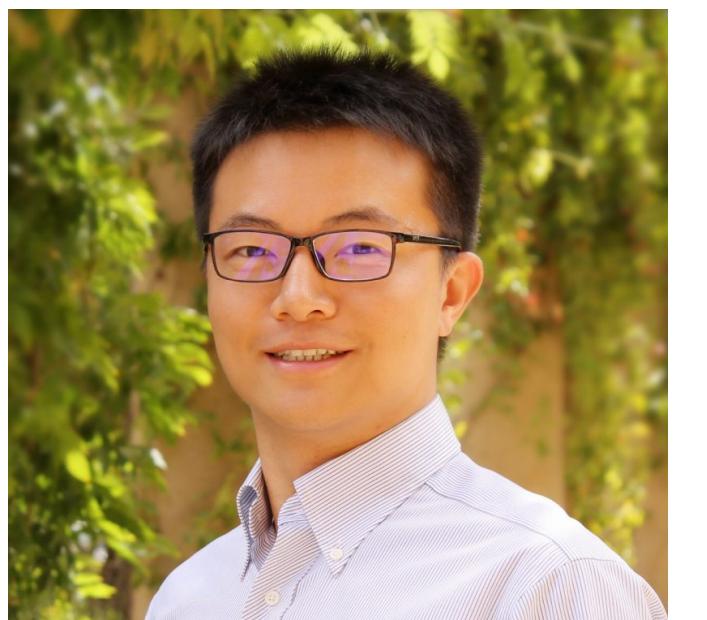


# EfficientML.ai Lecture 13

## Transformer and LLM

Part II



**Song Han**

Associate Professor, MIT  
Distinguished Scientist, NVIDIA

 @SongHan/MIT



# Lecture Plan

Today, we will cover:

## 1. Efficient inference algorithms for LLMs

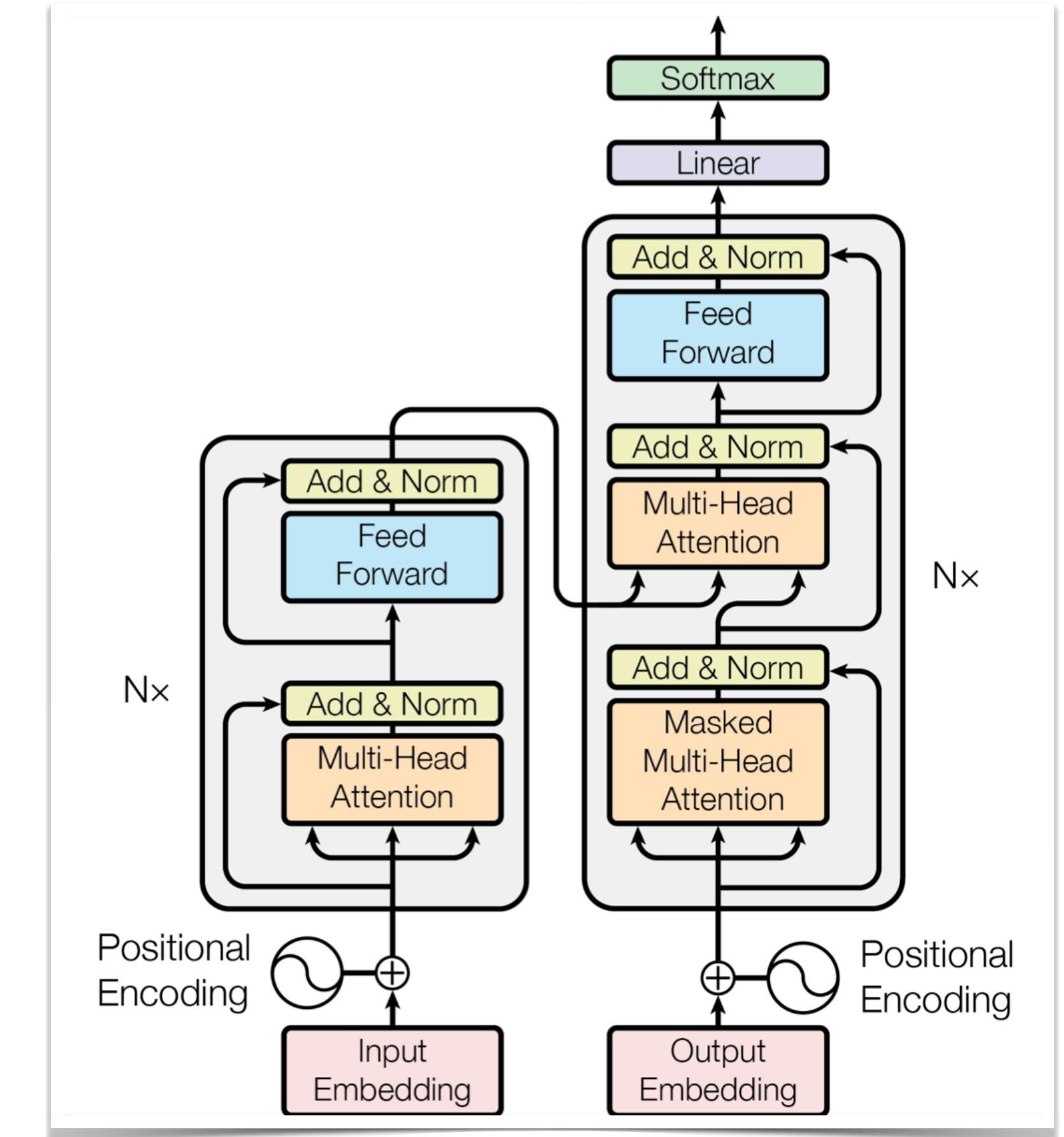
1. Quantization: SmoothQuant, AWQ, TinyChat
2. Pruning/sparsity: SpAtten, H2O, MoE

## 2. Efficient inference systems for LLMs

1. vLLM
2. StreamingLLM
3. FlashAttention
4. Speculative decoding

## 3. Efficient fine-tuning for LLMs

1. LoRA/QLoRA
2. Adapter
3. Prompt Tuning



# Lecture Plan

Today, we will cover:

## 1. Efficient inference algorithms for LLMs

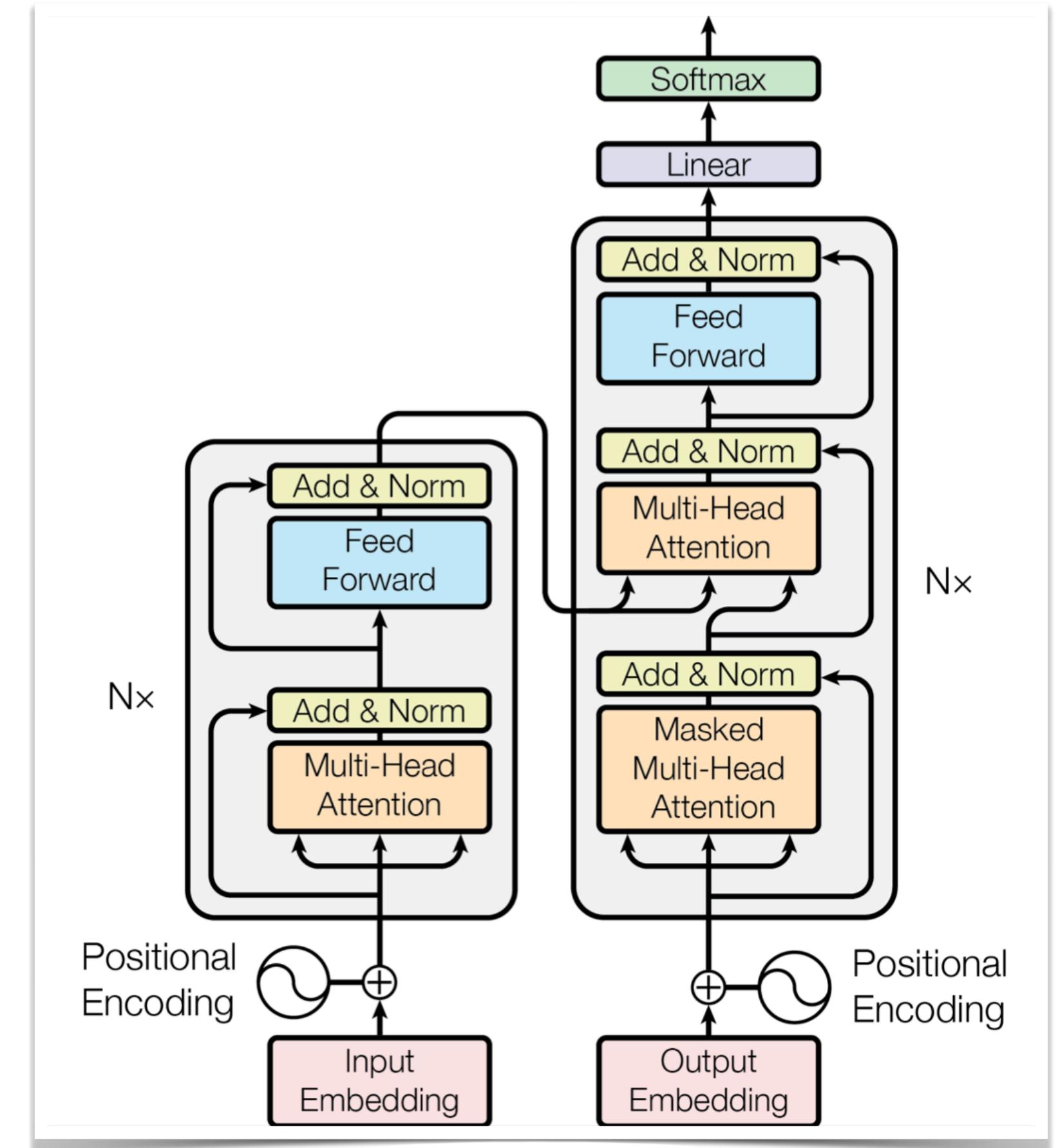
1. Quantization: SmoothQuant, AWQ, TinyChat
2. Pruning/sparsity: SpAtten, H2O, MoE

## 2. Efficient inference systems for LLMs

1. vLLM
2. StreamingLLM
3. FlashAttention
4. Speculative decoding

## 3. Efficient fine-tuning for LLMs

1. LoRA/QLoRA
2. Adapter
3. Prompt Tuning



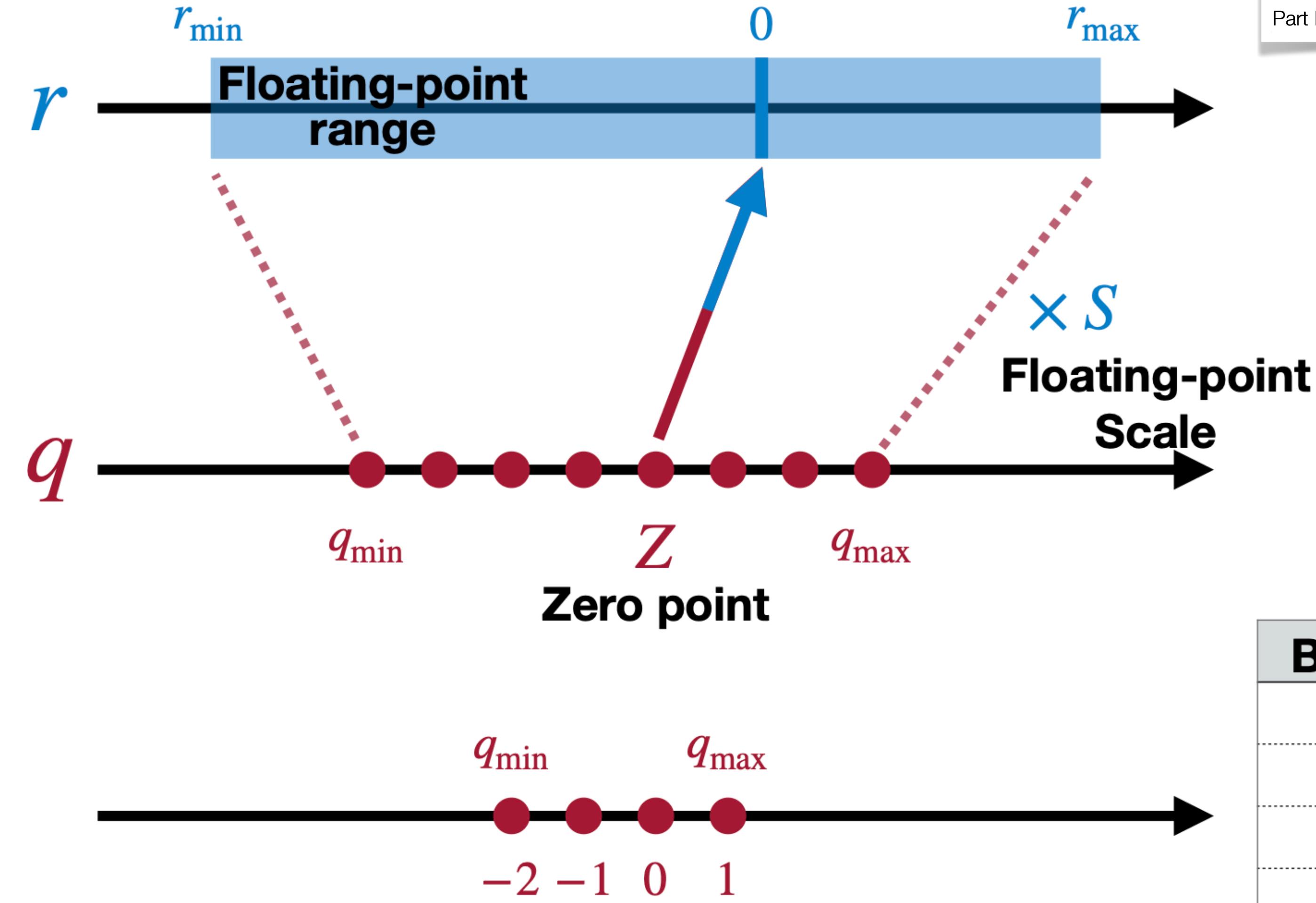
# Quantization

Lecture 05  
Quantization  
Part I



Lowers the bit-width and improves efficiency

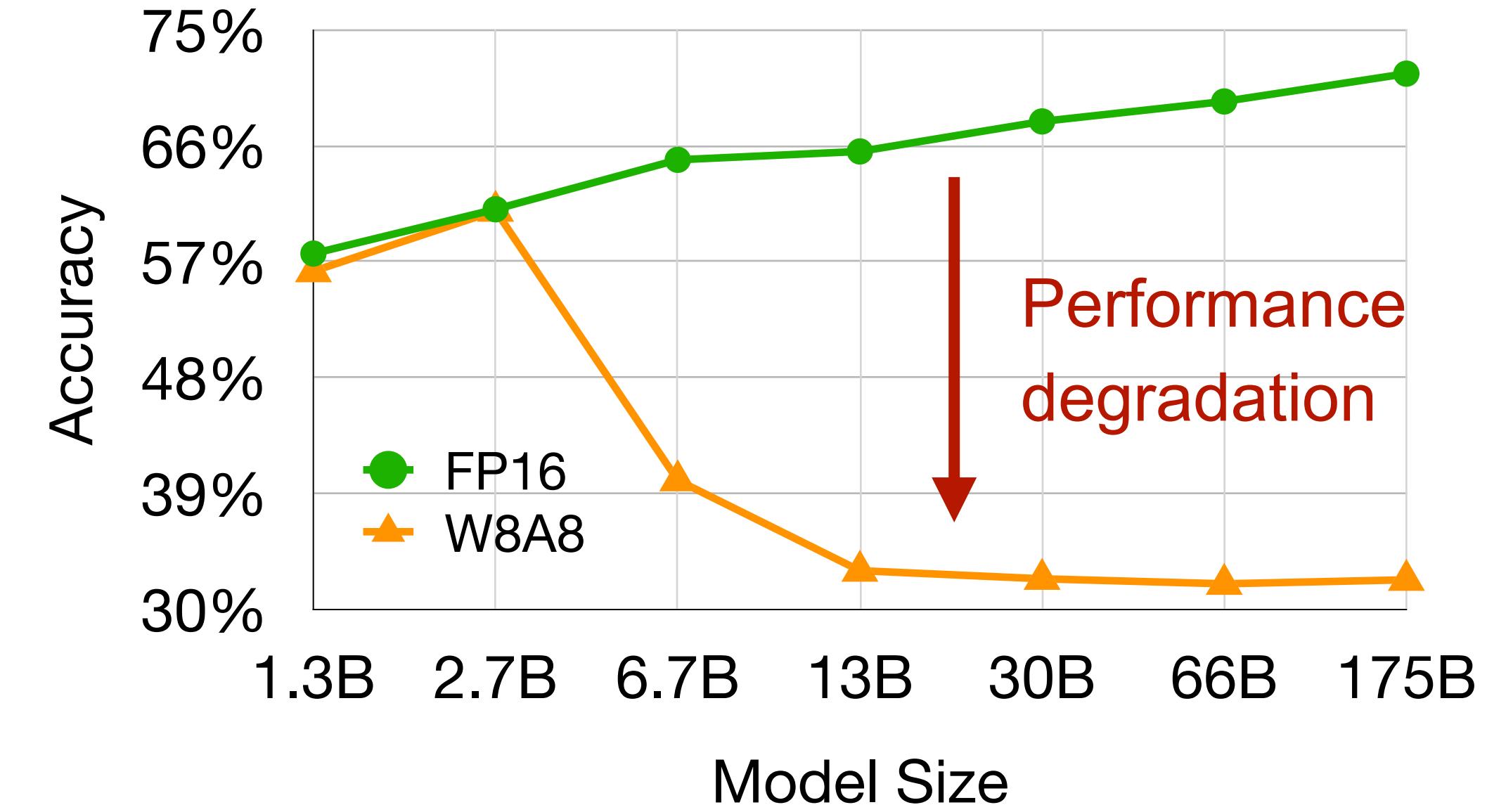
Lecture 06  
Quantization  
Part II



Binary	Decimal
01	1
00	0
11	-1
10	-2

# SmoothQuant (W8A8)

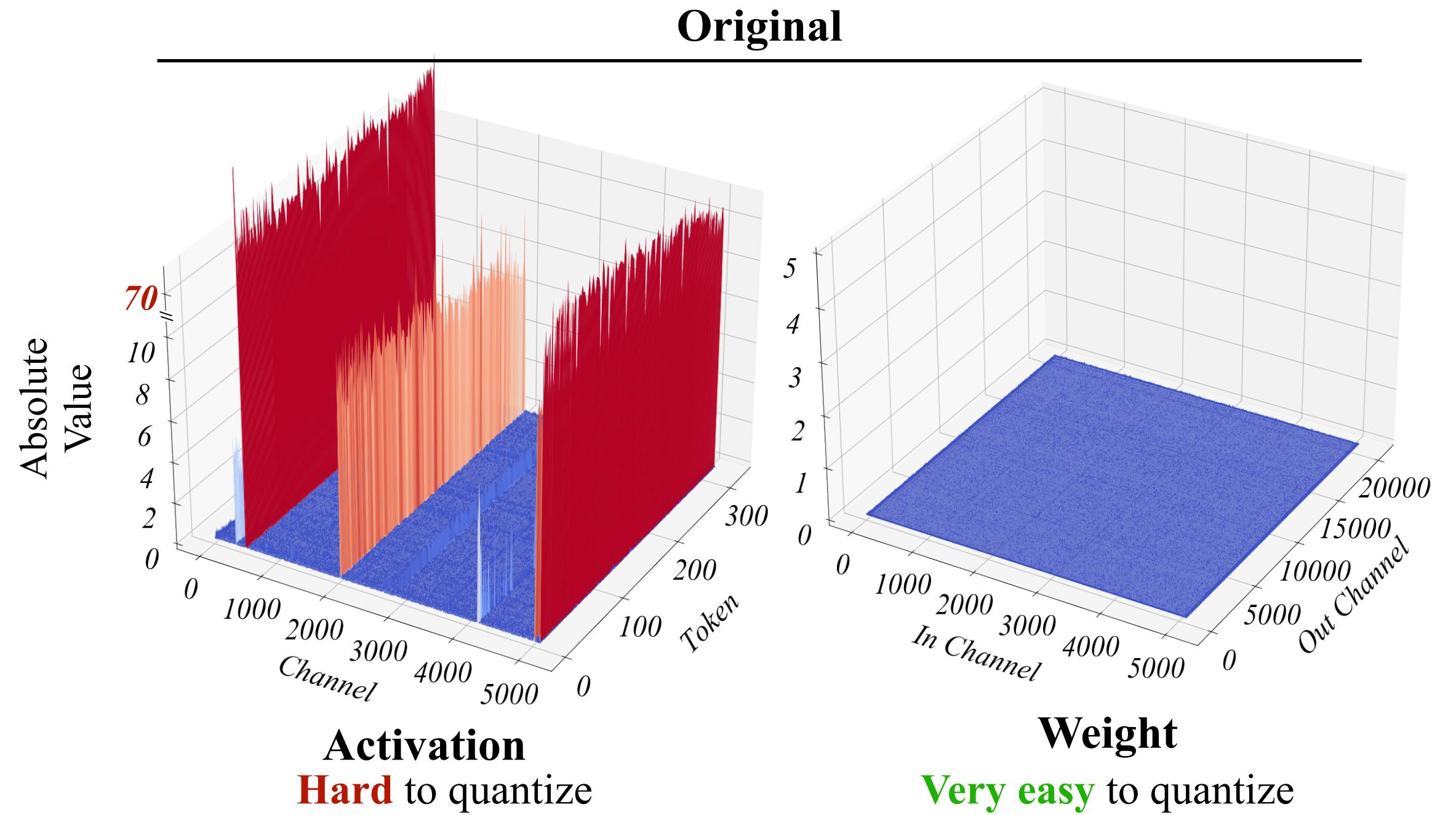
Activation outliers destroy quantized performance



- W8A8 quantization has been an industrial standard for CNNs, but not LLM. Why?
- Systematic outliers emerge in **activations** when we scale up LLMs beyond 6.7B. Traditional CNN quantization methods will destroy the accuracy.

# SmoothQuant (W8A8)

## Smoothing activation to reduce quantization error

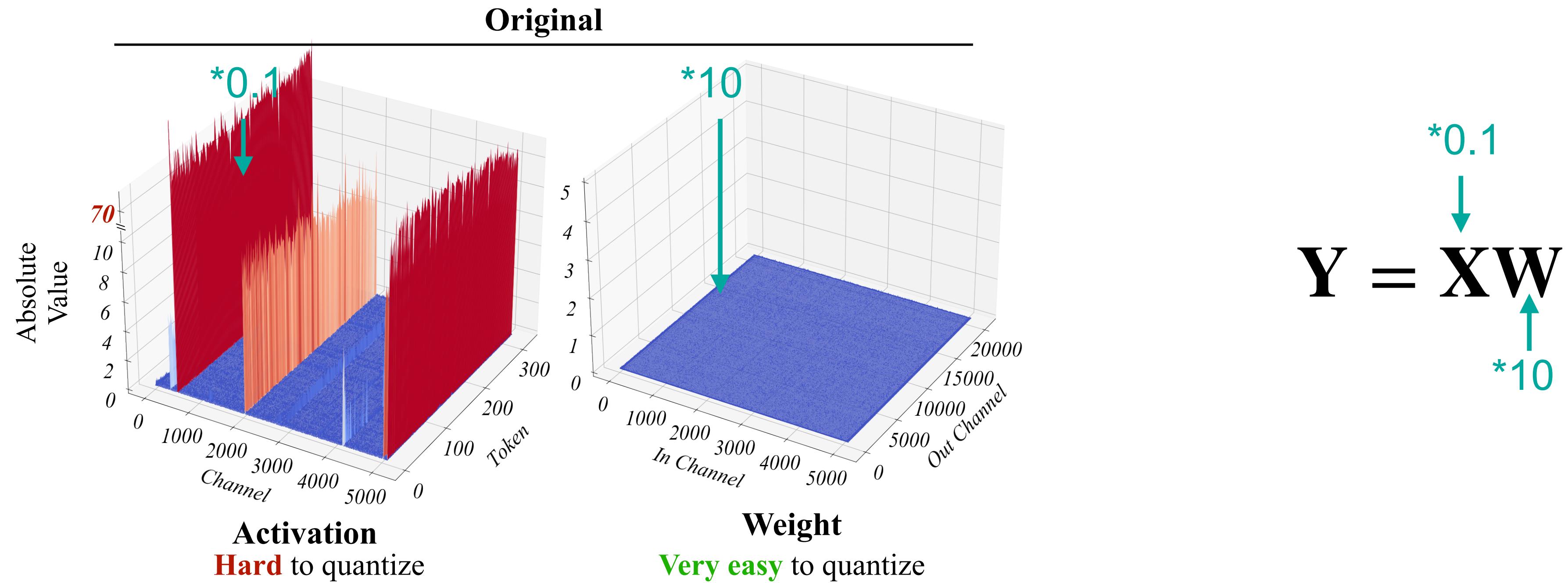


- Weights are easy to quantize, but activation is hard due to outliers
- Luckily, outliers persist in fixed channels

SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models [Xiao et al., 2022]

# SmoothQuant (W8A8)

## Smoothing activation to reduce quantization error

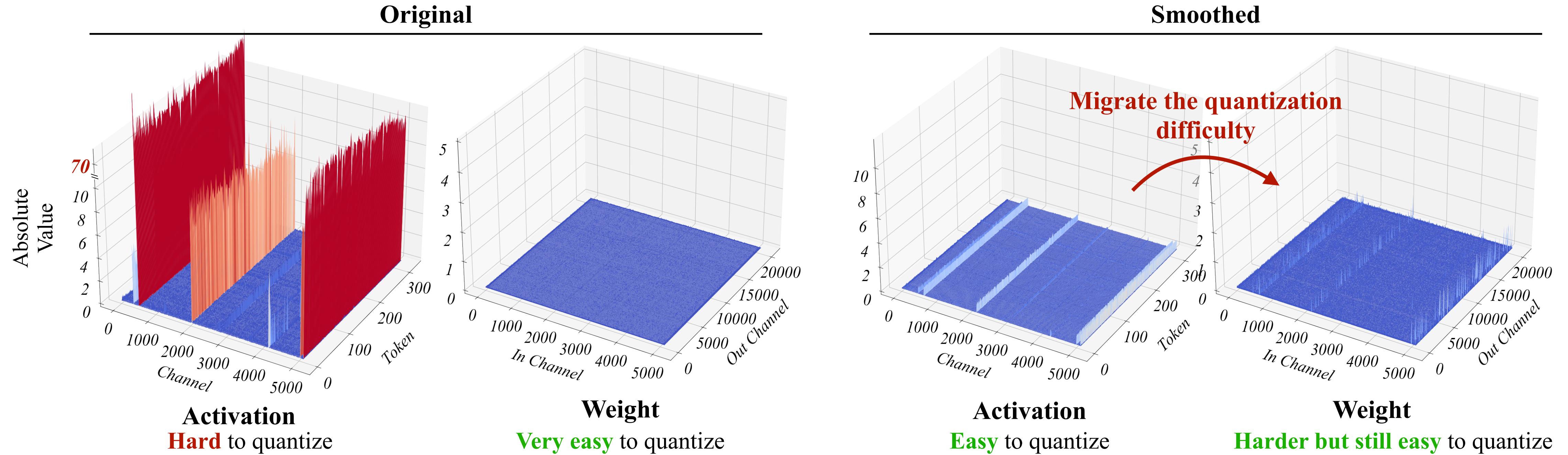


- Weights are easy to quantize, but activation is hard due to outliers
- Luckily, outliers persist in fixed channels

SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models [Xiao et al., 2022]

# SmoothQuant (W8A8)

## Smoothing activation to reduce quantization error



- Weights are easy to quantize, but activation is hard due to outliers
- Luckily, outliers persist in fixed channels
- Migrate the quantization difficulty from activation to weights, so both are easy to quantize

SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models [Xiao et al., 2022]

# SmoothQuant (W8A8)

Smoothing activation to reduce quantization error

$$\mathbf{X} \quad \begin{matrix} & \mathbf{W} \\ \begin{matrix} 1 & -16 & 2 & 6 \\ -2 & 8 & -1 & -9 \end{matrix} & * \quad \begin{matrix} 2 & 1 & -2 \\ 1 & -1 & -1 \\ 2 & -1 & -2 \\ -1 & -1 & 1 \end{matrix} \end{matrix}$$
$$\mathbf{s} \quad \boxed{\begin{matrix} 1 & 4 & 1 & 3 \end{matrix}}$$

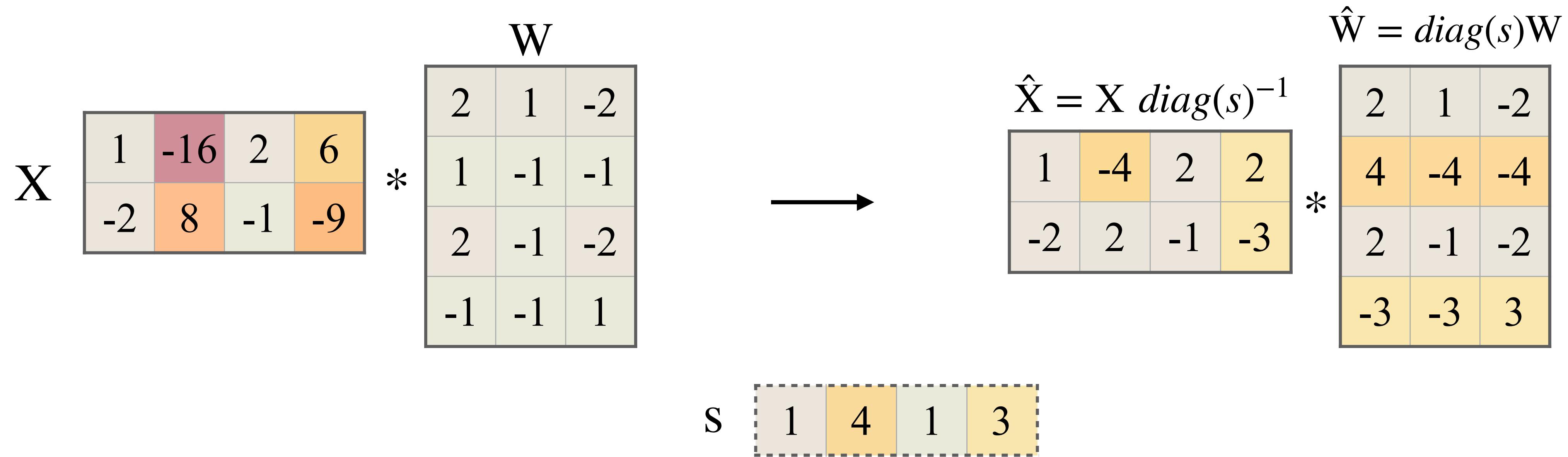
$$\mathbf{s}_j = \max(|\mathbf{X}_j|)^\alpha / \max(|\mathbf{W}_j|)^{1-\alpha}, j = 1, 2, \dots, C_i$$

$$\mathbf{Y} = (\mathbf{X} \text{diag}(\mathbf{s})^{-1}) \cdot (\text{diag}(\mathbf{s}) \mathbf{W}) = \hat{\mathbf{X}} \hat{\mathbf{W}}$$

SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models [Xiao et al., 2022]

# SmoothQuant (W8A8)

Smoothing activation to reduce quantization error



$$s_j = \max(|X_j|)^\alpha / \max(|W_j|)^{1-\alpha}, j = 1, 2, \dots, C_i$$

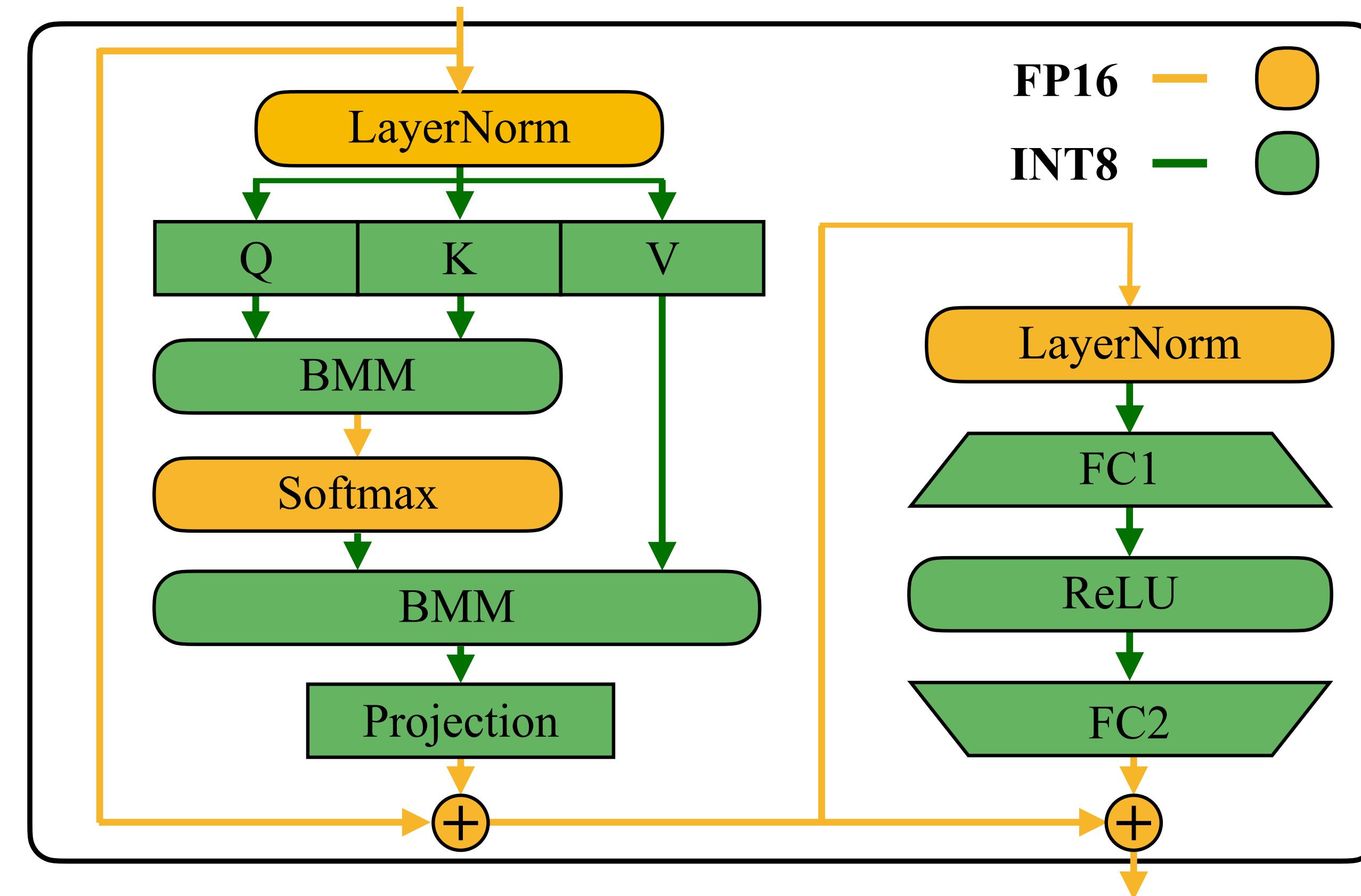
$$\mathbf{Y} = (\mathbf{X} \text{diag}(\mathbf{s})^{-1}) \cdot (\text{diag}(\mathbf{s})\mathbf{W}) = \hat{\mathbf{X}}\hat{\mathbf{W}}$$

SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models [Xiao et al., 2022]

# SmoothQuant (W8A8)

## Efficient System Implementation

- All compute-intensive operators (Linears, BMMs) are quantized

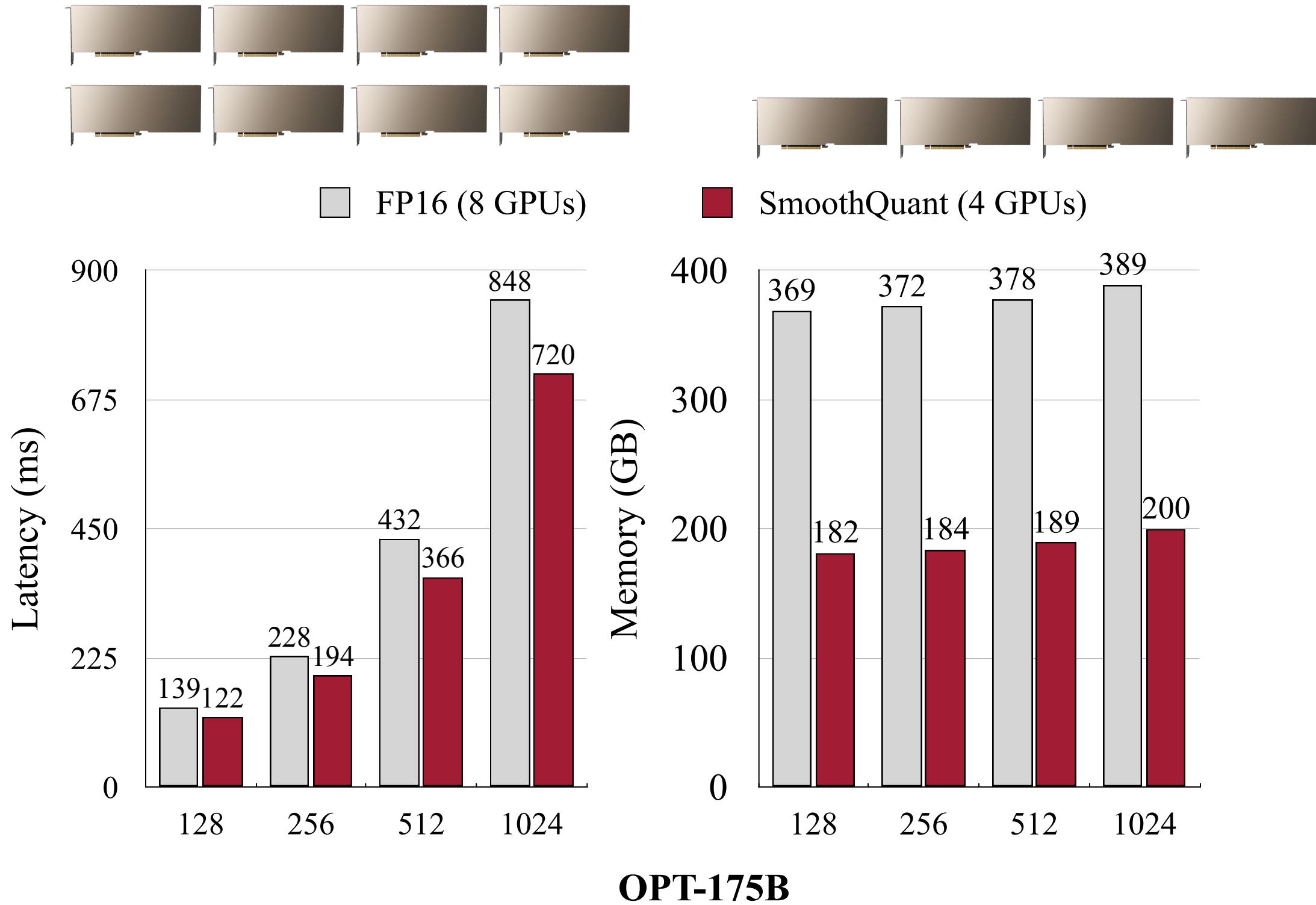
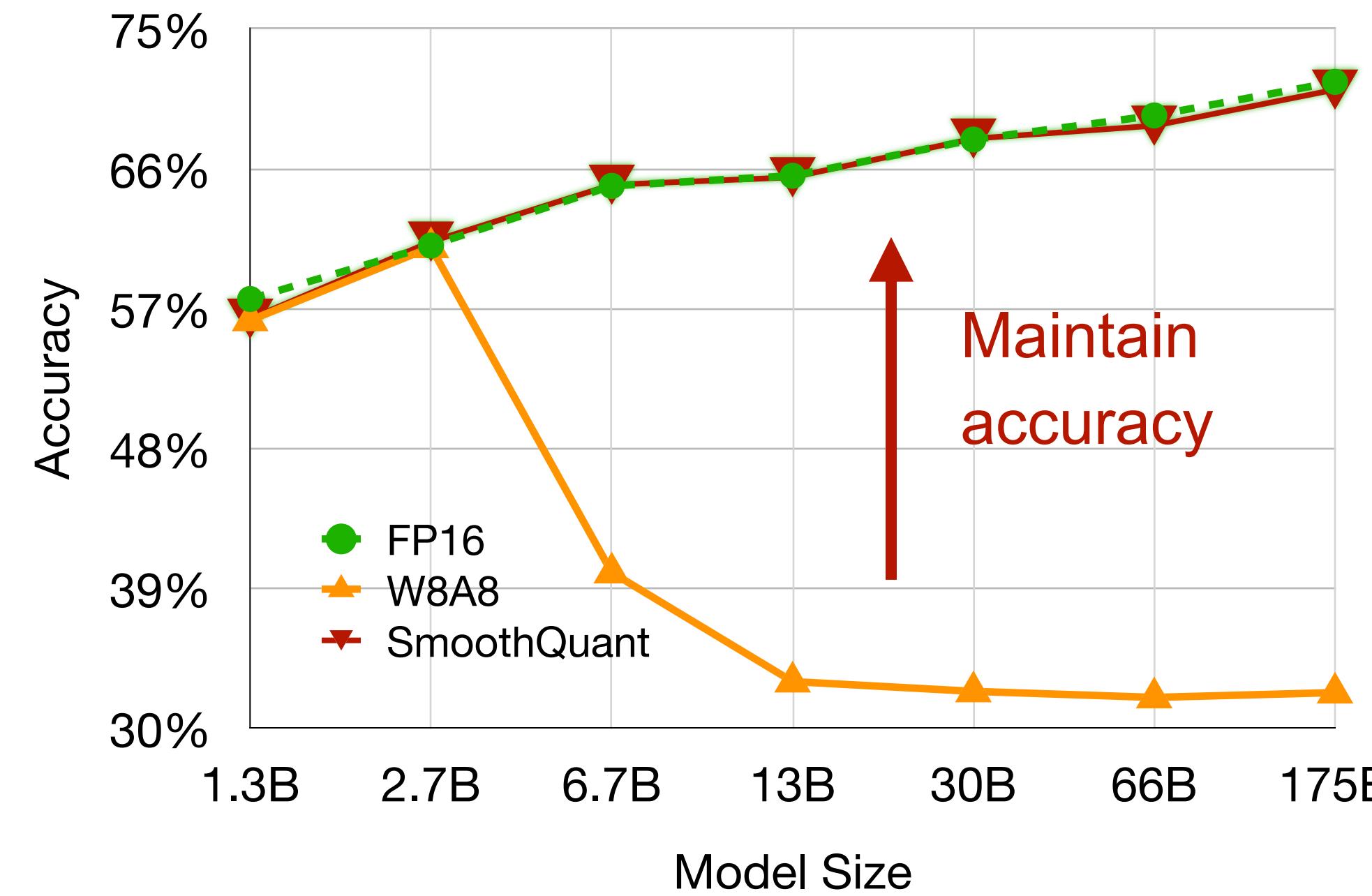


SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models [Xiao et al., 2022]

# SmoothQuant (W8A8)

## Accurate and efficient quantization of various LLMs

- SmoothQuant well maintains the accuracy without fine-tuning.
- SmoothQuant can both accelerate inference and halve the memory footprint.



SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models [Xiao et al., 2022]

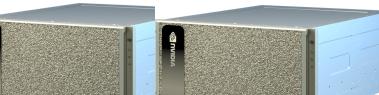
# SmoothQuant (W8A8)

## Scaling up: W8A8 quantization of MT-NLG 530B

MT-NLG 530B Accuracy

	LAMBADA	HellaSwag	PIQA	WinoGrande	Average
FP16	76.6%	62.1%	81.0%	72.9%	73.1%
INT8	77.2%	60.4%	80.7%	74.1%	73.1%

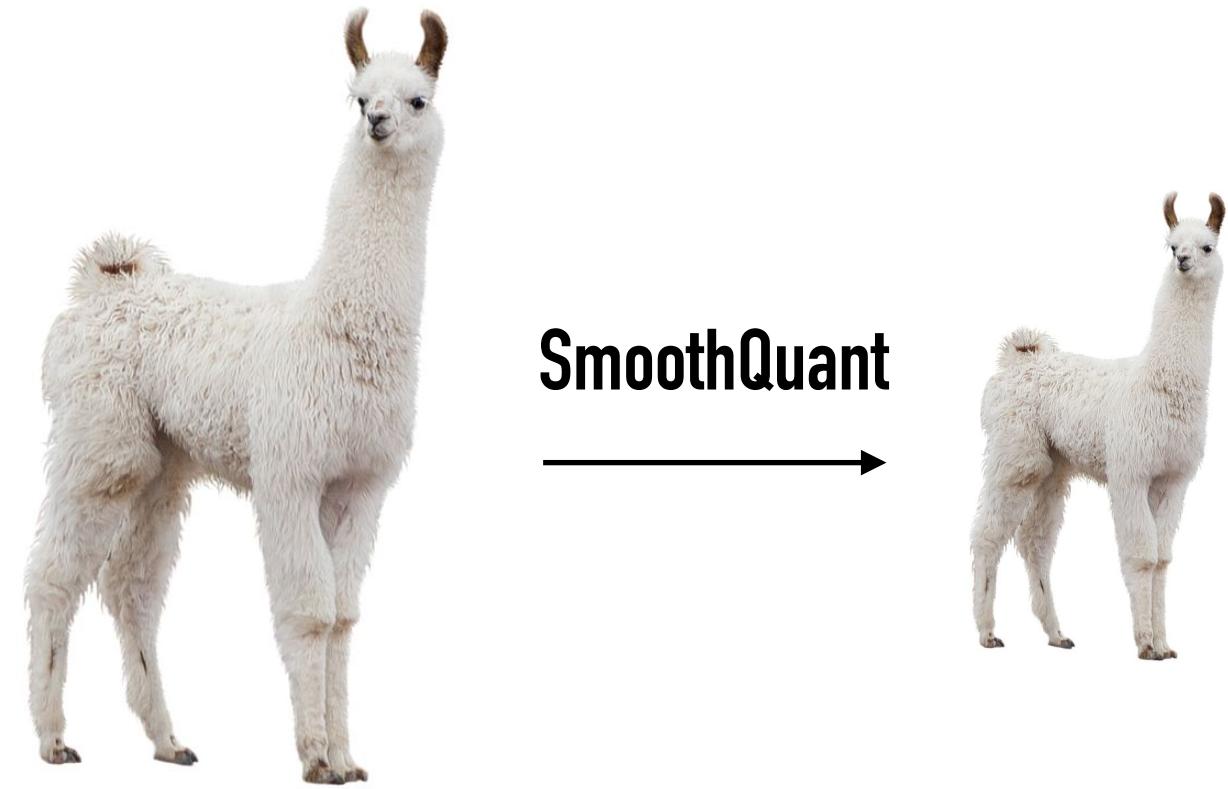
MT-NLG 530B Efficiency

	SeqLen	Prec.	#GPUs	Latency	Memory	
512	FP16	16	16	838ms	1068GB	
		8	8	839ms	545GB	
1024	FP16	16	16	1707ms	1095GB	
		8	8	1689ms	570GB	

- SmoothQuant can accurately quantize MT-NLG 530B model and reduce the serving GPU numbers by half at a similar latency, which allows serving the 530B model within a single node.

# SmoothQuant (W8A8)

Advancing new efficient open model LLaMA



- **LLaMA** (and its successors like Alpaca) are popular open-source LLMs, which introduced **SwishGLU, RoPE**. Will that impact quantization?
- SmoothQuant can losslessly quantize LLaMA families, further lowering the hardware barrier

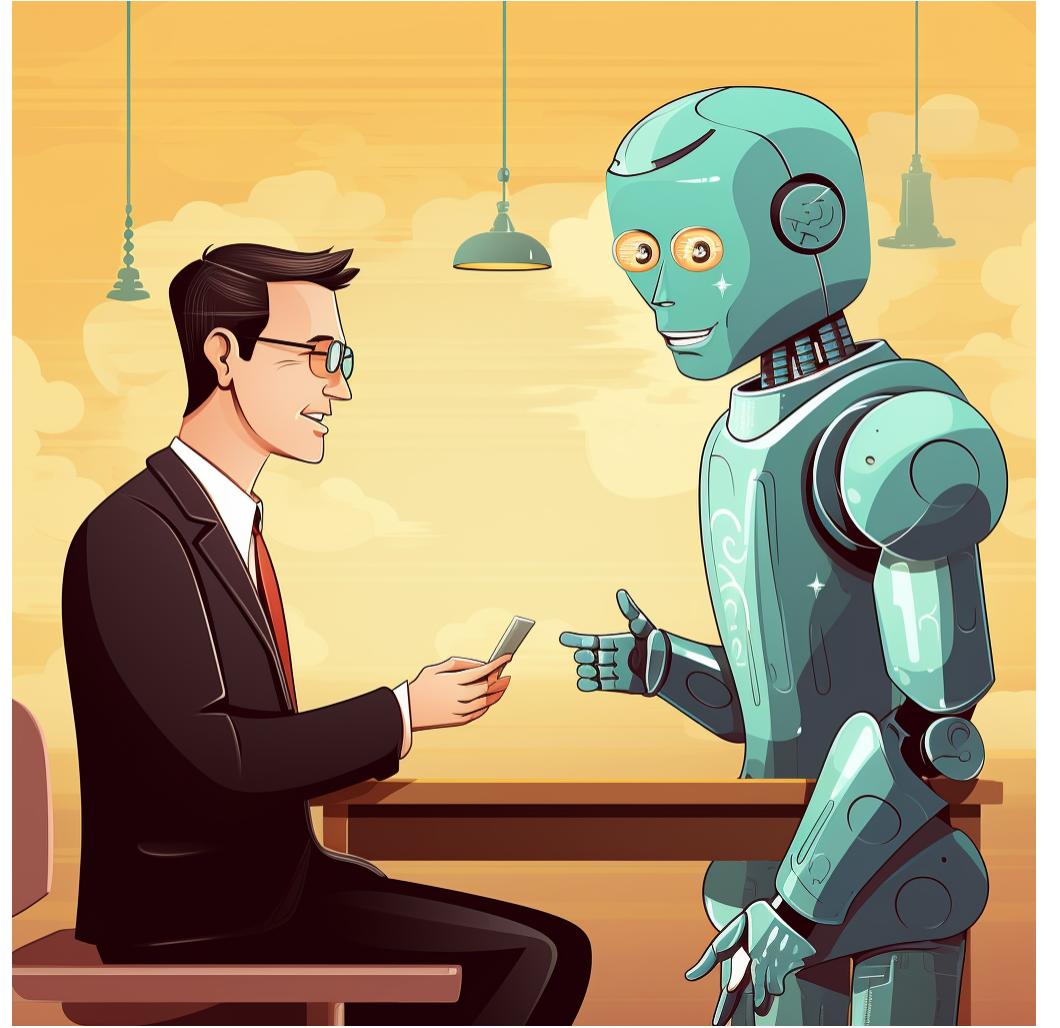
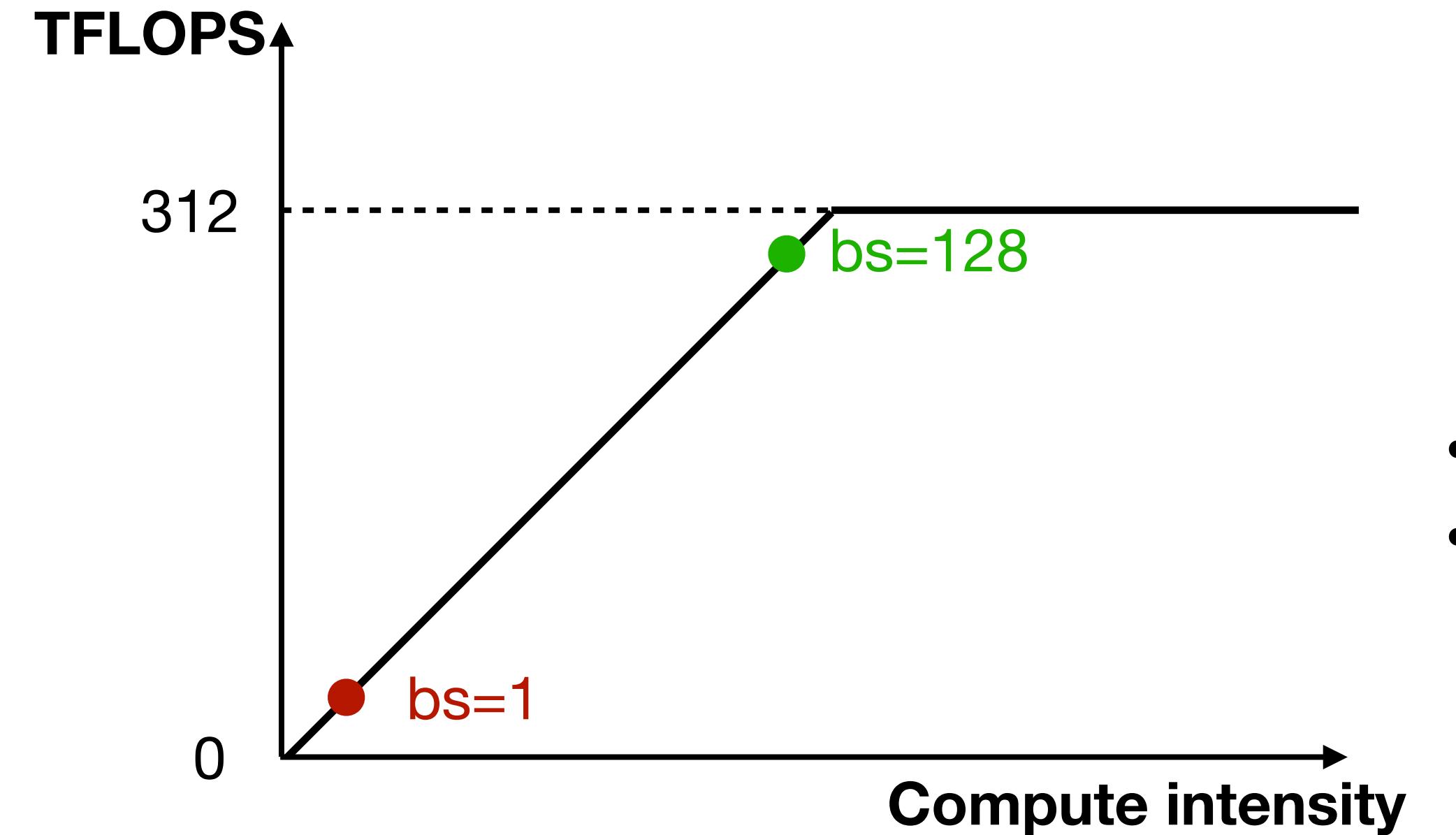
Wikitext↓	LLaMA 7B	LLaMA 13B	LLaMA 30B	LLaMA 65B
<b>FP16</b>	11.51	10.05	7.53	6.17
<b>SmoothQuant</b>	11.56	10.08	7.56	6.20

SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models [Xiao et al., 2022]

# W4A16 for Single-batch serving

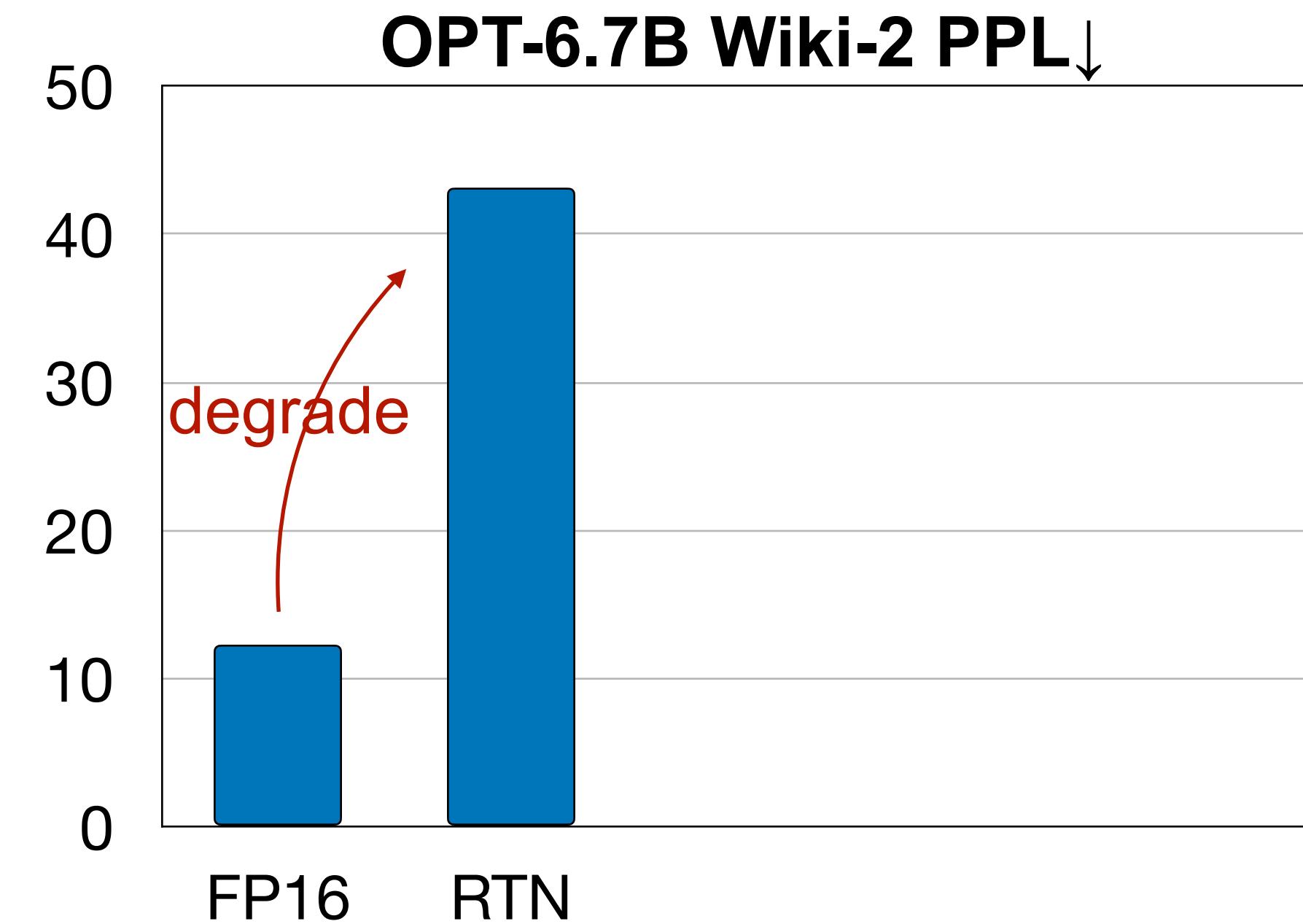
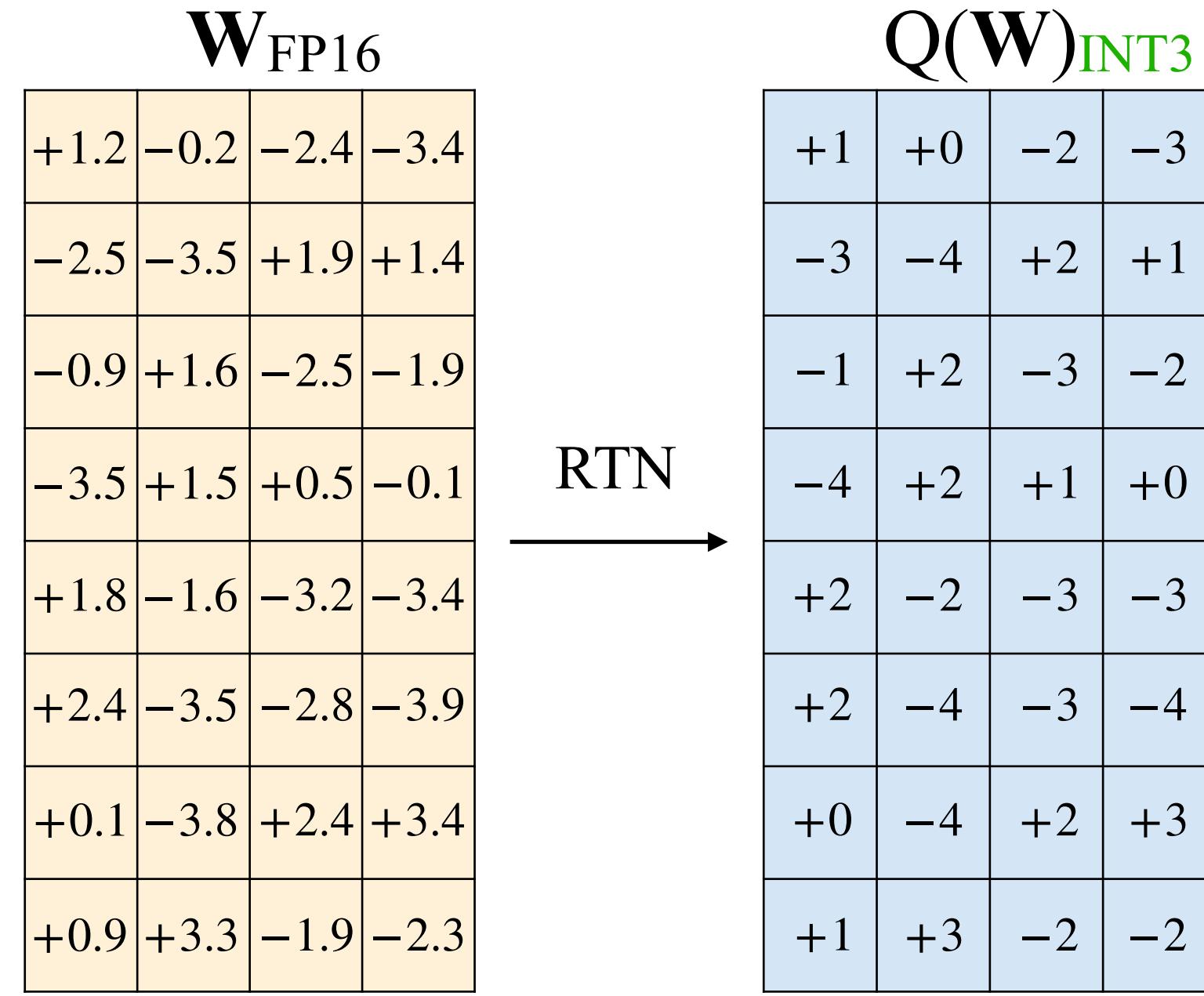
LLM decoding is highly memory-bounded; W8A8 is not enough

- W8A8 quantization is good for batch serving (e.g., batch size 128)
- But single-query LLM inference (e.g., local) is still highly memory-bounded
- We need **low-bit weight-only** quantization (e.g., W4A16) for this setting



# AWQ for Low-bit Weight-only Quantization

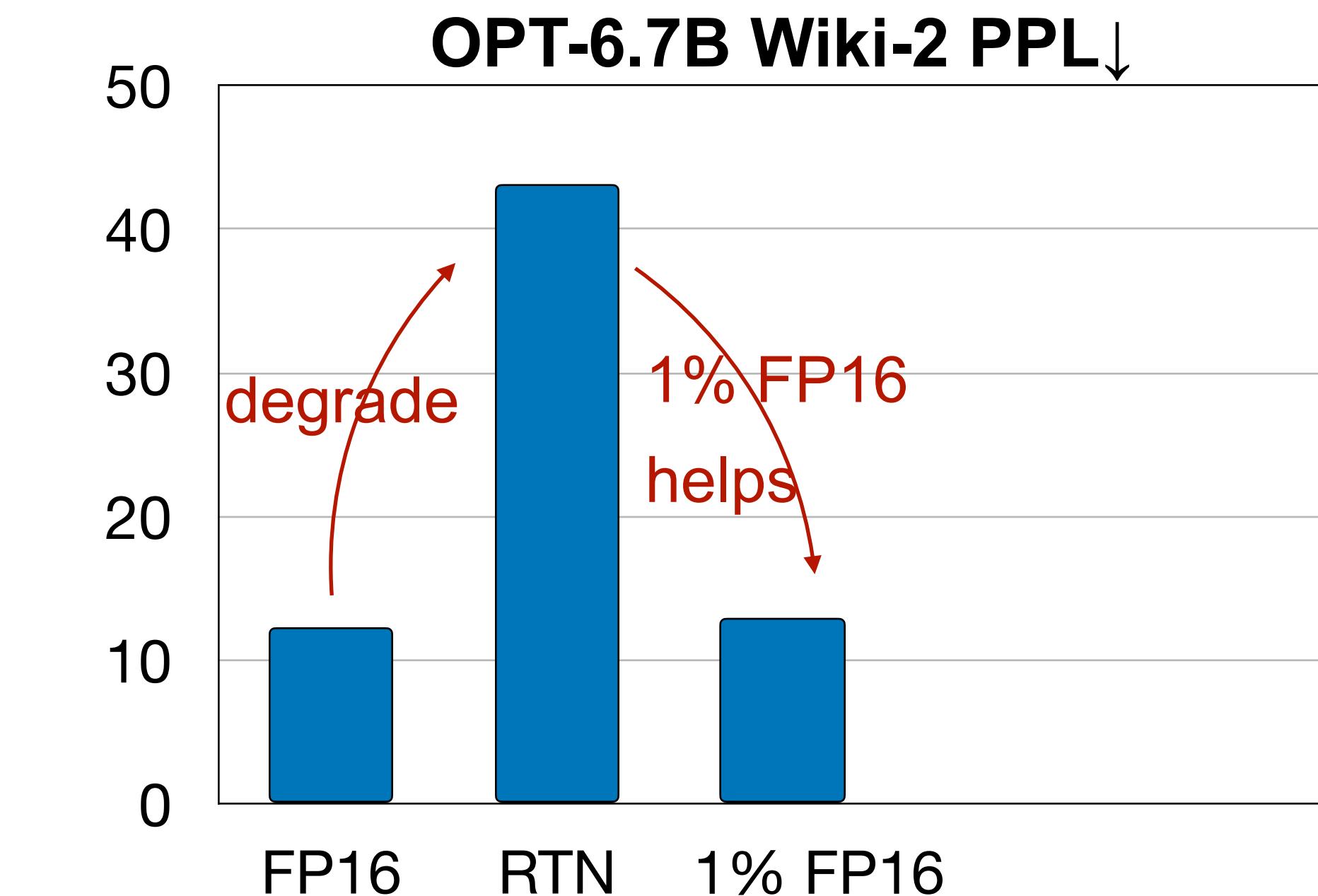
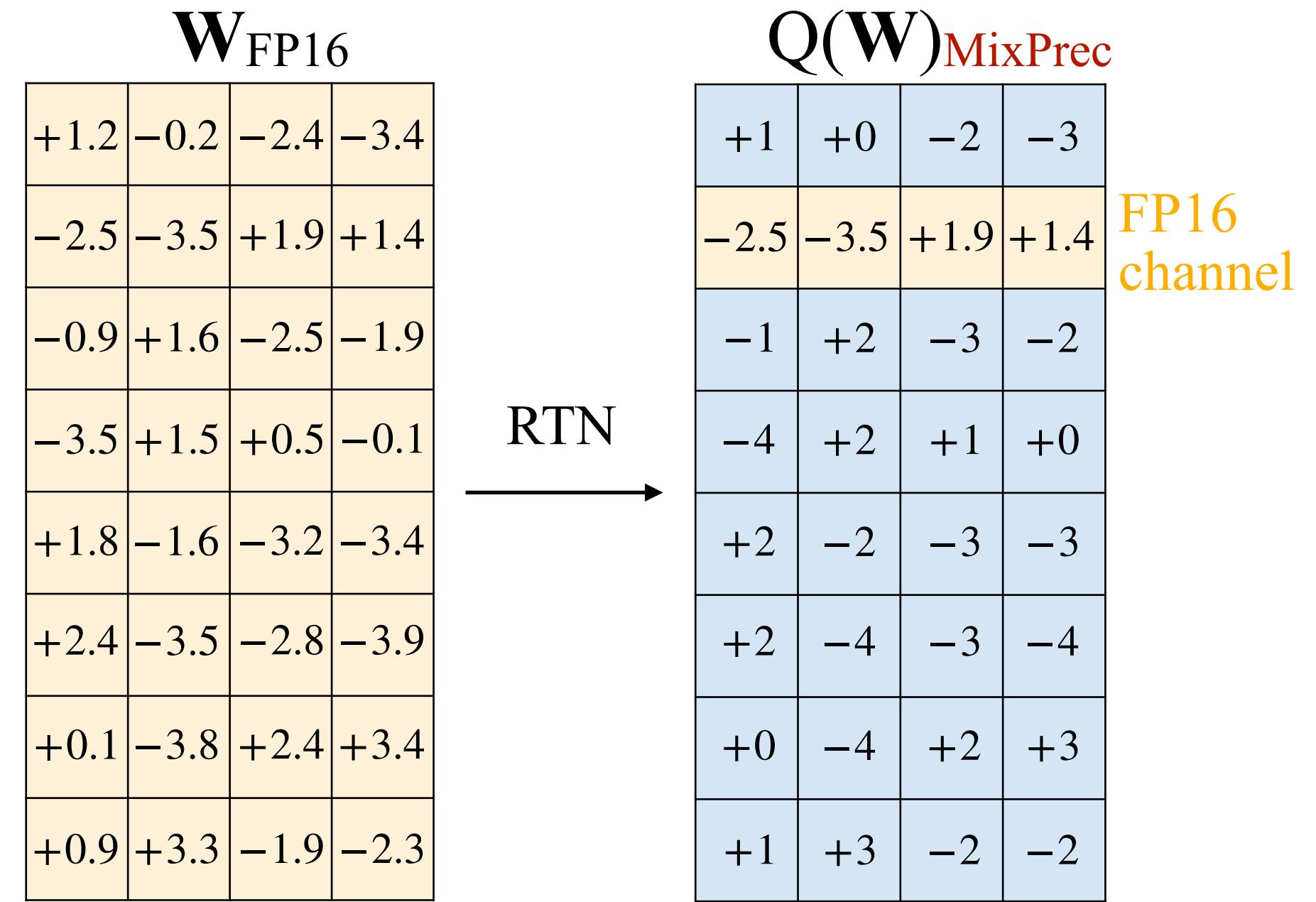
## Targeting group-wise W3/W4 quantization



- Weight-only quantization reduces the memory requirement, and accelerates token generation by alleviating the memory bottleneck.
- Group-wise/block-wise quantization (e.g., 64/128/256) offers a better accuracy-model size trade-off.
- But there is still a performance gap with round-to-nearest (RTN) quantization (INT3-g128)

# AWQ for Low-bit Weight-only Quantization

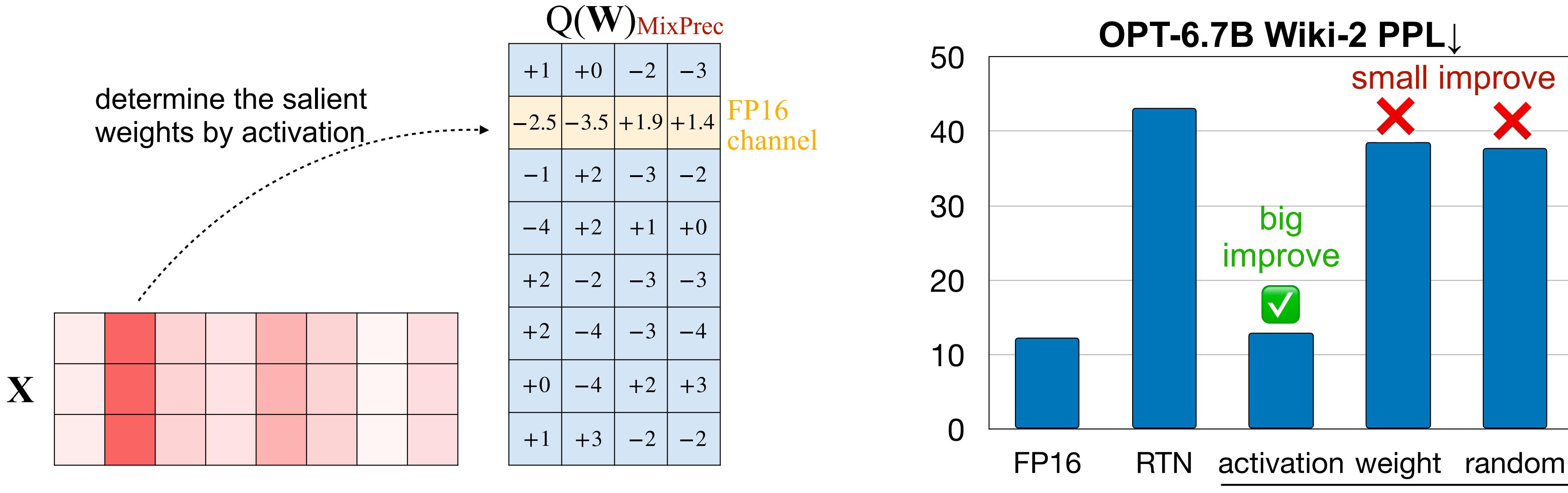
**Observation: Weights are not equally important; 1% salient weights**



- We find that weights are not equally important, keeping **only 1%** of salient weight channels in FP16 can greatly improve perplexity
- But how do we select salient channels? Should we select based on weight magnitude?

# AWQ for Low-bit Weight-only Quantization

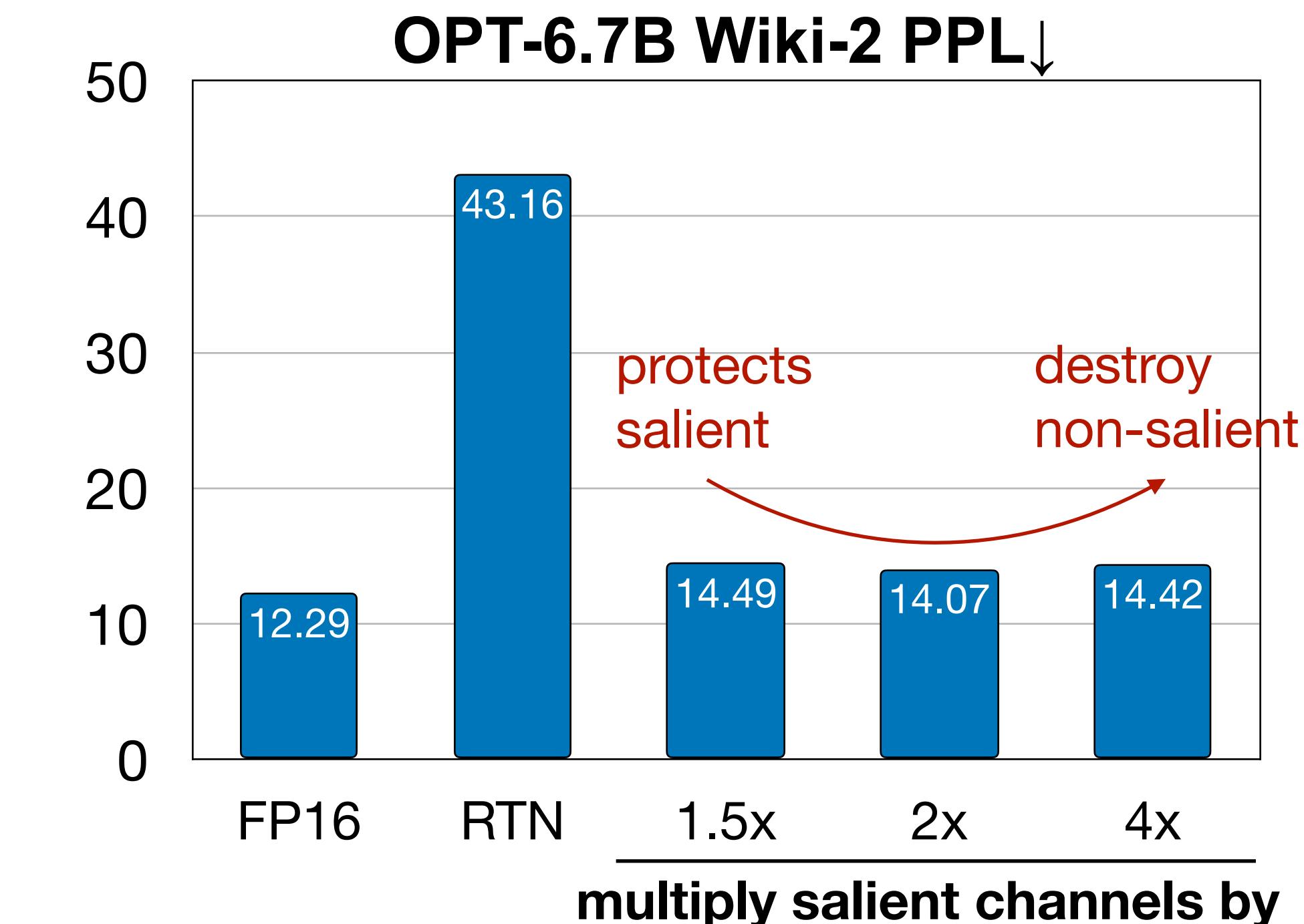
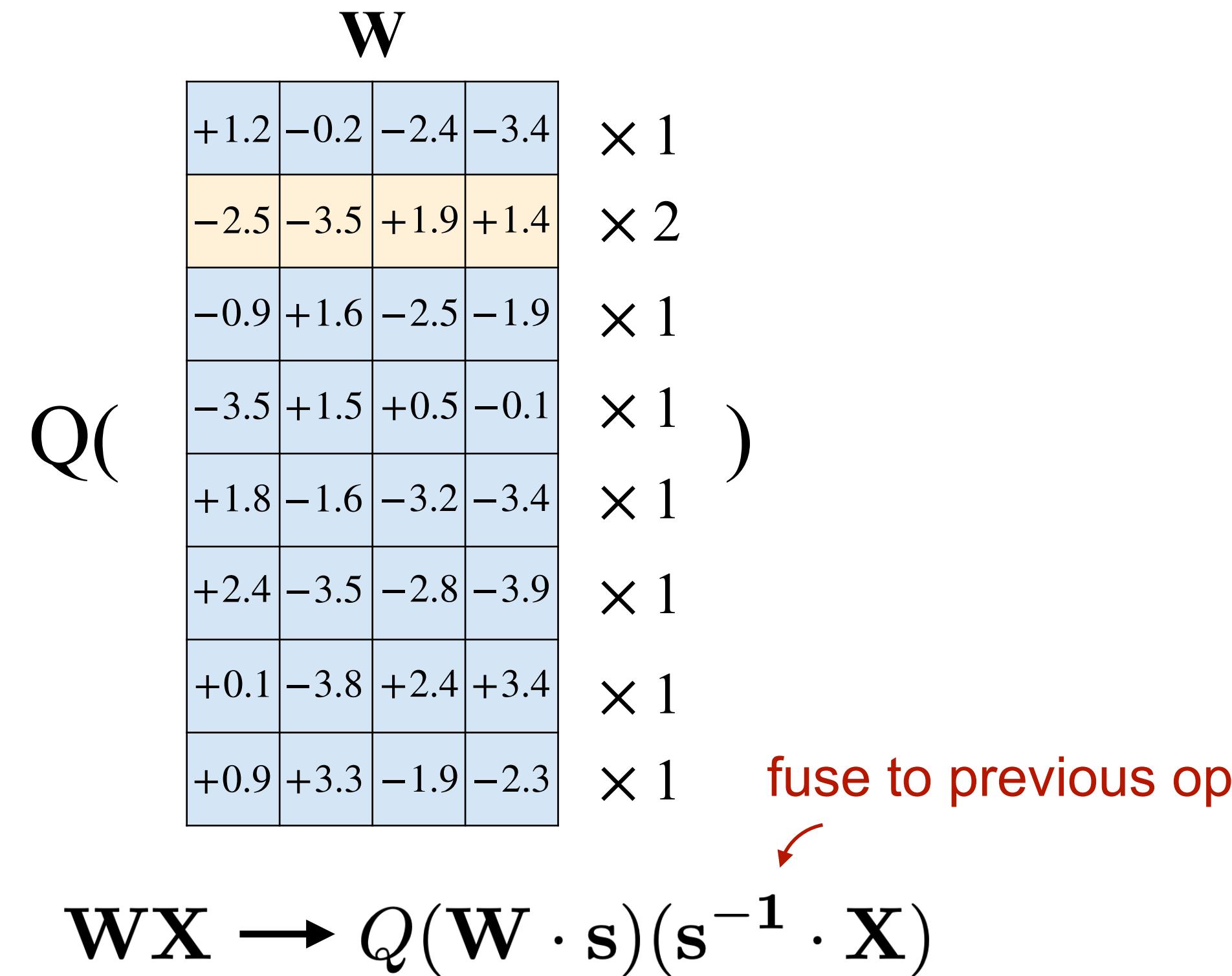
Salient weights are determined by activation distribution, not weight



- We find that weights are not equally important, keeping **only 1%** of salient weight channels in FP16 can greatly improve perplexity
- But how do we select salient channels? Should we select based on weight magnitude?
- No! We should look for **activation distribution, but not weight!**

# AWQ for Low-bit Weight-only Quantization

Protecting salient weights by scaling (no mixed prec.)



- Multiplying the salient channels with  $s > 1$  reduces its quantization error
- Why?

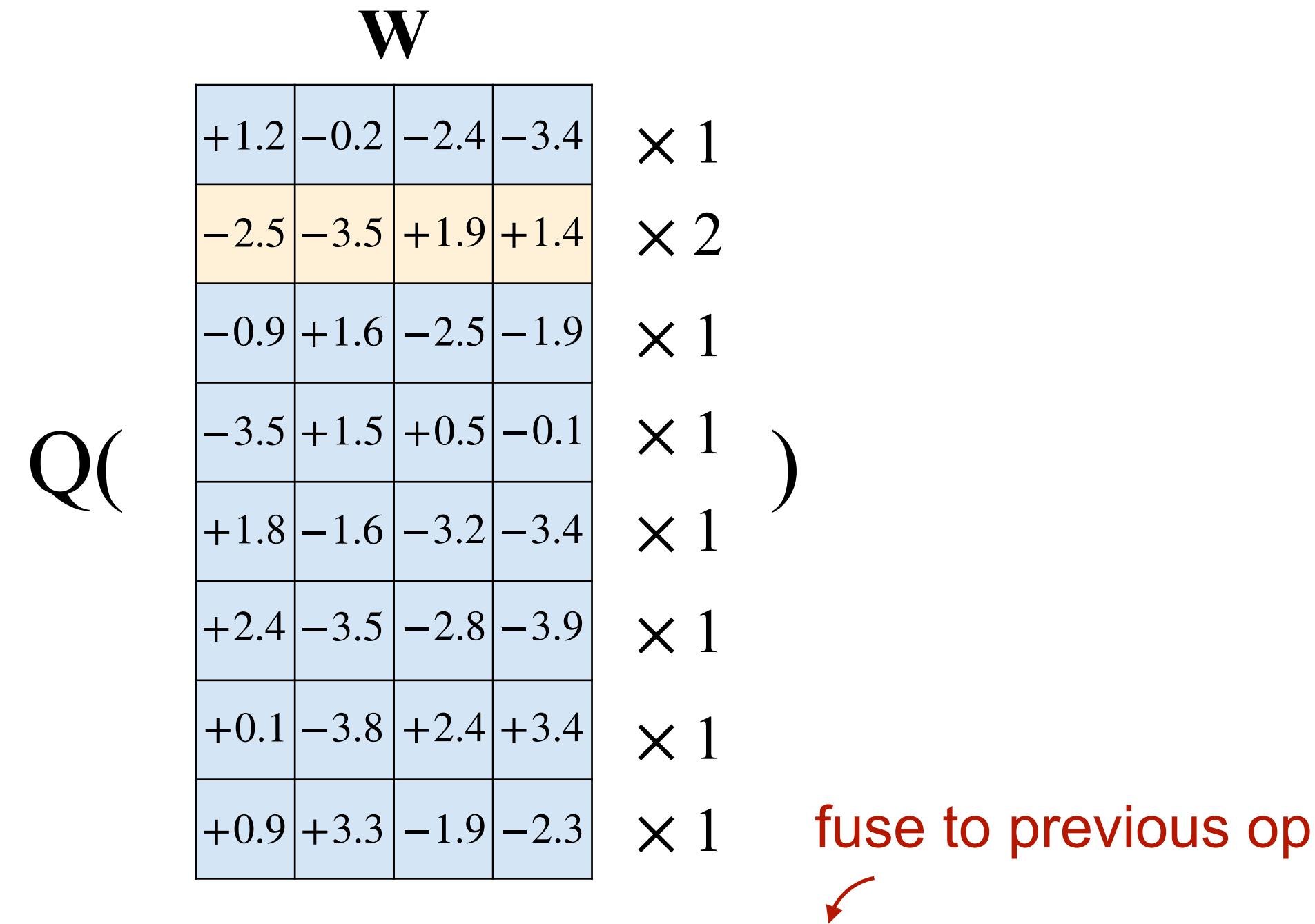
# AWQ for Low-bit Weight-only Quantization

## Protecting salient weights by scaling (no mixed precision)

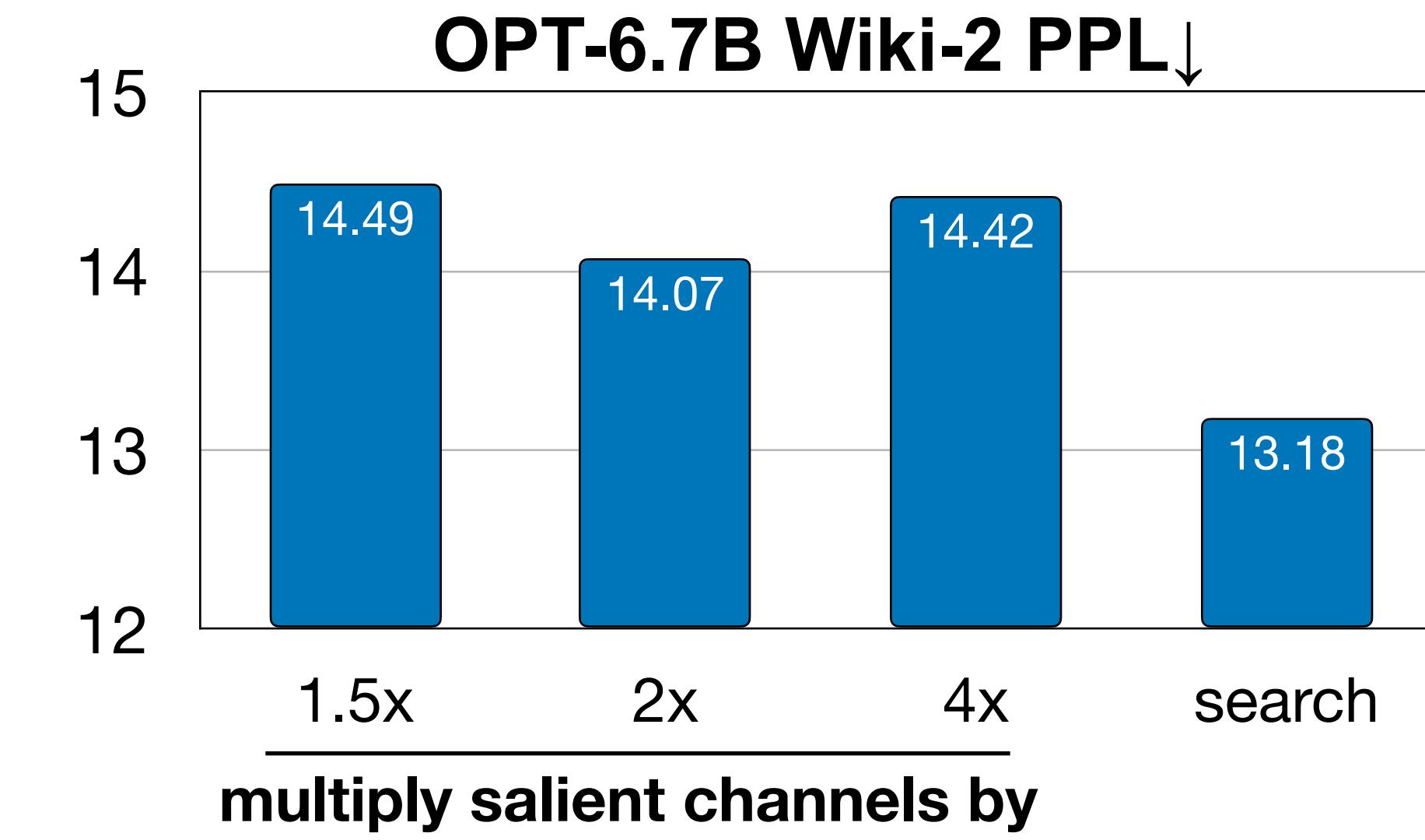
- Consider a linear layer channel  $\mathbf{y} = \mathbf{w}\mathbf{x}$  (from  $\mathbf{W}\mathbf{x}$ ). We care about the quantization error from  $Q(\mathbf{w})\mathbf{x}$
- $Q(\mathbf{w}) = \Delta \cdot \text{Round}(\mathbf{w}/\Delta), \quad \Delta = \frac{\max(|\mathbf{w}|)}{2^{N-1}}$
- The scaled version is  $Q(\mathbf{w} \cdot s)(x/s) = \underline{\Delta} \cdot \underline{\text{Round}(s\mathbf{w}/\Delta)} \cdot x \cdot \frac{1}{s}$
- We find that the error from Round() is always ~0.25 (average from 0-0.5)
- The maximum value in a group “usually” does not change if we just scale up a channel ->  $\Delta$  not changed
- With  $s > 1$ , the error is scaled down.

# AWQ for Low-bit Weight-only Quantization

Protecting salient weights by scaling (no mixed prec.)



$$\mathbf{WX} \rightarrow Q(\mathbf{W} \cdot \mathbf{s})(\mathbf{s}^{-1} \cdot \mathbf{X})$$



$$\mathcal{L}(\mathbf{s}) = \|Q(\mathbf{W} \cdot \mathbf{s})(\mathbf{s}^{-1} \cdot \mathbf{X}) - \mathbf{WX}\|$$

$$\mathbf{s} = s_{\mathbf{X}}^{\alpha}$$

**Activation-awareness** is important,  
but not weight-awareness

- Multiplying the salient channels with  $s > 1$  reduces its quantization error
- Take a data-driven approach with a fast **grid search**

# AWQ for Low-bit Weight-only Quantization

Better PPL under low-bit weight-only quantization

PPL↓		Llama-2			LLaMA			
		7B	13B	70B	7B	13B	30B	65B
FP16	-	5.47	4.88	3.32	5.68	5.09	4.10	3.53
INT3 g128	RTN	6.66	5.52	3.98	7.01	5.88	4.88	4.24
	GPTQ	6.43	5.48	3.88	8.81	5.66	4.88	4.17
	GPTQ-R	6.42	5.41	3.86	6.53	5.64	4.74	4.21
	AWQ	<b>6.24</b>	<b>5.32</b>	<b>3.74</b>	<b>6.35</b>	<b>5.52</b>	<b>4.61</b>	<b>3.95</b>
INT4 g128	RTN	5.73	4.98	3.46	5.96	5.25	4.23	3.67
	GPTQ	5.69	4.98	3.42	6.22	5.23	4.24	3.66
	GPTQ-R	5.63	4.99	3.43	5.83	5.20	4.22	3.66
	AWQ	<b>5.60</b>	<b>4.97</b>	<b>3.41</b>	<b>5.78</b>	<b>5.19</b>	<b>4.21</b>	<b>3.62</b>

AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration (Lin et al., 2023)

# AWQ for Low-bit Weight-only Quantization

Also works for multi-modal LLMs (OpenFlamingo-9B, captioning)

COCO (CIDEr $\uparrow$ )		0-shot	4-shot	8-shot	16-shot	32-shot	$\Delta(32\text{-shot})$
FP16	-	63.73	72.18	76.95	79.74	81.70	-
INT4 g128	RTN	60.24	68.07	72.46	74.09	77.13	-4.57
	GPTQ	59.72	67.68	72.53	74.98	74.98	-6.72
	AWQ	<b>62.57</b>	<b>71.02</b>	<b>74.75</b>	<b>78.23</b>	<b>80.53</b>	<b>-1.17</b>
INT3 g128	RTN	46.07	55.13	60.46	63.21	64.79	-16.91
	GPTQ	29.84	50.77	56.55	60.54	64.77	-16.93
	AWQ	<b>56.33</b>	<b>64.73</b>	<b>68.79</b>	<b>72.86</b>	<b>74.47</b>	<b>-7.23</b>



**W4-RTN:** A model airplane **flying in the sky**.

**W4-AWQ:** Two toy airplanes **sit on a grass field**.



**W4-RTN:** A man is **holding a baby elephant** in his arms.

**W4-AWQ:** A man and his daughter **pose with an elephant**.



**W4-RTN:** **A man and a dog** walking past some bushes.

**W4-AWQ:** **Two dogs** are walking on the street.

# AWQ for Low-bit Weight-only Quantization

Also works for multi-modal LLMs (LLaVA, visual reasoning)



Sometimes I just look at pictures of the earth from space and I marvel at how beautiful it all is.



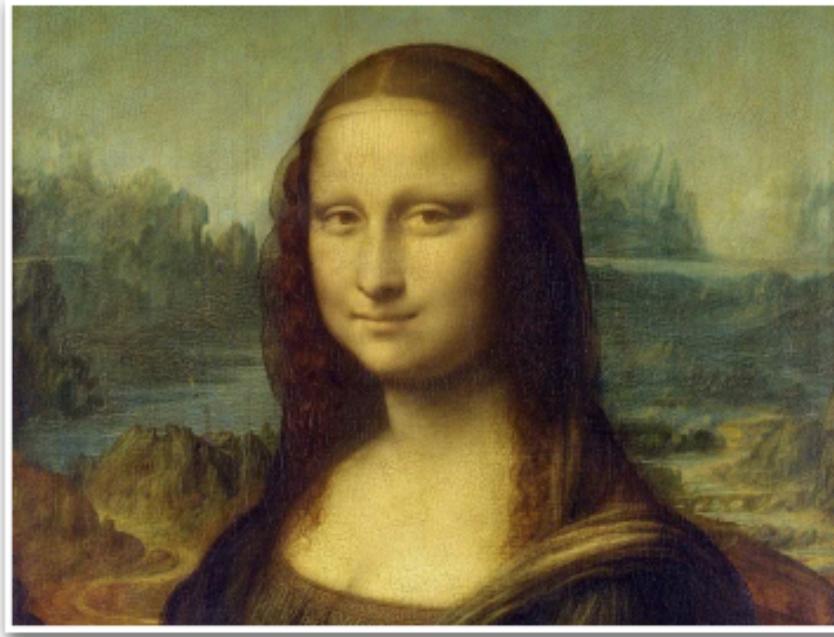
**Question:** Can you explain this meme in detail?

**INT4-g128 RTN:** The meme in the image features a plate of food, likely fried chicken, with a unique twist. **There are small pictures of the Earth and other planets placed on top of the food**, creating an interesting and amusing visual. Alongside the plate, a caption reads, "Sometimes I look at pictures of the Earth from space and I marvel at how beautiful it all is." This combination of food and planet images adds a touch of humor and curiosity, making the viewer wonder about the context and intent behind the meme.

**INT4-g128 AWQ:** The meme in the image is a **lighthearted and humorous take on the concept of looking at pictures of the Earth from space**. A plate of fried food, specifically chicken nuggets, is presented with the caption, "Sometimes I look at pictures of the Earth from space and I marvel at how beautiful it all is." The playful combination of a food item and an Earth-related context creates a fun and entertaining visual.

# AWQ for Low-bit Weight-only Quantization

Also works for multi-modal LLMs (LLaVA, visual reasoning)



**Question:** Do you know who drew this painting?

**INT4-g128 RTN:** The painting depicts a woman, who some believe to be a representation of the famous artist, Mona Lisa. However, it is not certain that the woman in the painting is indeed Mona Lisa. The artwork is a popular subject of artistic analysis and admiration, often considered a symbol of feminine beauty and enigma. The portrait is characterized by its intricate details, captivating the viewer with the woman's expressive gaze and the subtle, enigmatic smile that has become synonymous with the Mona Lisa.

**INT4-g128 AWQ:** The painting depicts a woman, who is believed to be a portrait of Mona Lisa, the famous artwork by the artist **Leonardo da Vinci**. Although the painting is widely recognized as a representation of Mona Lisa, it is not possible to confirm the artist who drew this particular painting without additional information. Leonardo da Vinci was a renowned artist known for his exceptional skill in painting detailed, realistic portraits, and the Mona Lisa is one of his most famous works.



# Industry & Community Impact

**SmoothQuant and AWQ are widely used:**



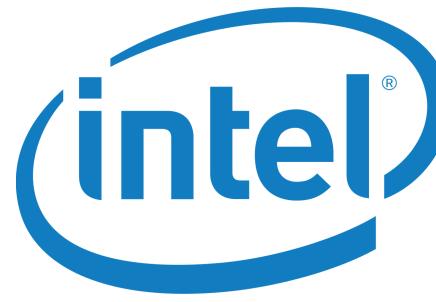
FasterTransformer  
TRT-LLM

[https://github.com/NVIDIA/FasterTransformer/blob/main/docs/gpt\\_guide.md](https://github.com/NVIDIA/FasterTransformer/blob/main/docs/gpt_guide.md)  
<https://github.com/NVIDIA/TensorRT-LLM#key-features>



text-generation-inference

[https://github.com/huggingface/text-generation-inference/tree/main/server/text\\_generation\\_server/utils/awq/quantize](https://github.com/huggingface/text-generation-inference/tree/main/server/text_generation_server/utils/awq/quantize)



Neural Compressor  
Q8-Chat

[https://github.com/intel/neural-compressor/blob/master/docs/source/smooth\\_quant.md](https://github.com/intel/neural-compressor/blob/master/docs/source/smooth_quant.md)



lmdeploy

<https://github.com/InternLM/lmdeploy/blob/main/lmdeploy/lite/quantization/awq.py>



[https://github.com/vllm-project/vllm/blob/main/vllm/model\\_executor/quantization\\_utils/awq.py](https://github.com/vllm-project/vllm/blob/main/vllm/model_executor/quantization_utils/awq.py)



oobabooga/text-generation-webui

<https://github.com/oobabooga/text-generation-webui/blob/main/modules/models.py>



lm-sys/FastChat

<https://github.com/lm-sys/FastChat/blob/main/docs/awq.md>

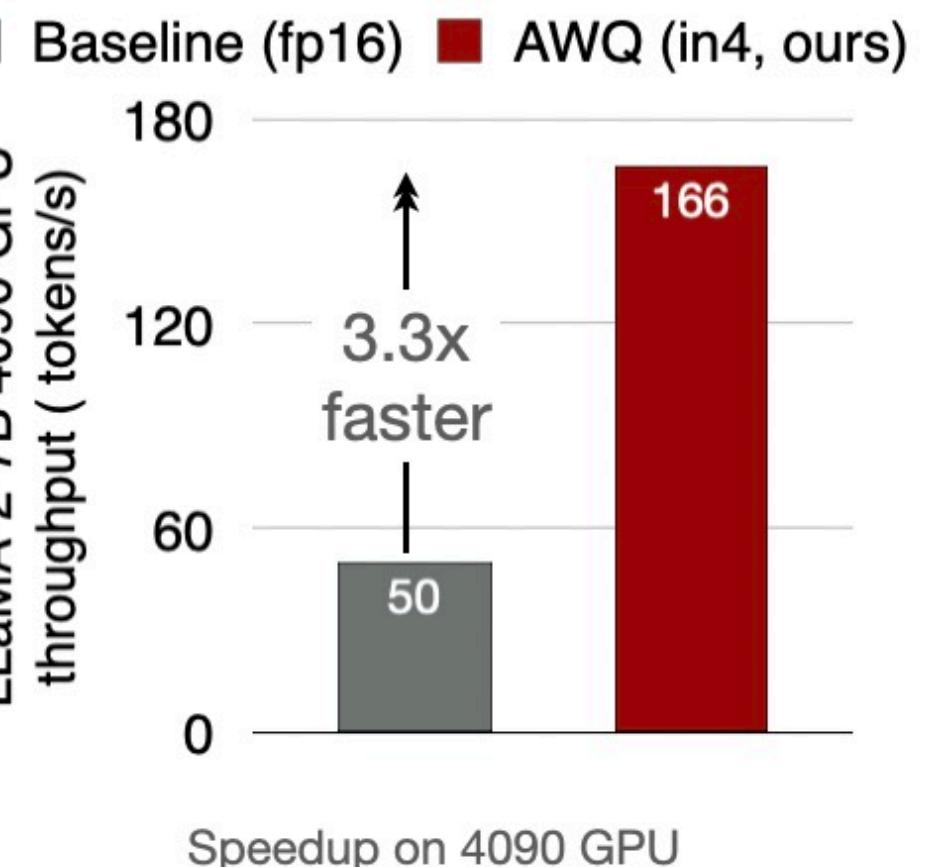
Replicate

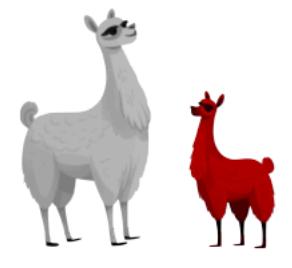
[https://github.com/replicate/vllm-with-loras/blob/main/vllm/model\\_executor/quantization\\_utils/awq.py](https://github.com/replicate/vllm-with-loras/blob/main/vllm/model_executor/quantization_utils/awq.py)



# TinyChat

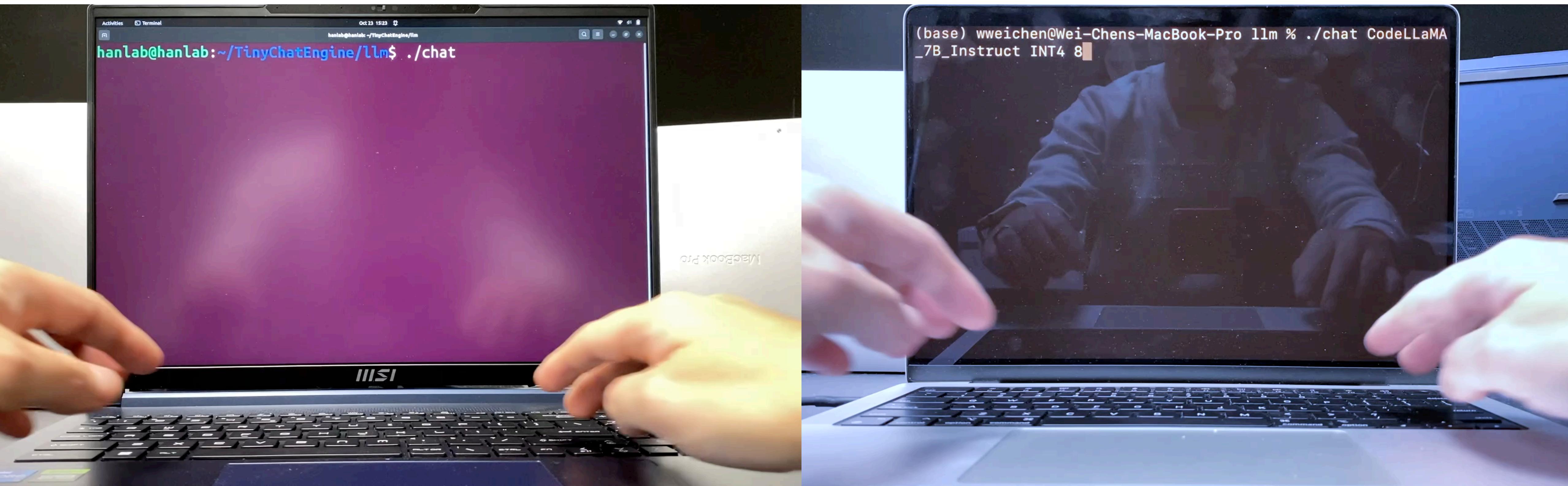
- A LightWeight Chatbot for LLMs on the edge
  - Deploying LLM on the edge is useful: running copilot services (code completion, office, game chat) locally on laptops, cars, robots, and more. These devices are **resource-constrained**, **low-power** and sometimes **do not have access to the Internet**.
  - **Data privacy** is important. Users do not want to share personal data with large companies.

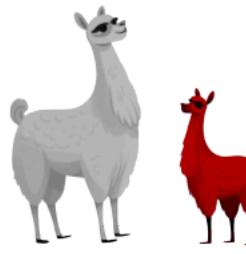




# TinyChat

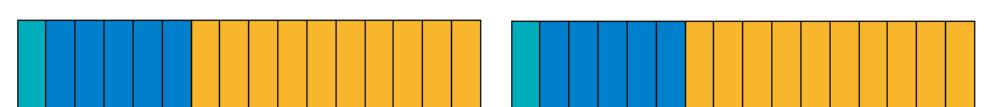
- TinyChatEngine implements the compressed AWQ 4bit model, built from C/C++ from scratch, easy to install and migrate to edge platforms
- Enables on-device LLM





# TinyChat brings about 3.3x speedup to LLaMA-2 on 4090

**LLaMA-2-7B (FP16): 50 tokens / s**



# Baseline: fp16 weight, fp16 activation

# LLaMA-2-7B (W4A16, AWQ): 166 tokens / s



# AWQ: int4 weight, fp16 activation



# TinyChat flexibly supports different LLM architectures



```
File Edit Selection View Go ... orin [SSH: orin] - Visual Studio Code PROBLEMS OUTPUT PORTS TERMINAL DEBUG CONSOLE sh - AWQ-Chat + ... x
PROBLEMS OUTPUT PORTS TERMINAL DEBUG CONSOLE sh - AWQ-Chat + ... x
o (AWQ-Chat) orin% ./mpt_awq_int4.sh 7b
real weight quantization... (init only): 100% | 32/32 [00:00<00:00, 314.76it/s]
Loading checkpoint: 100% | 1/1 [00:01<00:00, 1.54s/it]
USER: How do you compare MIT and Harvard?
ASSISTANT: As an AI language model, I don't have personal experiences or preferences, so I cannot provide an objective comparison between these two prestigious universities. However, both of them are well-known for their academic excellence in various fields such as science, technology, engineering, and mathematics (STEM). They consistently rank among the top universities worldwide in terms of academic reputation, research output, and employability of graduates. Ultimately, the choice between MIT and Harvard depends on individual preferences and goals, such as location, program offerings, admission requirements, financial aid opportunities, and career prospects after graduation. It is essential to conduct thorough research and consult with experts before making a decision about which university to attend.
=====
Speed of Inference
-----
Generation Stage : 31.29 ms/token
=====
USER: 
```

```
File Edit Selection View Go ... orin [SSH: orin] - Visual Studio Code PROBLEMS OUTPUT PORTS TERMINAL DEBUG CONSOLE sh - AWQ-Chat + ... x
PROBLEMS OUTPUT PORTS TERMINAL DEBUG CONSOLE sh - AWQ-Chat + ... x
o (AWQ-Chat) orin% ./falcon_awq_int4.sh 7b
real weight quantization... (init only): 100% | 32/32 [00:00<00:00, 127.73it/s]
Loading checkpoint: 100% | 1/1 [00:02<00:00, 2.72s/it]
USER: How do you compare MIT and Harvard?
ASSISTANT: There are many differences between MIT and Harvard, but both universities have a reputation for rigorous academic programs, cutting-edge research, and talented faculty. MIT is known for its science and engineering programs, while Harvard is known for its humanities and social science programs. Additionally, Harvard has a larger endowment and more alumni giving, making it financially stronger than MIT. However, both universities share a commitment to excellence and are highly selective in their admissions process.
=====
Speed of Inference
-----
Generation Stage : 45.90 ms/token
=====
USER: 
```

```
File Edit Selection View Go ... orin [SSH: orin] - Visual Studio Code PROBLEMS OUTPUT PORTS TERMINAL DEBUG CONSOLE sh - AWQ-Chat + ... x
PROBLEMS OUTPUT PORTS TERMINAL DEBUG CONSOLE sh - AWQ-Chat + ... x
o (AWQ-Chat) orin% ./vicuna_awq_int4.sh 7b
Loading checkpoint: 100%
USER: How do you compare MIT and Harvard?
ASSISTANT: MIT and Harvard are both highly prestigious universities with strong reputations in various fields of study. However, they have some key differences that set them apart from each other.
One significant difference is their academic focus. MIT is known for its emphasis on science, technology, engineering, and mathematics (STEM) fields, while Harvard has a more diverse range of programs across the humanities, social sciences, law, medicine, and business. As such, MIT is often seen as the "technology school" and Harvard as the "liberal arts school."
In terms of research, both institutions are among the top performers in terms of funding and number of patents granted, but MIT is generally considered to be more focused on applied research, while Harvard is known for its more theoretical and interdisciplinary research.
Overall, it's difficult to say which one is "better" since it depends on what you're looking for in a university education. If you're interested in STEM fields or want to pursue a career in academia, MIT might be the better choice. If you're looking for a broader liberal arts education or want to go into a field like law or business, Harvard might be the better fit.
=====
Speed of Inference
-----
Generation Stage : 30.50 ms/token
=====
USER: 
```

```
File Edit Selection View Go ... orin [SSH: orin] - Visual Studio Code PROBLEMS OUTPUT PORTS TERMINAL DEBUG CONSOLE sh - AWQ-Chat + ... x
PROBLEMS OUTPUT PORTS TERMINAL DEBUG CONSOLE sh - AWQ-Chat + ... x
Homework: Read about different types of parallelism and complete a short quiz.
Week 2: Parallel Programming Languages
- Introduction to parallel programming languages (MPI, OpenMP, LINUX threads)
- Basic syntax and features of each language
- Examples of how to use each language to solve parallel problems
Homework: Write a simple parallel program using MPI or OpenMP and submit it for grading.
Week 3: Performance Analysis and Optimization
- How to measure and analyze the performance of parallel programs?
- Techniques for optimizing parallel programs (vectorization, loop fusion, pipelining)
- Understanding the tradeoffs between parallelism and efficiency
Homework: Analyze the performance of a parallel program and optimize it using different techniques.
Week 4: Multicore and Manycore Systems
- Introduction to modern CPUs and GPUs
- Understanding the differences between multicore and manycore systems
- How to design and implement efficient parallel algorithms on these systems
Homework: Design and implement a parallel algorithm that scales well on a manycore system.
Week 5: Applications of Parallel Computing
- Overview of some important applications areas for parallel computing (scientific simulations, machine learning, data analytics)
- Case studies of successful parallel computing projects in these areas
- Discussion of challenges and opportunities in applying parallel computing to real-world problems
Home homework: Research and discuss a real-world application of parallel computing and its challenges
Note: The above syllabus is just a sample and can be modified based on specific requirements and preferences.
=====
Speed of Inference
-----
Generation Stage : 31.27 ms/token
=====
USER: 
```

```
File Edit Selection View Go ... orin [SSH: orin] - Visual Studio Code PROBLEMS OUTPUT PORTS TERMINAL DEBUG CONSOLE sh - AWQ-Chat + ... x
PROBLEMS OUTPUT PORTS TERMINAL DEBUG CONSOLE sh - AWQ-Chat + ... x
o (AWQ-Chat) orin% ./falcon_awq_int4.sh 7b
real weight quantization... (init only): 100% | 32/32 [00:00<00:00, 140.33it/s]
Loading checkpoint: 100% | 1/1 [00:02<00:00, 2.67s/it]
USER: Make a detailed 5-week syllabus for Parallel Computing.
ASSISTANT: Week 1: Introduction to Parallel Computing - Understanding the basics of parallel computing and its applications.
Week 2: Single Processor Systems - Understanding the concept of a single processor and the limitations of a single processor system.
Week 3: Multi-Processor Systems - Understanding the concept of multi-processor systems and their advantages.
Week 4: Parallel Programming Models - Understanding the different types of parallel programming models, including shared memory, distributed memory, and many-to-many communication patterns.
Week 5: Parallel Programming - Writing and executing parallel programs using a programming language and understanding the concepts of parallelism and data locality.
=====
Speed of Inference
-----
Generation Stage : 45.25 ms/token
=====
USER: 
```

```
File Edit Selection View Go ... orin [SSH: orin] - Visual Studio Code PROBLEMS OUTPUT PORTS TERMINAL DEBUG CONSOLE sh - AWQ-Chat + ... x
PROBLEMS OUTPUT PORTS TERMINAL DEBUG CONSOLE sh - AWQ-Chat + ... x
* Parallel computing architectures (SMP, MPP, GPU)
Week 2: Parallel Programming Languages
* Introduction to parallel programming languages (MPI, OpenMP, CUDA, etc.)
* Features and limitations of each language
* Best practices for writing efficient parallel code in each language
Week 3: Parallel Algorithms
* Introduction to parallel algorithms (divide and conquer, map reduce, etc.)
* Analysis of algorithmic complexity
* Performance characteristics of different types of algorithms
Week 4: Parallel Architectures and System Design
* Introduction to parallel architectures (SMP, MPP, GPU)
* System design considerations for parallel computing systems
* Memory hierarchy and cache design
Week 5: Case Studies in Parallel Computing
* Real-world applications of parallel computing
* Case studies on large scale data processing, scientific simulations, and high performance computing
* Discussion of best practices and optimization strategies for different workloads.
=====
Speed of Inference
-----
Generation Stage : 30.46 ms/token
=====
USER: 
```



MPT-7B: 31 tokens / s



Falcon-7B: 22 tokens / s



Vicuna-7B: 33 tokens / s



# TinyChat delivers 30 tokens / second performance for LLaMA2



```
(AWQ-Chat) orin% ./llama2_awq_int4.sh 7b
Loading checkpoint: 100% | 1/1 [00:02<00:00, 2.99s/it]
USER: Describe five attractions in the Greater Boston Area.
ASSISTANT: Of course! The Greater Boston Area has a plethora of exciting attractions that cater to diverse interests. Here are five popular ones:
1. The Freedom Trail: This 2.5-mile trail takes you through historic sites in Boston, including the Old State House, Faneuil Hall, and the USS Constitution. You'll learn about the city's rich history and witness its iconic landmarks along the way.
2. Museum of Fine Arts: As one of the largest and most visited art museums in the country, the MFA features over 450,000 works of art from around the world, including European, American, and Asian pieces. Don't miss the impressive Egyptian, Greek, and Roman collections.
3. Fenway Park: Known as "America's Most Beloved Ballpark," Fenway Park is home to the Boston Red Sox baseball team. Take a guided tour of the stadium, explore the team's Hall of Fame, and enjoy the lively atmosphere during a game.
4. New England Aquarium: Located on the waterfront, the NEAQ showcases a diverse array of marine life, including penguins, seals, and fish of all shapes and sizes. Be sure to check out the giant Pacific octopus and the shark and ray touch tanks.
5. Harvard University: Established in 1636, Harvard is one of the oldest and most prestigious universities in the US. Take a stroll across the campus, visit the Widener Library, and marvel at the architecture of the Harvard Yard. You might even catch a glimpse of the elusive Harvard Bridge.
Each of these attractions offers a unique experience, so be sure to add them to your itinerary when visiting the Greater Boston Area!
=====
Speed of Inference
-----
Generation Stage : 33.07 ms/token
=====
USER: 
```

**LLaMA-2-7B (W4A16, AWQ): 30 tokens / s**

<https://github.com/mit-han-lab/llm-awq>

# Lecture Plan

Today, we will cover:

## 1. Efficient inference algorithms for LLMs

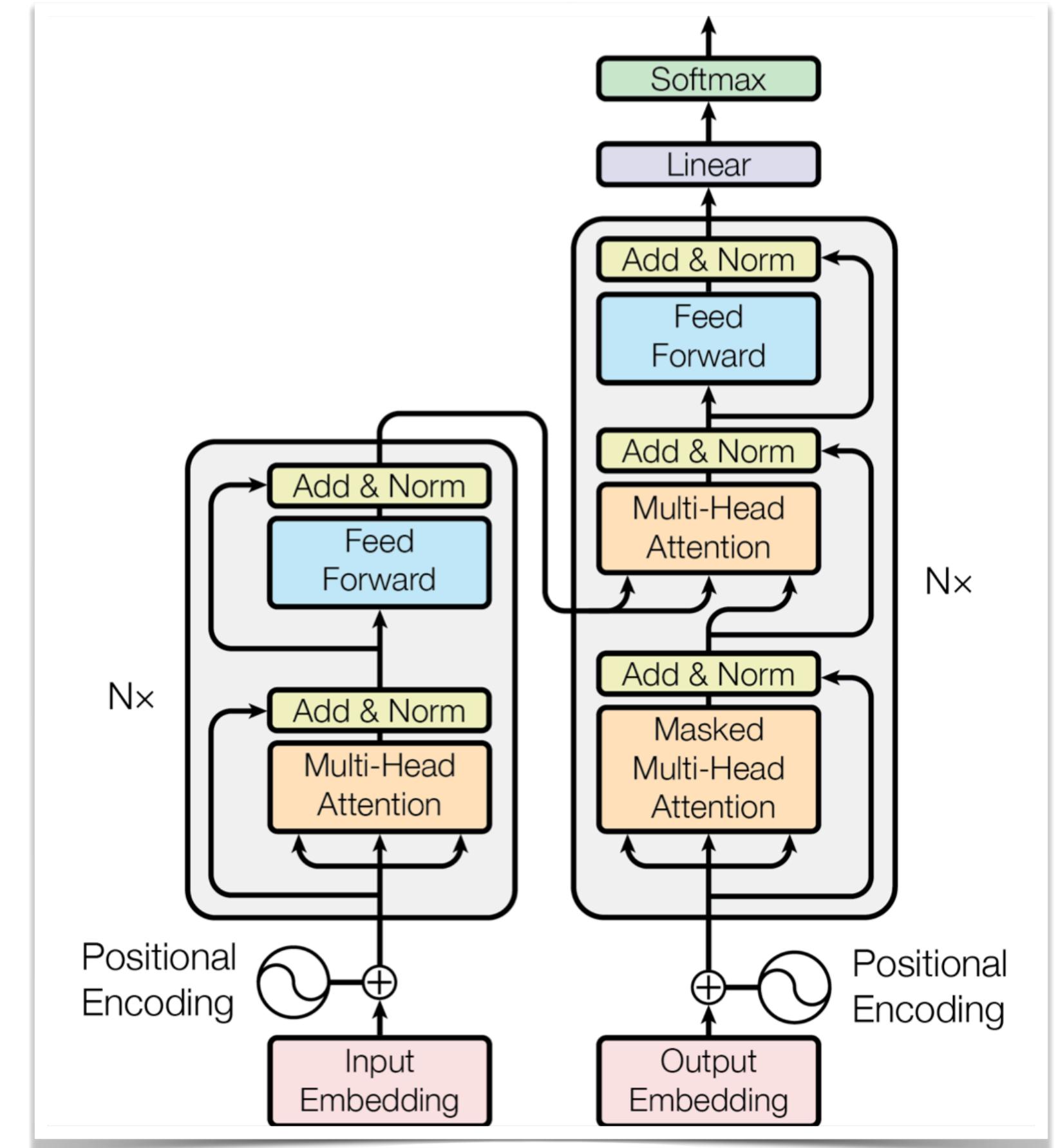
1. Quantization: SmoothQuant, AWQ, TinyChat
2. Pruning/sparsity: SpAtten, H2O, MoE

## 2. Efficient inference systems for LLMs

1. vLLM
2. StreamingLLM
3. FlashAttention
4. Speculative decoding

## 3. Efficient fine-tuning for LLMs

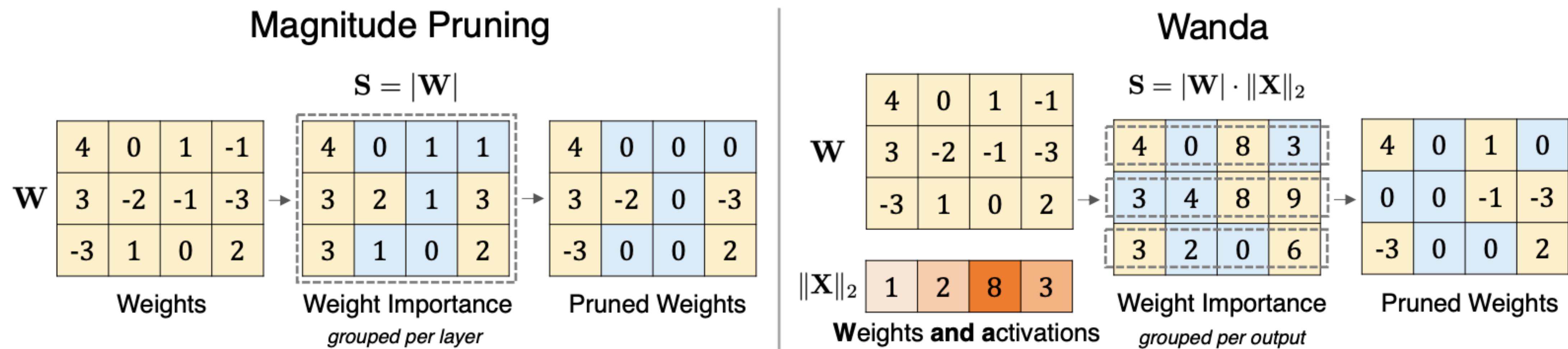
1. LoRA/QLoRA
2. Adapter
3. Prompt Tuning



# Pruning

## Wanda: pruning by considering weights and activations

- Similar idea compared to AWQ: we should also consider activation distribution when pruning weights!
- Use  $|\text{weight}| * \|\text{activation}\|$  as the criteria for pruning



# Pruning

## Wanda: pruning by considering weights and activations

- Similar idea compared to AWQ: we should also consider activation distribution when pruning weights!
- Use  $|\text{weight}| * \|\text{activation}\|$  as the criteria for pruning
- Consistently outperforms magnitude-based pruning

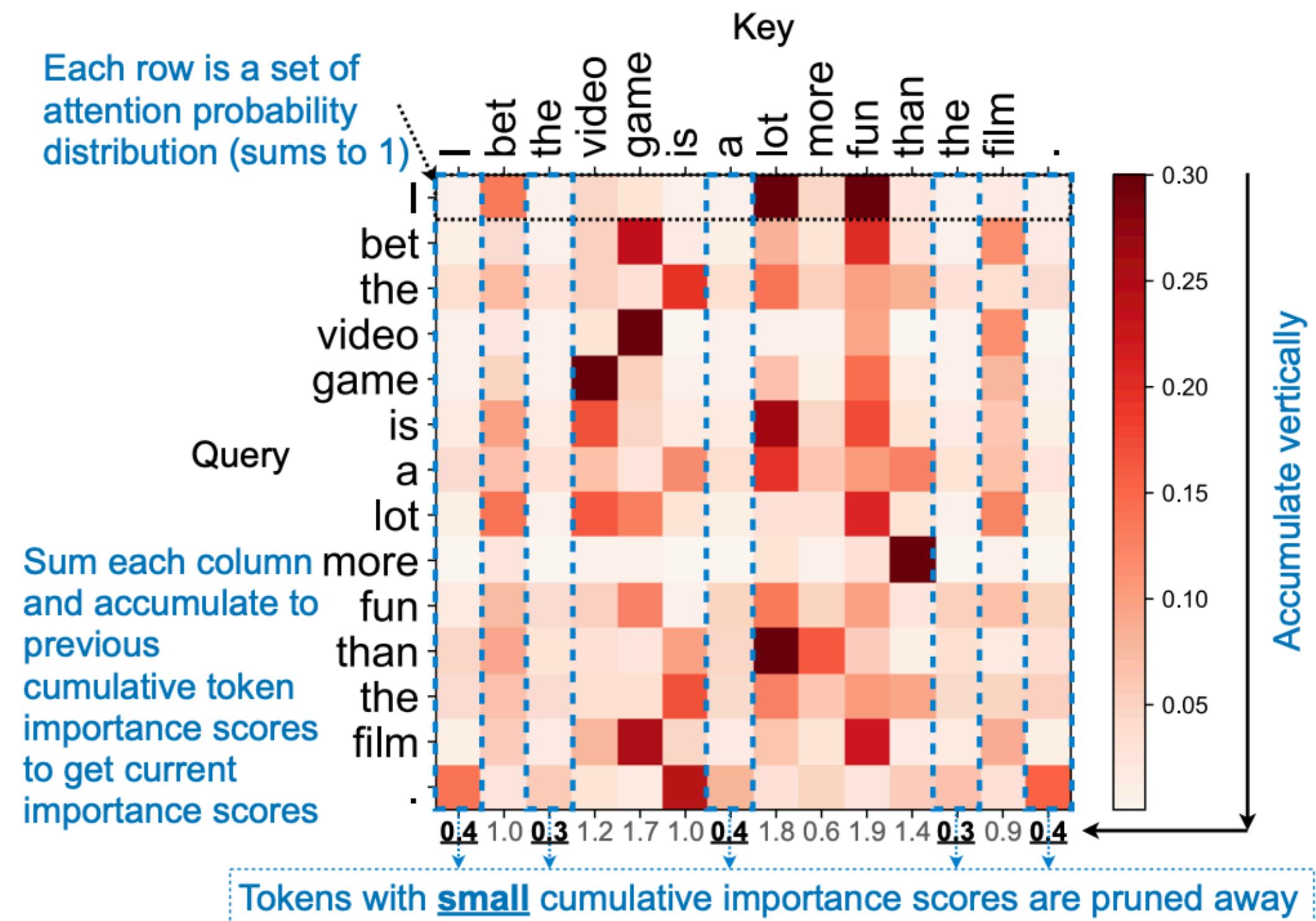
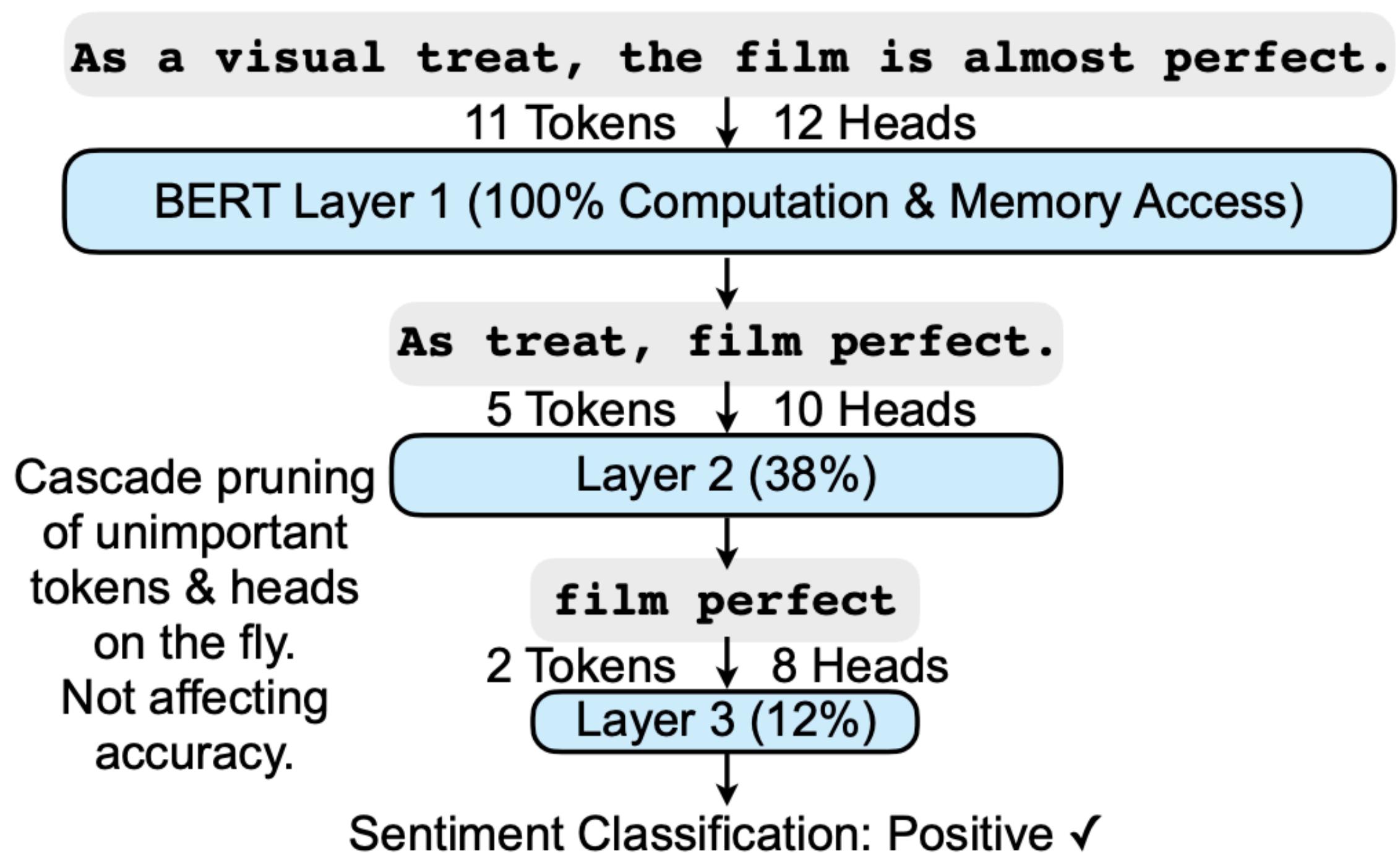
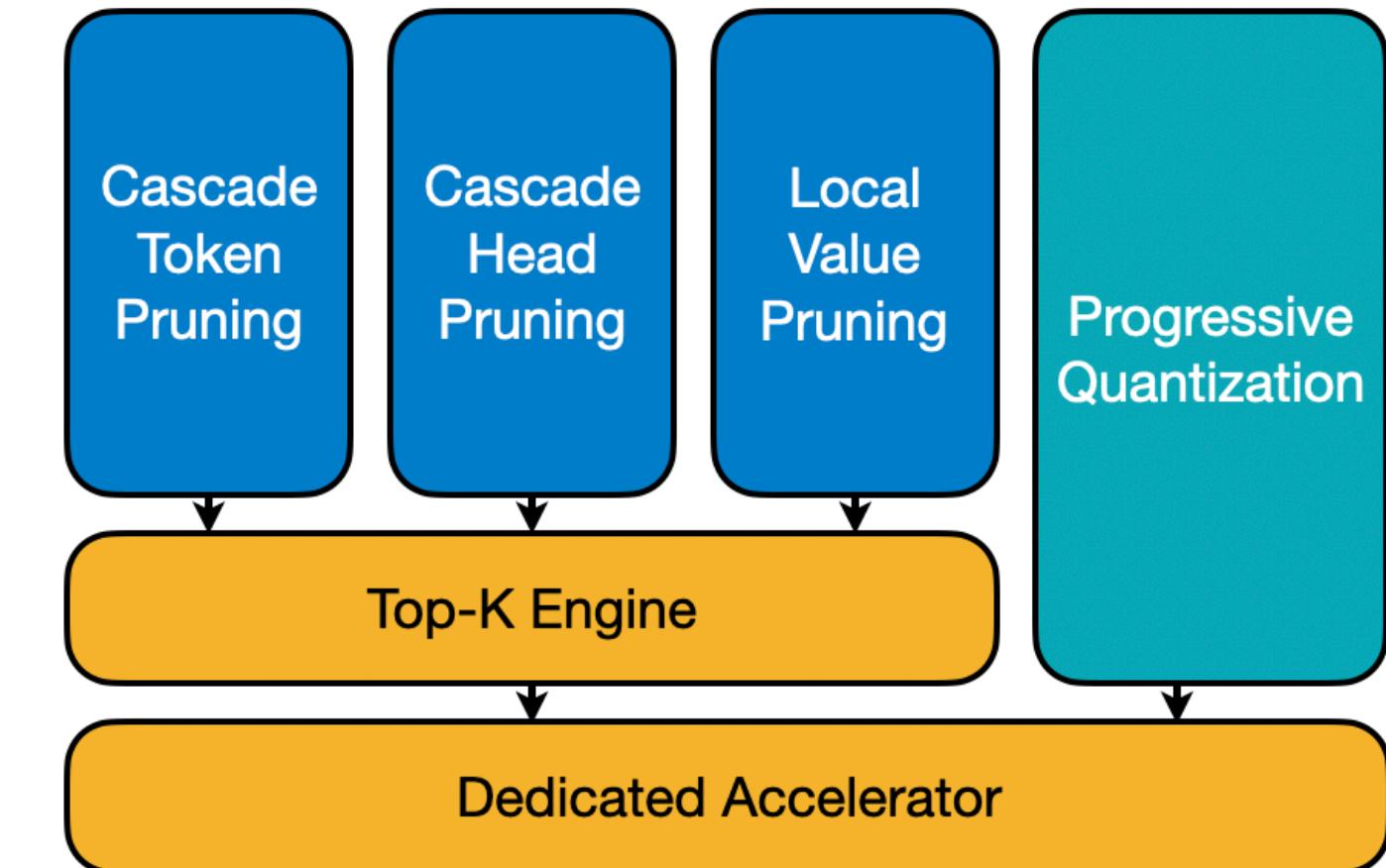
Method	Weight Update	Sparsity	LLaMA				LLaMA-2		
			7B	13B	30B	65B	7B	13B	70B
Dense	-	0%	59.99	62.59	65.38	66.97	59.71	63.03	67.08
Magnitude	✗	50%	46.94	47.61	53.83	62.74	51.14	52.85	60.93
SparseGPT	✓	50%	<b>54.94</b>	58.61	63.09	66.30	<b>56.24</b>	60.72	<b>67.28</b>
Wanda	✗	50%	54.21	<b>59.33</b>	<b>63.60</b>	<b>66.67</b>	<b>56.24</b>	<b>60.83</b>	67.03
Magnitude	✗	4:8	46.03	50.53	53.53	62.17	50.64	52.81	60.28
SparseGPT	✓	4:8	<b>52.80</b>	55.99	60.79	64.87	<b>53.80</b>	<b>59.15</b>	65.84
Wanda	✗	4:8	52.76	<b>56.09</b>	<b>61.00</b>	<b>64.97</b>	52.49	58.75	<b>66.06</b>
Magnitude	✗	2:4	44.73	48.00	53.16	61.28	45.58	49.89	59.95
SparseGPT	✓	2:4	<b>50.60</b>	<b>53.22</b>	58.91	62.57	<b>50.94</b>	54.86	63.89
Wanda	✗	2:4	48.53	52.30	<b>59.21</b>	<b>62.84</b>	48.75	<b>55.03</b>	<b>64.14</b>

A Simple and Effective Pruning Approach for Large Language Models (Sun et al., 2023)

# Attention Sparsity

## SpAtten: token pruning & head pruning

- Cascade pruning of unimportant tokens and heads.
- Tokens with a small cumulative attention are pruned away.
- V pruning: don't fetch V if QK is small.
- Progressive quantization: low precision first, if not confident => high precision.

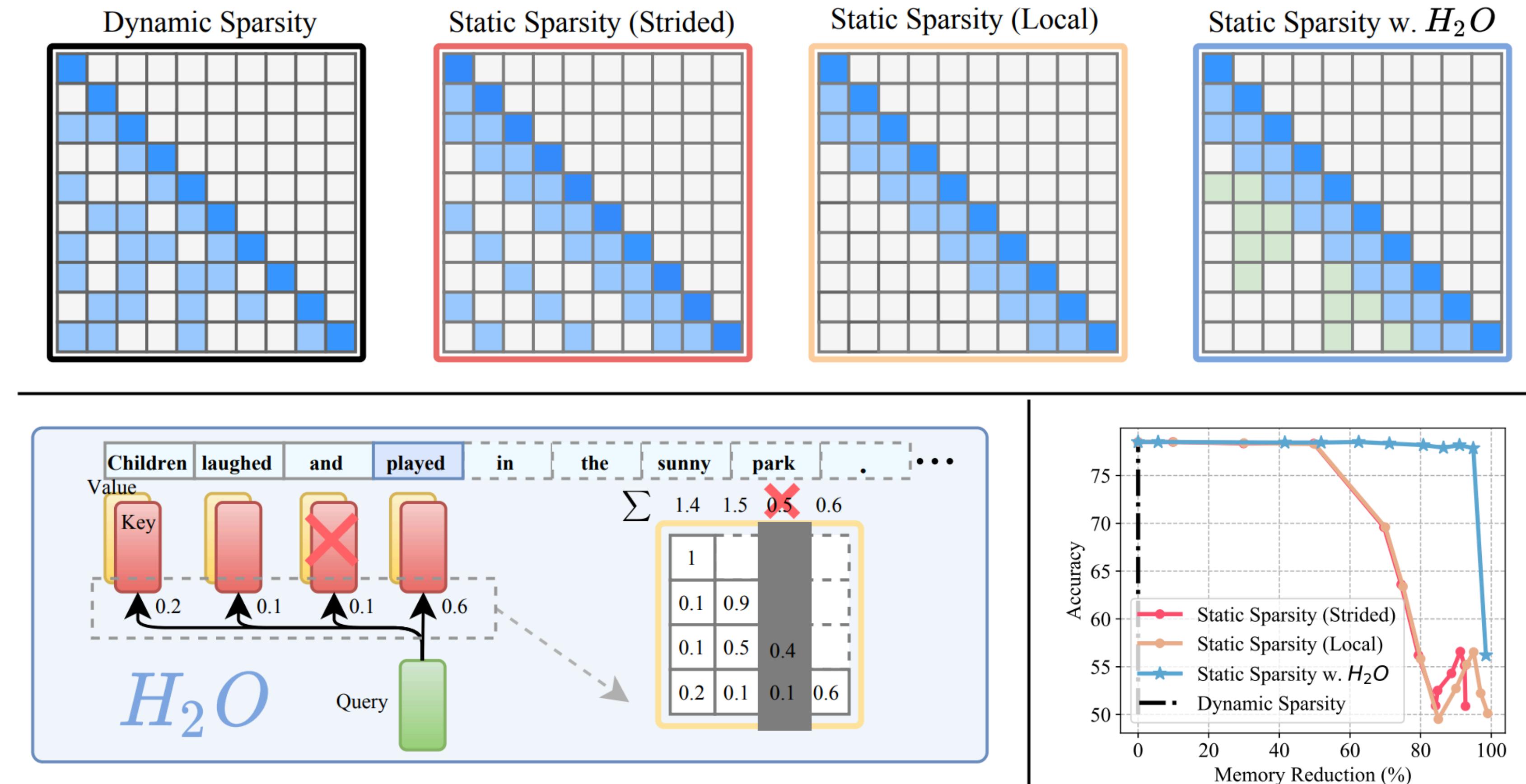


SpAtten: Efficient Sparse Attention Architecture with Cascade Token and Head Pruning (Wang et al., 2020)

# Attention Sparsity

## H<sub>2</sub>O: token pruning in KV cache

- Keep the local tokens and heavy Hitter Tokens (H<sub>2</sub>) in the cache

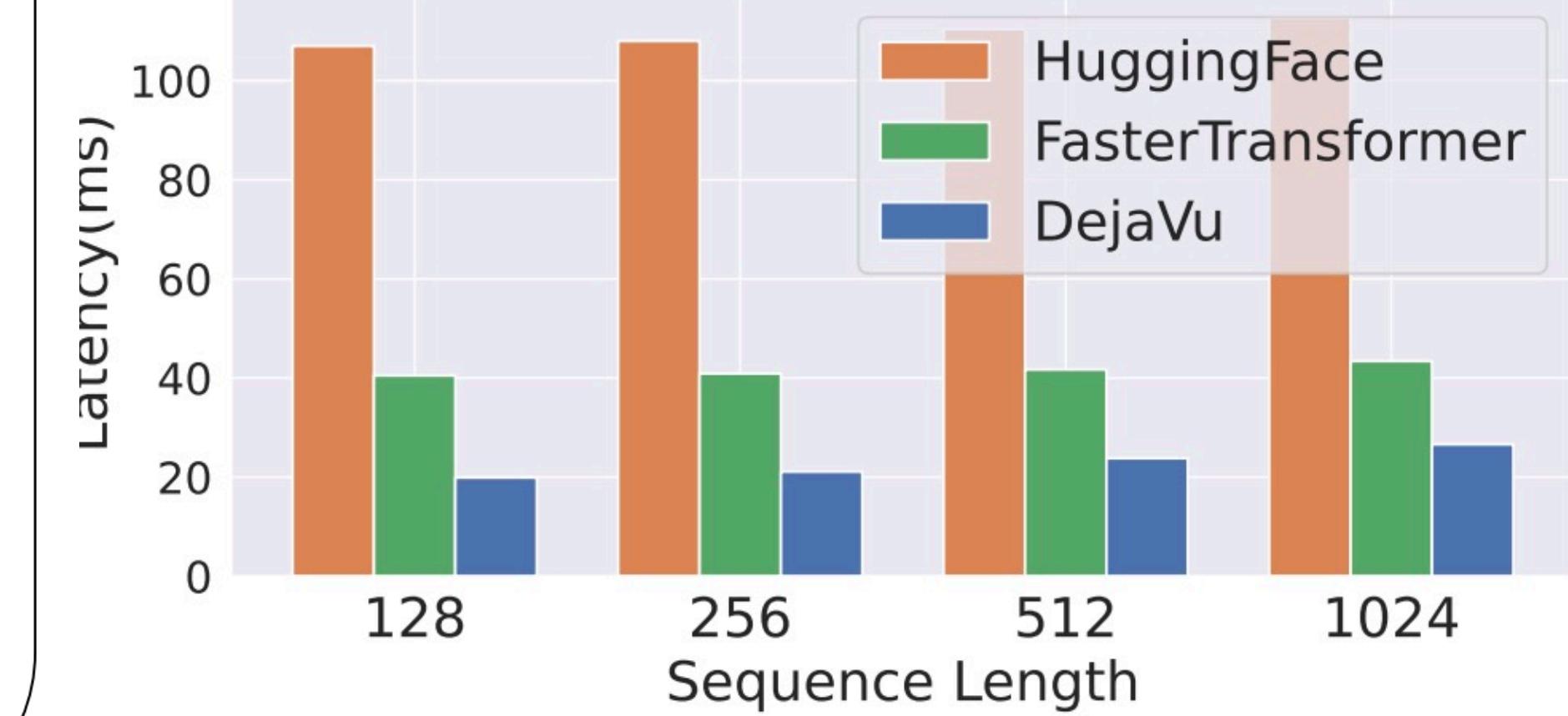
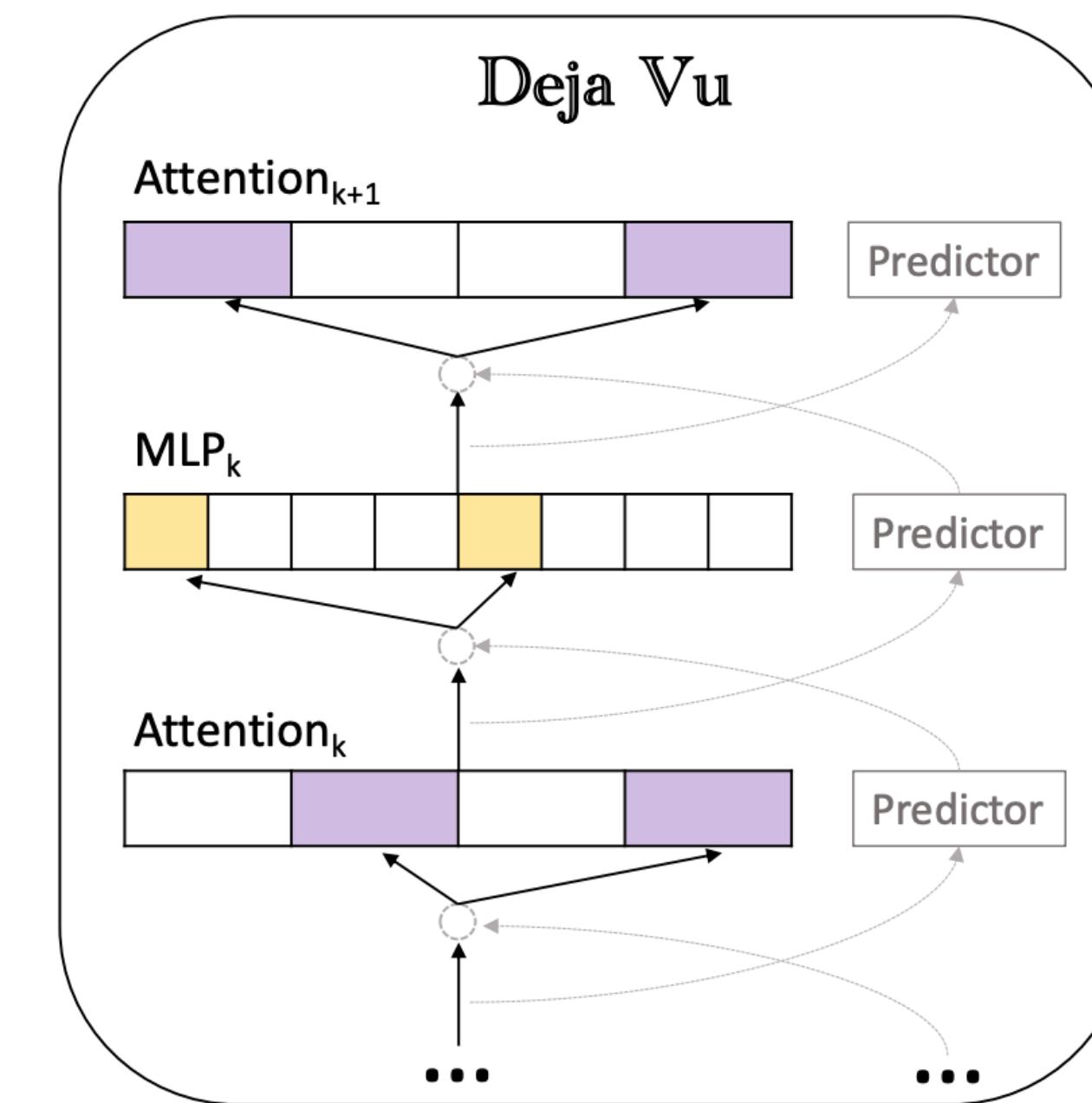
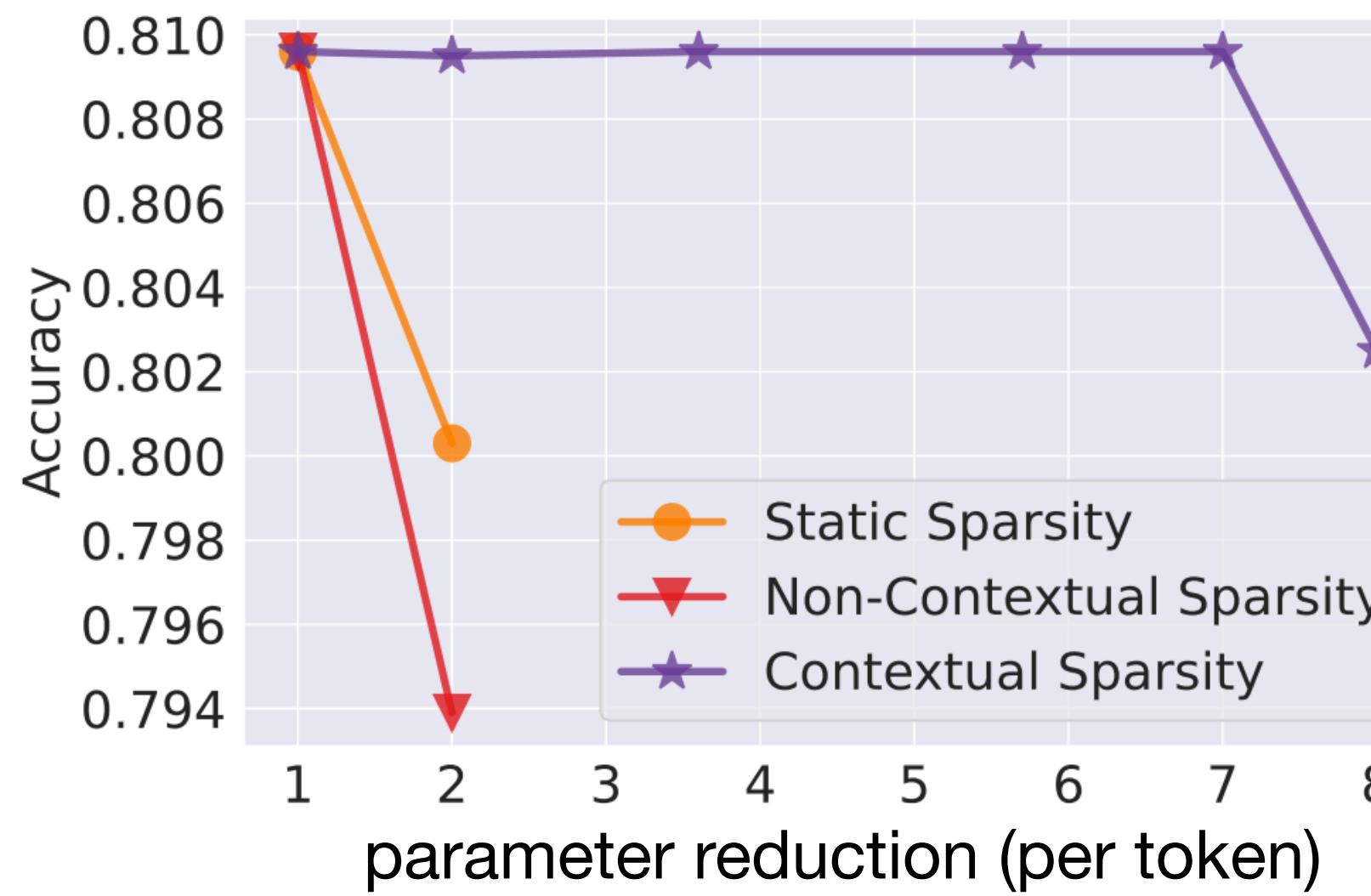


H<sub>2</sub>O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models (Zhang et al., 2023)

# Contextual Sparsity

## DejaVu (input dependent sparsity)

- *Static* sparsity: hurts the accuracy with a medium-high sparsity
- *Contextual* sparsity: small, input-dependent sets of redundant heads and features
- Contextual sparsity exists and can be predicted (using an async predictor head)
- Accelerate inference without hurting model quality

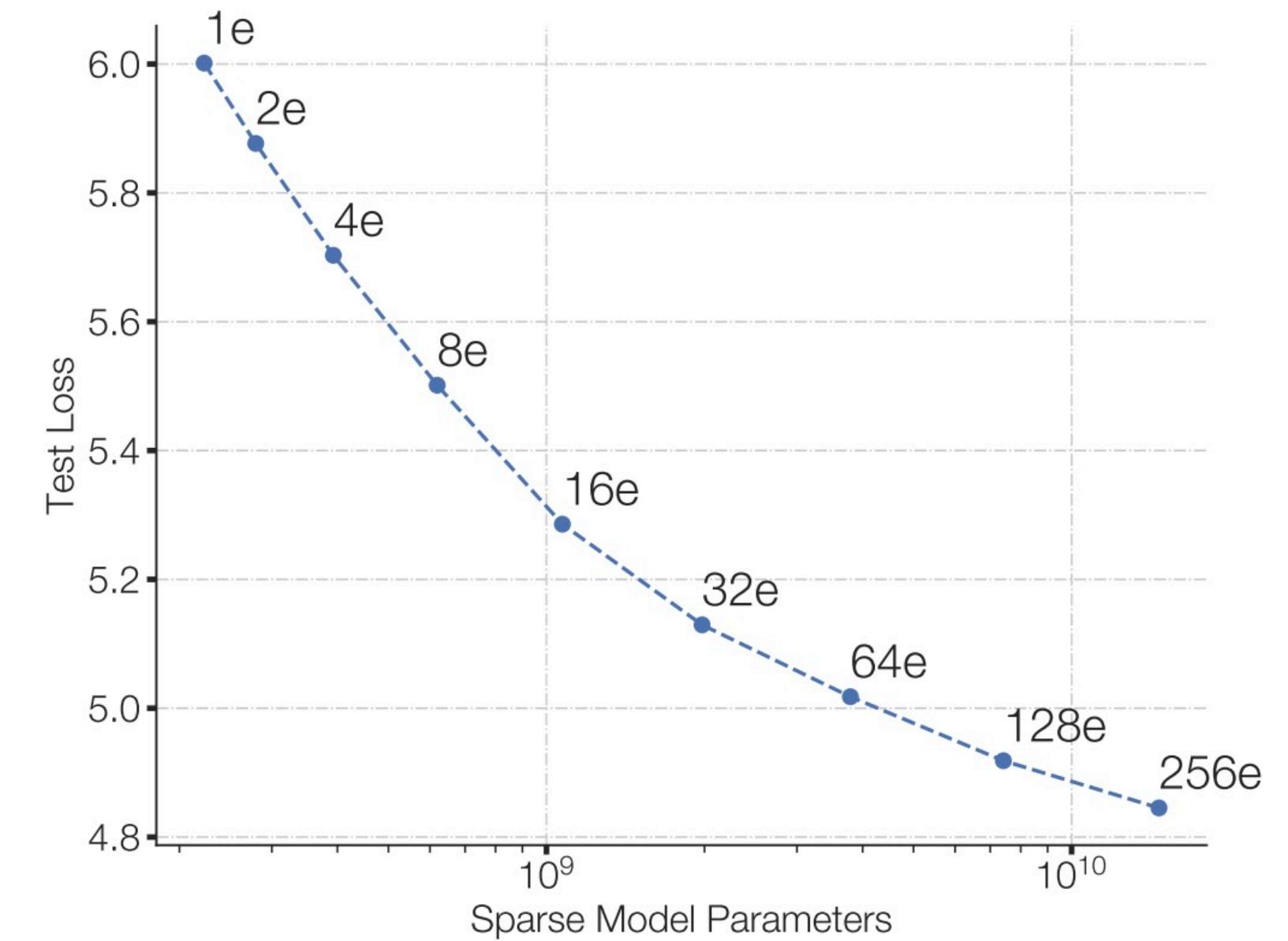
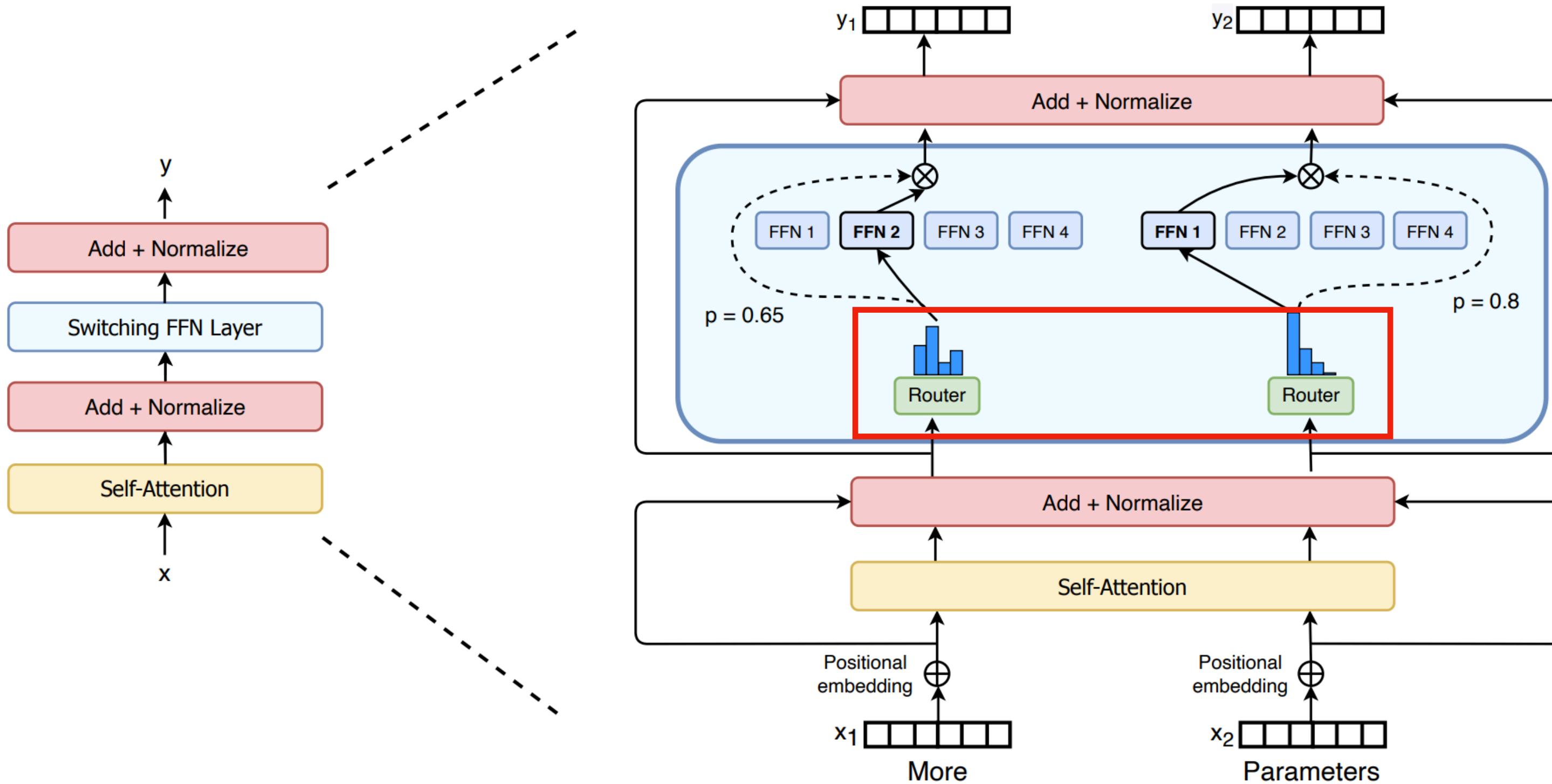


Deja Vu: Contextual Sparsity for Efficient LLMs at Inference Time (Liu et al., 2023)

# Mixture-of-Experts (MoE)

## Sparsely activated for each token

- MoE allows us to sparsely use part of the parameters for each token during inference
- It can increase the **total** amount of parameters without increasing inference costs **per token**
- A router will distribute the workload among different experts
- More experts -> larger total model size -> lower loss/better perplexity

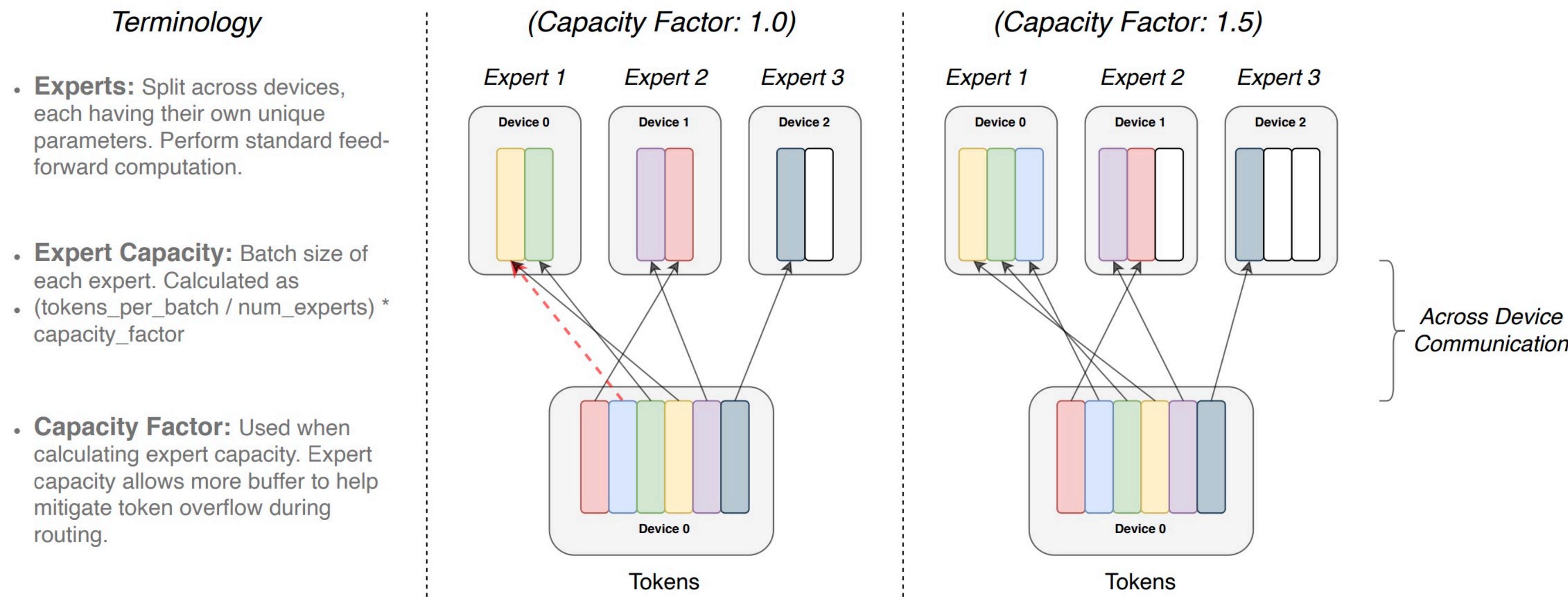


Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity [Fedus et al., 2022]

# Mixture-of-Experts (MoE)

## Sparsely activated for each token

- A key component of MoE is the routing function
- Capacity factor  $C = \left( \frac{\text{tokens per batch}}{\text{number of experts}} \right) \times \text{capacity factor}$ .
- $C = 1$ , every expert can process at most  $6/3*1=2$  tokens; one token is skipped
- $C = 1.5$ , every expert can process at most  $6/3*1.5=3$  tokens; slack capacity for expert 2&3

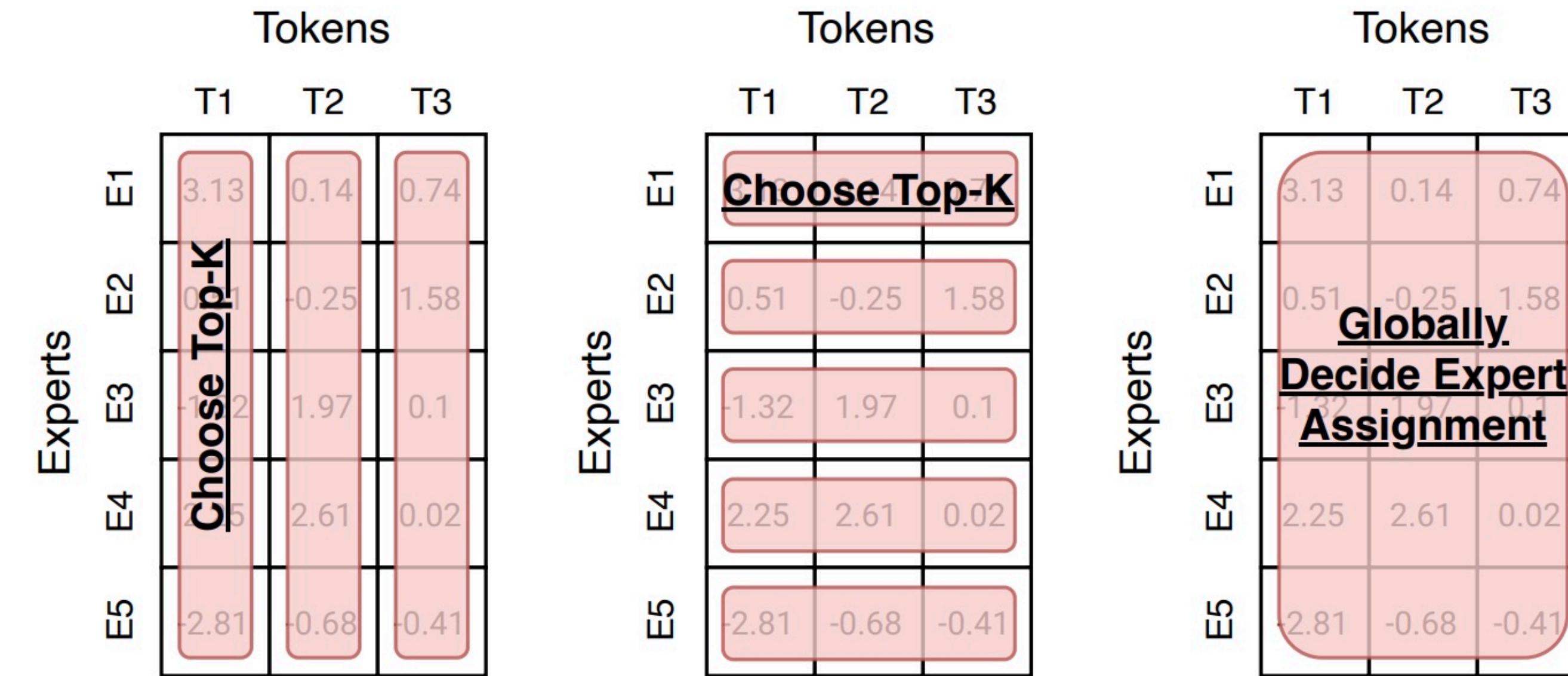


Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity [Fedus et al., 2022]

# Mixture-of-Experts (MoE)

Sparsely activated for each token

- Different routing mechanisms

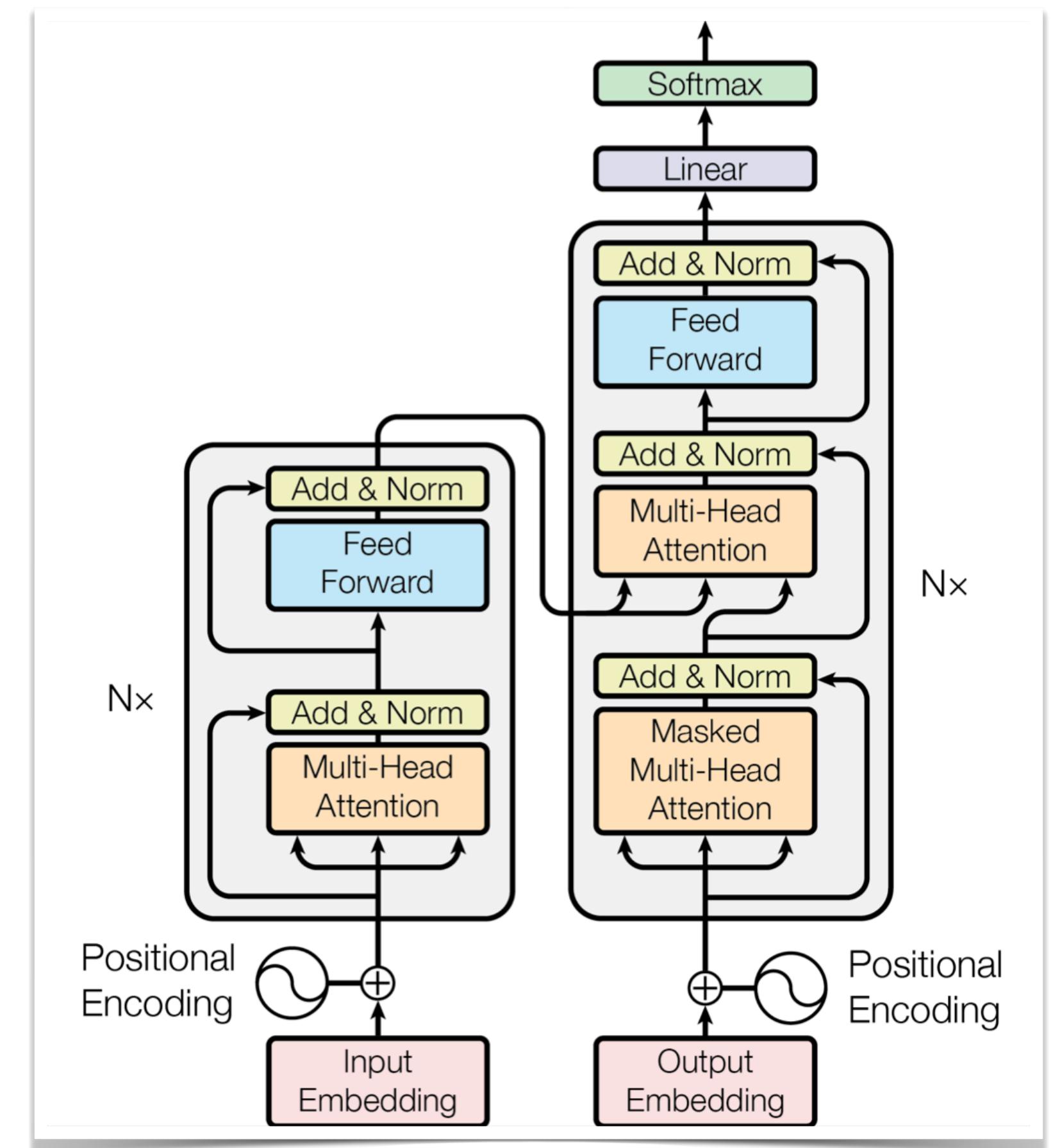


A Review of Sparse Expert Models in Deep Learning [Fedus et al., 2022]

# Lecture Plan

Today, we will cover:

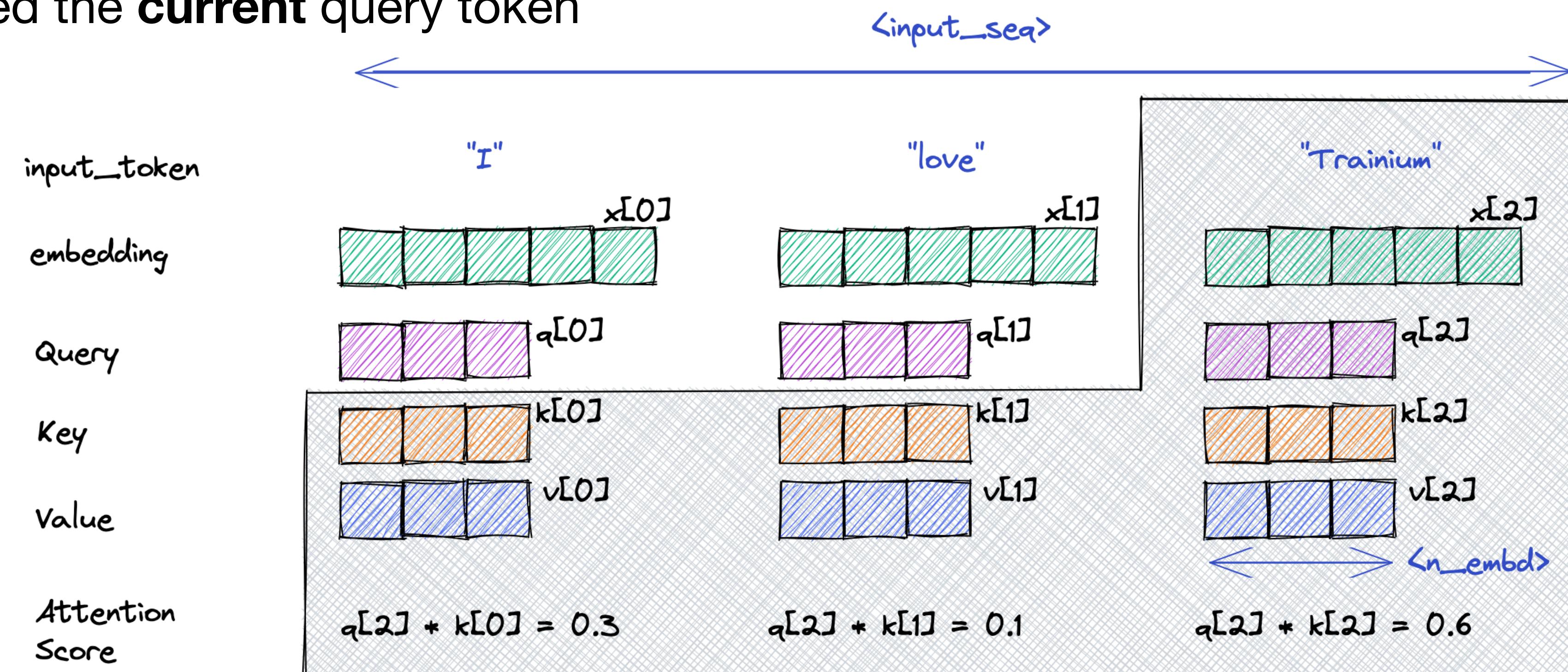
1. Efficient inference algorithms for LLMs
  1. Quantization
  2. Pruning/sparsity
2. Efficient inference systems for LLMs
  1. vLLM
  2. StreamingLLM
  3. FlashAttention
  4. Speculative decoding
3. Efficient fine-tuning for LLMs
  1. LoRA/QLoRA
  2. Adapter
  3. Prompt Tuning



# Recap: KV Cache

The KV cache could be large with long context

- During Transformer decoding (GPT-style), we need to store the **Keys** and **Values** of **all previous** tokens so that we can perform the attention computation, namely the **KV cache**
  - Only need the **current** query token



$$a_{ij} = \frac{\exp(q_i^\top k_j / \sqrt{d})}{\sum_{t=1}^i \exp(q_i^\top k_t / \sqrt{d})}, \quad o_i = \sum_{j=1}^i a_{ij} v_j$$

Image credit: <https://awsdocs-neuron.readthedocs-hosted.com/en/latest/general/appnotes/transformers-neuronx/generative-llm-inference-with-neuron.html>

# Recap: the Challenge of Large KV Cache

## The KV cache could be large with long context

- We can calculate the memory required to store the KV cache
- Consider Llama-2-70B (if using MHA), KV cache requires

$$\underbrace{BS}_{minibatch} * \underbrace{80}_{layers} * \underbrace{64}_{kv-heads} * \underbrace{128}_{n_{emd}} * \underbrace{N}_{length} * \underbrace{2}_{K\&V} * \overbrace{2\text{bytes}}^{\text{FP16}} = 2.5\text{MB} \times BS \times N$$

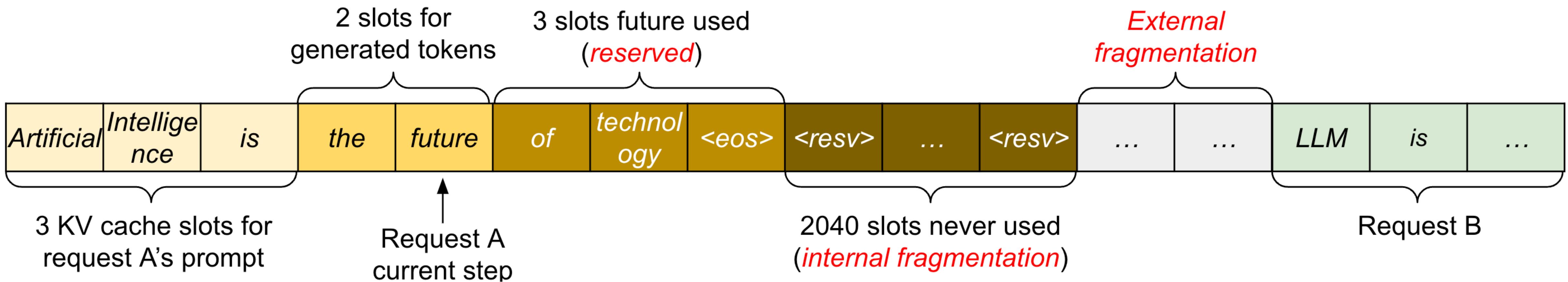
- bs=1, n\_seq=512: 1.25GB
- bs=1, n\_seq=4096 : 10GB (~ a paper)
- bs=16, n\_seq=4096: 160GB (requires two A100 GPU!)

<https://www.databricks.com/blog/llm-inference-performance-engineering-best-practices>

# vLLM and Paged Attention

## High-throughput and memory-efficient serving

- Analyzing the waste in KV Cache usage:
  - *Internal fragmentation*: over-allocated due to the unknown output length.
  - *Reservation*: not used at the current step, but used in the future.
  - *External fragmentation*: due to different sequence lengths.

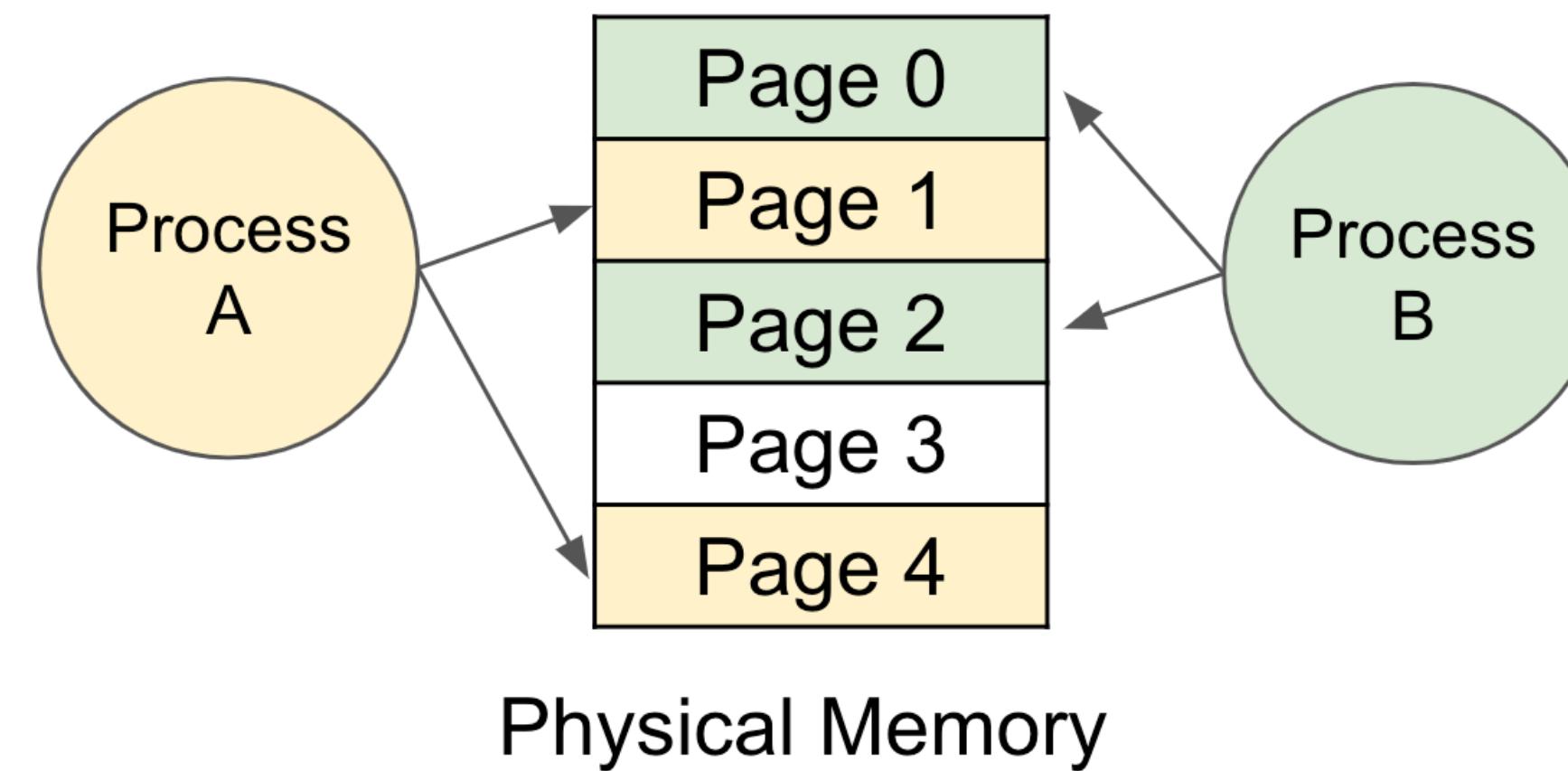


# vLLM and Paged Attention

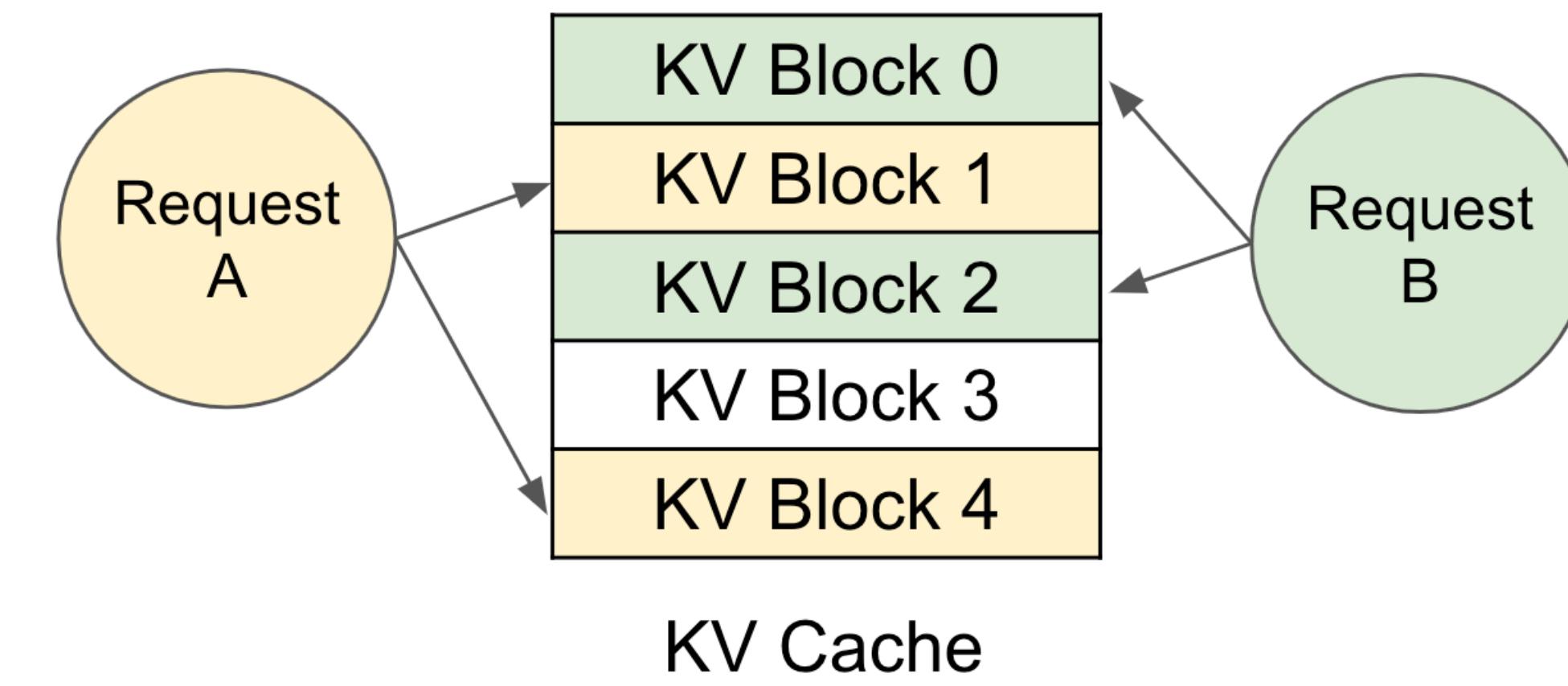
## High-throughput and memory-efficient serving

- Inspiration from operating systems (OS): virtual memory and paging!

### Memory management in OS



### Memory management in vLLM

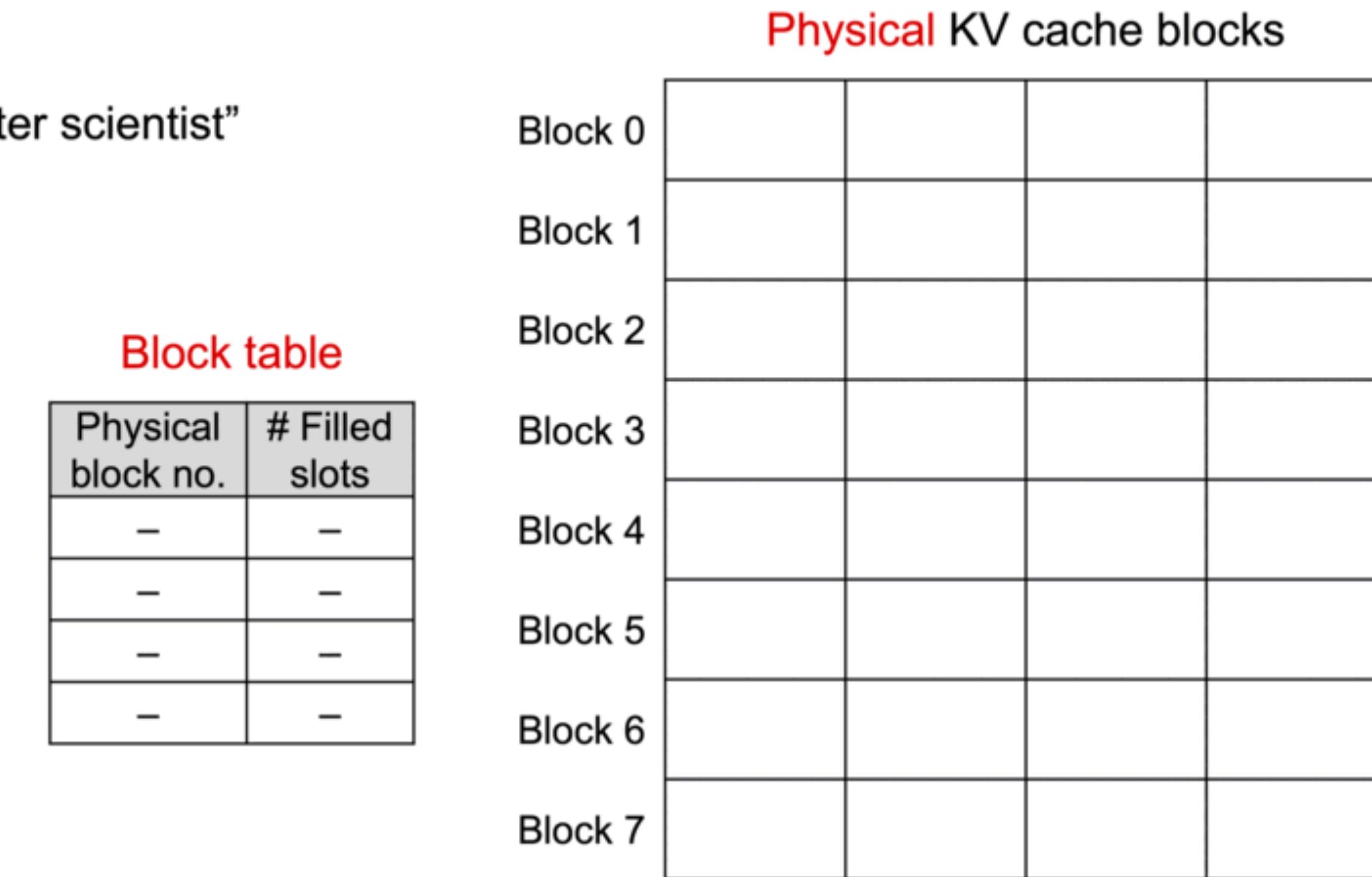
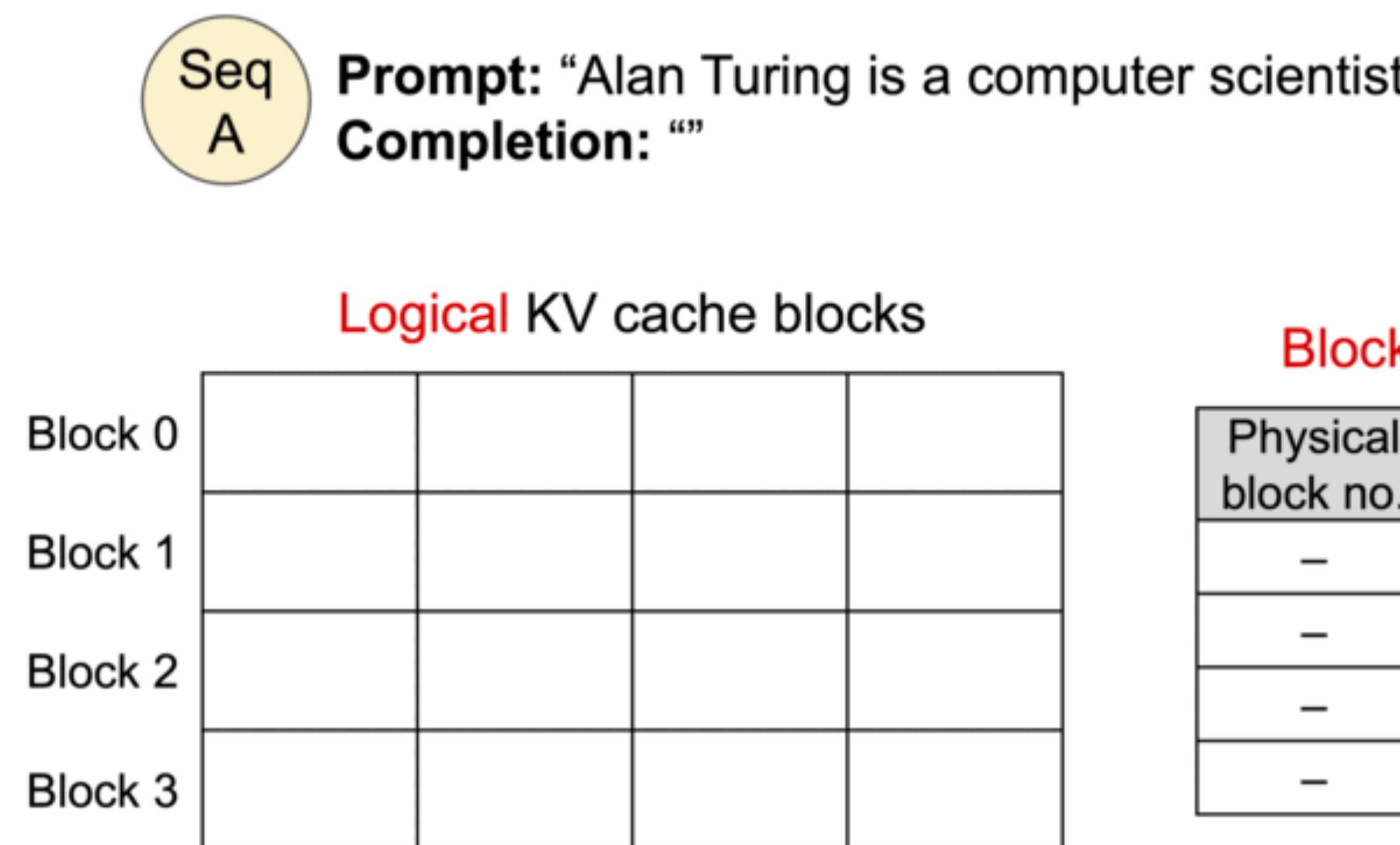


# vLLM and Paged Attention

## PagedAttention

- PagedAttention addresses the KV-cache memory fragmentation
- It allows for storing continuous keys and values in non-contiguous memory space

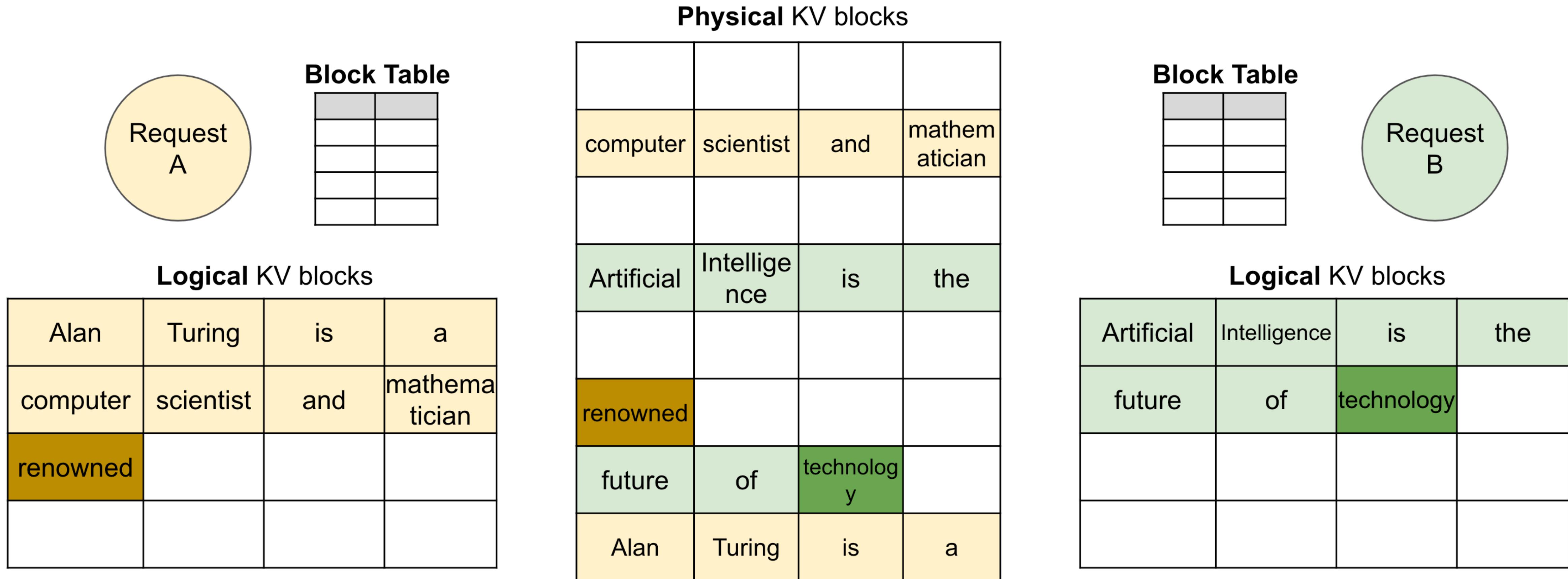
0. Before generation.



# vLLM and Paged Attention

## PagedAttention

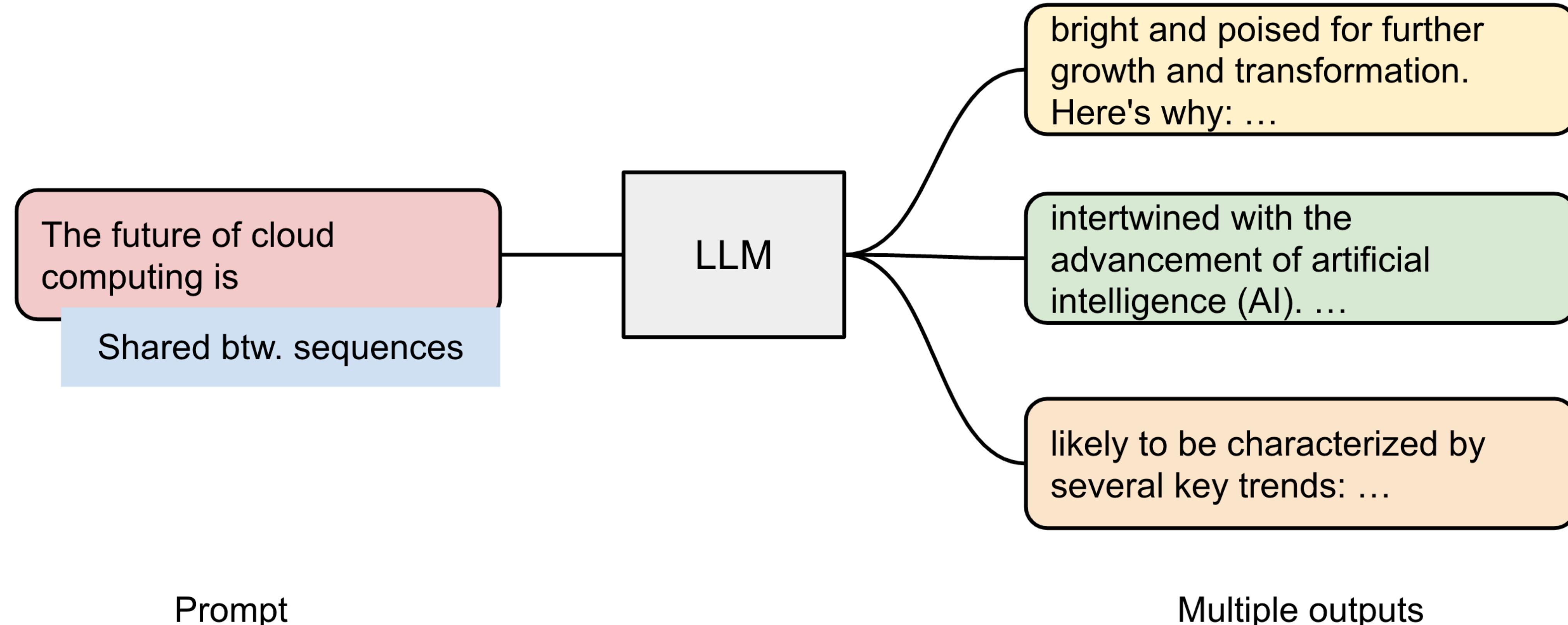
- The case for multiple requests



# vLLM and Paged Attention

## PagedAttention

- Dynamic block mapping enables prompt sharing in parallel sampling:

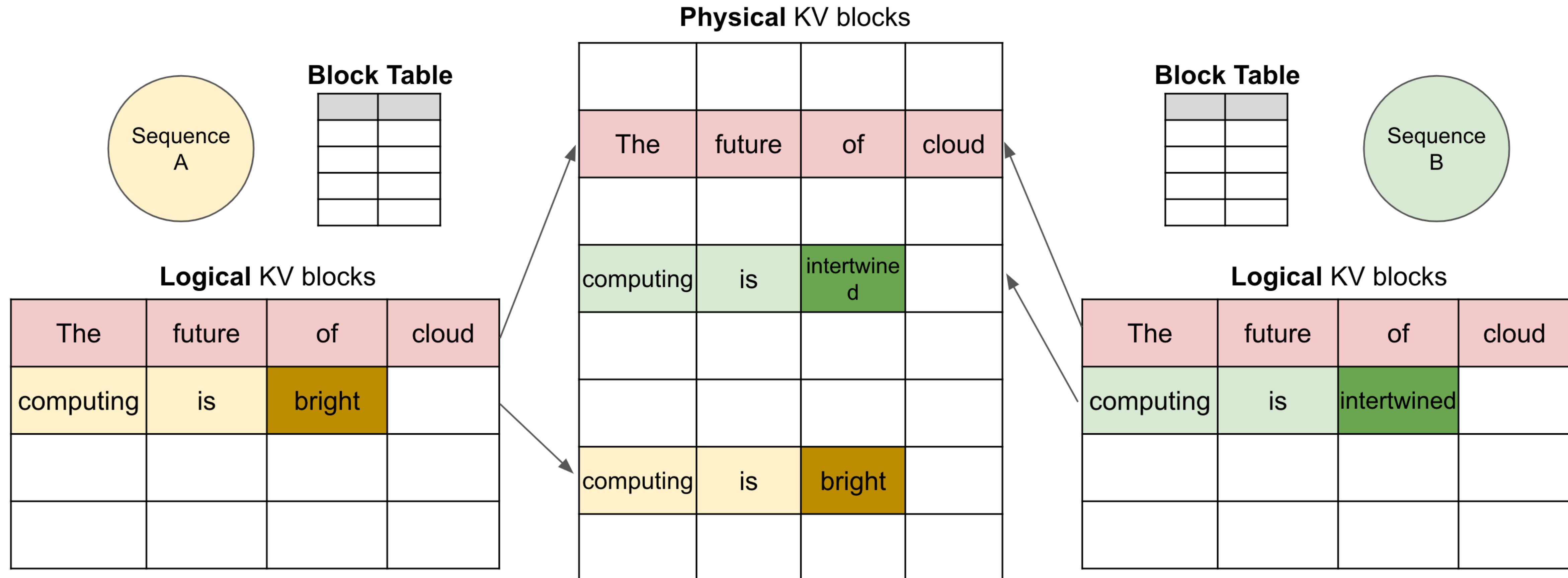


Efficient Memory Management for Large Language Model Serving with PagedAttention (Kwon et al., 2023)

# vLLM and Paged Attention

## PagedAttention

- Dynamic block mapping enables prompt sharing in parallel sampling



# Lecture Plan

Today, we will cover:

## 1. Efficient inference algorithms for LLMs

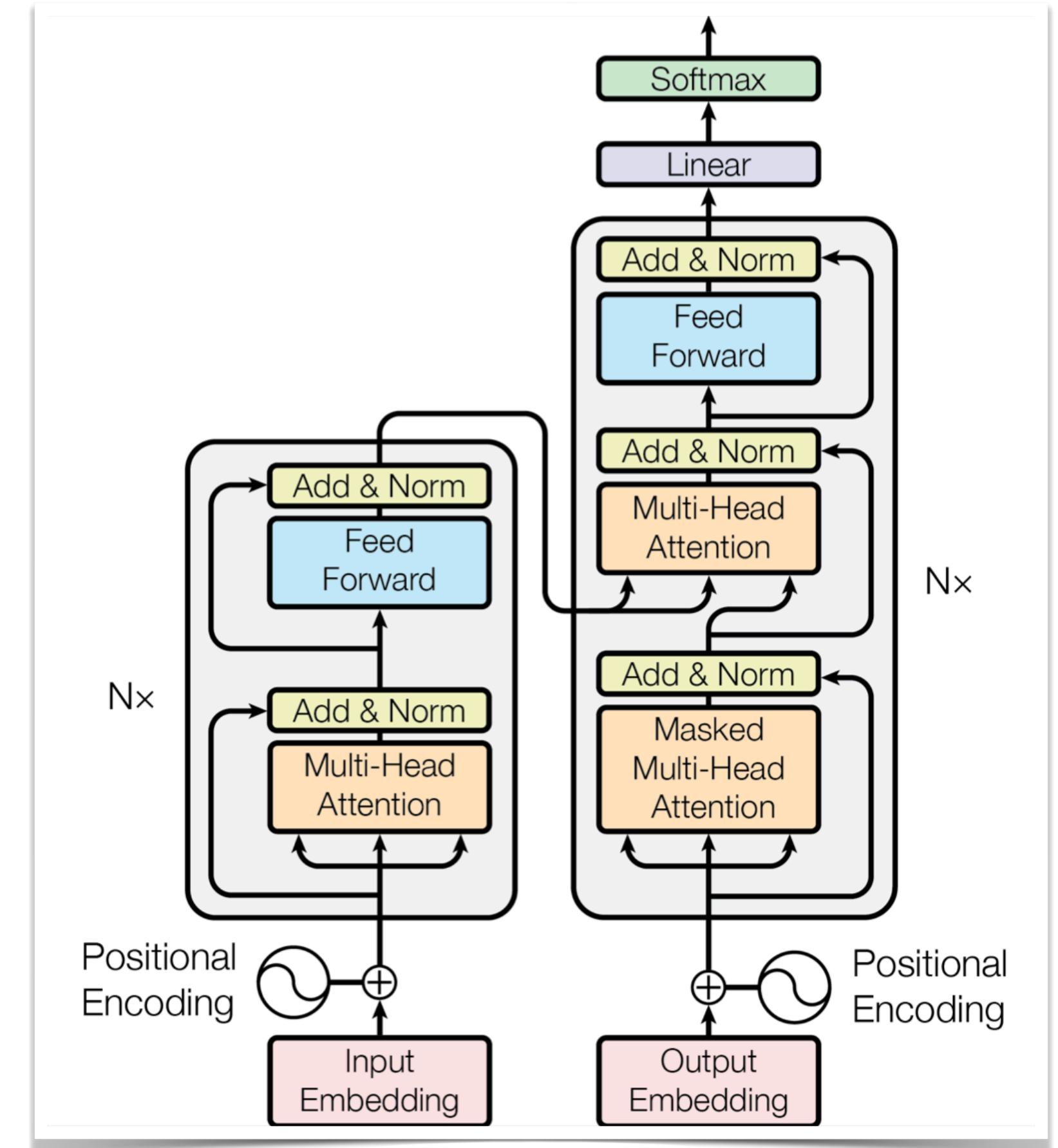
1. Quantization: SmoothQuant, AWQ, TinyChat
2. Pruning/sparsity: SpAtten, H2O, MoE

## 2. Efficient inference systems for LLMs

1. vLLM
2. StreamingLLM
3. FlashAttention
4. Speculative decoding

## 3. Efficient fine-tuning for LLMs

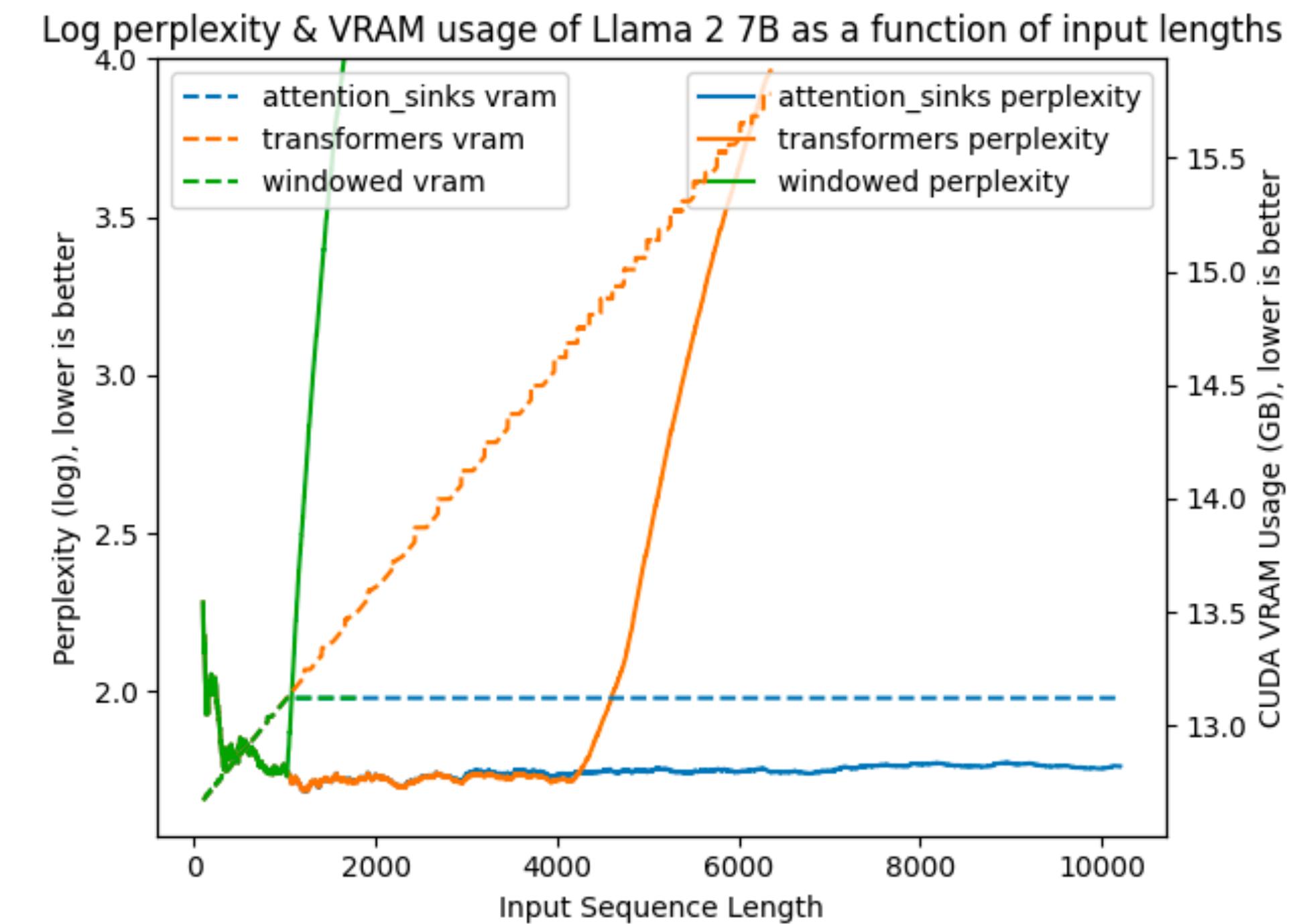
1. LoRA/QLoRA
2. Adapter
3. Prompt Tuning



# StreamingLLM

## Deploying LLM in streaming applications is challenging

- Urgent need for LLMs in streaming applications such as multi-round dialogues, where long interactions are needed.
- Challenges:
  - Extensive memory consumption during the decoding stage.
  - Inability of popular LLMs to generalize to longer text sequences.



Efficient Streaming Language Models with Attention Sinks

# Challenges of Deploying LLMs in Streaming Applications

w/o StreamingLLM

```
(streaming) guangxuan@l29:~/workspace/streaming-llm$ CUDA_VISIBLE_DEVICES=0 python examples/run_streaming_llama.py  
Loading model from lmsys/vicuna-13b-v1.3 ...  
Loading checkpoint shards: 67%|██████████| 2/3 [00:09<00:04, 4.94s/it]
```

w/ StreamingLLM

```
(streaming) guangxuan@l29:~/workspace/streaming-llm$ CUDA_VISIBLE_DEVICES=1 python examples/run_streaming_llama.py --enable_streaming  
Loading model from lmsys/vicuna-13b-v1.3 ...  
Loading checkpoint shards: 67%|██████████| 2/3 [00:09<00:04, 4.89s/it]
```

# Challenges of Deploying LLMs in Streaming Applications

## w/o StreamingLLM

# Model Performance Breaks

ASSISTANT: 0000000-t-t-t-t

USER: Write a C++ program to print Fibonacci number using recursion.

USER: Now we define a sequence of numbers in which each number is the sum of the three preceding ones. The first three numbers are 0, -1, -1. Write a program to find the nth number.

ASSTANT: 0-a-a-a-eah000000000000

**USER:** Write a simple website in HTML. When a user clicks the button, it shows a random joke from a list of 4 jokes.

ASSISTANT: ■

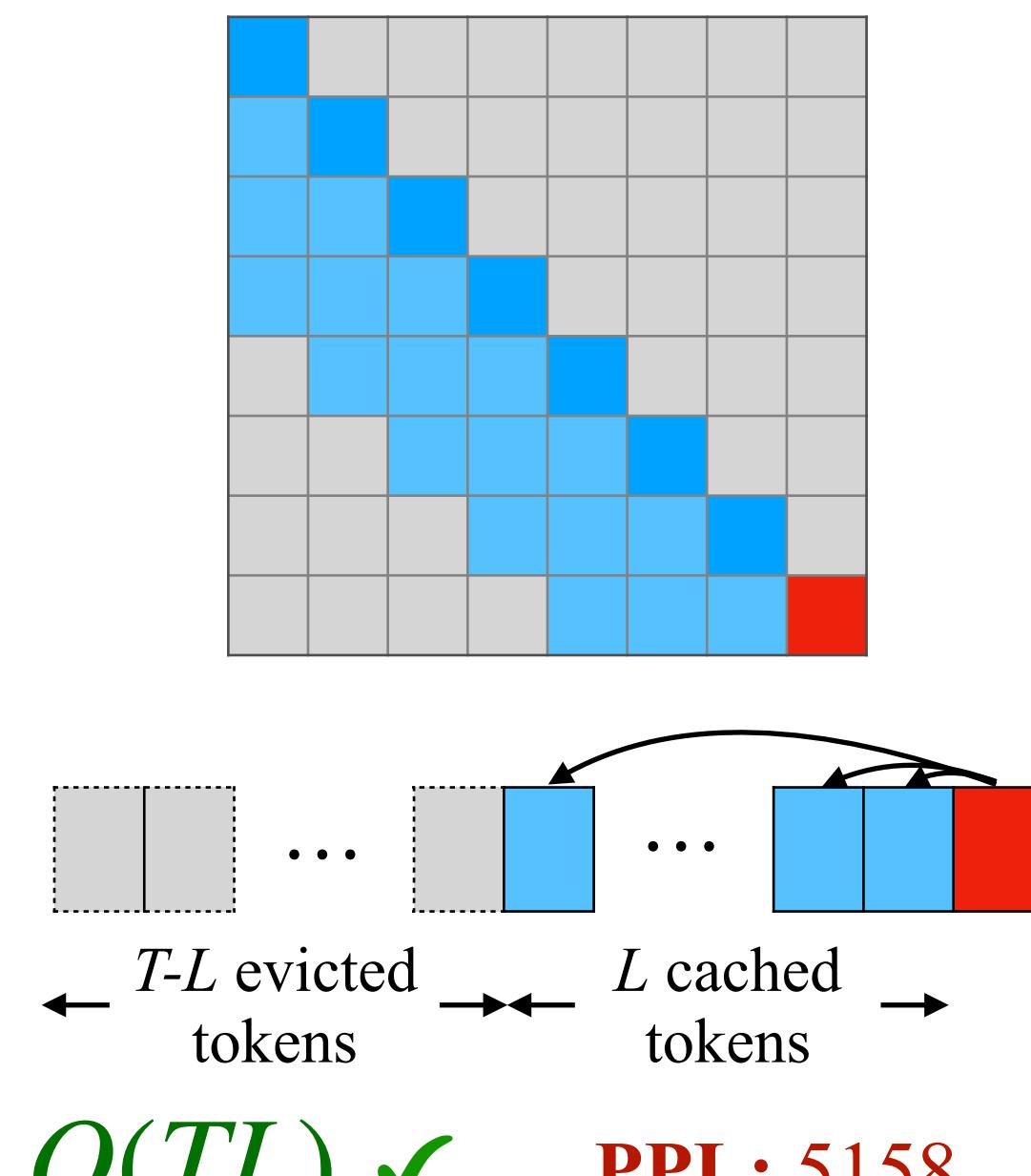
# w/o StreamingLLM

```
outputs = model(
    File "/home/guangxuan/miniconda3/envs/streaming/lib/python3.8/site-packages/torch/nn/modules/module.py", line 1501, in __call__
        return forward_call(*args, **kwargs)
    File "/home/guangxuan/miniconda3/envs/streaming/lib/python3.8/site-packages/transformers/models/llama/modeling_llama.py", line 820, in forward
        outputs = self.model(
            File "/home/guangxuan/miniconda3/envs/streaming/lib/python3.8/site-packages/torch/nn/modules/module.py", line 1501, in __call__
                return forward_call(*args, **kwargs)
            File "/home/guangxuan/miniconda3/envs/streaming/lib/python3.8/site-packages/transformers/models/llama/modeling_llama.py", line 708, in forward
                layer_outputs = decoder_layer(
                    File "/home/guangxuan/miniconda3/envs/streaming/lib/python3.8/site-packages/torch/nn/modules/module.py", line 1501, in __call__
                        return forward_call(*args, **kwargs)
                    File "/home/guangxuan/miniconda3/envs/streaming/lib/python3.8/site-packages/transformers/models/llama/modeling_llama.py", line 424, in forward
                        hidden_states, self_attn_weights, present_key_value = self.self_attn(
                            File "/home/guangxuan/miniconda3/envs/streaming/lib/python3.8/site-packages/torch/nn/modules/module.py", line 1501, in __call__
                                return forward_call(*args, **kwargs)
                            File "/home/guangxuan/miniconda3/envs/streaming/lib/python3.8/site-packages/transformers/models/llama/modeling_llama.py", line 337, in forward
                                key_states = torch.cat([past_key_value[0], key_states], dim=2)
torch.cuda.OutOfMemoryError: CUDA out of memory. Tried to allocate 90.00 MiB (GPU 0; 47.54 GiB total capacity; 44.53 GiB already allocated; 81.06 MiB free; 46.47 GiB reserved in total by PyTorch) If reserved memory is >> allocated memory try setting max_split_size_mb to avoid fragmentation. See documentation for Memory Management and PYTORCH_CUDA_ALLOC_CONF
(streaming) quangxuan@l29:~/workspace/streaming-llm$
```

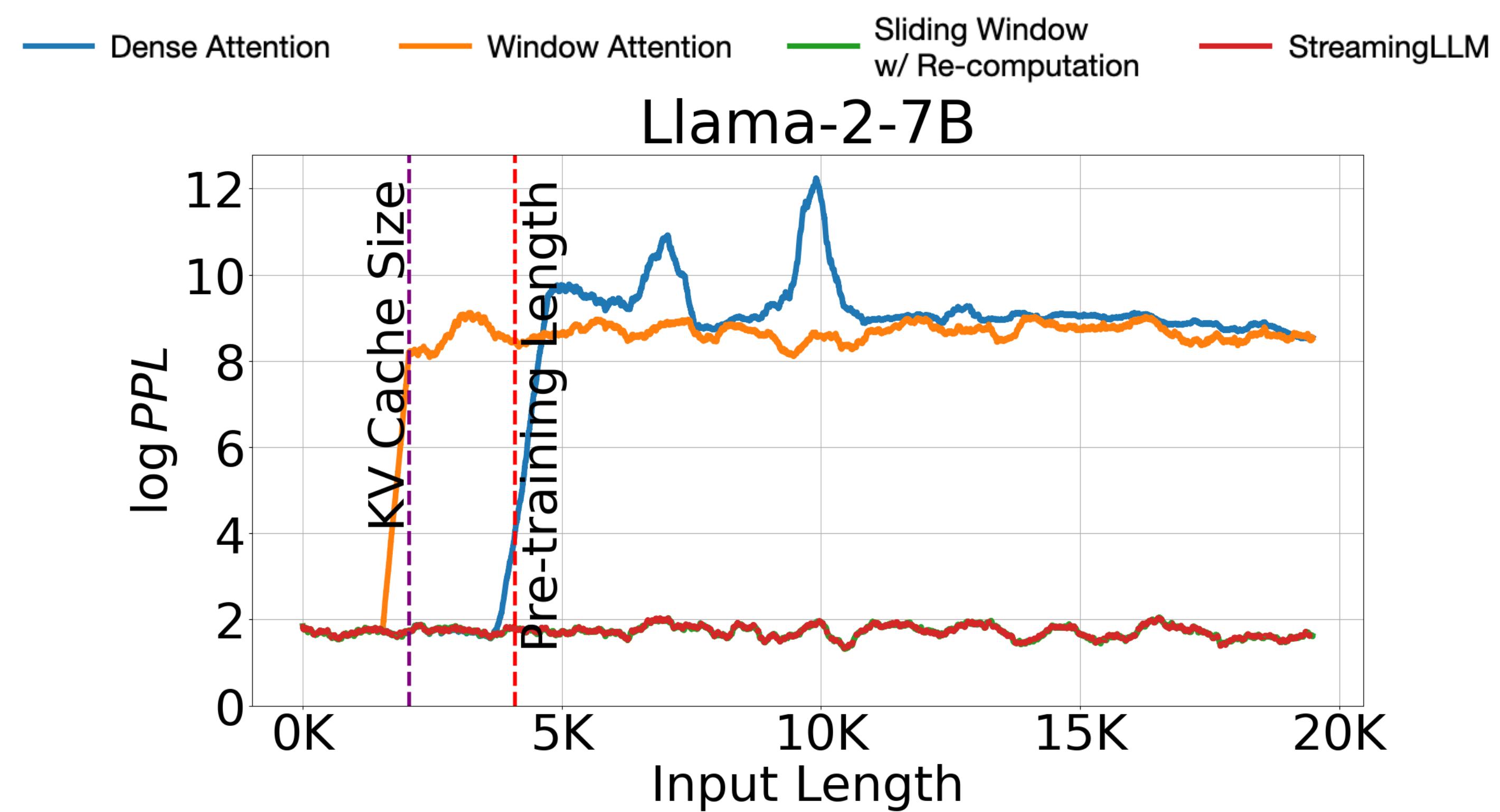
# The Limits of Window Attention

- A natural approach – window attention: caching only the most recent Key-Value states.
- Drawback: model collapses when the text length surpasses the cache size, when the initial token is evicted.

(b) Window Attention



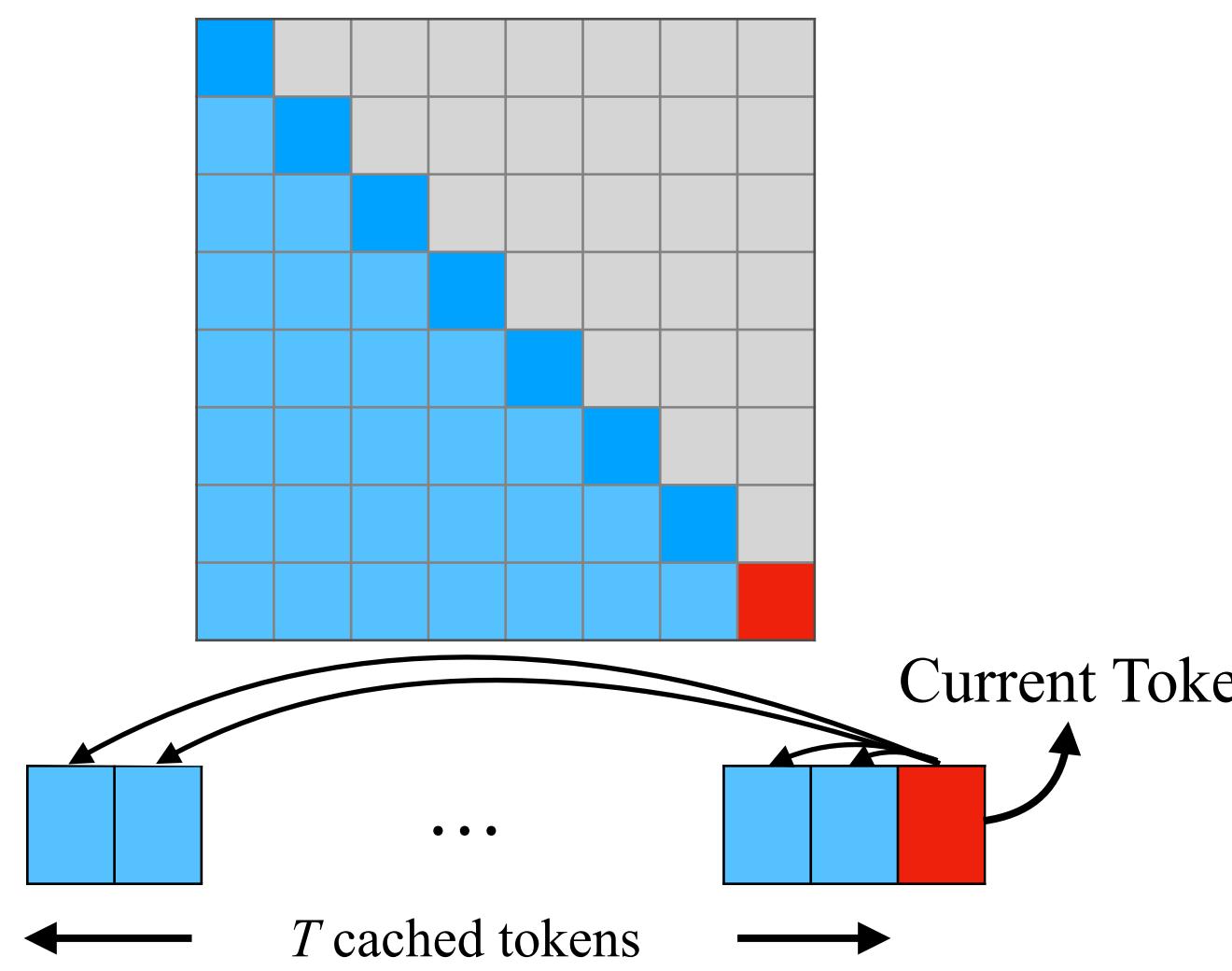
Breaks when initial tokens  
are evicted.



# Difficulties of Other Methods

- A natural approach – window attention: caching only the most recent Key-Value states.
- Drawback: model collapses when the text length surpasses the cache size, when the initial token is evicted.

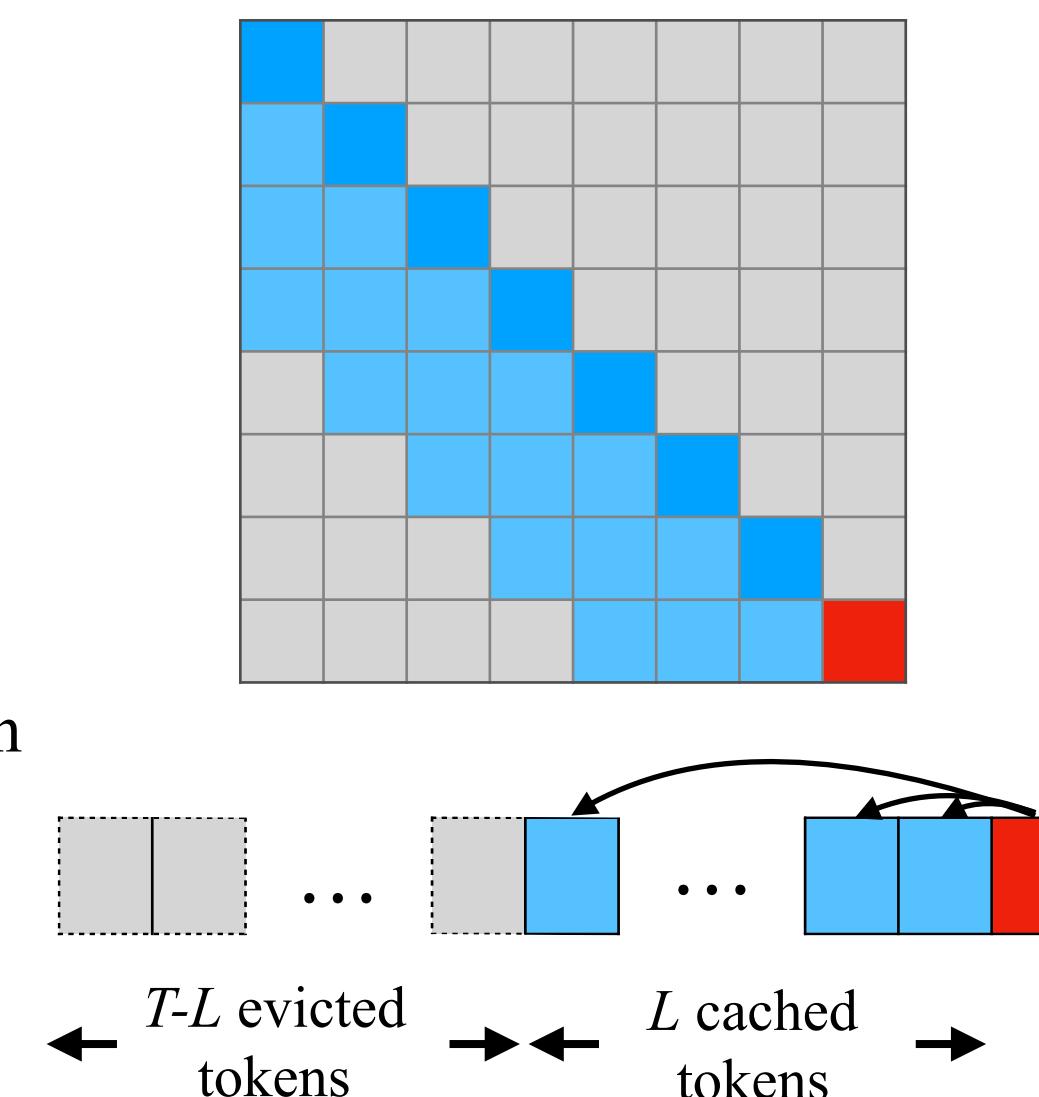
(a) Dense Attention



$O(T^2) \times$  PPL: 5641  $\times$

Has poor efficiency and performance on long text.

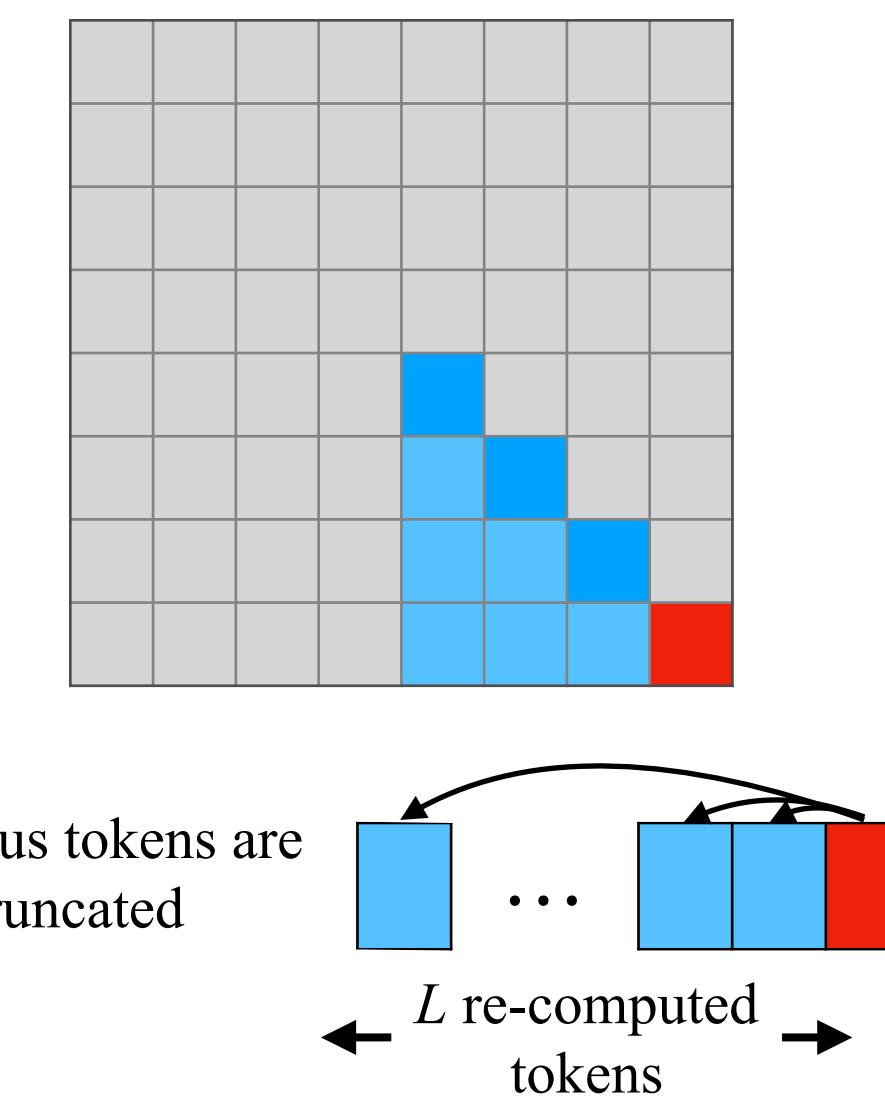
(b) Window Attention



$O(TL) \checkmark$  PPL: 5158  $\times$

Breaks when initial tokens are evicted.

(c) Sliding Window w/ Re-computation



$O(TL^2) \times$  PPL: 5.43  $\checkmark$

Has to re-compute cache for each incoming token.

# The “Attention Sink” Phenomenon

- **Observation:** initial tokens have large attention scores, even if they're not semantically significant.
- **Attention Sink:** Tokens that disproportionately attract attention irrespective of their relevance.

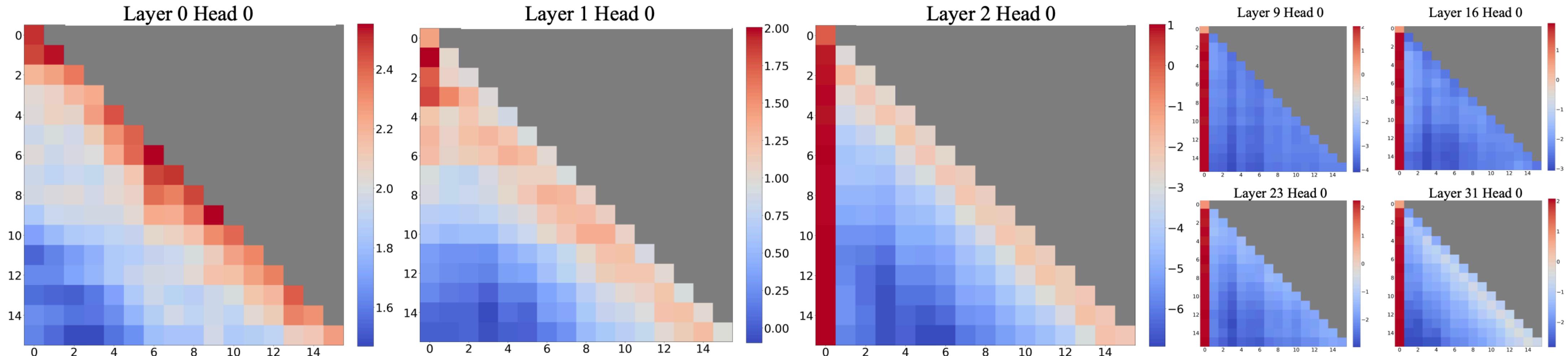


Figure 2: Visualization of the *average* attention logits in Llama-2-7B over 256 sentences, each with a length of 16. Observations include: (1) The attention maps in the first two layers (layers 0 and 1) exhibit the "local" pattern, with recent tokens receiving more attention. (2) Beyond the bottom two layers, the model heavily attends to the initial token across all layers and heads.

$$\text{SoftMax}(x)_i = \frac{e^{x_i}}{e^{x_1} + \sum_{j=2}^N e^{x_j}}, \quad x_1 \gg x_j, j \in 2, \dots, N$$

# The “Attention Sink” Phenomenon

- This phenomenon was observed in the SpAtten paper three years ago:

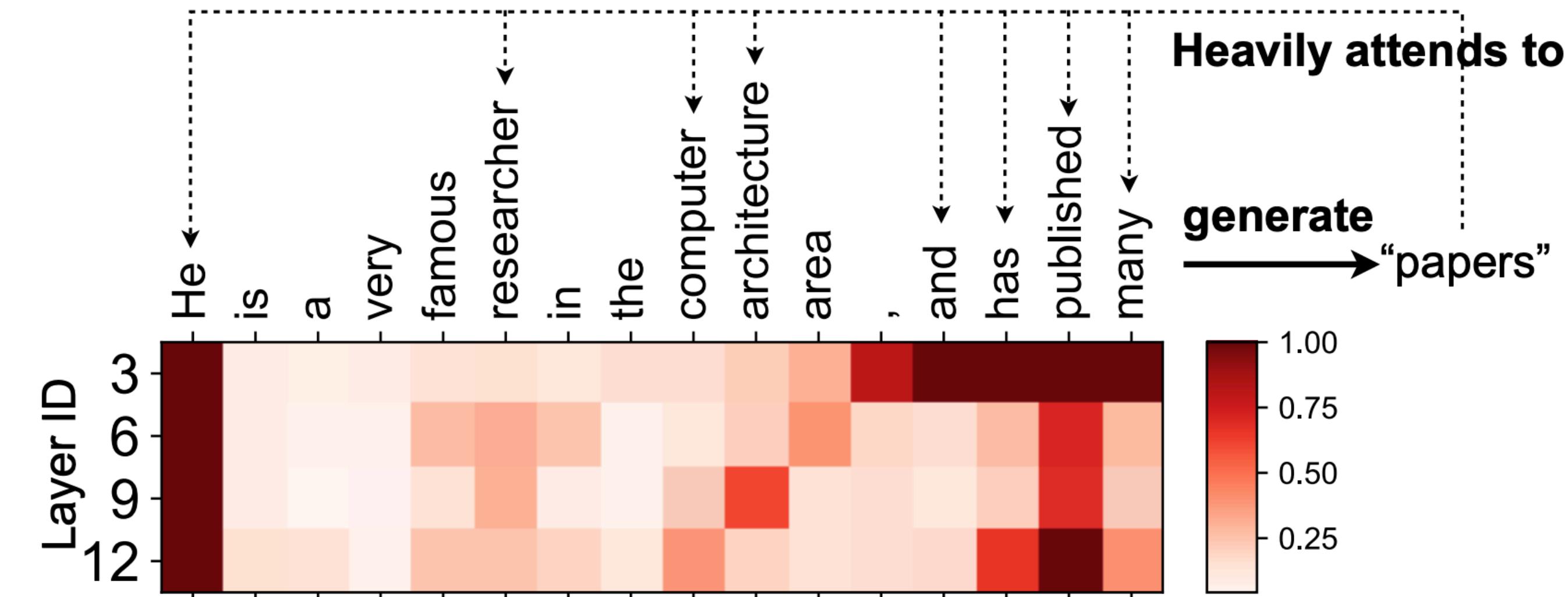


Fig. 23. Cumulative importance scores in GPT-2. Unimportant tokens are pruned on the fly. Important tokens are heavily attended.

GPT-2 for Language Modeling  
Du Fu was a great poet of the Tang dynasty. Recently a variety of styles have been used in efforts to translate the work of Du Fu into English  
Du Fu was a great poet of the Tang dynasty. Recently a variety of styles have been used in efforts to translate the work of Du Fu into English  
Du Fu was a great poet of the Tang dynasty. Recently a variety of styles have been used in efforts to translate the work of Du Fu into English  
  
'English' is the generated token.

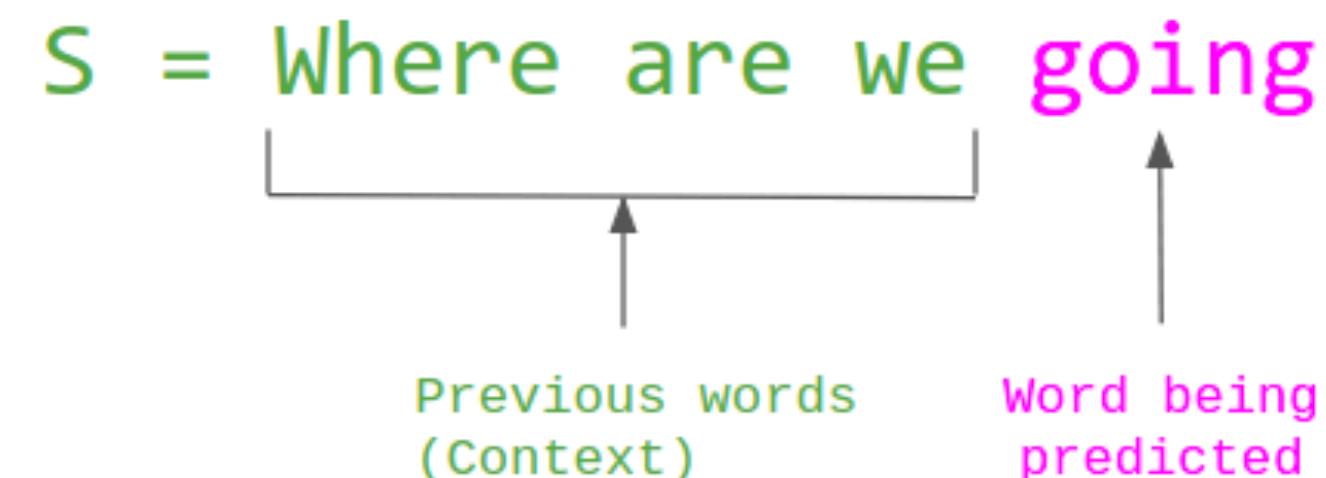
# Understanding Why Attention Sinks Exist

## The Rationale Behind Attention Sinks

- SoftMax operation's role in creating attention sinks – attention scores have to sum up to one for all contextual tokens.
- Does the importance of the initial tokens arise from their **position** or their **semantics**?

$$\text{SoftMax}(x)_i = \frac{e^{x_i}}{e^{x_1} + \sum_{j=2}^N e^{x_j}}, \quad x_1 \gg x_j, j \in 2, \dots, N$$

- Initial tokens' advantage in becoming sinks due to their visibility to subsequent tokens, rooted in autoregressive language modeling.



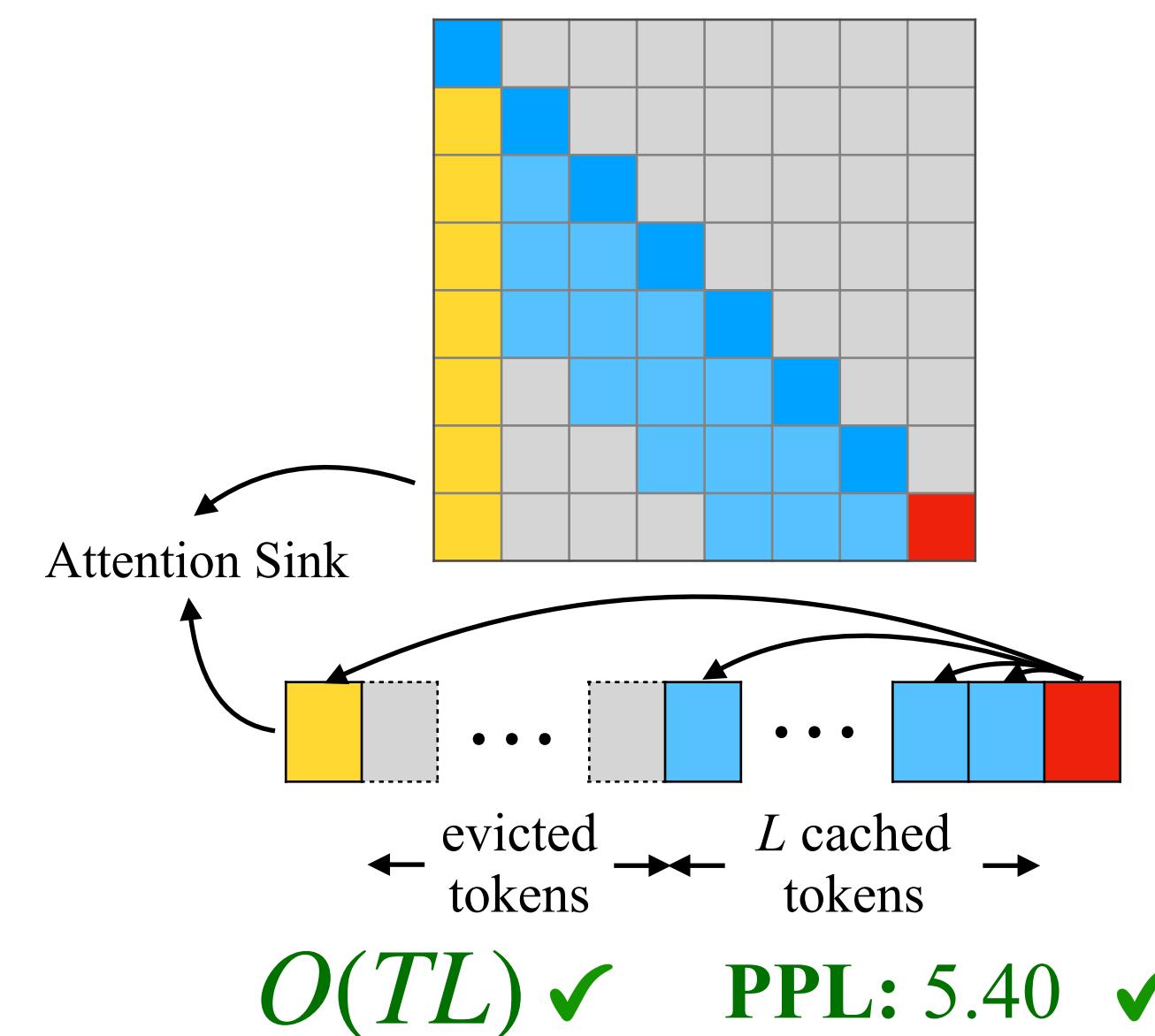
$$P(S) = P(\text{Where}) \times P(\text{are} \mid \text{Where}) \times P(\text{we} \mid \text{Where are}) \times P(\text{going} \mid \text{Where are we})$$

Llama-2-13B	PPL ( $\downarrow$ )
0 + 1024 (Window)	5158.07
4 + 1020	5.40
4"\n"+1020	5.60

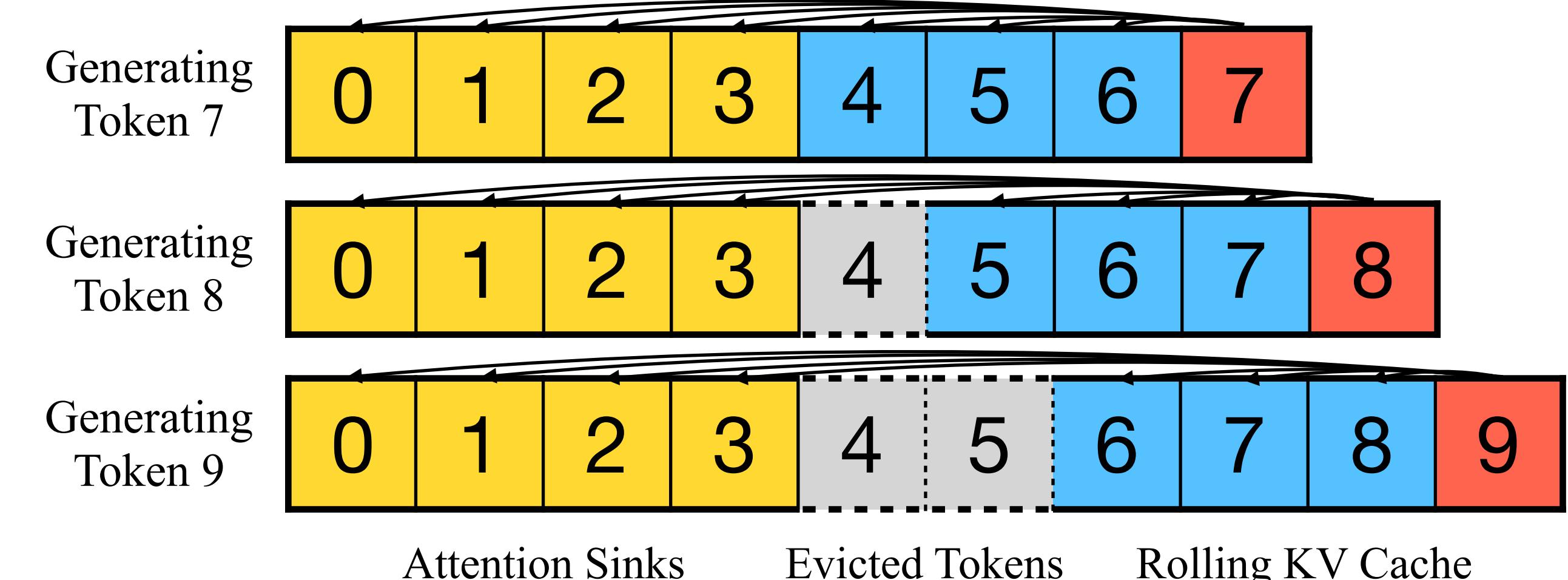
# StreamingLLM: Using Attention Sinks for Infinite Streams

- **Objective:** Enable LLMs trained with a finite attention window to handle infinite text lengths without additional training.
- **Key Idea:** **preserve the KV of attention sink tokens**, along with the sliding window's KV to stabilize the model's behavior.

(d) StreamingLLM (ours)

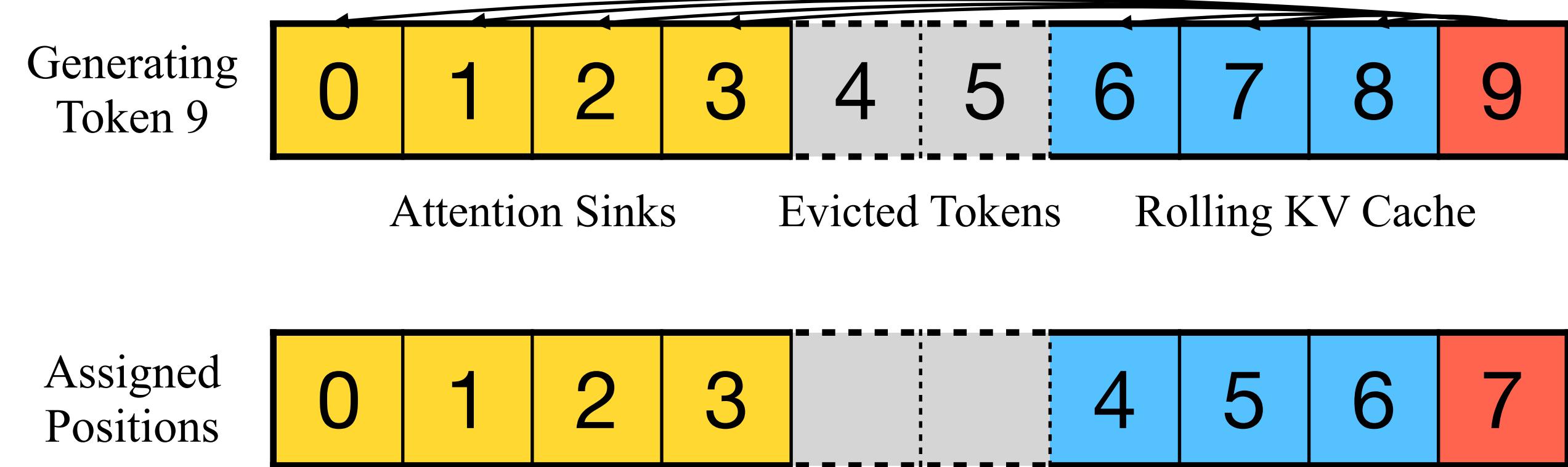


Can perform efficient and stable language modeling on long texts.



# Positional Encoding Assignment

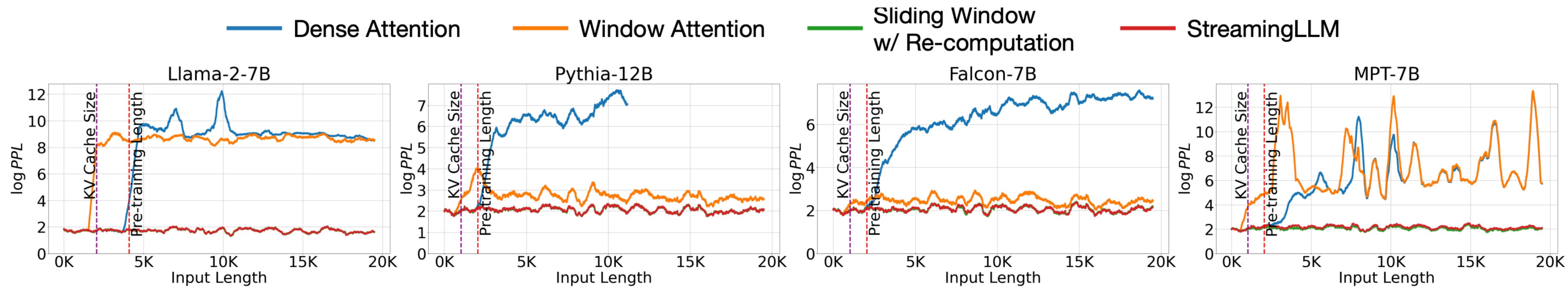
- Use positions *in the cache* instead of those *in the original text*.



- How to implement StreamingLLM with Paged Attention?

# Streaming Performance

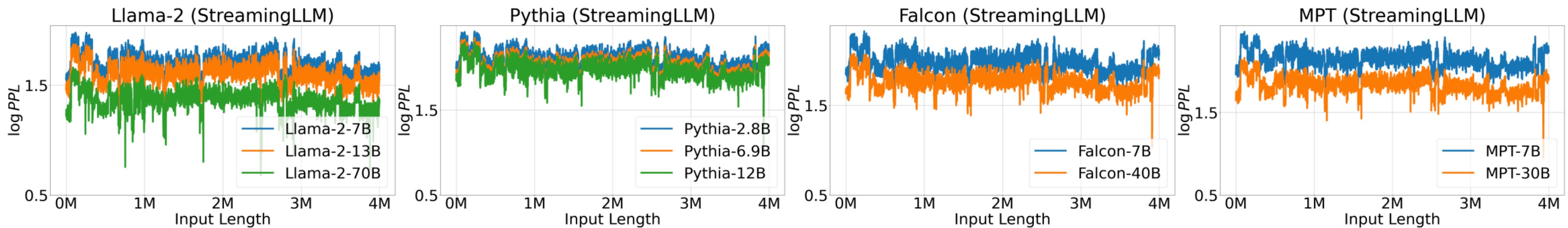
- Comparison between dense attention, window attention, and sliding window w/ re-computation.



- Dense attention fails beyond pre-training attention window size.
- Window attention fails after input exceeds cache size (initial tokens evicted).
- StreamingLLM shows stable performance; perplexity close to sliding window with re-computation baseline.

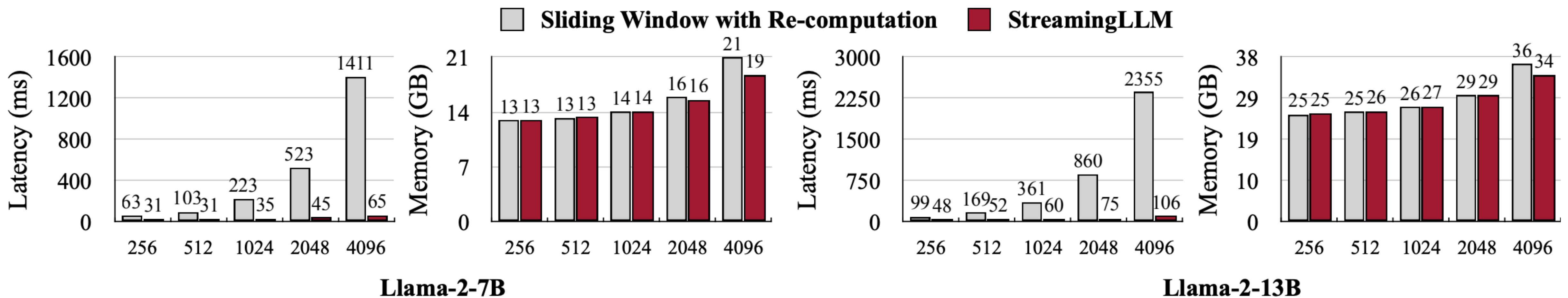
# Streaming Performance

- With StreamingLLM, model families include Llama-2, MPT, Falcon, and Pythia can now effectively model up to 4 million tokens.



# Efficiency

- **Comparison baseline:** The sliding window with re-computation, a method that is computationally heavy due to quadratic attention computation within its window.
- StreamingLLM provides up to 22.2x speedup over the baseline, making LLMs for real-time streaming applications feasible.



# Ablation Study: #Attention Sinks

- The number of attention sinks that need to be introduced to recover perplexity.
  - 4 attention sinks are generally enough.

Cache Config	0+2048	1+2047	2+2046	4+2044	8+2040
Falcon-7B	17.90	12.12	12.12	12.12	12.12
MPT-7B	460.29	14.99	15.00	14.99	14.98
Pythia-12B	21.62	11.95	12.09	12.09	12.02

Cache Config	0+4096	1+4095	2+4094	4+4092	8+4088
Llama-2-7B	3359.95	11.88	10.51	9.59	9.54

# Pre-training with a Dedicated Attention Sink Token

- **Idea: Why 4 attention sinks?** Can we train a LLM that need only one single attention sink? **Yes!**
- **Method:** Introduce an extra learnable token at the start of all training samples to act as a dedicated attention sink.
- **Result:** This pre-trained model retains performance in streaming cases with just this single sink token, contrasting with vanilla models that require multiple initial tokens.

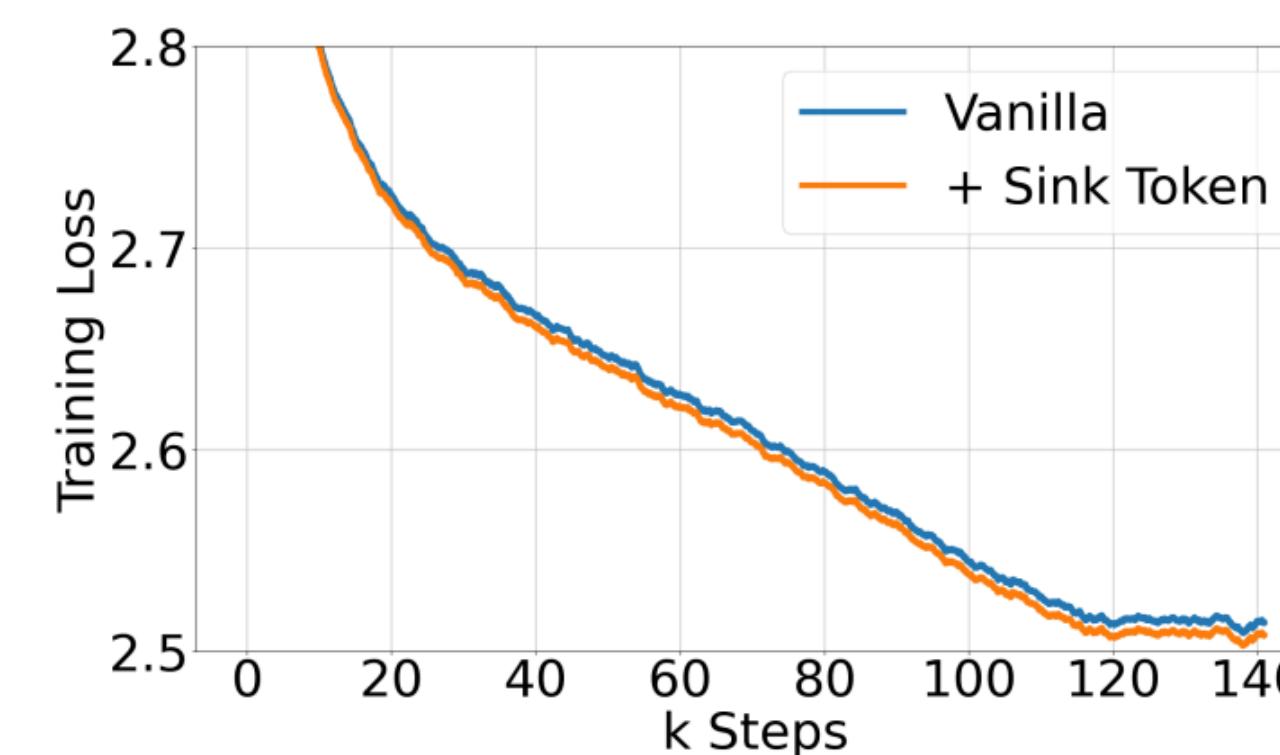


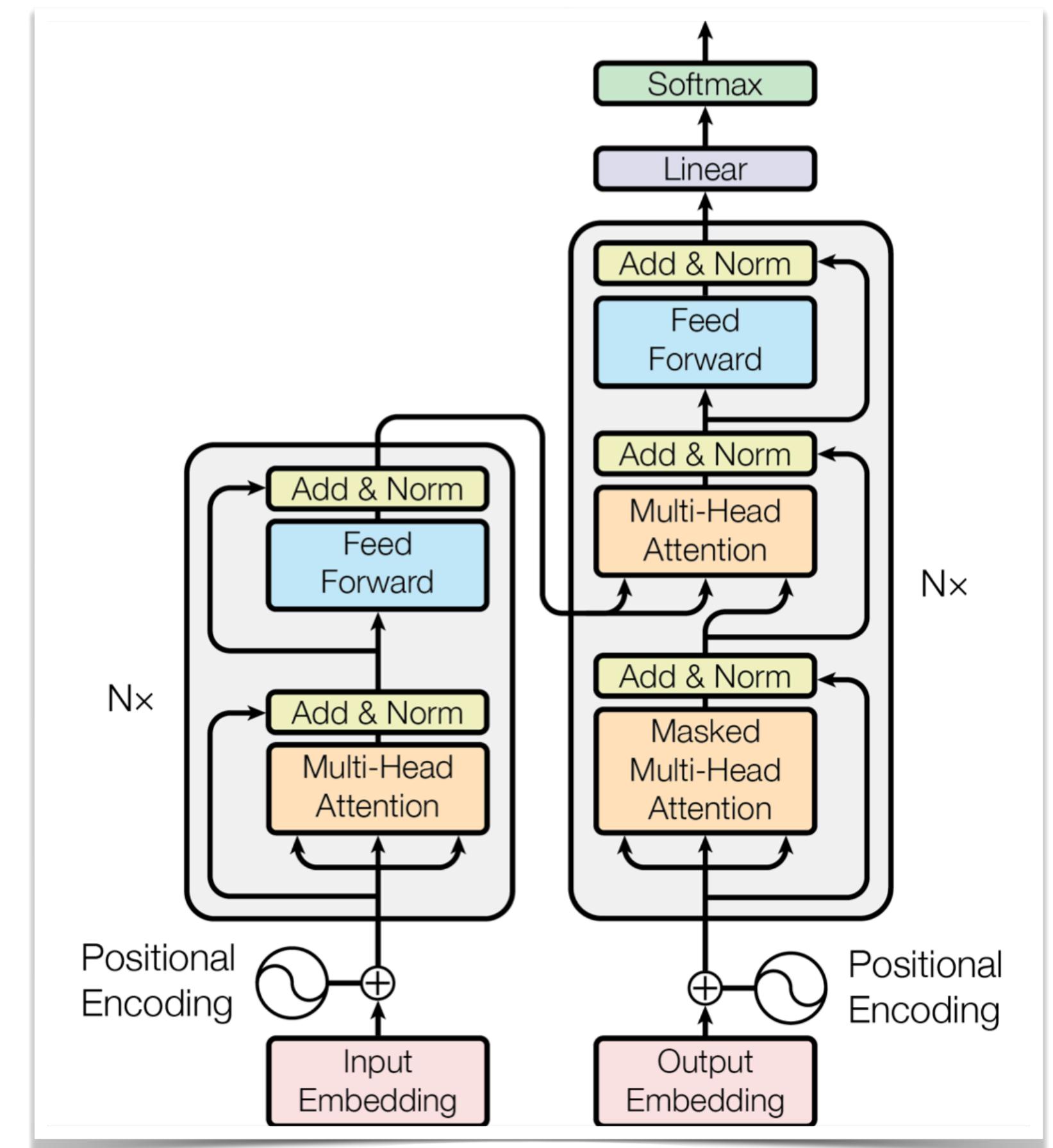
Figure 6: Pre-training loss curves of models w/ and w/o sink tokens. Two models have a similar convergence trend.

Cache Config	0+1024	1+1023	2+1022	4+1020
Vanilla	27.87	18.49	18.05	18.05
Zero Sink	29214	19.90	18.27	18.01
Learnable Sink	1235	18.01	18.01	18.02

# Lecture Plan

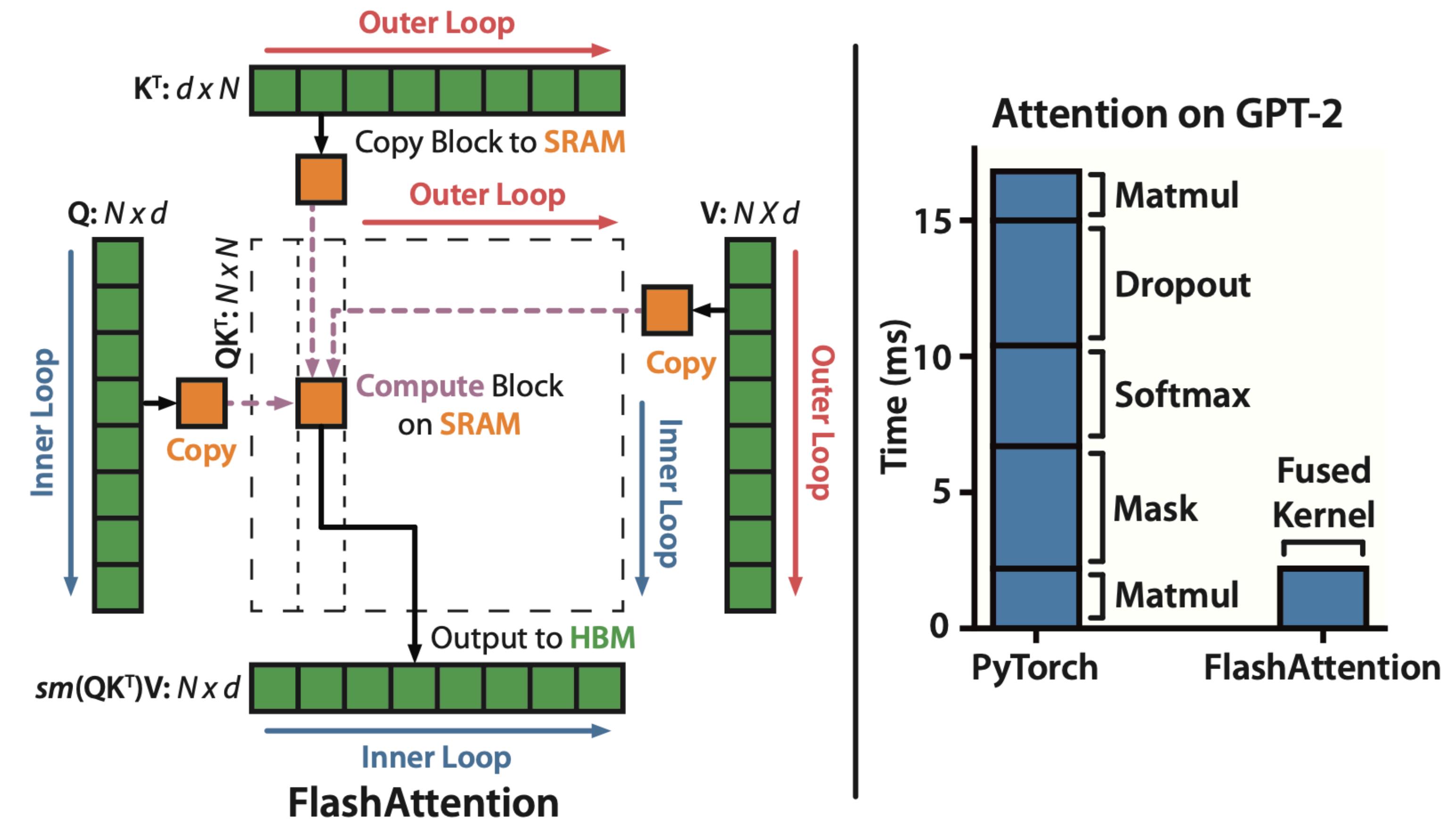
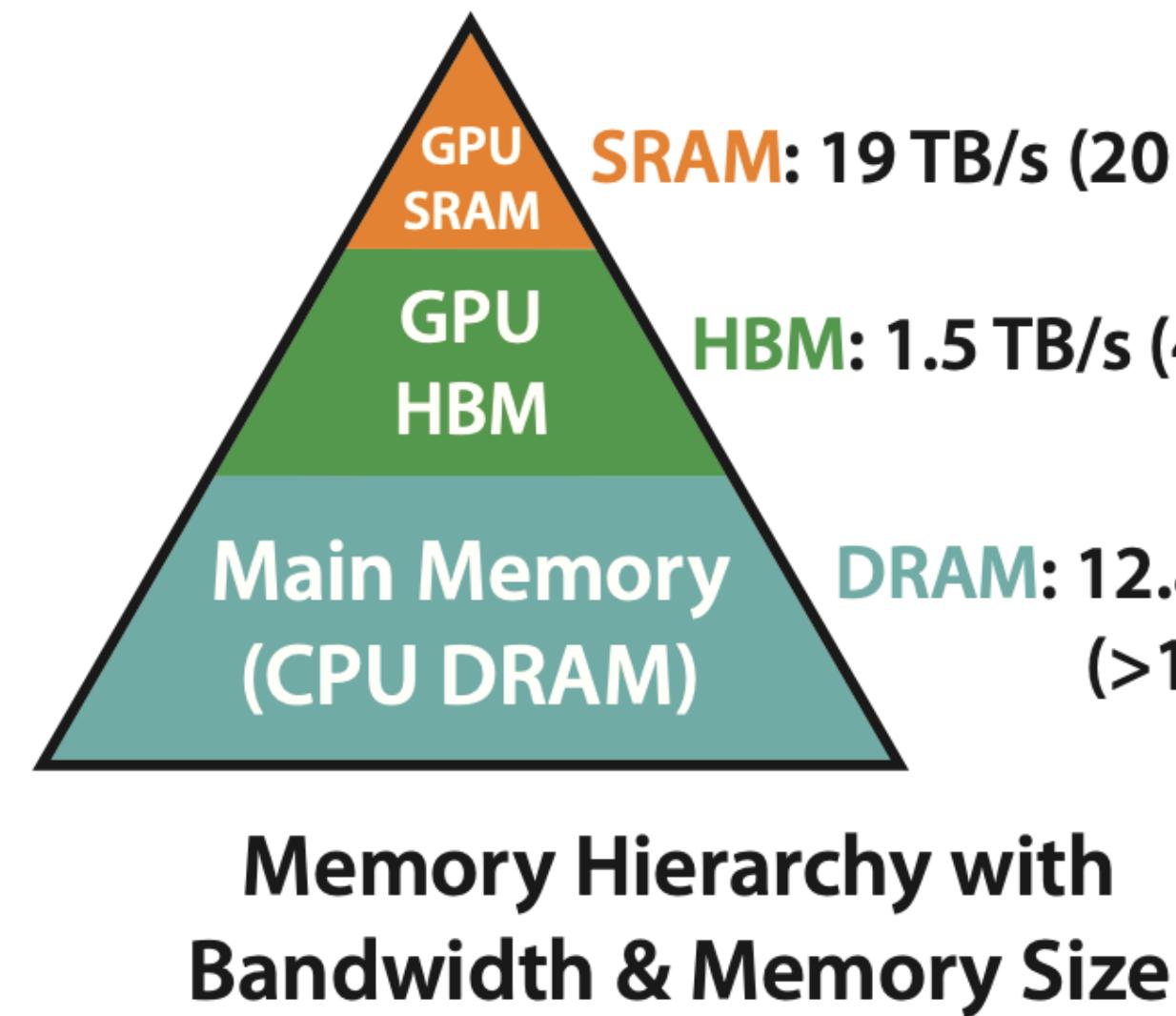
Today, we will cover:

1. Efficient inference algorithms for LLMs
  1. Quantization
  2. Pruning/sparsity
2. Efficient inference systems for LLMs
  1. vLLM
  2. StreamingLLM
  3. FlashAttention
  4. Speculative decoding
3. Efficient fine-tuning for LLMs
  1. LoRA/QLoRA
  2. Adapter
  3. Prompt Tuning



# FlashAttention

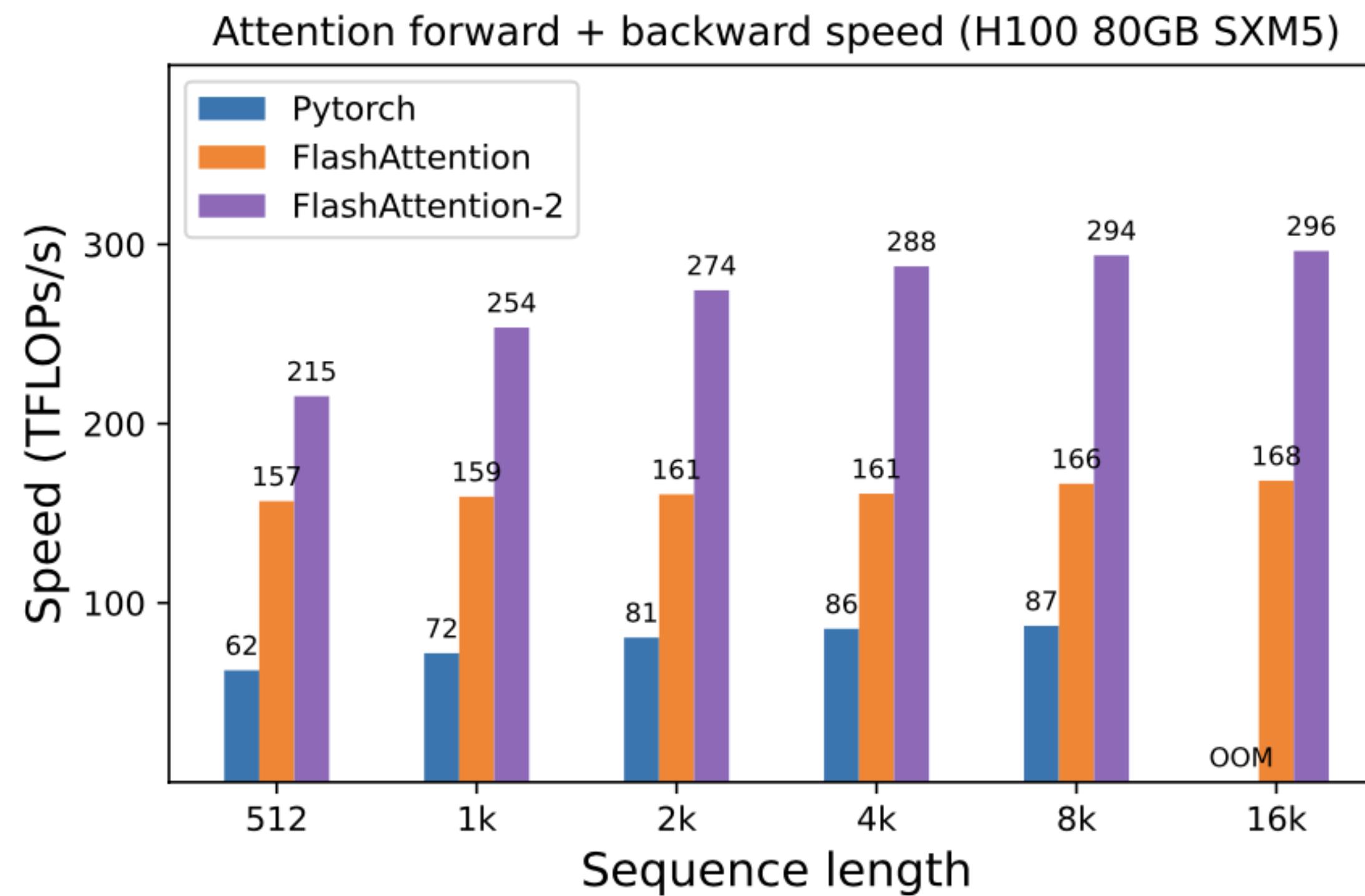
- Use tiling to prevent the materialization of the large  $N \times N$  attention matrix, thus avoid using the slow HBM; kernel fusion.



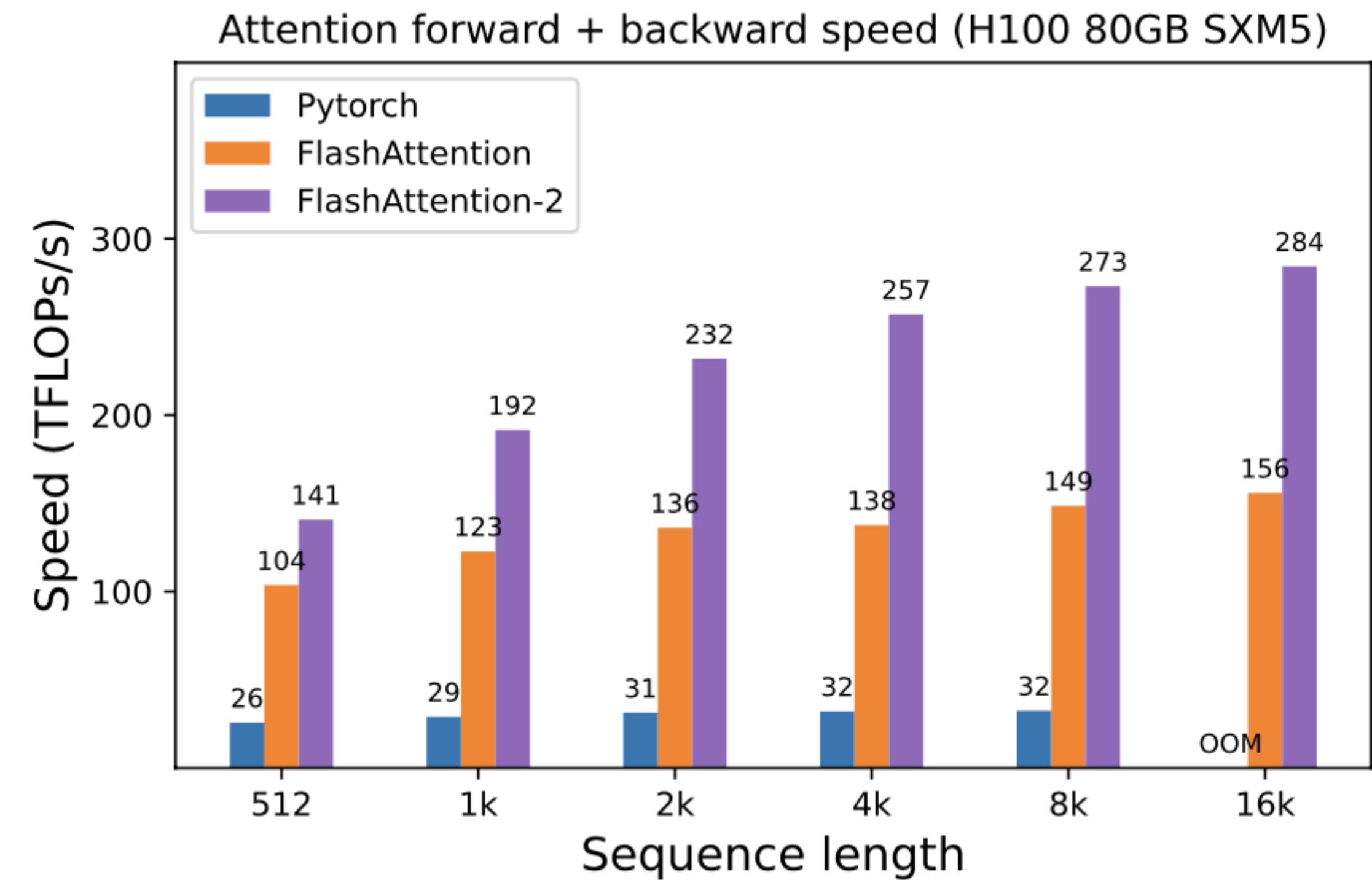
FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness [Dao et al., 2022]

# FlashAttention

- Training acceleration results on NVIDIA H100



(a) Without causal mask, head dimension 64



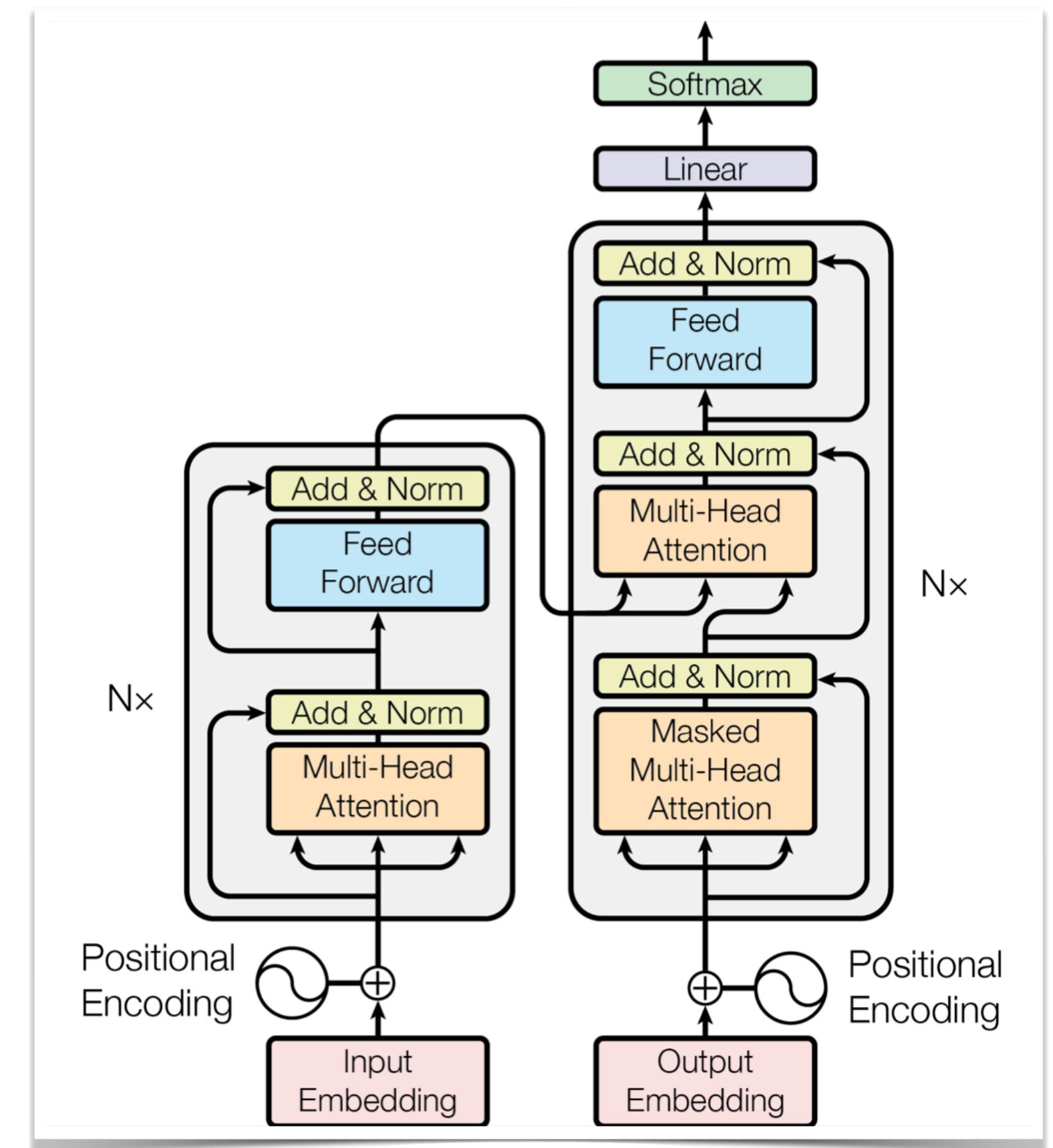
(b) With causal mask, head dimension 64

FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness [Dao et al., 2022]

# Lecture Plan

Today, we will cover:

1. Efficient inference algorithms for LLMs
  1. Quantization
  2. Pruning/sparsity
2. Efficient inference systems for LLMs
  1. vLLM
  2. StreamingLLM
  3. FlashAttention
  4. Speculative decoding
3. Efficient fine-tuning for LLMs
  1. LoRA/QLoRA
  2. Adapter
  3. Prompt Tuning



# Speculative Decoding

## Accelerating memory-bounded generation

- The decoding phase of LLM generates outputs token by token, which is highly memory-bounded (especially at a small batch size)
- There are two models in speculative decoding:
  - *Draft model*: a small LLM (e.g., 7B)
  - *Target model*: a large LLM (e.g., 175B, the one we are trying to accelerate)
- Procedure:
  - The draft model decodes  $K$  tokens **autoregressively**
  - Feed the  $K$  generated tokens **in parallel** into the target model and get the predicted probabilities on each location
  - Decide if we want to keep the  $K$  tokens or reject them

# Speculative Decoding

## Accelerating memory-bounded generation

- From generator to verifier

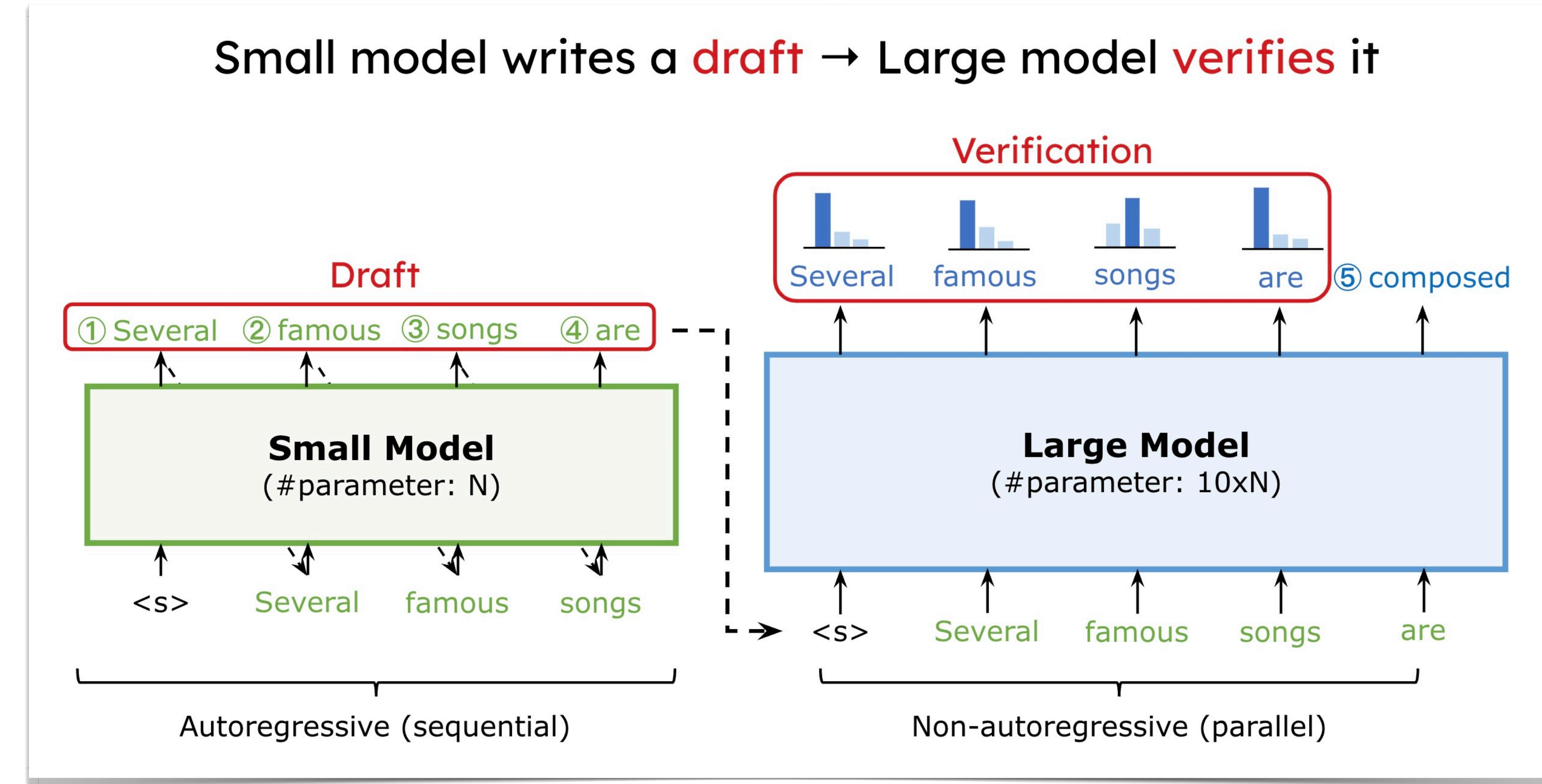


Image credit: vLLM First SF Meetup Slides

# Speculative Decoding

## Accelerating memory-bounded generation

- The decoding phase of LLM generates outputs token by token, which is highly memory-bounded (especially at a small batch size)
- Since multiple tokens are fed to the target model in parallel, it lifts the memory bottleneck
- An example from speculative decoding
  - green: accepted, red: rejected, blue: correction
- 2-3x speed up with **identical** output

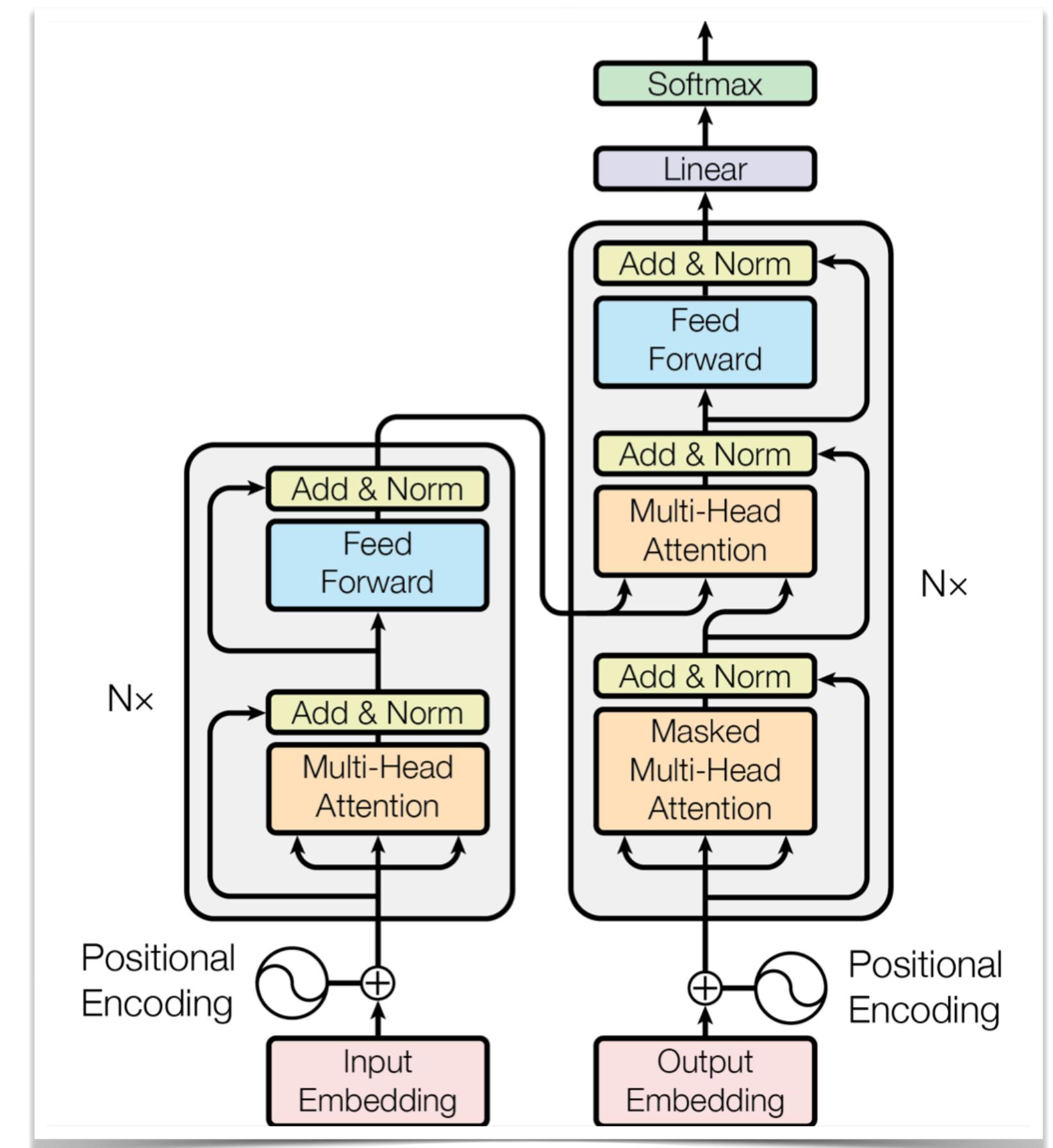
```
[START] japan : s benchmark bond n
[START] japan : s benchmark nikkei 22 75
[START] japan : s benchmark nikkei 225 index rose 22 76
[START] japan : s benchmark nikkei 225 index rose 226 . 69 7 points
[START] japan : s benchmark nikkei 225 index rose 226 . 69 points , or 0 1
[START] japan : s benchmark nikkei 225 index rose 226 . 69 points , or 1 . 5 percent , to 10 , 9859
[START] japan : s benchmark nikkei 225 index rose 226 . 69 points , or 1 . 5 percent , to 10 , 989 . 79 7 in
[START] japan : s benchmark nikkei 225 index rose 226 . 69 points , or 1 . 5 percent , to 10 , 989 . 79 in tokyo late
[START] japan : s benchmark nikkei 225 index rose 226 . 69 points , or 1 . 5 percent , to 10 , 989 . 79 in late morning trading . [END]
```

Leviathan et al., Fast Inference from Transformers via Speculative Decoding, 2023

# Lecture Plan

Today, we will cover:

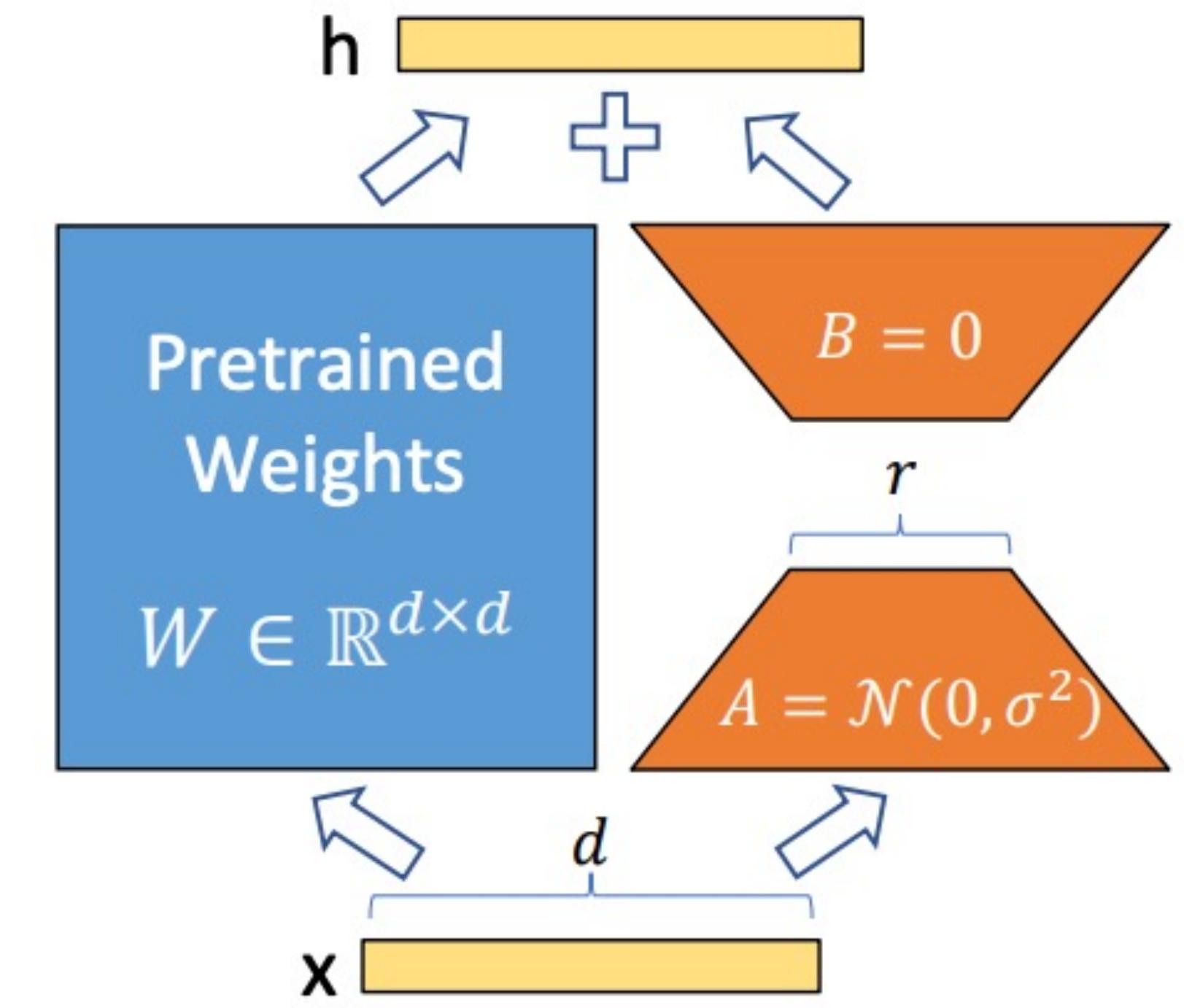
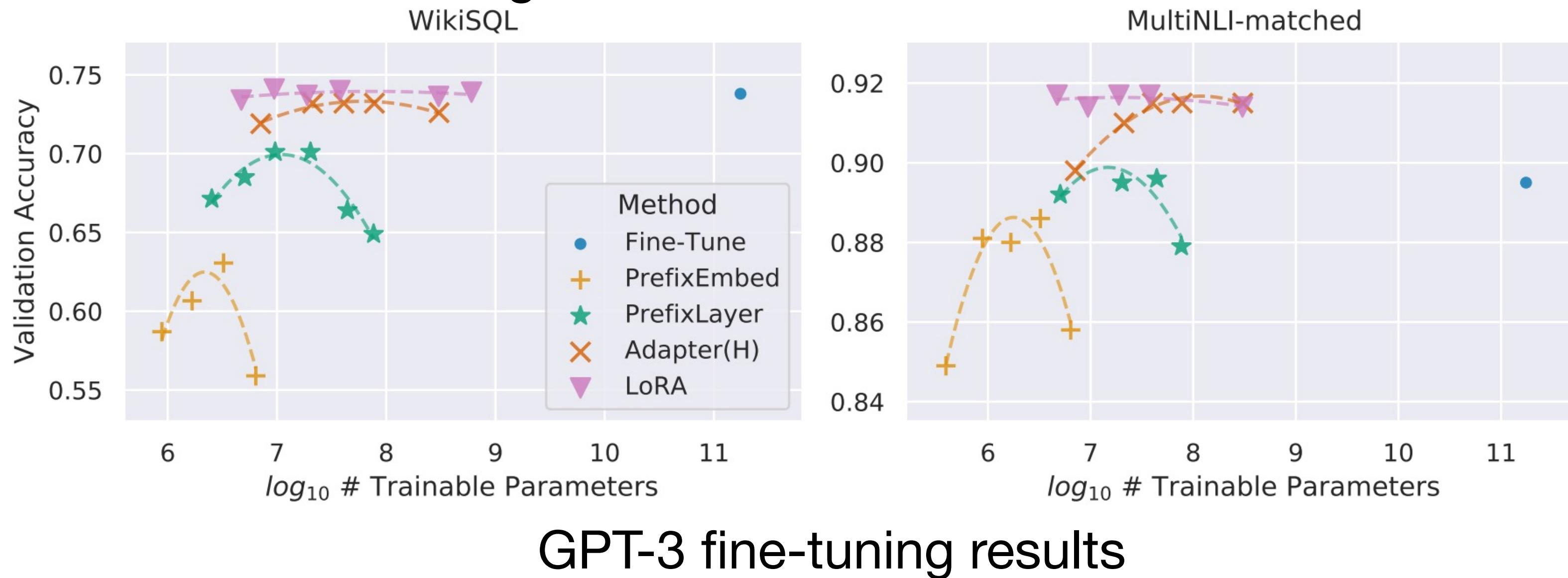
1. Efficient inference algorithms for LLMs
  1. Quantization
  2. Pruning/sparsity
2. Efficient inference systems for LLMs
  1. vLLM
  2. StreamingLLM
  3. FlashAttention
  4. Speculative decoding
3. Efficient fine-tuning for LLMs
  1. LoRA/QLoRA
  2. Adapter
  3. Prompt Tuning



# LoRA

## Low-rank adaptation of large language models/diffusion models

- Instead of updating the full model weights, update a small low-rank component
- Benefits:
  - Speed up fine-tuning by skipping gradient computation
  - Save fine-tuning memory by reducing optimizer states
  - Prevent catastrophic forgetting
  - Low-rank weights can be fused

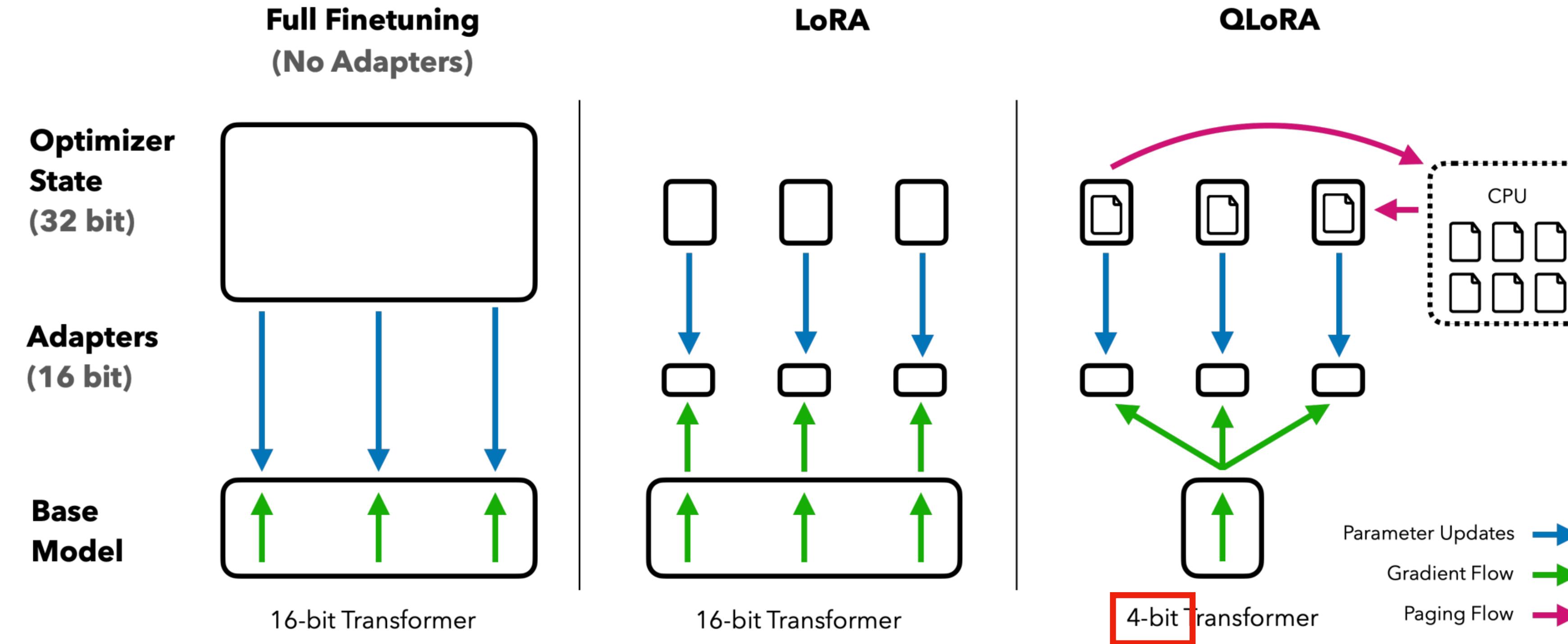


LoRA: Low-Rank Adaptation of Large Language Models (Hu et al., 2021)

# QLoRA

## LoRA with quantized base model weights

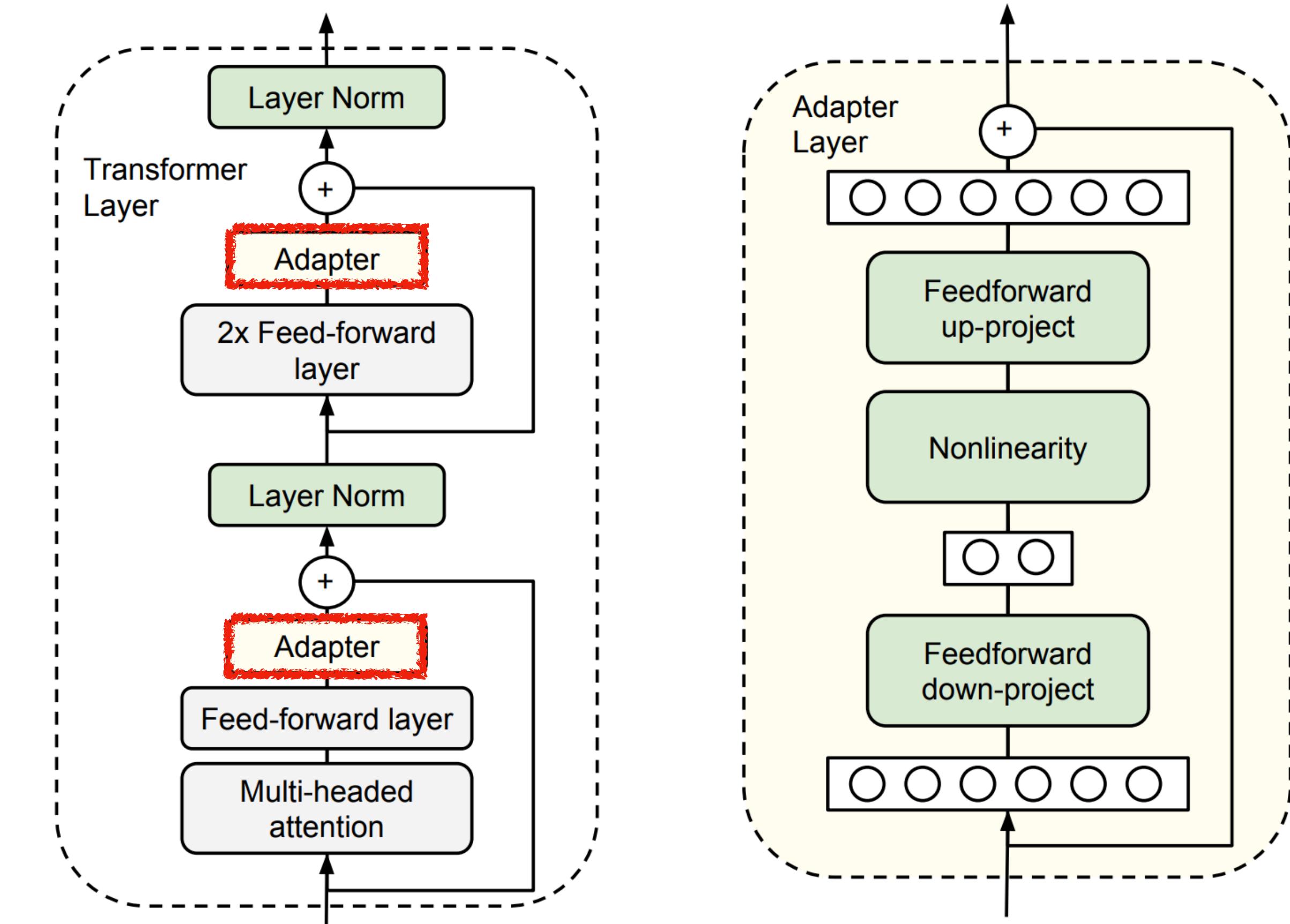
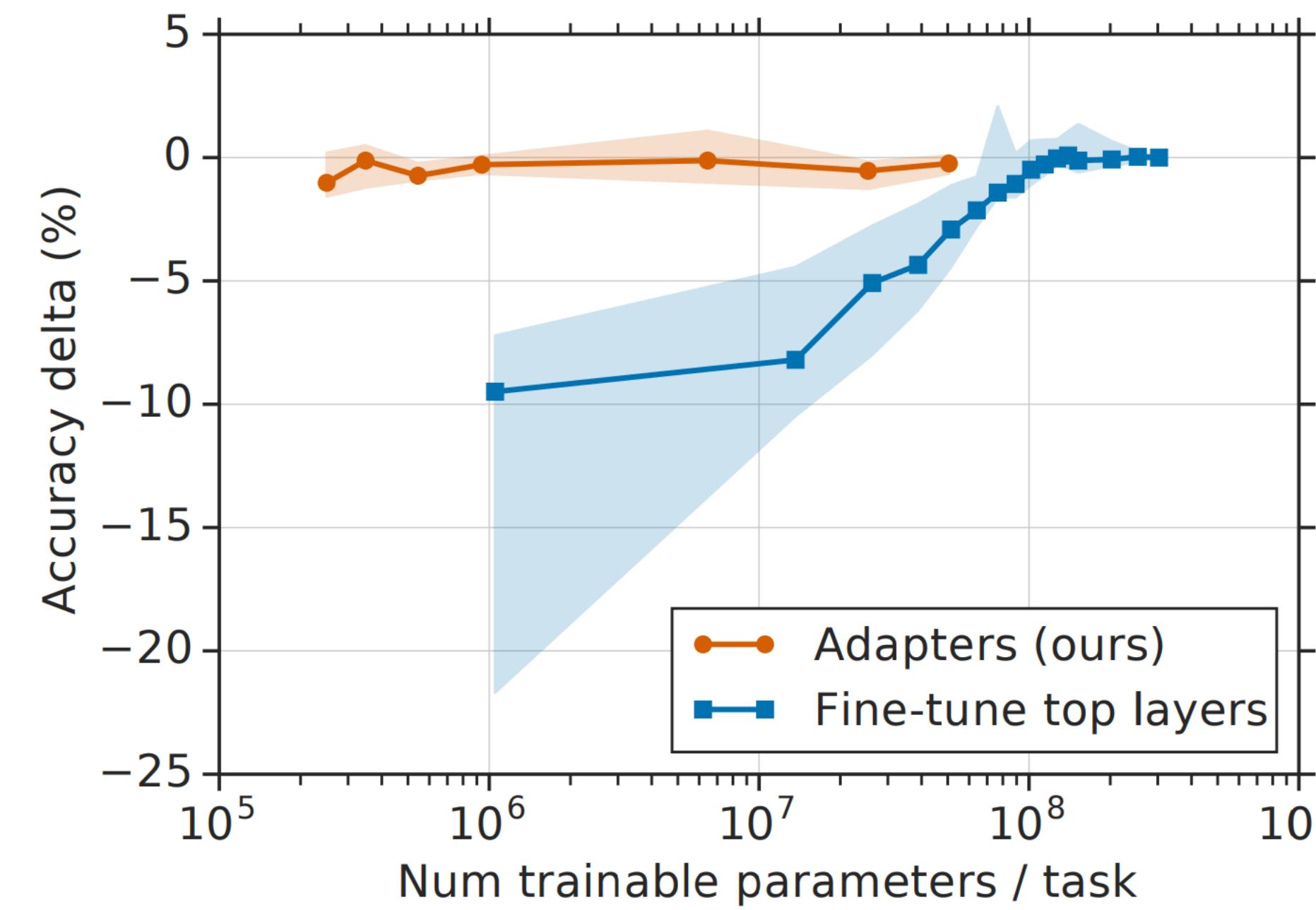
- NormalFloat (NF4), a new data type for LLM weight quantization
- Double quantization (apply quantization on scaling factors) to further reduce base model size
- Paged Optimizers with CPU offloading



QLORA: Efficient Finetuning of Quantized LLMs (Dettmers et al., 2023)

# Adapter

- Smaller adapter for transfer learning
- Cannot be fused

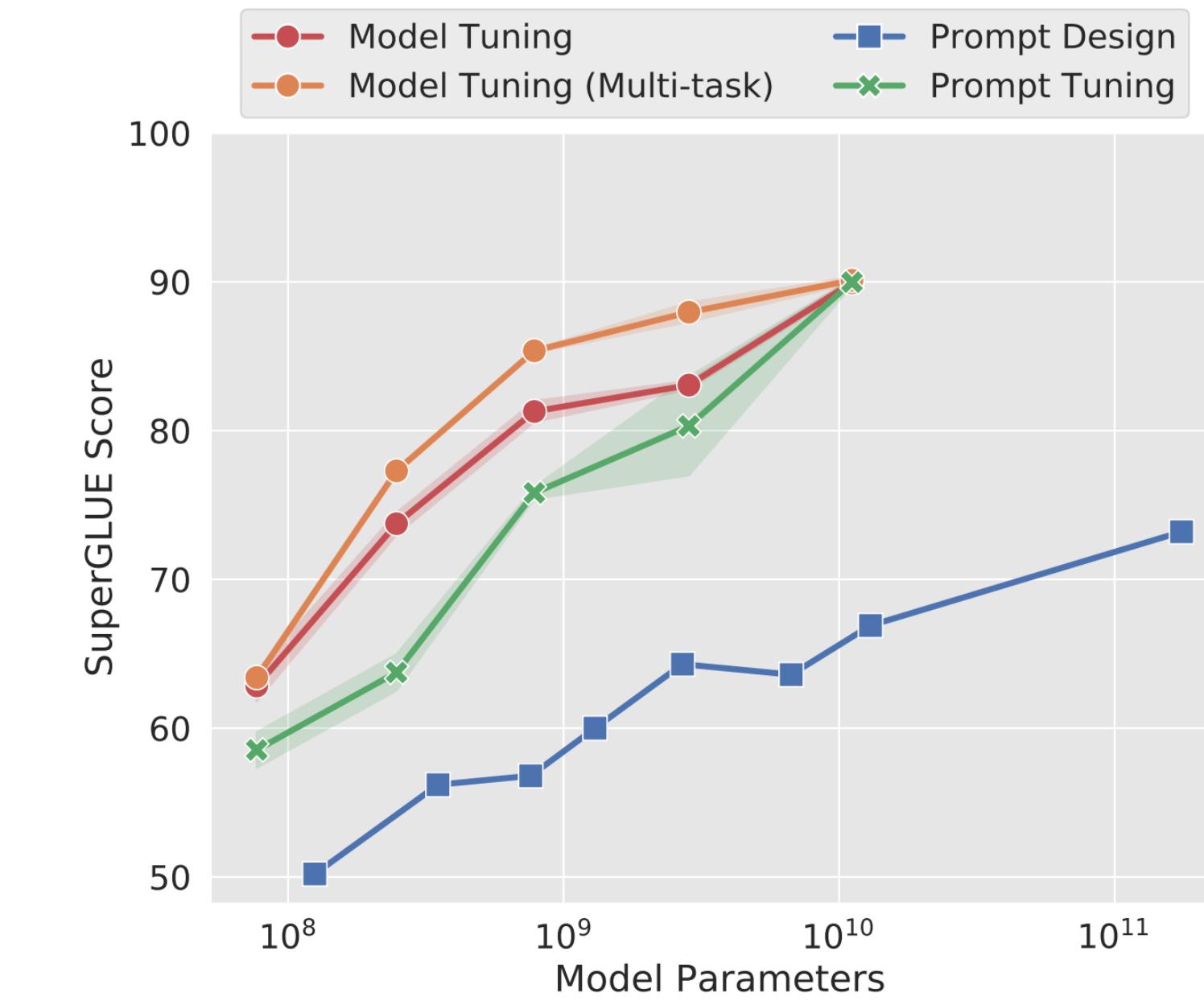
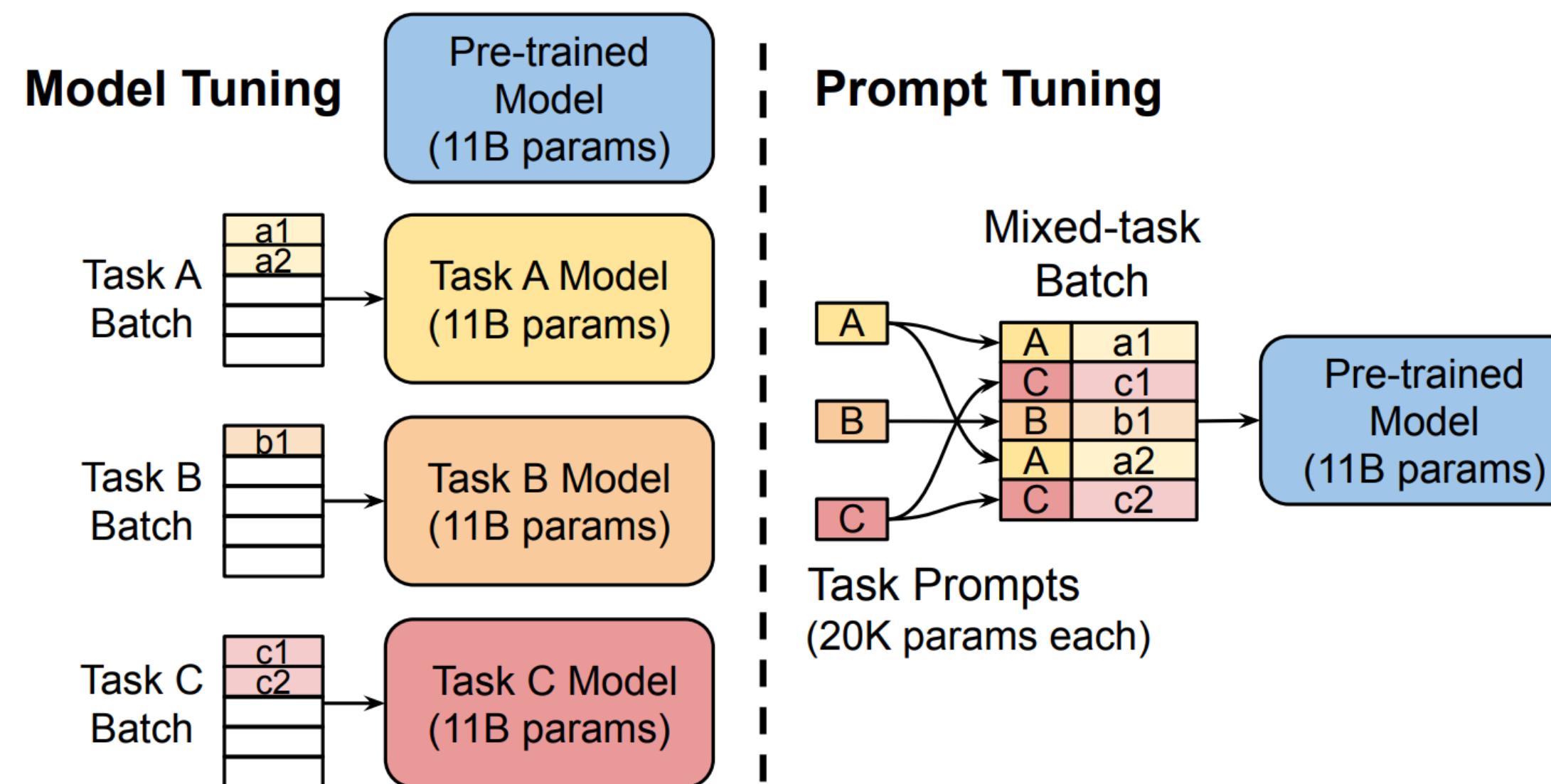


Parameter-Efficient Transfer Learning for NLP (Houlsby et al., 2019)

# Prompt Tuning

## From discrete prompt to continuous prompt

- Prompt engineering can help LLM work on different downstream applications
  - E.g., “Please summarize the following text: ”
- We can train a continuous prompt that is prepended to inputs for each task
- We can **mix different learned prompts in a single batch**
- Comparable accuracy as fine-tuning as the model gets larger.



The Power of Scale for Parameter-Efficient Prompt Tuning (Lester et al., 2021)

# TensorRT-LLM Public Release

## State-of-the-art LLM serving infra from NVIDIA

### Key Features

TensorRT-LLM contains examples that implement the following features.

- Multi-head Attention([MHA](#))
- Multi-query Attention ([MQA](#))
- Group-query Attention([GQA](#))
- In-flight Batching
- Paged KV Cache for the Attention
- Tensor Parallelism
- Pipeline Parallelism

- INT4/INT8 Weight-Only Quantization (W4A16 & W8A16)

- [SmoothQuant](#)
- [GPTQ](#)
- [AWQ](#)
- [FP8](#)

- Greedy-search
- Beam-search
- RoPE

## TensorRT-LLM

### A TensorRT Toolbox for Large Language Models

[docs](#) [latest](#) [python 3.10.12](#) [cuda 12.2](#) [TRT 9.1](#) [release 0.5.0](#) [license Apache 2](#)

[Architecture](#) | [Results](#) | [Examples](#) | [Documentations](#)

 covered in the lecture

 will introduce in future lectures

<https://github.com/NVIDIA/TensorRT-LLM>

# Summary of Today's Lecture

## 1. Efficient inference algorithms for LLMs

1. Quantization: SmoothQuant, AWQ, TinyChat
2. Pruning/sparsity: SpAtten, H2O, MoE

## 2. Efficient inference systems for LLMs

1. vLLM
2. StreamingLLM
3. FlashAttention
4. Speculative decoding

## 3. Efficient fine-tuning for LLMs

1. LoRA/QLoRA
2. Adapter
3. Prompt Tuning

