PRIMATE: Processing in Memory Acceleration for Dynamic Token-pruning Transformers

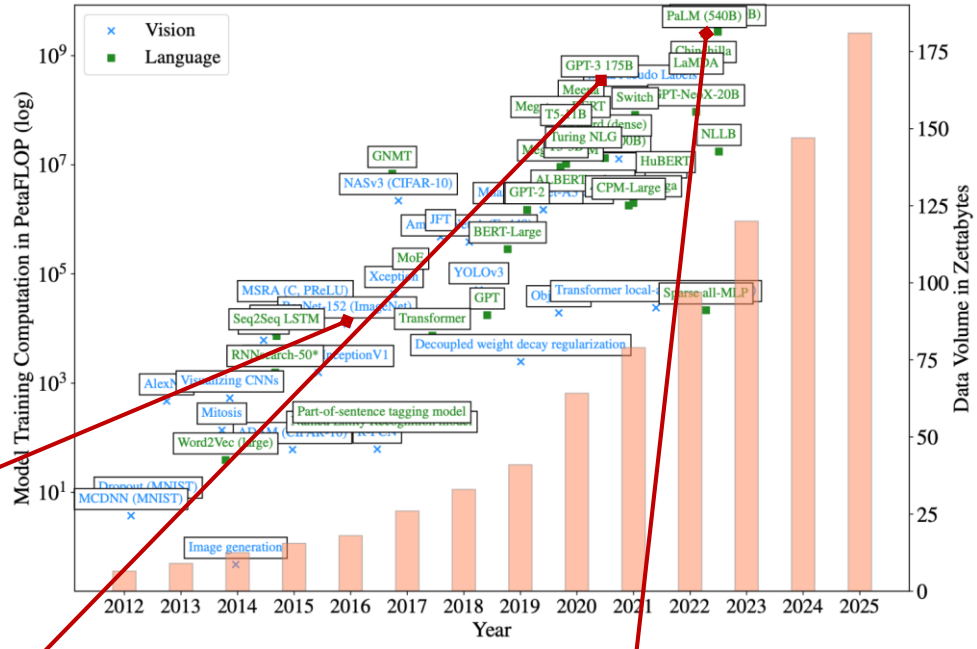
Yue Pan[1], Minxuan Zhou[1], Chonghan Lee[2], Zheyu Li[2],
Rishika Kushwah[2], Vijaykrishnan Narayanan[2], Tajana Rosing[1]
University of California San Diego[1]
Pennsylvania State University[2]

Speaker: Chonghan Lee

System Energy Efficiency Lab
seelab.ucsd.edu

Large-Scale Deep Learning Era

- The overall volume of stored data worldwide projected to reach 181 Zettabytes (10^{21})
- Rise of Transformer Architecture: State-of-the-art models in various NLP tasks and vision tasks
- More parameter → more performance

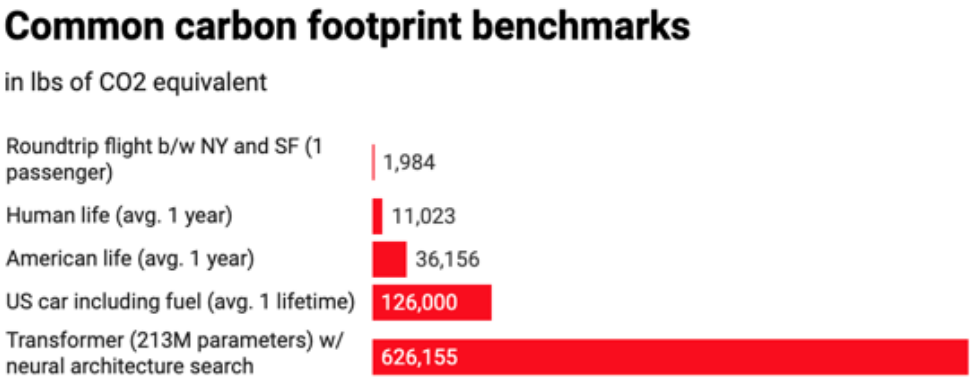


Resnet-152
60M parameters

GPT-3
175B parameters

Pathways Language Model (Palm)
540B parameters

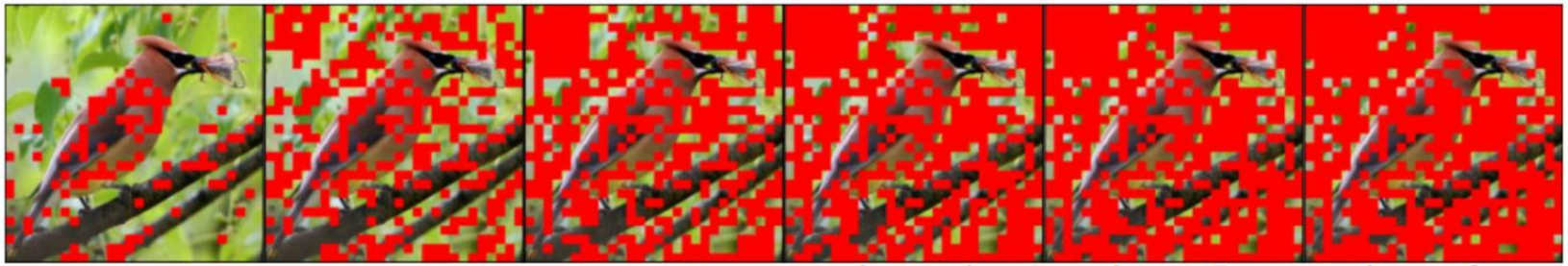
Large-Scale Deep Learning Era



- Large ML models require enormous computational resources and incur substantial energy and environmental impact
  - GPT-3 training requires 190,000 KWh of energy and 187,400 lbs CO_2
  - Serving one instance of ChatGPT requires 8x Nvidia A100 with 400W each
- Strong necessitation for environmentally friendly and efficient AI

Background: Token Adaptive Vision Transformer (TAVT)

L1: The mesmerizing performance of the leads keep the film grounded and the audience riveted
L6: The mesmerizing leads keep the film grounded and the audience riveted
L12: Leads grounded keep the audience riveted



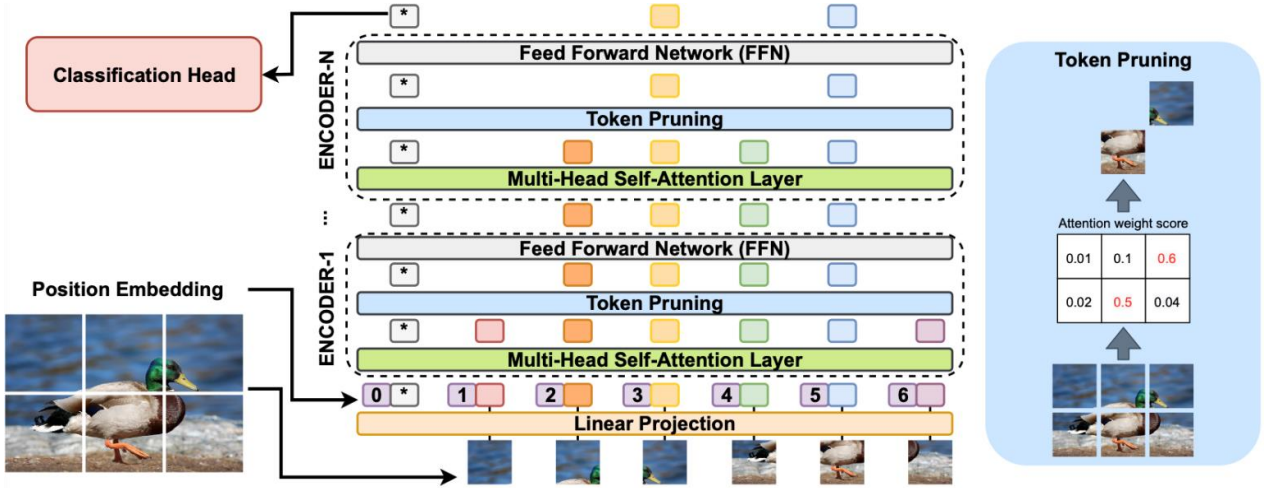
Input data contains redundancies that are inconsequential to the output

C. Lee, R. B. Brufau, K. Ding and V. Narayanan, "Token Adaptive Vision Transformer with Efficient Deployment for Fine-Grained Image Recognition," *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Antwerp, Belgium, 2023, pp. 1-6, doi: 10.23919/DATE56975.2023.10137239.

Background: Token Adaptive Vision Transformer (TAVT)

To reduce redundant information,

- Splits the input image into a series of ordered patches
- Drop less important patches each layer using self-attention scores
- Improve accuracy by removing noise and background patches



C. Lee, R. B. Brufau, K. Ding and V. Narayanan, "Token Adaptive Vision Transformer with Efficient Deployment for Fine-Grained Image Recognition," *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Antwerp, Belgium, 2023, pp. 1-6, doi: 10.23919/DATE56975.2023.10137239.

Background: Token Adaptive Vision Transformer (TAVT)

Performance with token pruning:

- Stanford Birds dataset: 200 classes, 5994 training 5972 testing images
- Performance model – TAVT-P
  - 0.5% higher accuracy while requiring 43% compute of ViT
  - 2x faster on GPU / 2.6x faster on CPU
  - Reduce memory usage by 2.5x
- Efficient model – TAVT-E
  - Within 1% accuracy, requiring 33% compute of ViT
  - 2.7x faster on GPU / 3.6x faster on CPU

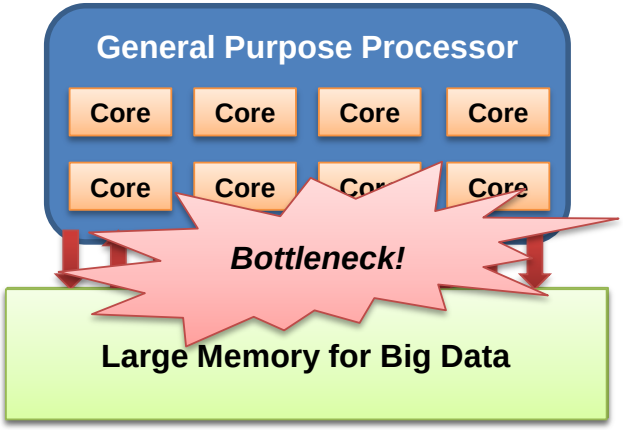
Model	Acc	Latency(ms)	FLOPs	Mem usage (GB)
VIT	90.6	134 / 2232*	1.00x	9.41
TAVT-P	91.1	63 / 850*	0.43x	3.8
TAVT-E	89.8	50 / 623*	0.33x	2.9

TAVT serves as the algorithm design of Primate

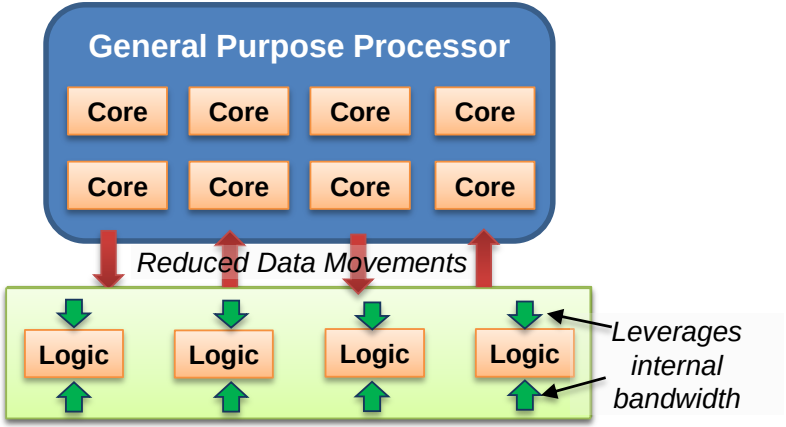
C. Lee, R. B. Brufau, K. Ding and V. Narayanan, "Token Adaptive Vision Transformer with Efficient Deployment for Fine-Grained Image Recognition," *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Antwerp, Belgium, 2023, pp. 1-6, doi: 10.23919/DATE56975.2023.10137239.

Processing in Memory (PIM)

- Traditional Von Neumann architecture suffers from performance bottleneck at memory interface connecting compute and storage
- Processing in Memory (PIM) alleviates this issue by exploiting internal memory bandwidth and parallelism using logic units embedded into memory



Conventional Architecture



Processing In-Memory

PIM can be achieved on analog devices (RRAM) and digital devices (DRAM, SRAM)

Motivation: combining PIM and token pruning for transformers

Existing works have proposed in-memory architecture for transformer inference based on HBM-based DRAM architecture.

However, naively combining PIM and token pruning yields unsatisfactory results.

- With 3.2x less FLOPS compared to the unpruned model, we get 1.5x speedup

How to effectively synergize PIM and token pruning?

Hardware challenges

- No support for in-memory token selection

Scheduling and mapping challenges

- Memory partitioning and mapping difficulties due to different layer dimensions
- Low and inconsistent memory utilization

Goal: bridging the gap between reduced computation and overall speedup

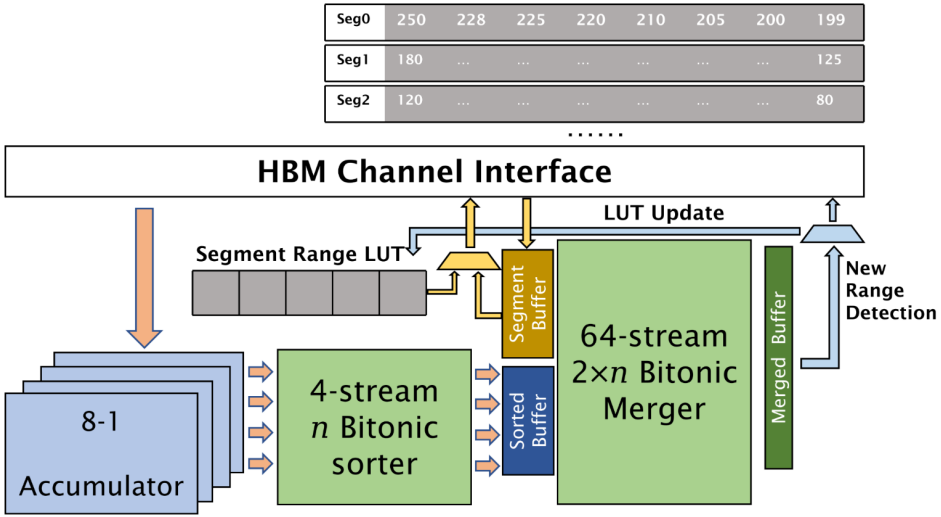
M. Zhou, W. Xu, J. Kang and T. Rosing, "TransPIM: A Memory-based Acceleration via Software-Hardware Co-Design for Transformer," *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, Seoul, Korea, Republic of, 2022, pp. 1071-1085, doi: 10.1109/HPCA53966.2022.00082.

HW Challenge: in-memory token selection

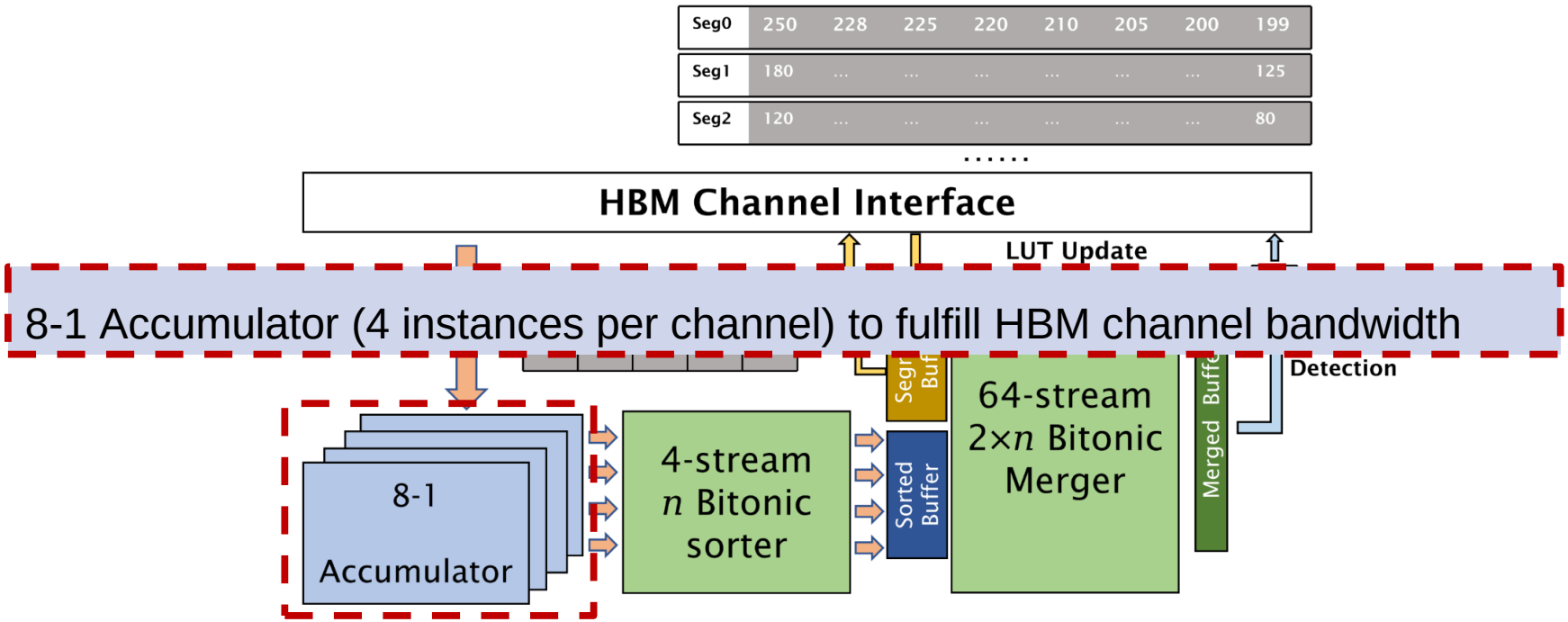
Challenge: lack of token sorting and selection mechanism in PIM architecture.

- Naive solution: forward tokens to host CPU for sorting,
  But it incurs up to 25% overhead in inference latency

PRIMATE Solution: dedicated channel-level in-memory Top-k Engine (TE)

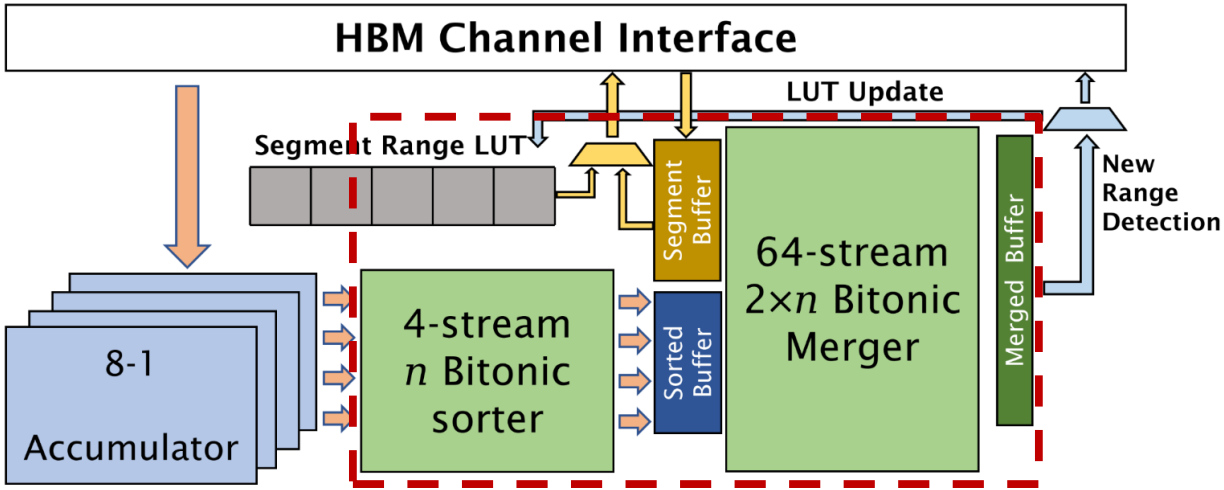


Top-K Engine (TE) design

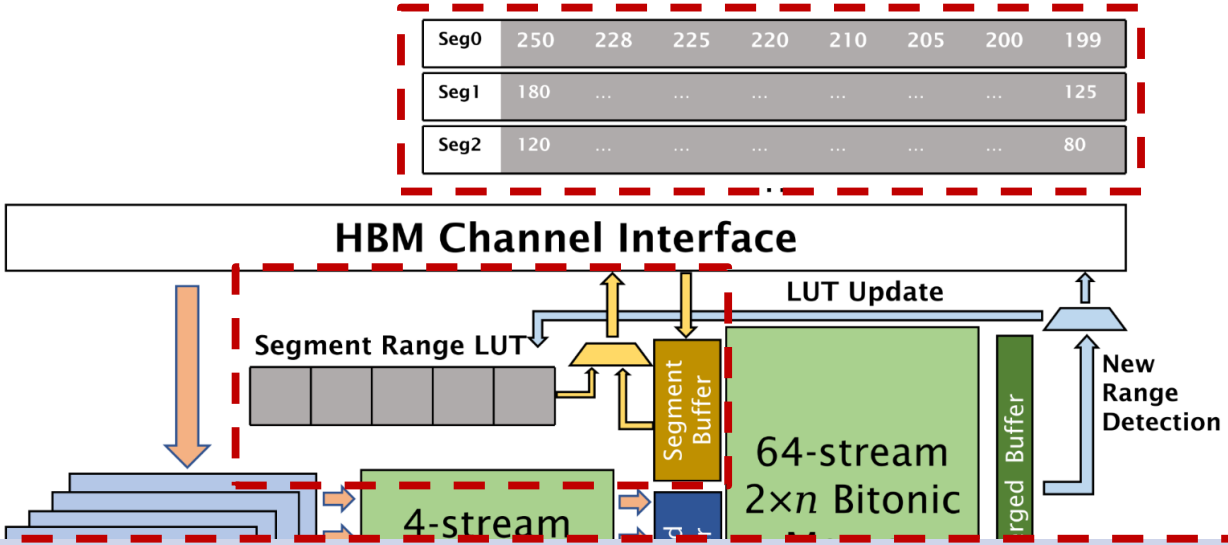


Top-K Engine (TE) design

- Bitonic sorter and merger
  - 4-stream (4 input values per cycle) high-throughput parallel bitonic sorter
  - 64-stream bitonic merger



Top-K Engine (TE) design



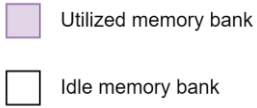
Segmented top-k buffer:

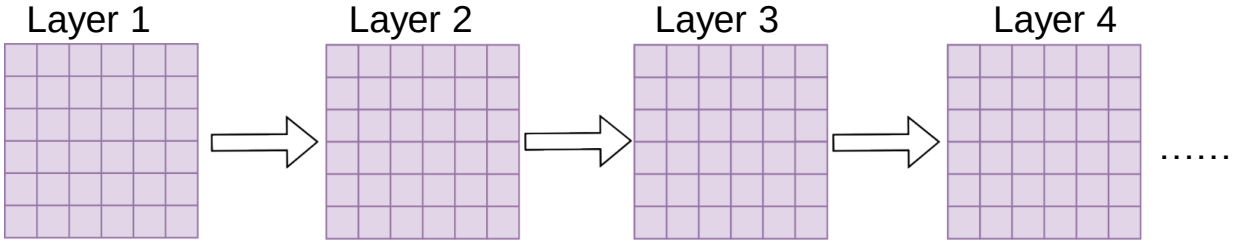
- Keeps results of current top-k tokens and ranges look-up tables
- Segmented design reduces insertion overhead
- Keeps track of ranges of each segment for easy location of target segment

Utilization challenge with diminishing layer sizes

Existing works like TransPIM assumes consistent layer dimensions during mapping.

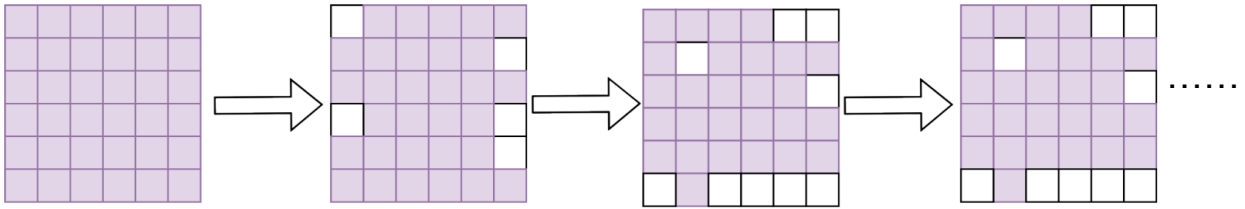
TransPIM on regular transformer





Without pruning, TransPIM keeps high utilization throughout inference

TransPIM on token-pruning workloads

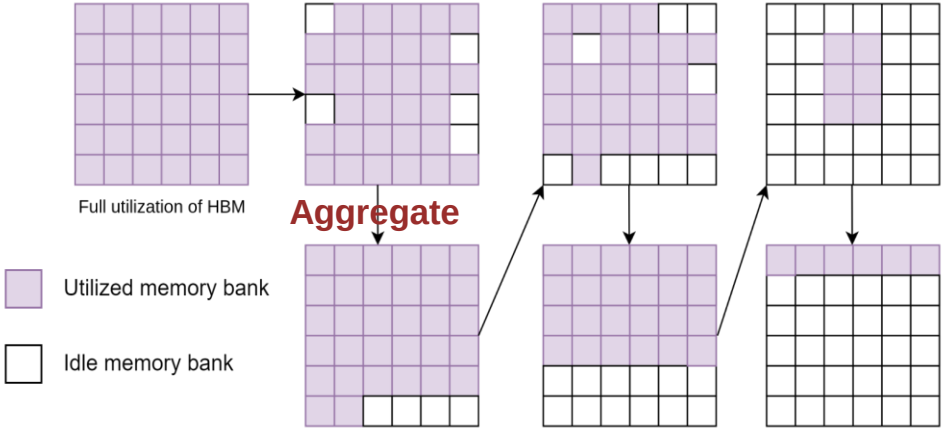


With pruning, SoTA cannot retain utilization due to sparse pruning patterns

Sparse token layout from pruning ➔ larger communication overhead
Sparse memory utilization ≠ memory saving as PIM highly depends on

Utilization challenge with diminishing layer sizes

- Naive solution: include a token aggregation operation after each layer
- Results:
  - Achieved limited throughput improvement
    - Result of improved data locality
  - Consolidated memory utilization but leaves chunks of underutilization, still
    - Can we utilize idle banks?

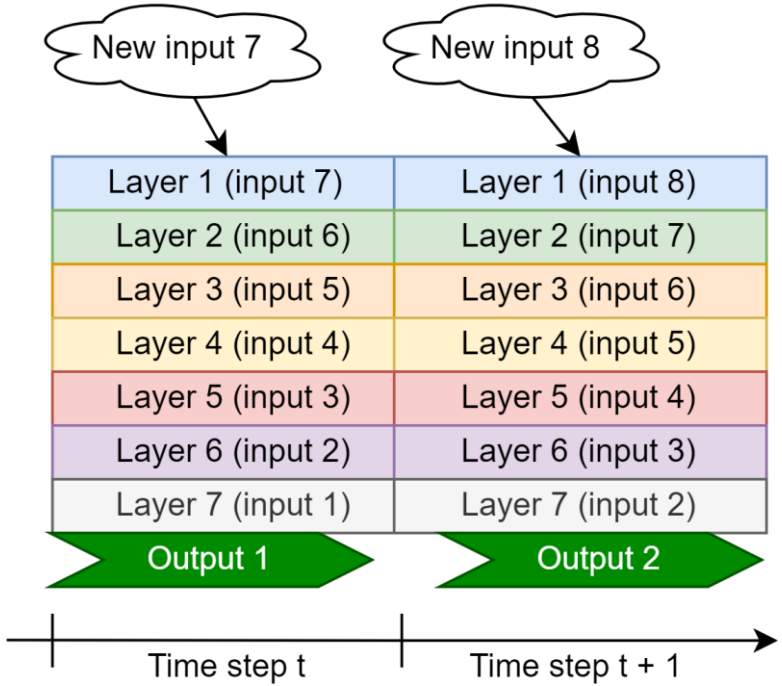


Scheduling and mapping challenge: diminishing layer sizes

PRIMATE solution: in-memory pipelining

We leverage the important fact that “how much to prune” is predefined at inference time.

- Allocate fixed size partitions for each layer
- Maximize memory utilization by allowing simultaneous processing of multiple layers corresponds to different input queries
- With inter-layer data forwarding, we construct an in-memory pipeline to significantly improve throughput



Naive Pipeline Partitioning

First, we try two naive schemes in mapping layers to HBM partitions.

- Evenly slice memory into equally sized partitions
- Weighted partitioning considering number of tokens per layer

Both schemes results in poor performance.

Tradeoff space between partition size and overhead:

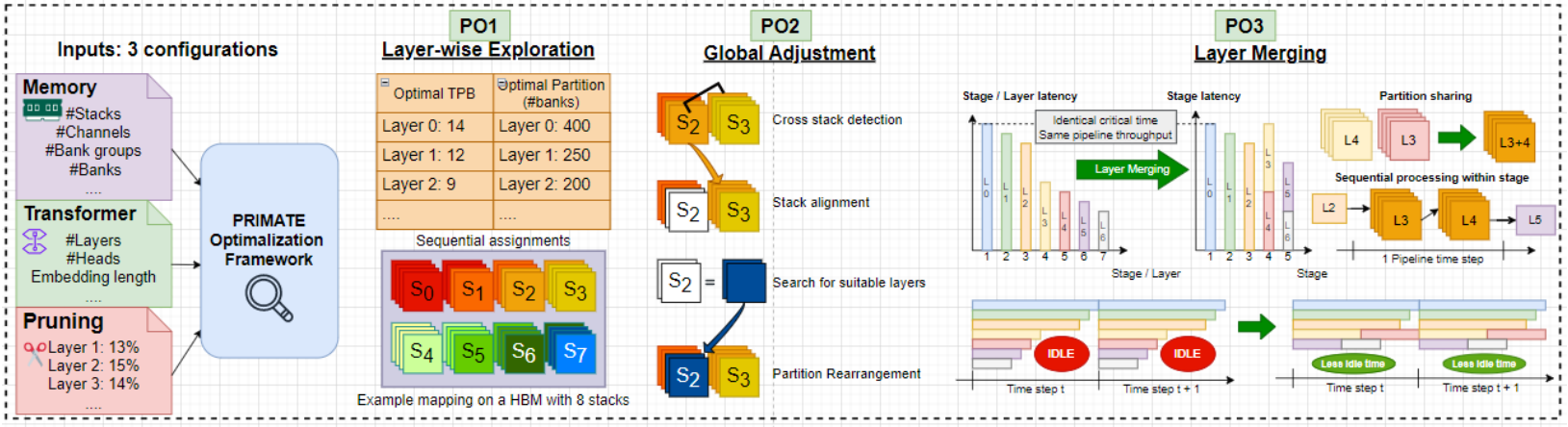
- Larger partition size → more communication overheads
- Smaller partition size → more congestion in sharing compute units

This tradeoff space needs careful exploration.

Primate pipeline partitioning

- Aspect 1: optimal partition sizes for all transformer layers
- Aspect 2: scheduling and mapping with knowledge of memory configuration
- Aspect 3: pipeline stage runtime balancing

We propose a comprehensive, progressive optimization framework, PrimateOpt.



PrimateOpt: Layer-wise Exploration (PO1)

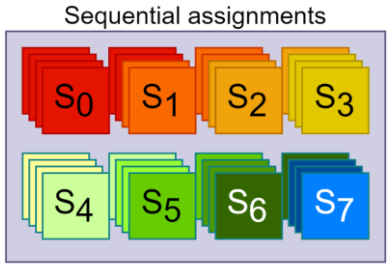
Goal: explore the optimal memory partition size for a given layer

Tradeoff space:

- Metric: token density / tokens per bank (TPB), dictates layer partition size given # tokens
- Smaller TPB: more communication overhead during computation
- Larger TPB: queuing delays in sharing compute units

Search among valid partition configurations and arrive at best partition sizes.

Optimal TPB	Optimal Partition (#banks)
Layer 0: 14	Layer 0: 400
Layer 1: 12	Layer 1: 250
Layer 2: 9	Layer 2: 200
....	....



Result: Sequential assignments of layer partitions across available memory space

PrimateOpt: Global Adjustment (PO2)

Start: sequentially allocated partitions from PO1

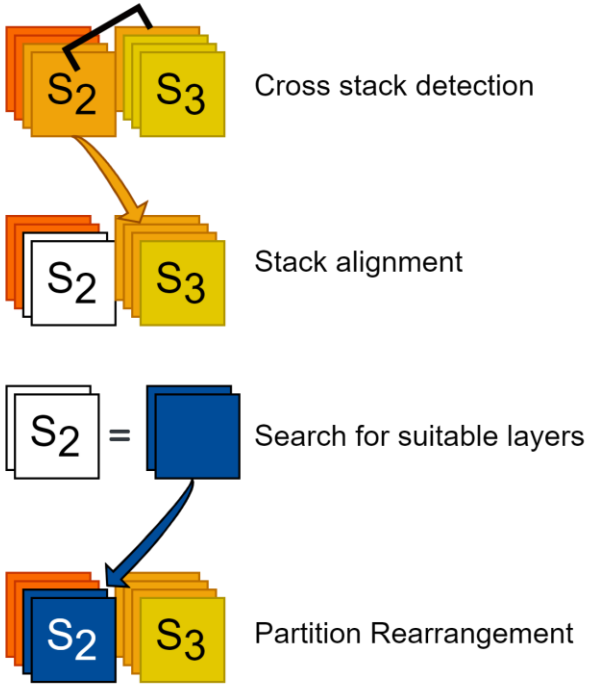
Problem:

- Some partitions are placed cross HBM stack
- Additional communication overheads

Solution: *Stack Alignment*

- Identify layers assigned to cross-stack partitions
- “Push back” partitions to align with stack boundaries
- Find another partition to fill the gap created by stack alignment
- Iterative process that minimizes stack crossing

Result: a mapping with minimal communication overhead due to stack crossing



PrimateOpt: Layer merging (PO3)

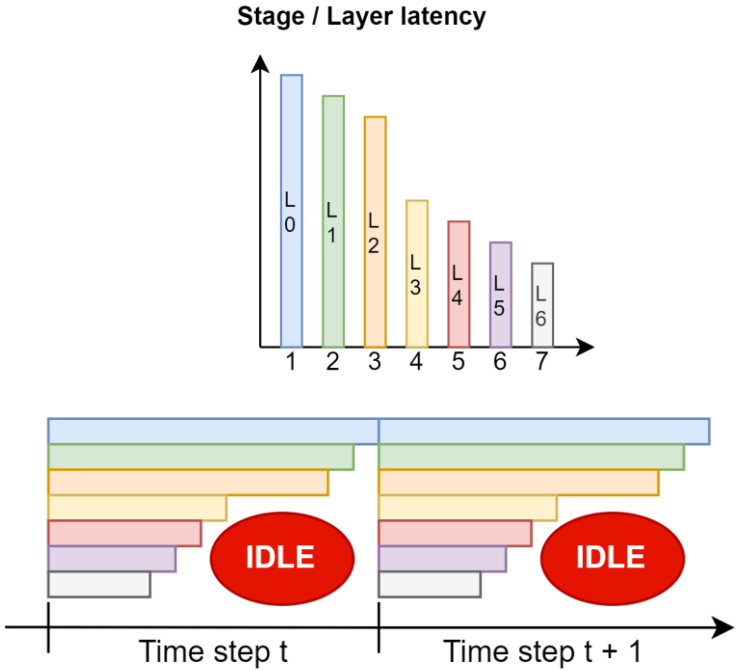
Start: stack-aligned mapping from PO2

Problem:

- *Temporal* underutilization in memory due to uneven layerwise runtimes

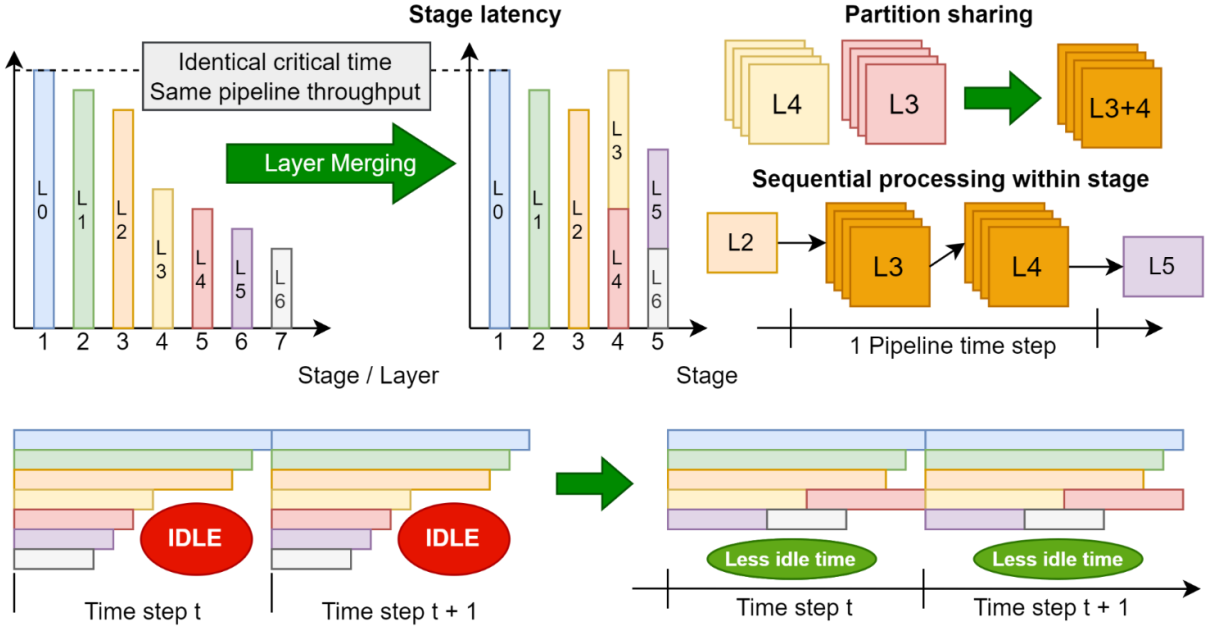
Solution: *layer merging*

- Allow 1+ faster running partitions to be merged
- These layers share the largest partition among them, in which they are sequentially executed
- Does not impact throughput as critical stage runtime in pipeline is unchanged



Result: optimized pipeline mapping with balanced stages

PrimateOpt: Layer merging (PO3)



Layer merging constraint: layers merged must be in immediately sequential order in the model

Evaluation: experiments setup

Baseline: TransPIM (HPCA'22)

Architectural simulation:

- HBM2E in 10nm technology
- Memory organization
  - 8 stacks, 16GB capacity each
  - 256b channel bandwidth
  - 16 channels per stack
- HBM frequency: 500 MHz
- Performanced based on PIM simulator
- Simulator parameters:

t_{RC}	t_{RCD}	t_{RAS}	t_{FAW}	e_{ACT}	e_{LSA}	e_{GSA}	e_{IO}
45 ns	16 ns	29 ns	12 ns	909 pJ	1.51 pJ/b	1.17 pJ/b	0.80 pJ/b

Evaluation: experiments setup

- Workloads

Workload	Model	Dataset	Layers	Sequence len
W1	ViT	Stanford Dogs	12	786
W2	ViT	CUB-200-2011	12	3137
W3	BERT	SST-2	12	128
W4	RoBERTa	Hyperpartisan News	12	4096

- Experiments conducted

BS1	Current SOTA (TransPIM [27])
BS2	TransPIM [27] with pruning and without aggregation (Sec. III-A)
BS3	TransPIM [27] with pruning and aggregation support (Sec. III-A)
NP1	Naive Pipeline: Equal partitioning of entire memory (Sec. V-B)
NP2	Naive Pipeline: Weighted partitioning by input size (Sec. V-B)
PO1	PrimateOpt: Layer-wise Exploration results (Sec. V-C1)
PO2	PrimateOpt: Global Adjustments (Sec. V-C2)
PO3	PrimateOpt: Layer merging (Sec. V-C3)

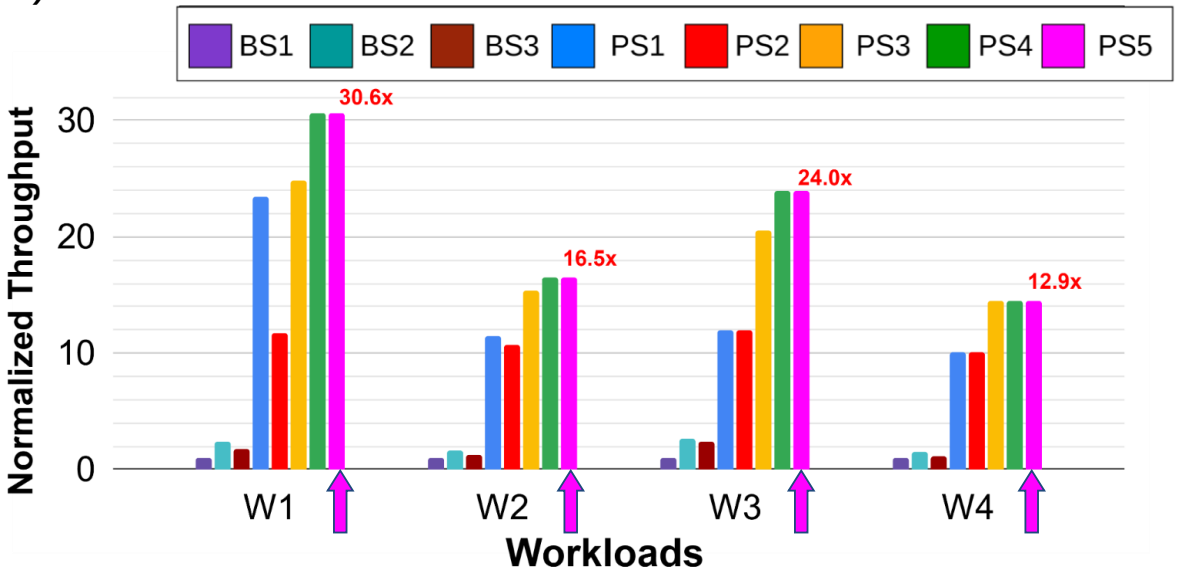
BS1-3: Baseline Schemes

NP1-2: Naive Optimizations

PO1-3: This work

Evaluation: normalized throughput

With all optimizations, Primate (PO3) achieves up to 30.6x better throughput (inferences / s) over baseline

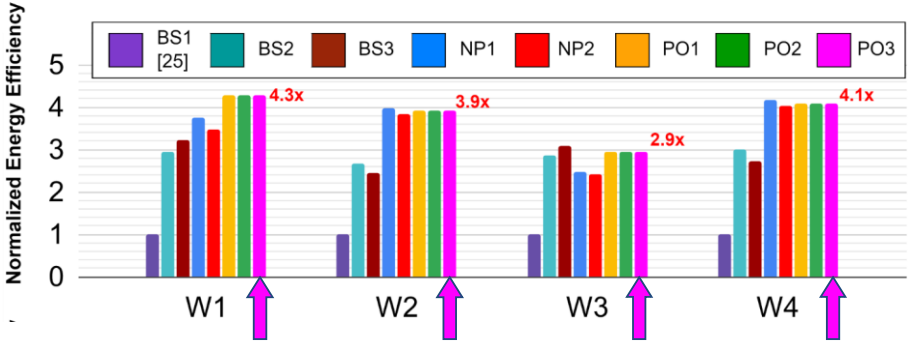


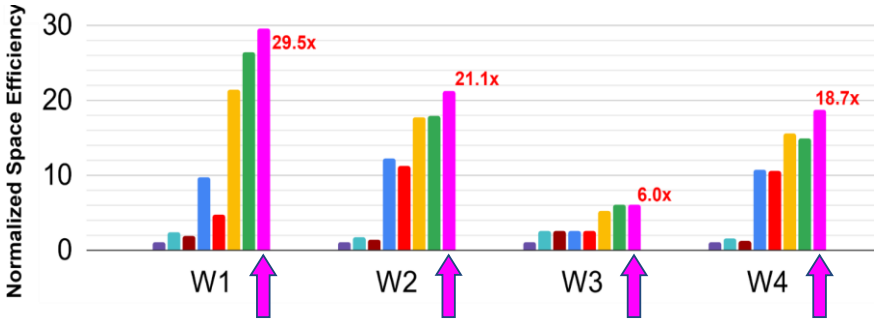
This work (final optimization)

Evaluation: energy and space efficiency

With all optimizations, Primate achieves

- up to 29.5x better space efficiency (inferences / s / GB)
- up to 4.3x better energy efficiency (inferences / s / J)





↑ This work (final optimization)

Evaluation: overhead analysis

Top-K Engines:

- Implementation:
  - Verilog generated by the Spiral Project and functionally verified in Xilinx Vivado
  - Design synthesized with Synopsys Design Compiler
  - Considered overheads for mixing designs of memory and compute
- Evaluation:
  - Area: 2.53 mm^2 (total area of 16 channel-level sorters)
  - 2.3% of HBM2E area per stack
  - Power: 1.3W

There is no impact on HBM capacity due to small size of TEs.

Primate Opt

- Memory partitioning and data forwarding are implemented as memory routines run before each time step. These overheads are included in performance valuations.

Summary

- We propose Primate to synergize token pruning and Processing in Memory (PIM) to improve throughput and efficiency of transformer inference
- HW challenge: in-memory sorting mechanism
  - We propose low cost in-memory top-k selection units
- Scheduling and mapping challenge:
  - We propose a pipelining scheme to maximize memory utilization
  - We propose PrimateOpt, a comprehensive optimization framework to optimize pipelining configurations
- Results
  - Baseline: TransPIM [HPCA'22]
  - We achieve 30.6× better throughput, 29.5× better space efficiency, and 4.3× better energy efficiency

Thank you!

System Energy Efficiency Lab
seelab.ucsd.edu

Evaluation: per-layer runtimes

- NP1 & NP2: naive memory partitioning yields low performance
- PO1: improves layer runtime due to adequate partition sizes, but suffers from cross stack communications (spikes on curve)
- PO2 & 3: Further improves layer runtimes by eliminating cross stack partitions

Primate progressively optimizes layerwise runtimes until it monotonically declines, adhering to the progressive pruning pattern.