SSDI8: Accurate and Efficient 8-Bit Quantization for State Space Duality

Hyunwoo Kim[1], Byoungchan Ko[2], Minseok Kang[2], Minwoo Kim[2], Dongin Lee[3], Jaehoon Lee[4], Sungroh Yoon[3,4,5†], Dahuin Jung[1†]

[1]Department of Artificial Intelligence, Chung–Ang University

[2]School of Computer Science and Engineering, Soongsil University

[3]Department of Electrical and Computer Engineering, Seoul National University

[4]Interdisciplinary Program in Artificial Intelligence, Seoul National University

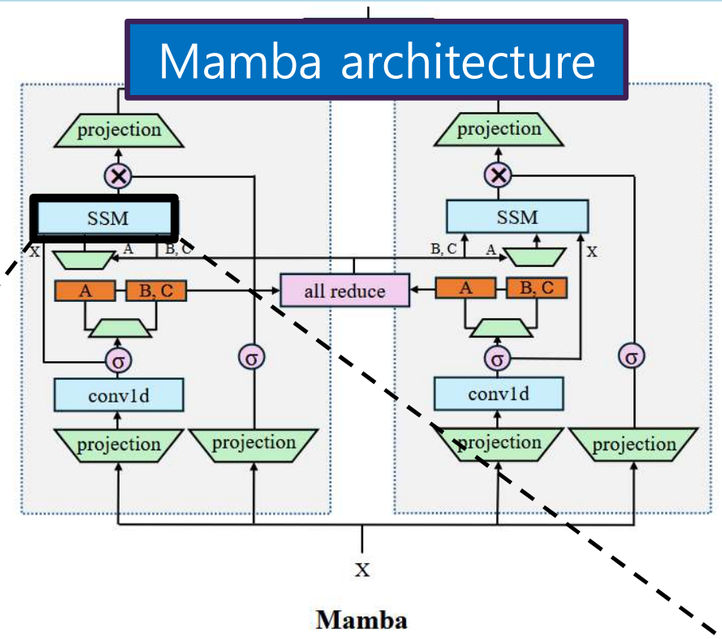
[5]AIIS, ASRI, INMC, and ISRC, Seoul National University

[*]Equal contribution

[†]Corresponding Authors

Background & Motivation

- Transformer-based LLMs incur computation and memory costs that scale quadratically with sequence length.
- Mamba, an SSM-based architecture, achieves performance comparable to or better than recent architectures.
- However, SSM is difficult to parallelize on modern accelerators and challenging to scale to larger model sizes.



State Space Model Equation

Continuous

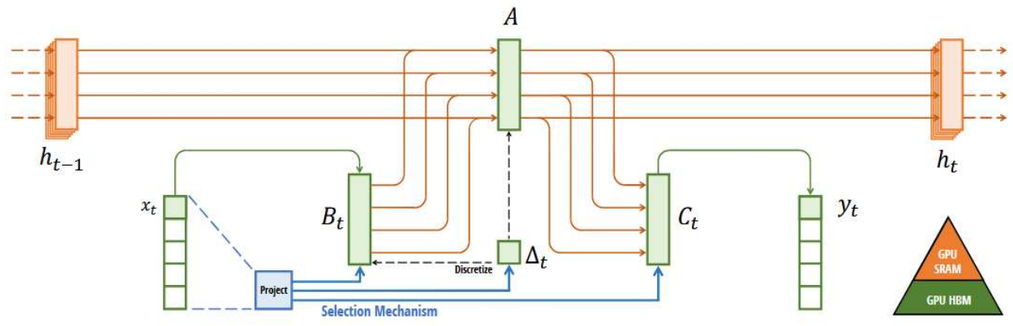
$$h'(t) = Ah(t) + Bx(t)$$
$$y(t) = Ch(t)$$

Discretized (ZOH)

$$h_t = \bar{A}h_{t-1} + \bar{B}x_t$$
$$y_t = Ch_t$$

$$\bar{A} = \exp(\Delta A), \qquad \bar{B} = (\Delta A)^{-1}(\exp(\Delta A) - I) \cdot \Delta B$$

Selective State Space Model

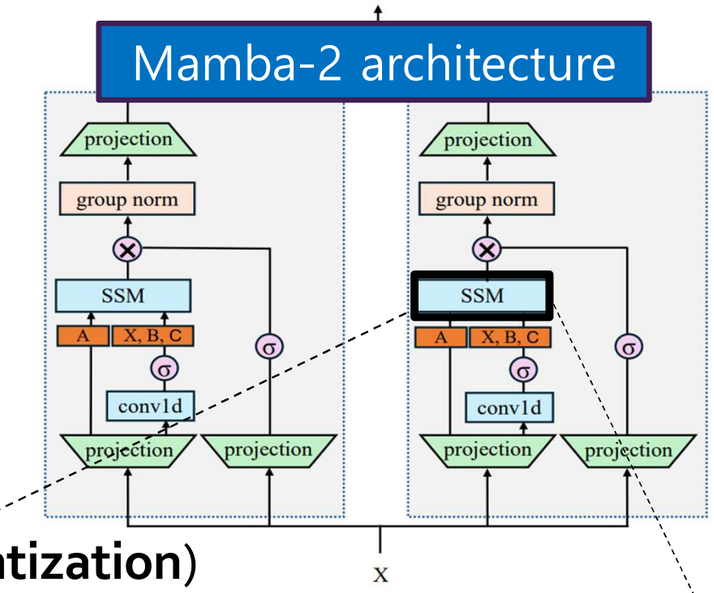


CAU HAI Human-centered AI

https://openreview.net/forum?id=tEYskw1VY2&nesting=2&sort=date-desc
https://arxiv.org/abs/2406.07887

Background & Motivation

- Mamba-2 introduces the **Structured State Space Duality(SSD)**, a hybrid design that integrates recurrent model with attention mode.
- It employs a dual representation that improves GEMM utilization, yielding higher throughput on GPUs and TPUs.
- Mamba-2 scales effectively to over 8B parameters, but the resulting memory and latency overhead calls for efficient compression. (**SSD-Quantization**)



State Space Model Equation

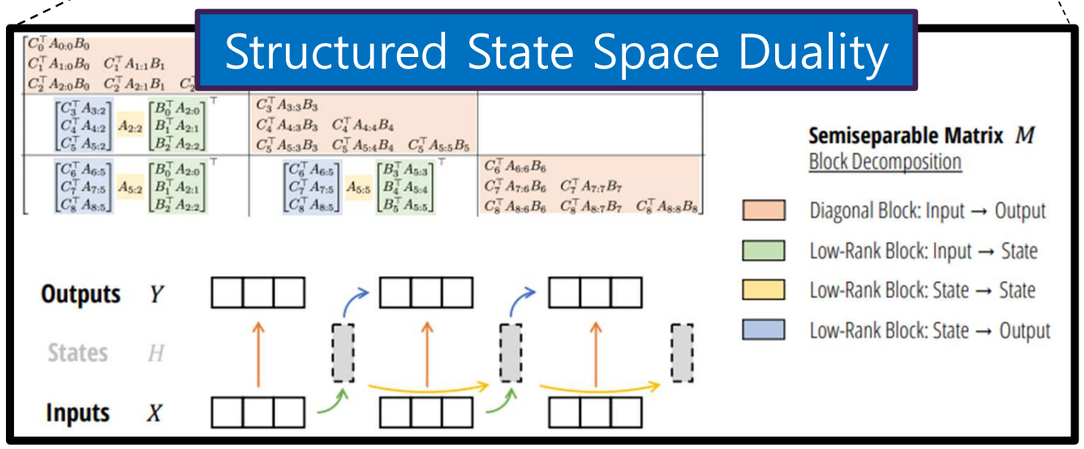
Continuous

$$h'(t) = Ah(t) + Bx(t)$$
$$y(t) = Ch(t)$$

Discretized (ZOH)

$$h_t = \bar{A}h_{t-1} + \bar{B}x_t$$
$$y_t = Ch_t$$

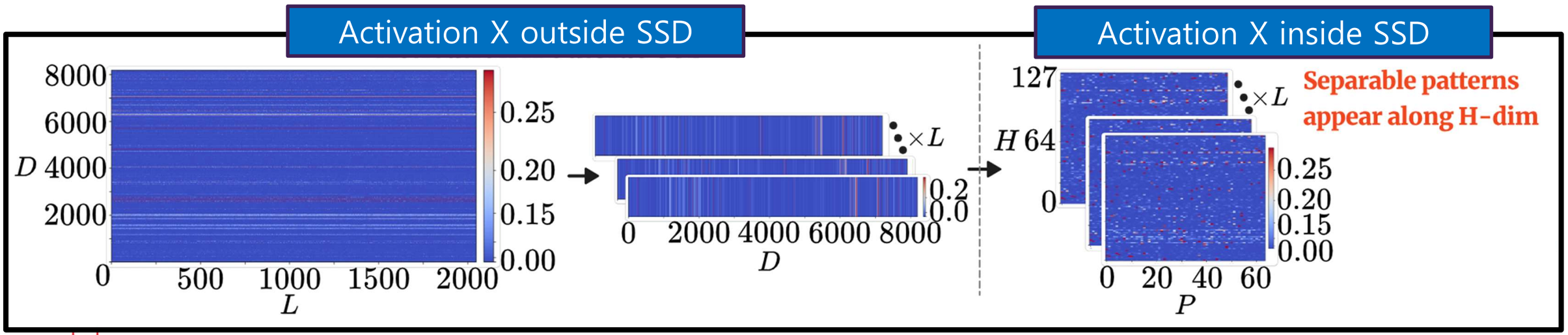
$$\bar{A} = \exp(\Delta A), \qquad \bar{B} = (\Delta A)^{-1}(\exp(\Delta A) - I) \cdot \Delta B$$



https://arxiv.org/abs/2406.07887
https://proceedings.mlr.press/v235/dao24a.html

Problems

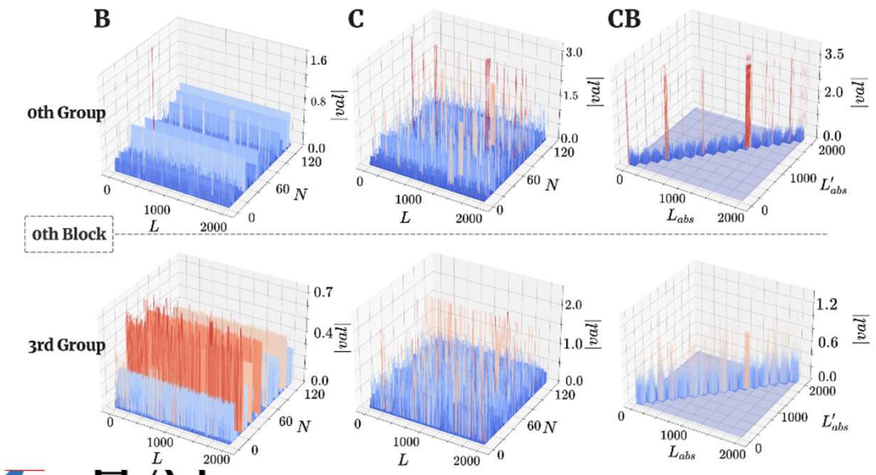
- SSD involves external-to-internal dimension changes
  that must be considered to preserve quantization performance.
- Activations are repeatedly reused across multiple modules and
  exhibit distinct outlier characteristics.
- Element-wise multiplications are tightly intertwined with matrix multiplications,
  complicating quantization.



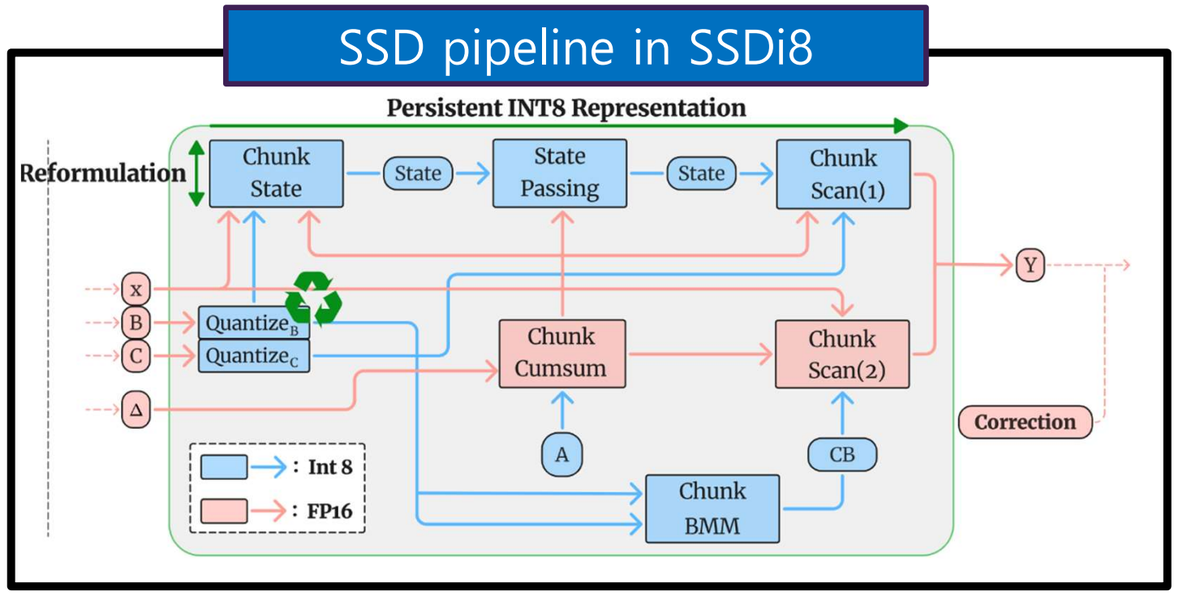
SSDi8 : Persistent INT8 Representation

- **Our Goal**: Establish a **Persistent INT8 path Representation (PIR)** within SSD to minimize latency while preserving accuracy.
- **Quantize once and reuse**: Quantize reused activation B and C once and reuse them across SSD submodules to reduce memory traffic and quantization overhead.
- **Mean correction**: Apply a per-channel mean correction after SSD to mitigate accumulated quantization error with minimal latency overhead.

Activations B and C

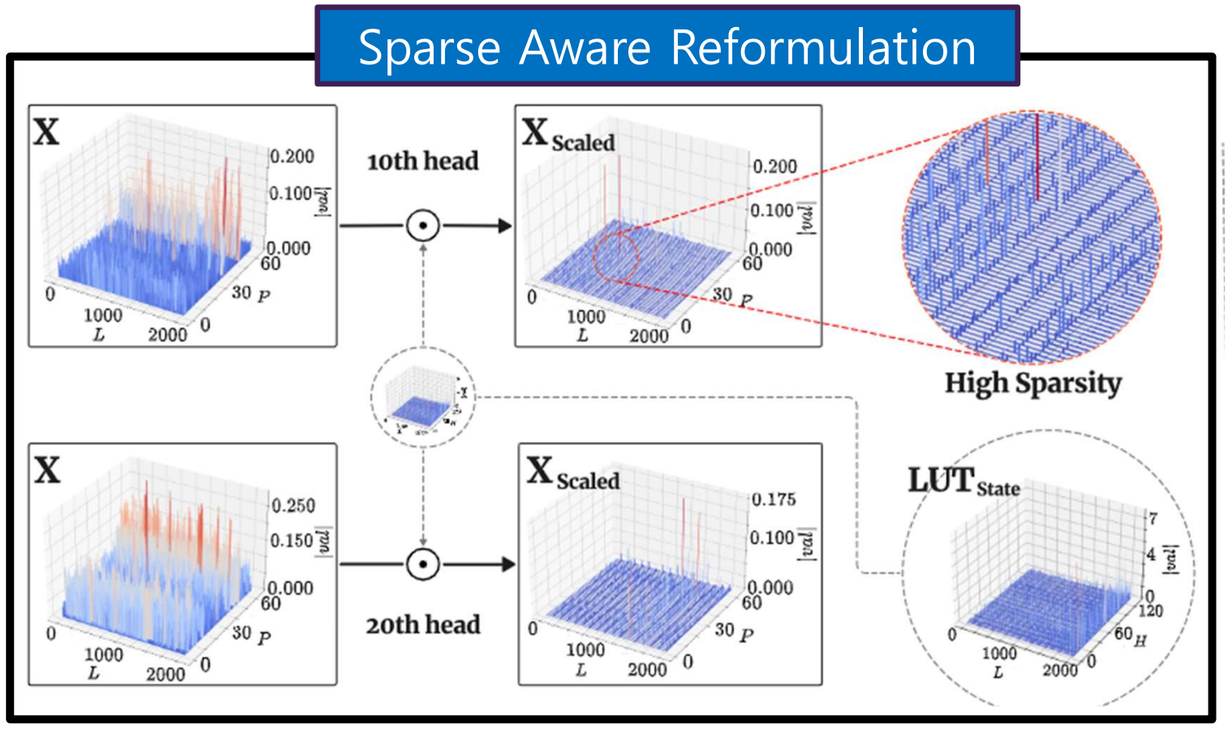


SSD pipeline in SSDi8



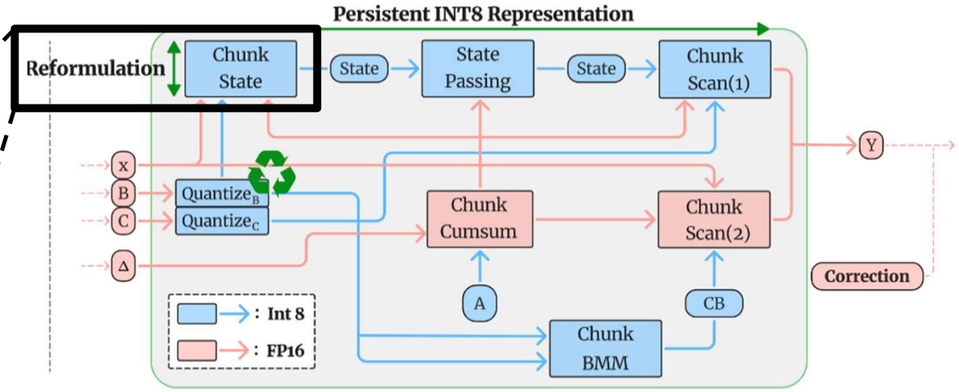
SSDi8 : Sparse Aware Reformulation

- Multiplying reused INT8 B by FP16 LUT_{state} incurs additional overhead.
- **Sparse Aware Reformulation(SAR)** preserves INT8 execution path and avoids the overhead.
- Although X_{scaled} is highly sparse, its quantization does not increase error and is theoretically justified.
- As a result, PIR is completed, enabling the SSD internal path to remain in INT8.



SAR Equation

$$State = X \times (B \odot LUT_{state}) \Rightarrow Q(State) = Q(X_{scaled}) \times Q(B)$$
$$X_{scaled} = LUT_{state} \odot X$$



Experiments

Zero-shot task results

Model	Size	Methods	Bitwidth	LA	HS	PIQA	Arc-E	Arc-C	WG	Avg.
Mamba-2	1.3B	-	FP16	65.6%	59.9%	73.3%	64.1%	33.3%	60.8%	59.5%
		Quamba	W8A8	49.8%	58.5%	71.2%	61.9%	32.1%	58.1%	55.2%
		Quamba2	W8A8	62.0%	59.2%	72.5%	63.4%	32.7%	60.0%	58.3%
			W4A8	61.0%	58.8%	72.4%	62.7%	32.6%	59.1%	57.7%
		SSDi8 (Ours)	W8A8	**64.7%**	**59.7%**	**72.7%**	**64.0%**	**32.8%**	**60.9%**	**59.1%**
			W4A8	**63.6%**	**59.2%**	**72.7%**	**63.5%**	**33.5%**	**60.4%**	**58.8%**
	2.7B	-	FP16	69.5%	66.6%	76.4%	69.5%	36.4%	64.2%	63.8%
		Quamba	W8A8	52.4%	60.4%	71.6%	62.9%	33.7%	58.0%	56.5%
		Quamba2	W8A8	66.1%	65.5%	74.4%	68.4%	**37.1%**	**63.7%**	62.5%
			W4A8	65.6%	65.1%	74.7%	68.1%	**36.1%**	62.8%	62.1%
		SSDi8 (Ours)	W8A8	**68.3%**	**66.2%**	**75.6%**	**69.0%**	36.8%	63.4%	**63.2%**
			W4A8	**67.4%**	**65.3%**	**75.6%**	**68.9%**	35.2%	**63.5%**	**62.6%**
	8B	-	FP16	70.9%	77.7%	79.7%	76.0%	48.0%	72.0%	70.7%
		Quamba	W8A8	54.0%	74.6%	77.1%	73.5%	44.2%	65.5%	64.8%
		Quamba2	W8A8	69.8%	**77.8%**	79.1%	**75.9%**	46.9%	69.0%	69.8%
			W4A8	68.8%	**77.1%**	79.1%	75.0%	46.0%	68.7%	69.1%
		SSDi8 (Ours)	W8A8	**70.4%**	77.2%	**79.6%**	75.5%	**47.2%**	**71.2%**	**70.2%**
			W4A8	**69.9%**	76.5%	**79.1%**	**75.4%**	**46.2%**	**70.6%**	**69.6%**

Perplexity results

Methods	Bitwidth	Wikitext2 Perplexity (↓)		
		1.3B	2.7B	8B
-	FP16	10.42	9.06	7.25
HAD	W8A8	11.31	11.42	8.57
	W4A8	11.63	11.85	8.79
Quamba2	W8A8	10.80	9.32	7.79
	W4A8	11.08	9.54	7.94
SSDi8 (Ours)	W8A8	**10.63**	**9.22**	**7.49**
	W4A8	**10.92**	**9.43**	**7.62**

- SSD8 preserves strong zero-shot accuracy under quantization.
- SSDi8 achieve perplexity **close to FP16**, preserving linguistic fluency and generalization.

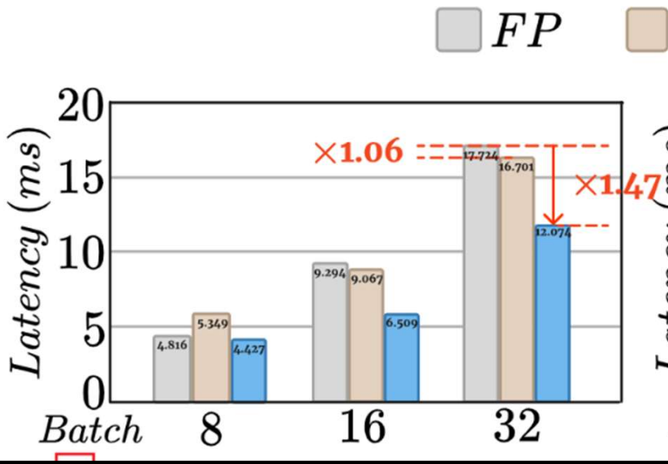
Experiments

- SSDi8 achieves substantial latency reduction, reaching up **to 1.47x speedup over FP16.**
- With Persistent INT8 Representation, $ChunkScan$ and $StatePssing$ achieve up to **1.77x and 2.25x speedup over FP16**, respectively.
- SSDi8 consistently reduces SSD latency on Orin NX 16G across sequence length, demonstrating robustness in resource-constrained environments.

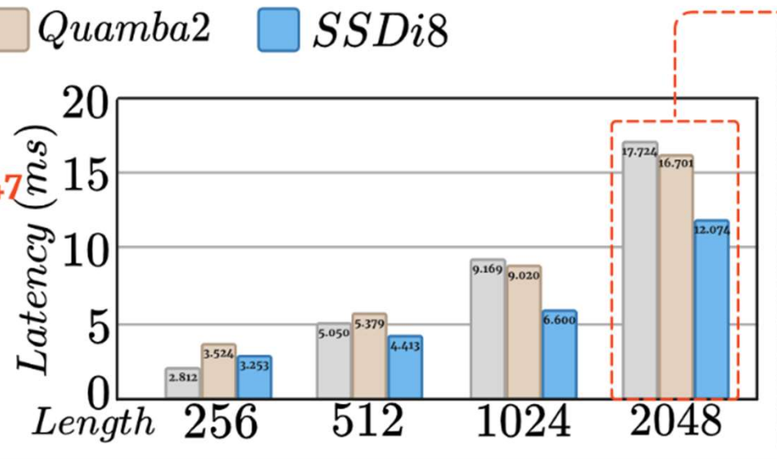
SSD latency on Orin NX 16G

GPU	Orin NX 16G			
Bitwidth	W4A8		W8A8	
Method	Quamba2	SSDi8	Quamba2	SSDi8
$L = 256$	55.30	44.71	51.03	41.30
$L = 512$	76.10	68.00	70.95	60.49
$L = 1024$	134.40	127.51	139.10	114.36
$L = 2048$	262.90	240.54	249.29	217.69

Latency vs. Batch size



Latency vs. Sequence length



Module wise latency breakdown