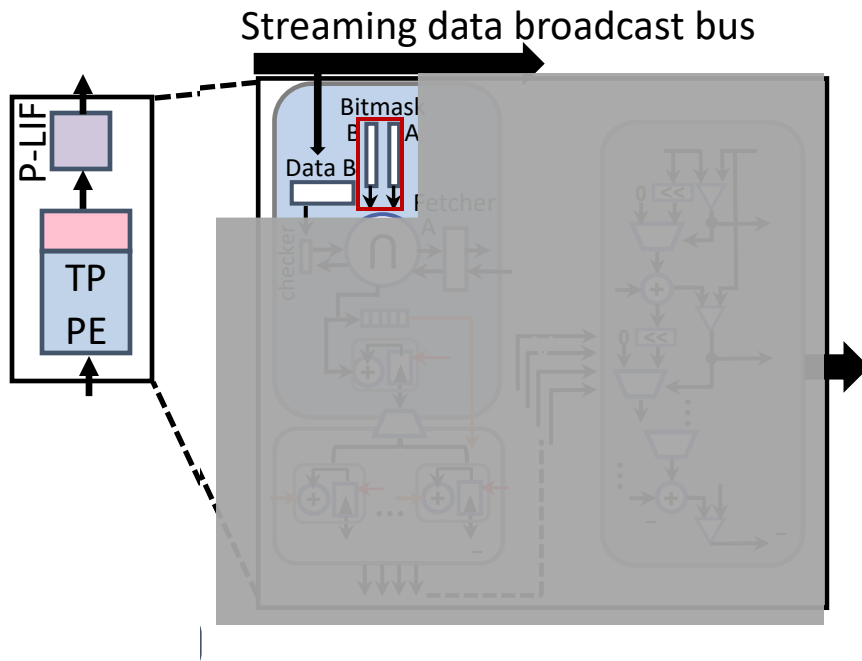
# LoAS



Data B is sent in as the compressed form:



# LoAS



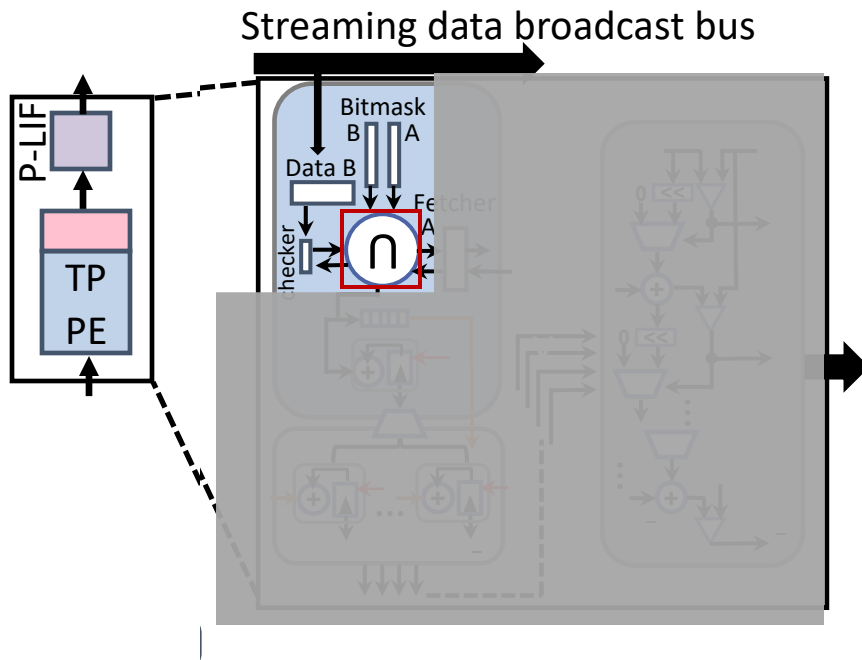
A and B's bitmasks are also sent.



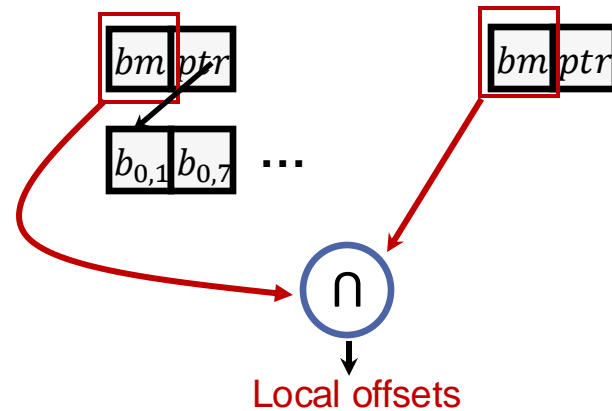
(spikes) $bm_A$  0100101...

(weights) $bm_B$  1101001...

# LoAS



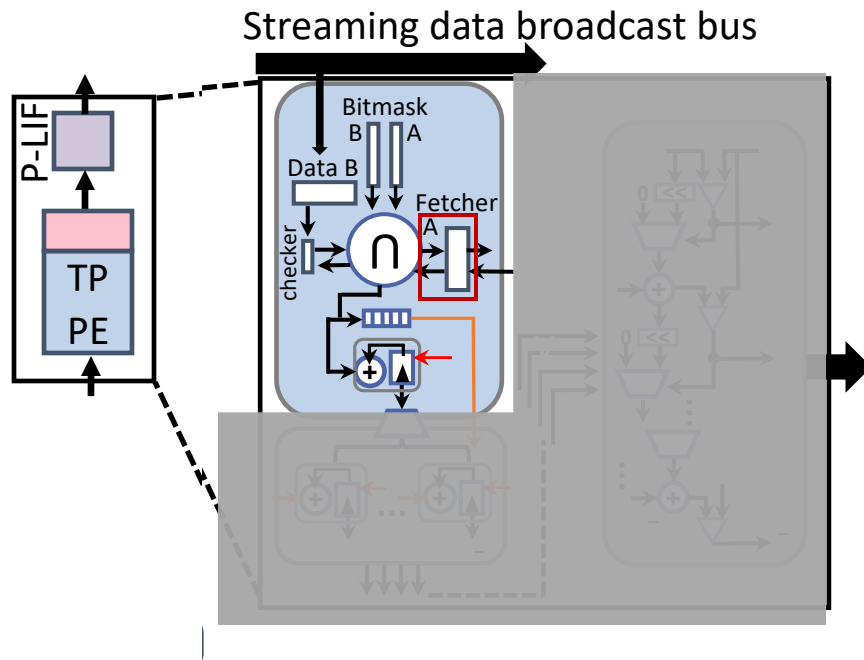
Bitmasks are sent through the inner-join unit



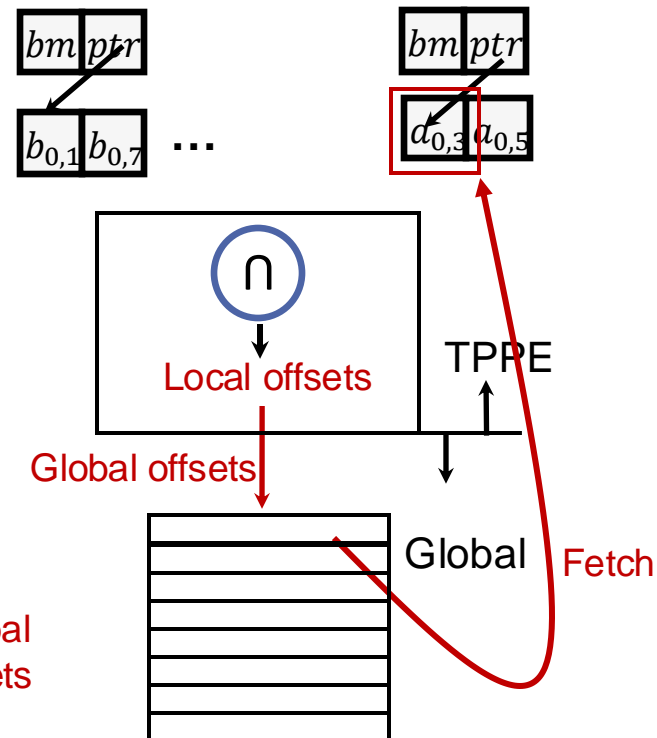
(spikes) $bm_A$  0100101...  
 (weights) $bm_B$  1101001...  $\cap$  → 1,6 ...  
 Local offsets



# LoAS

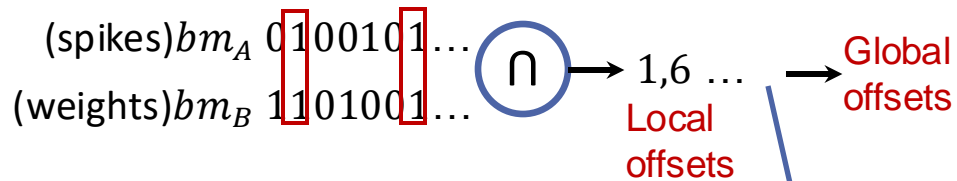


Use the local offsets to get global offsets and then fetch data and perform computation.

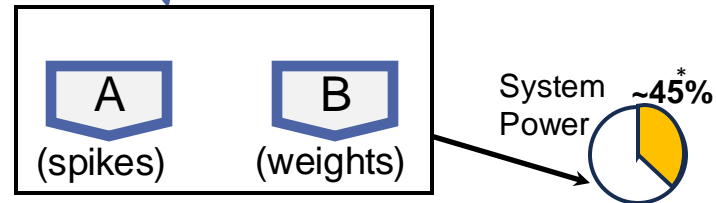


(spikes)  $bm_A$  0100101...  
 (weights)  $bm_B$  1101001...  
 $\cap \rightarrow 1,6 \dots$   
 Local offsets  $\rightarrow$  Global offsets

# LoAS



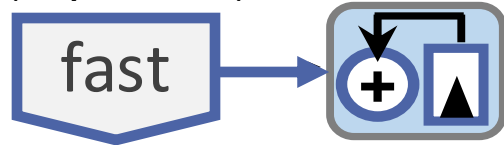
To get the global offsets, we need expensive prefix sum circuits for each operands.



# LoAS

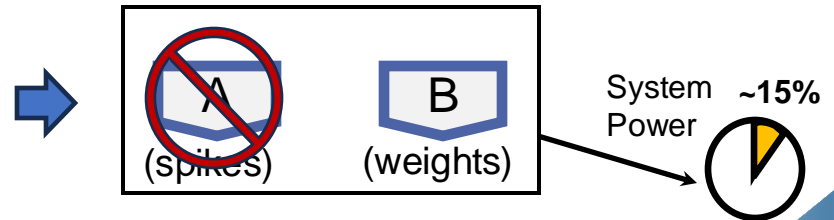
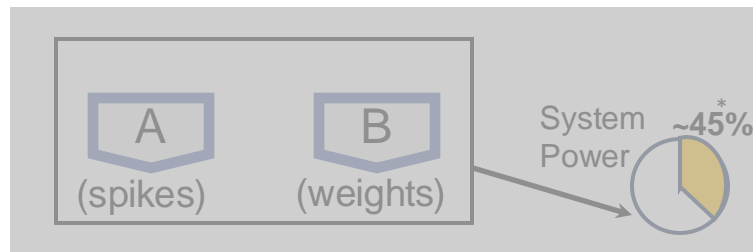
We propose an SNN-friendly inner-join units with a combination of fast prefix sum (for B) and laggy prefix sum (for A).

(expensive)



(cheap)

**Intuition:** **preemptively decode and accumulate** B using the fast prefix sum, then later make **corrections** on pre-computed results using the decoded A from the laggy prefix sum.



# Contents

---

- Motivation of accelerating dual-sparse SNNs
- Challenges of accelerating dual-sparse SNNs
- FTP (fully temporal-parallel) dataflow
- FTP-friendly compression
- LoAS and SNN-friendly inner-joint
- **Results**



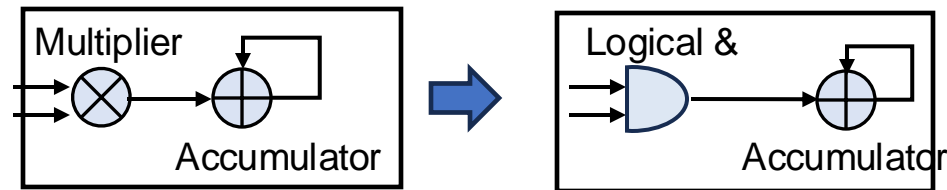
# Baselines

Simplify and modify the original ANN spMspM accelerators for running SNNs.

***Inner-Product***  
***SparTen MICRO '19***

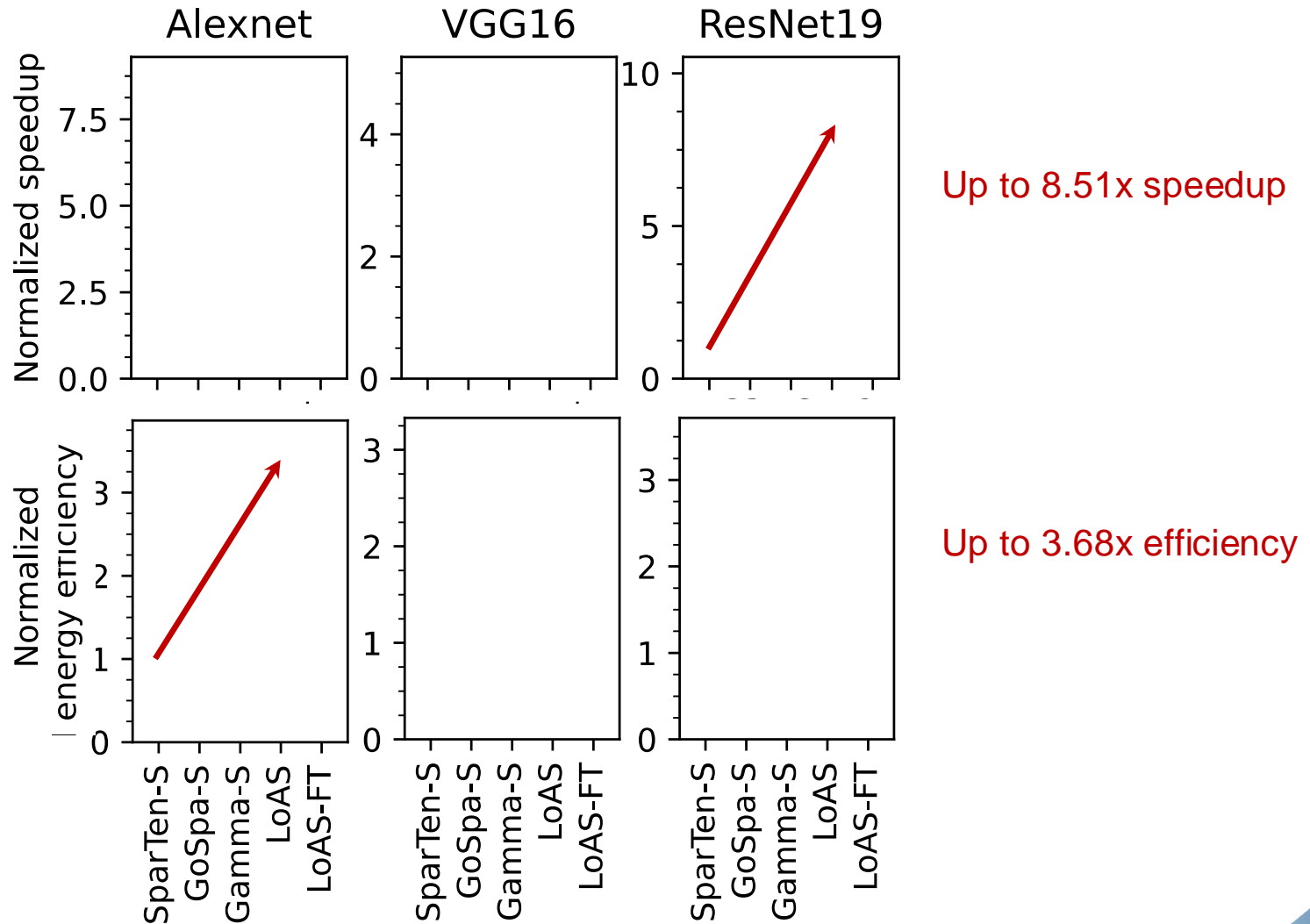
***Outer-Product***  
***GoSPA ISCA '21***

***Gustavson's***  
***Gamma ASPLOS '21***



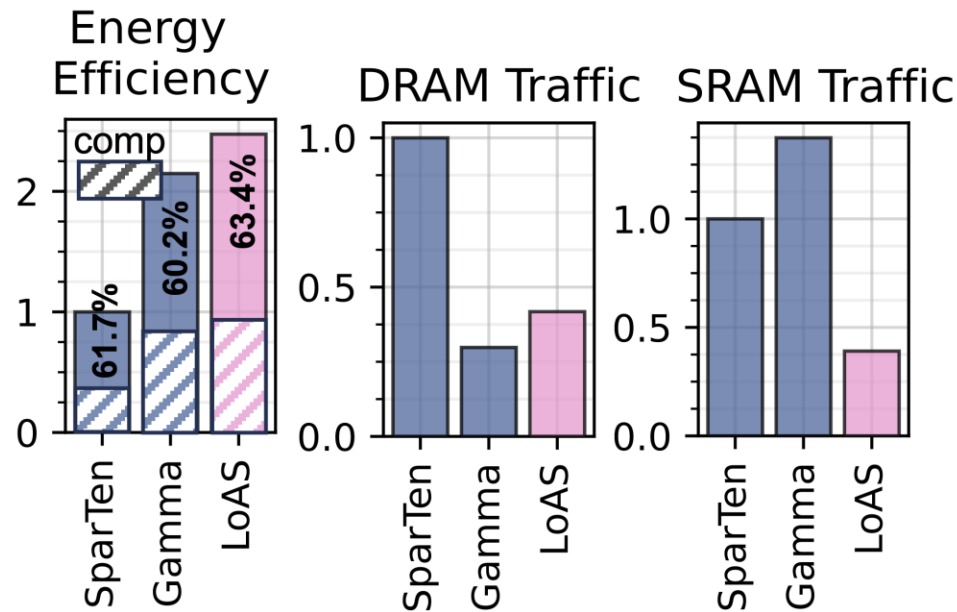
Simplify the MAC units to leverage SNN's unary spikes.

# LoAS (T=4) vs. Baselines (T=4)



# LoAS (T=4) vs. ANN SparTen (8bit)

Compare to dual-sparse ANNs (VGG16) running on SparTen and Gamma, LoAS achieves on average 1.9x more energy efficiency.



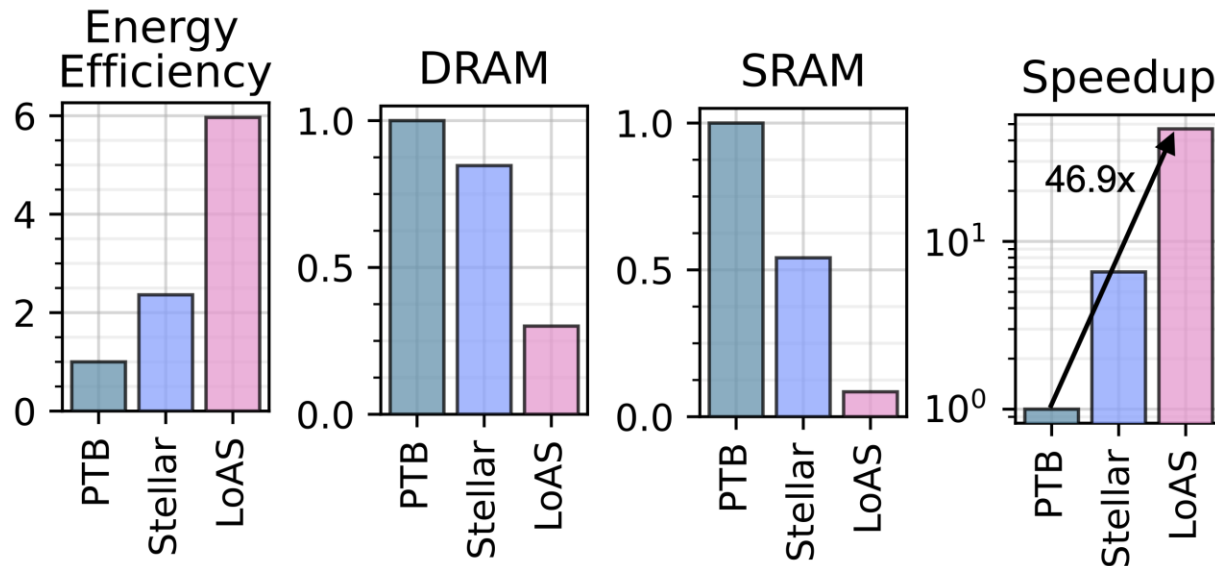
Both SNNs and ANNs have weights in 8-bits

Mainly due to

1. the higher sparsity in LIF neuron (79%) vs. ReLU (43.9%).
2. the lower input bandwidth (4-bit vs. 8-bit)

# LoAS (T=4) vs. dense SNN accelerators

Compared to existing dense SNN accelerators (VGG16), LoAS achieves on average 2.8x more energy efficiency and up to 46.9x speedup.



Mainly due to the leverage of dual-sparsity (less data fetch of weights).



# Conclusion

---

- There lacks prior hardware deployment solutions for dual-sparse SNN workloads (pruned SNNs).
- We explore the dataflow design space for dual-sparse SNNs and propose the fully temporal parallel (FTP) dataflow.
- We further propose an FTP-friendly compression method for unary spikes that ensures the compression efficiency and contiguous memory access for FTP dataflow.
- We encapsulate those techniques in LoAS together with an SNN-friendly inner-joint unit design.
- LoAS is more energy efficient and faster compared to its SNN-like spMspM baselines, ANN spMspM designs, and dense SNN designs.
- We hope this work can start a new round of design space exploration in SNN's spMspM accelerations.



---

# Thank you! Q&A

Code available at: <https://github.com/Intelligent-Computing-Lab-Yale/LoAS>

Feel free to email me questions!

[ruokai.yin@yale.edu](mailto:ruokai.yin@yale.edu)

Also here!

