Microsoft

LeanAttention

Hardware-Aware Scalable Exact Attention Mechanism

Rya Sanovar, Srikant Bharadwaj, Renee St. Amant, Victor Rühle, Saravan Rajmohan

Why is attention execution important?

*Support **long contexts** >1M tokens*

***Ragged Batches:** Batch requests of **unequal** context lengths*

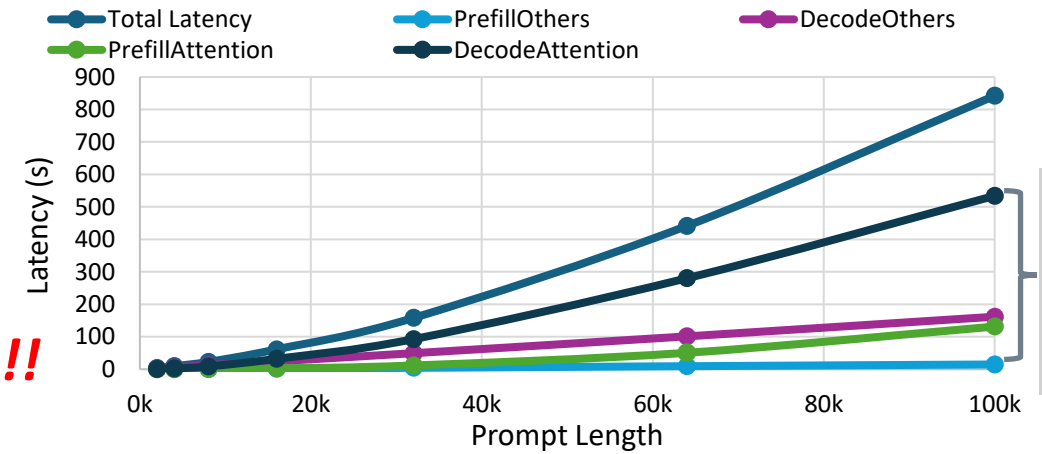
Grok Claude

Attention

GPU

Very slow !!

Improves model utility and delivers throughput



*Decode Attention takes up **50-60% of total processing time !***

8:1 token ratio on Phi-3 Medium. Measured on ONNXRuntime.

Why? SOTA **attention mechanisms** have **low hardware occupancy** for **these workloads**.

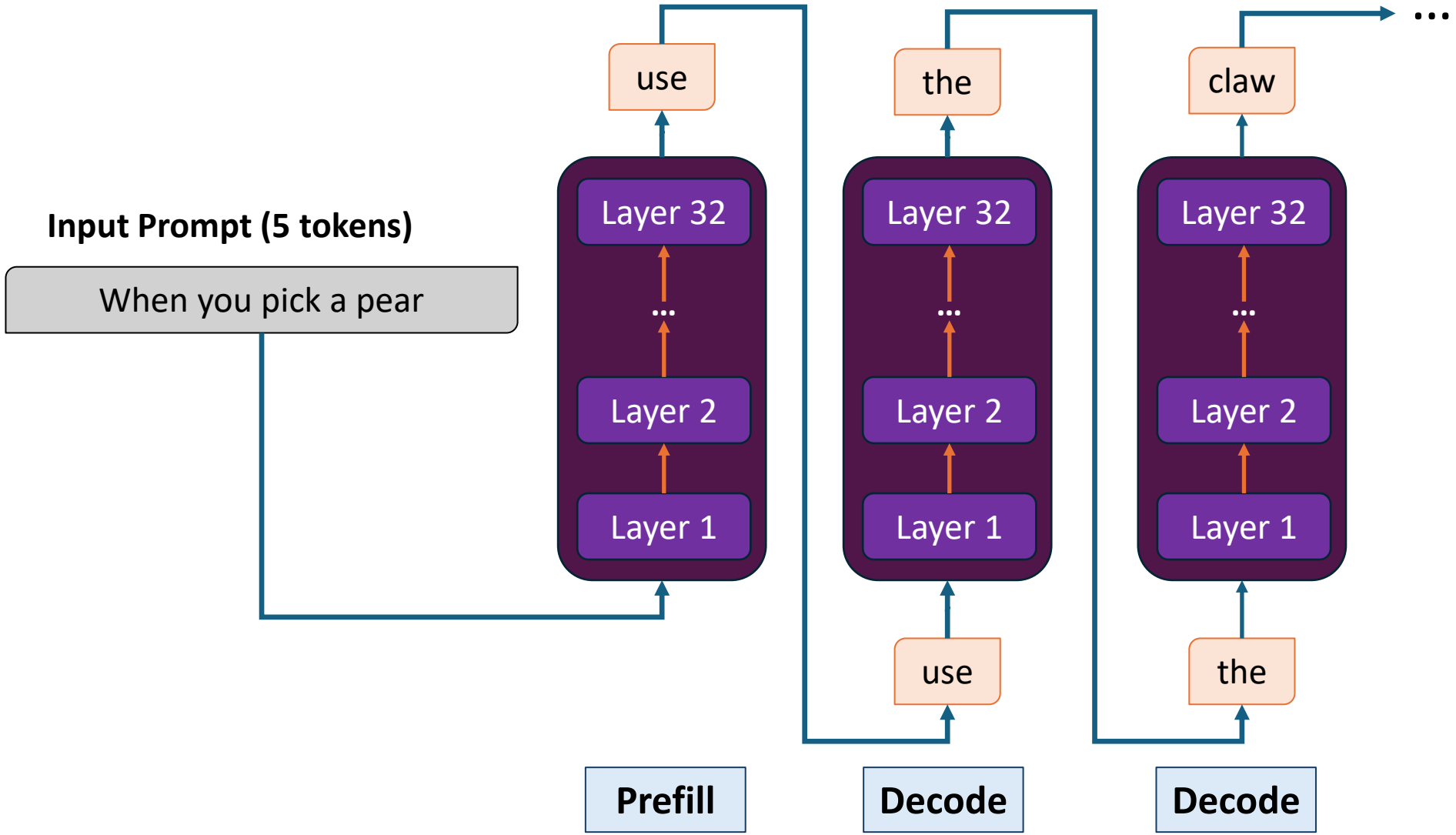
Need: A ***hardware-efficient*** **decomposition of** ***any*** **workload. Essential for:**

- ***Faster*** inference
- ***Higher*** scalability
- ***Lower*** costs

Roadmap

- **LLM Inference 101**
- **Challenges with current attention mechanisms**
- **LeanAttention's Design**
- **Evaluation Results**

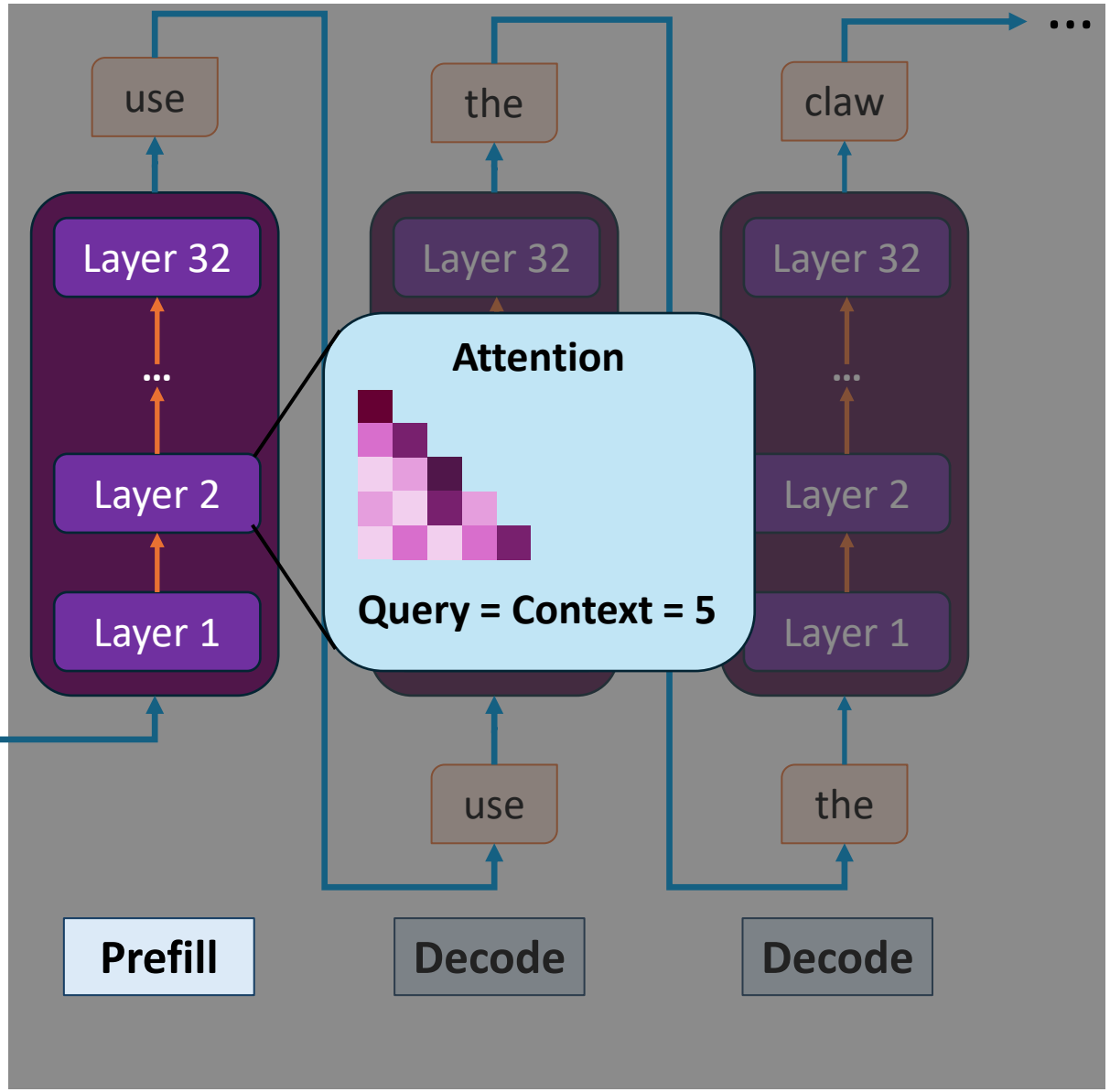
Two stages of LLM Inference



Prefill Stage

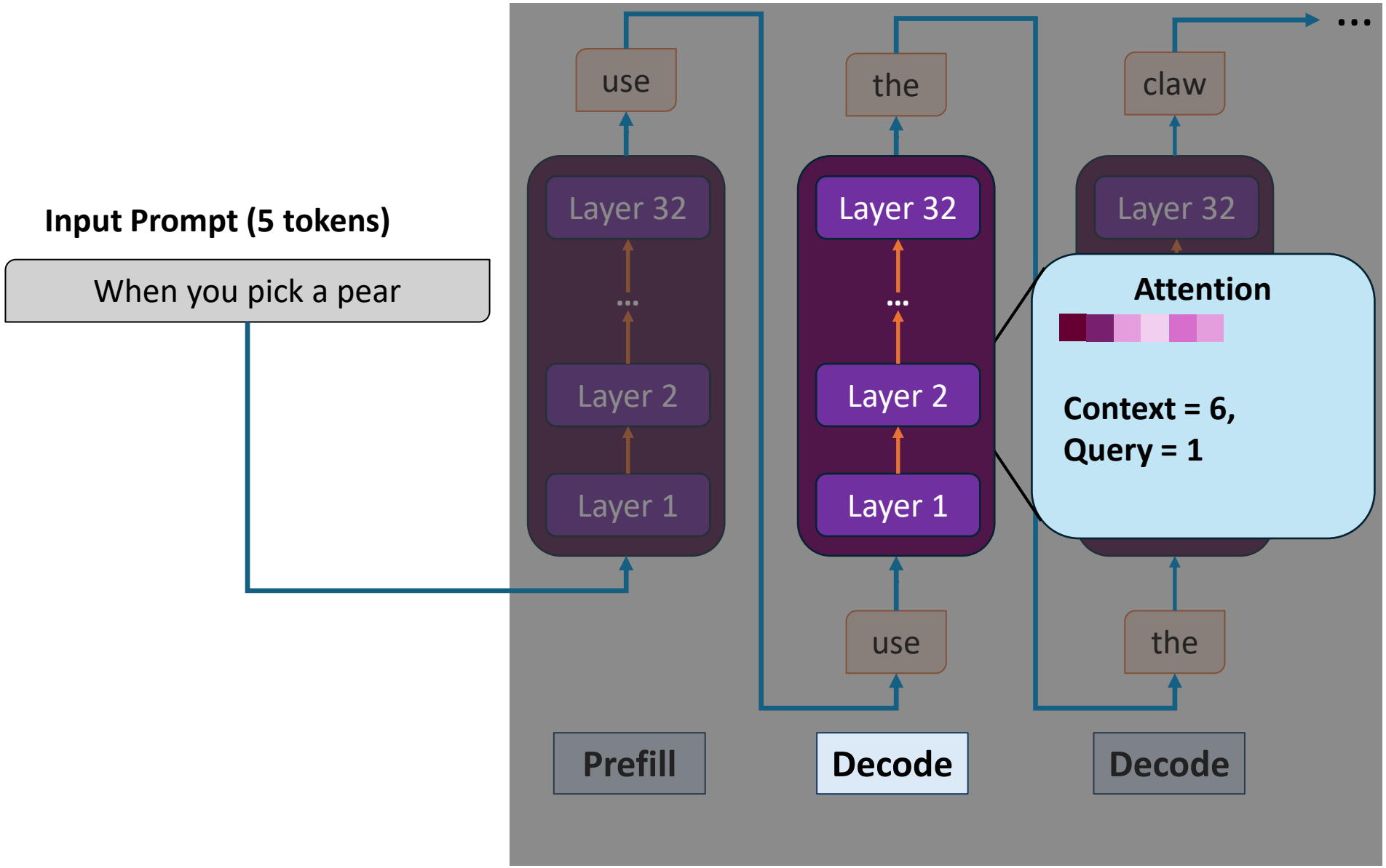
Input Prompt (5 tokens)

When you pick a pear

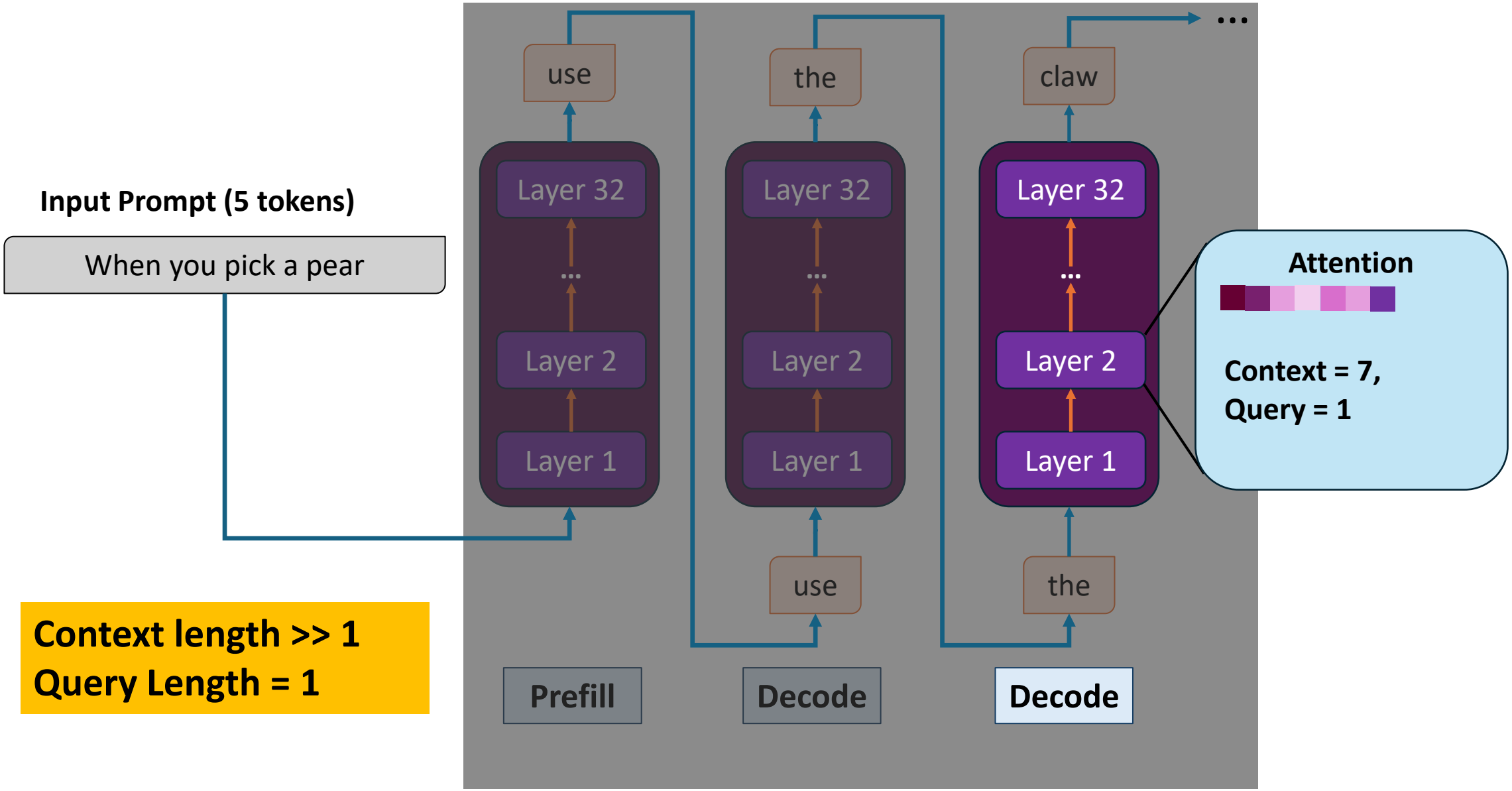


Query and Context lengths are the same !

Decode Stage

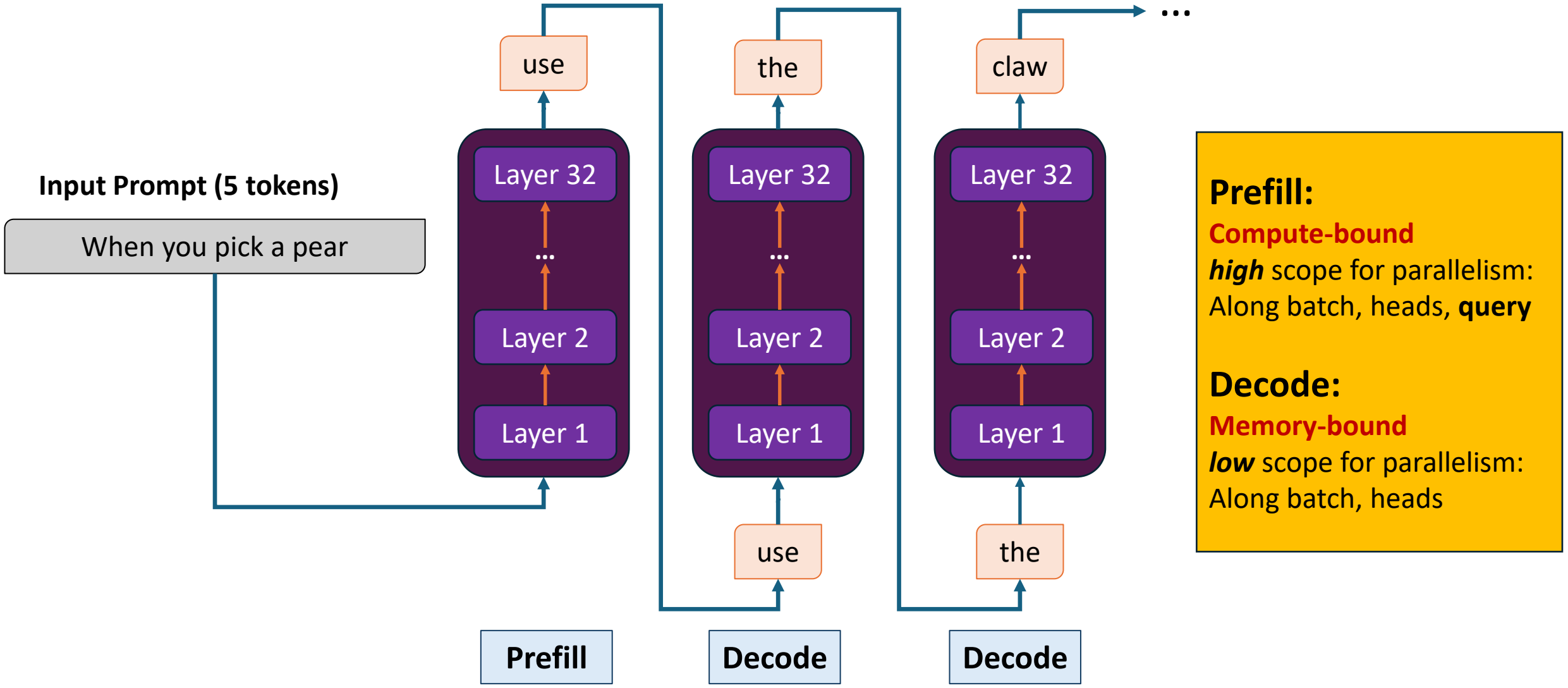


Decode Stage



Context length >> 1
Query Length = 1

Scope for parallelism



Prefill:

Compute-bound

high scope for parallelism:
Along batch, heads, **query**

Decode:

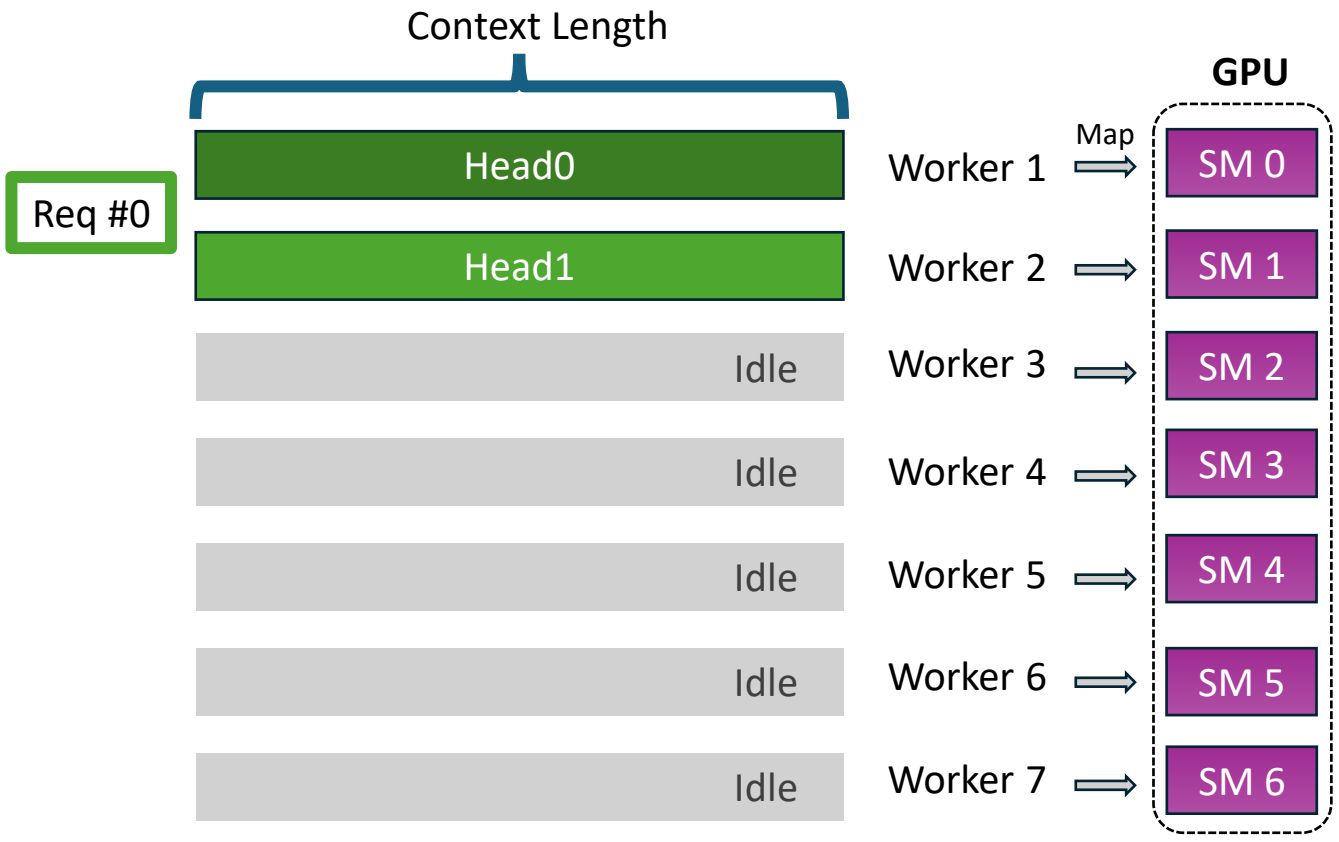
Memory-bound

low scope for parallelism:
Along batch, heads

Roadmap

- LLM Inference 101
- **Challenges with current attention mechanisms**
- LeanAttention's Design
- Evaluation Results

Imbalanced loads with FlashAttention



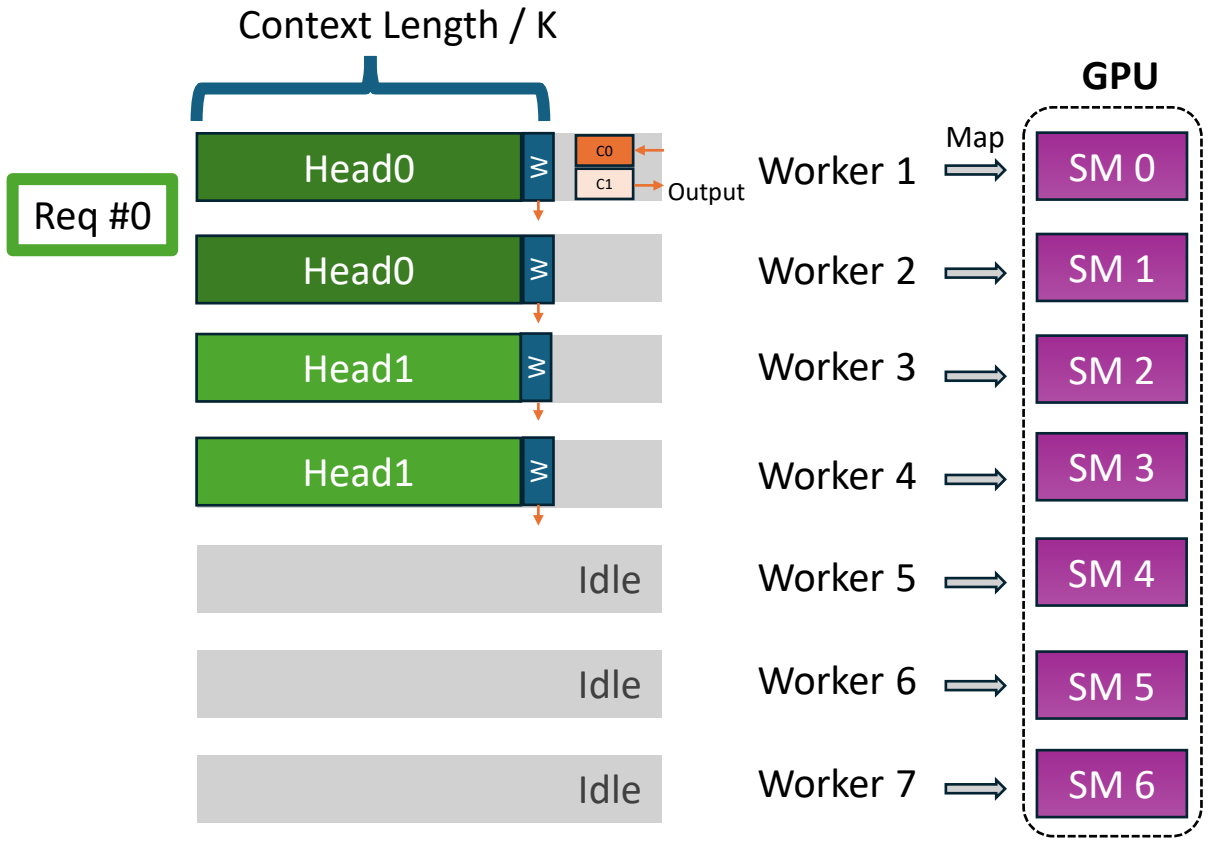
* SM = Streaming Multiprocessor i.e. an independent GPU core

FlashAttention leverages **parallelism along batch and heads** in Decode.

Ex: **Severe Underutilization** for Model with 128 heads on a 8x A100 system with 864 compute cores.

Question:
What if we split the work of each head along the context length ?

Imbalanced loads with FlashDecoding



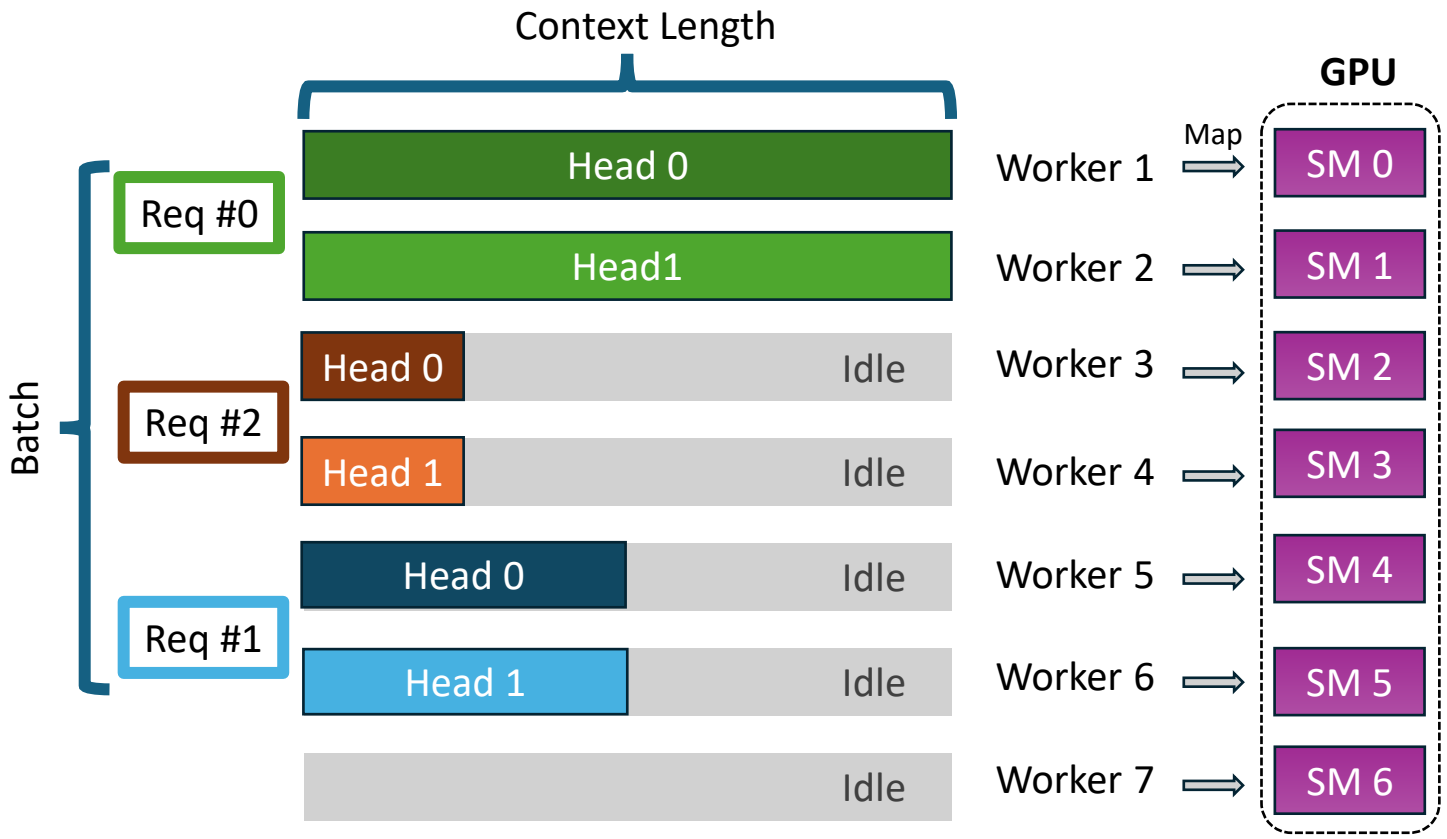
FlashDecoding employs fixed-split partitioning

- The **context of a each head is split** to different SMs.
- **Fixed number of splits K** is chosen (based on context size, # total heads, # SMs.)

- **GPU occupancy is dependent on workload**
  - Perfect occupancy only in cases where #SMs = 2n (#total_heads)

- **Reduction overheads scale with K**
- **More splits per head → inefficient use of register space per SM**

Imbalanced loads with Ragged Batching



Ragged Batching: Batching requests of unequal context lengths

- **Highly unlikely for requests in a batch to have the same context length.**
- **Memory Capacity Bounded**

Therefore, batching **decodes, decode + prefill, decode + chunked prefill** can **still give imbalanced loads** to the system.

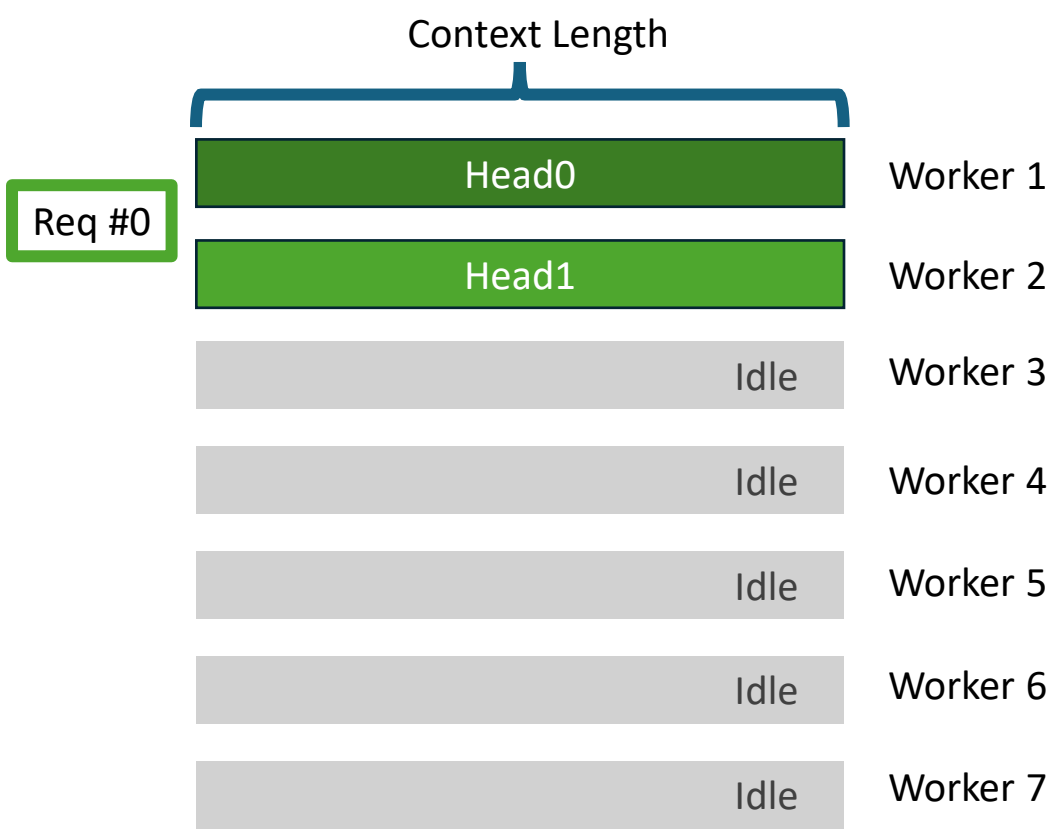
Question: How do we **decompose workload** to achieve **perfect quantization efficiency** (100% GPU Occupancy) for ***any*** **workload size?**

Roadmap

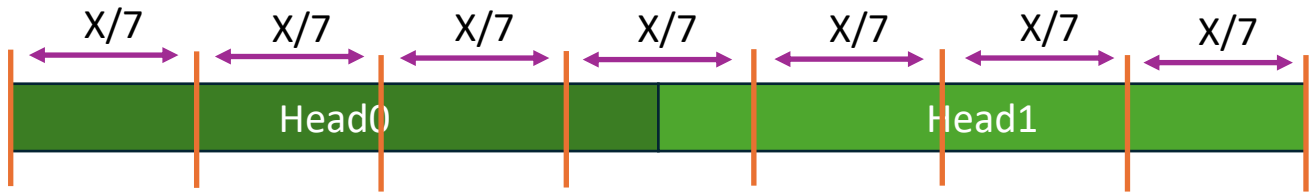
- LLM Inference 101
- Challenges with current attention mechanisms
- **LeanAttention's Design**
- Evaluation Results

A *Lean* Decomposition

100% GPU occupancy → ***Equal amount of work per SM*** → ***Splits of unequal sizes per head***



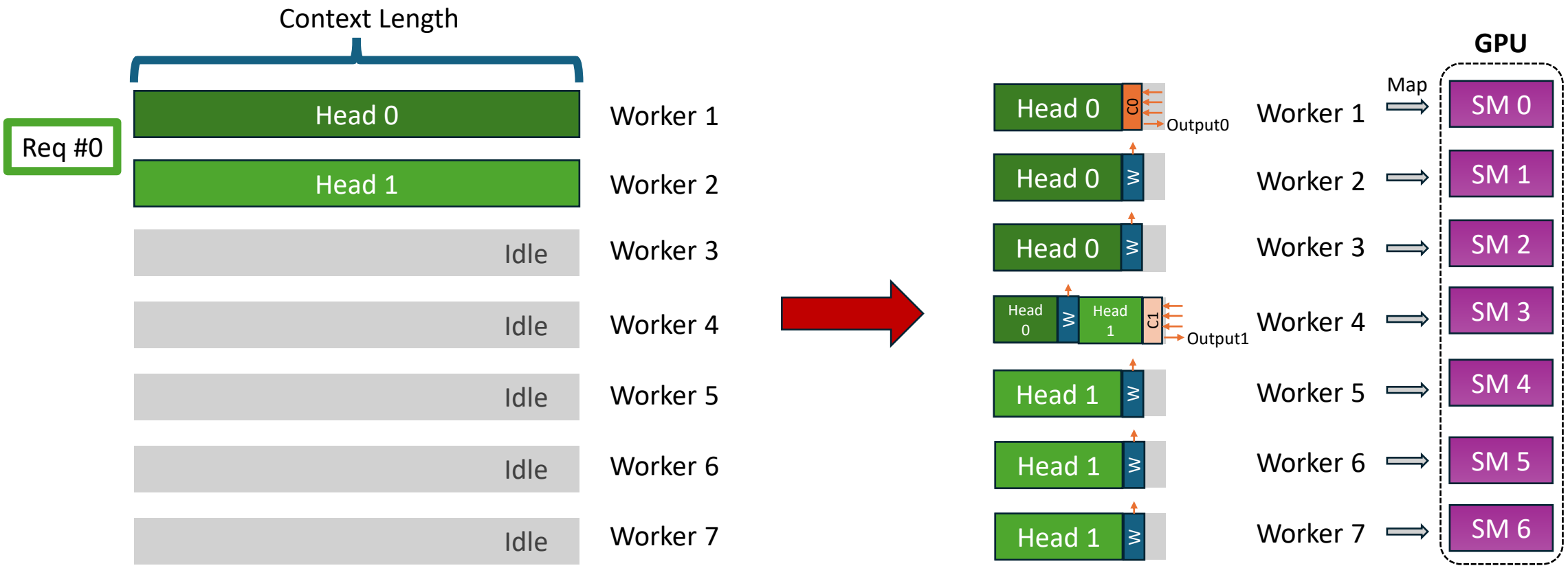
Equal work per SM = Total work X / #SMs



A *Lean* Decomposition

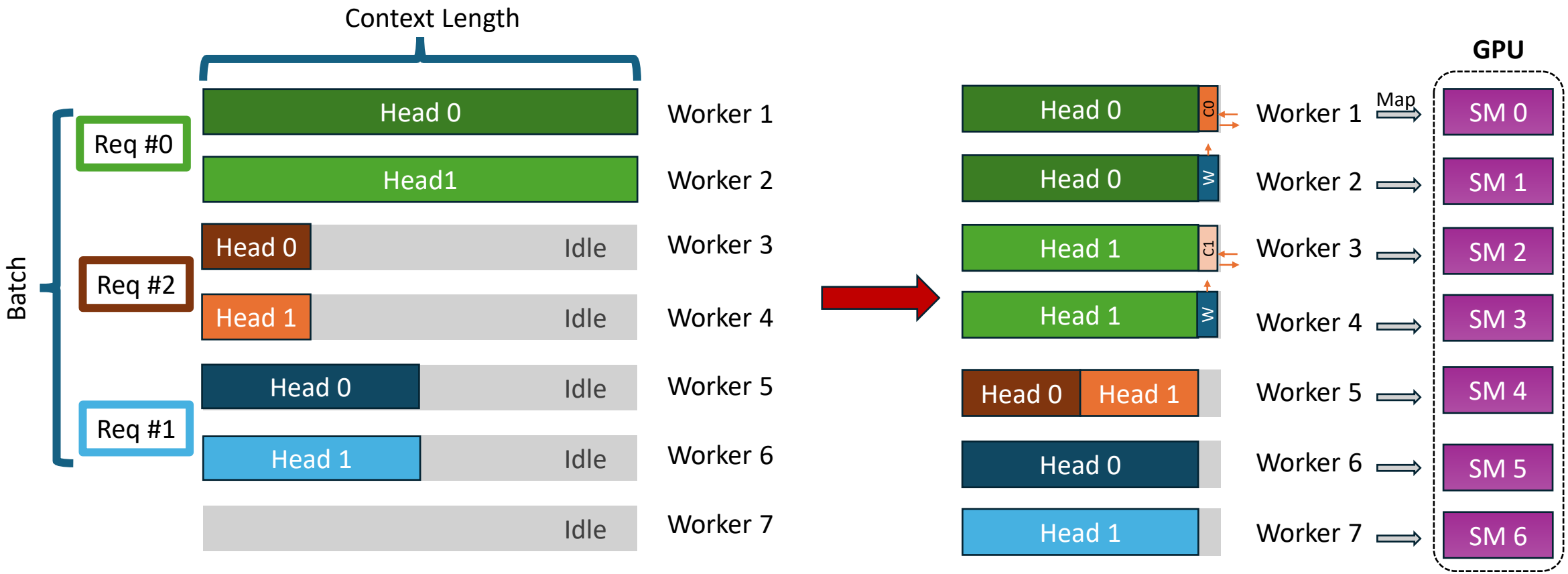
100% GPU occupancy → ***Equal amount of work per SM*** → ***Splits of unequal sizes per head***

Inspired by StreamK which does this for GEMMS !

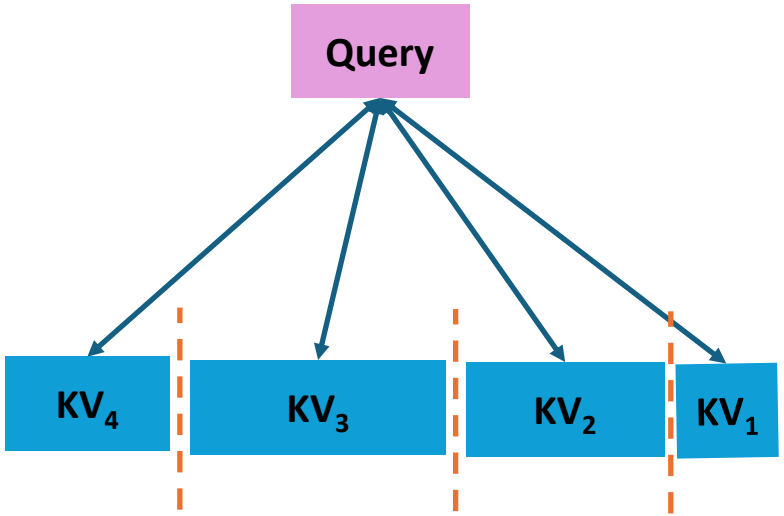


Lean Ragged Batching

Ragged batches can also be decomposed to give equal work per SM



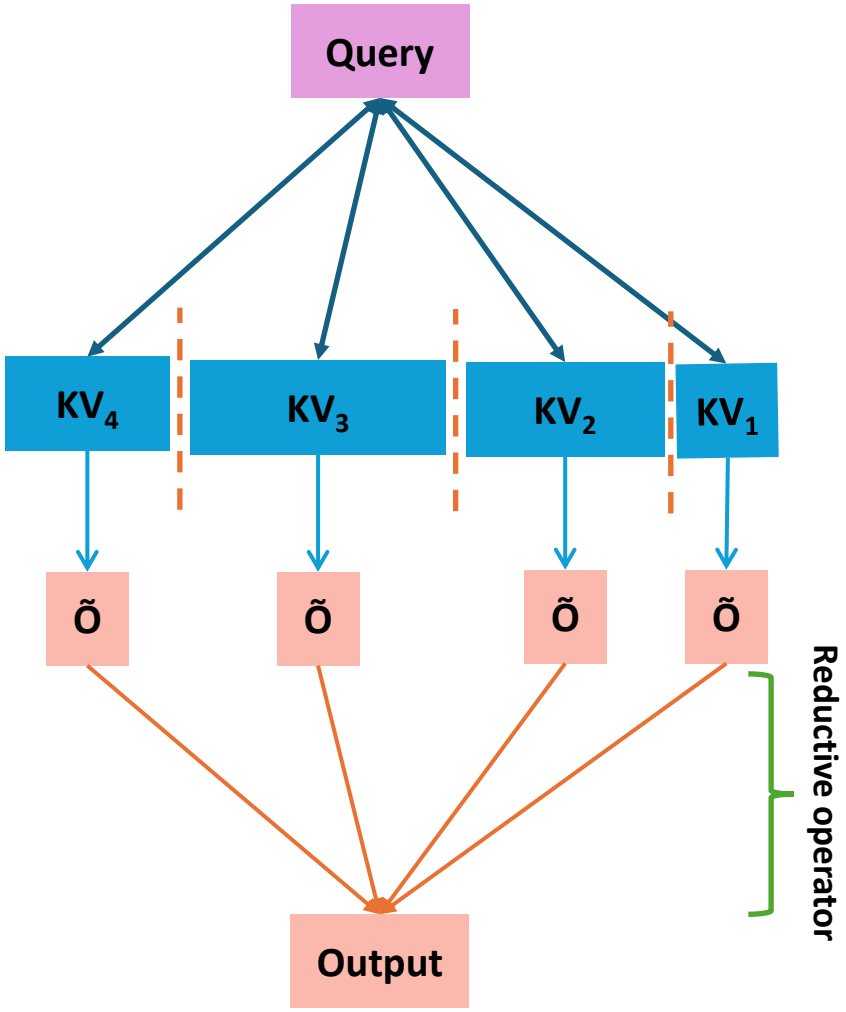
LeanAttention: Enabling StreamK-style decomposition



In LeanAttention's decomposition:

- Some heads get split into ***unequal*** portions

LeanAttention: Enabling StreamK decomposition

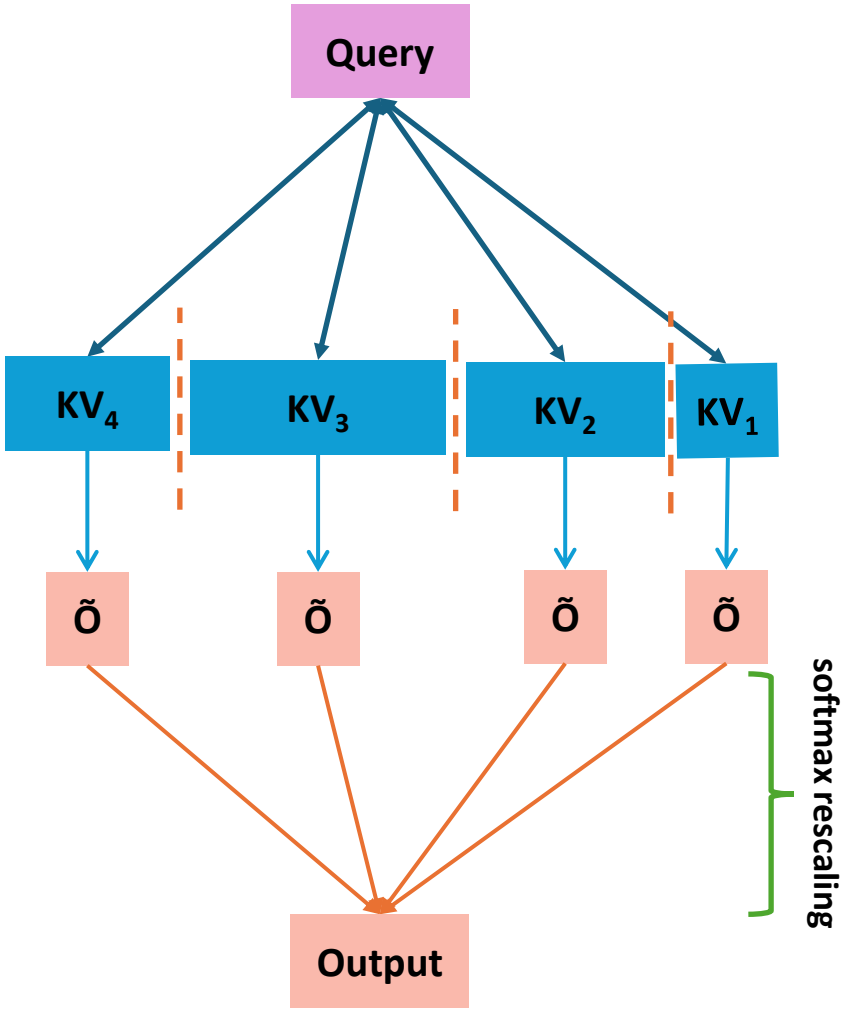


In LeanAttention's decomposition:

- Some heads get split into unequal portions

- To ***correctly reduce*** their partial outputs, the reductive operator must be **associative** in nature.

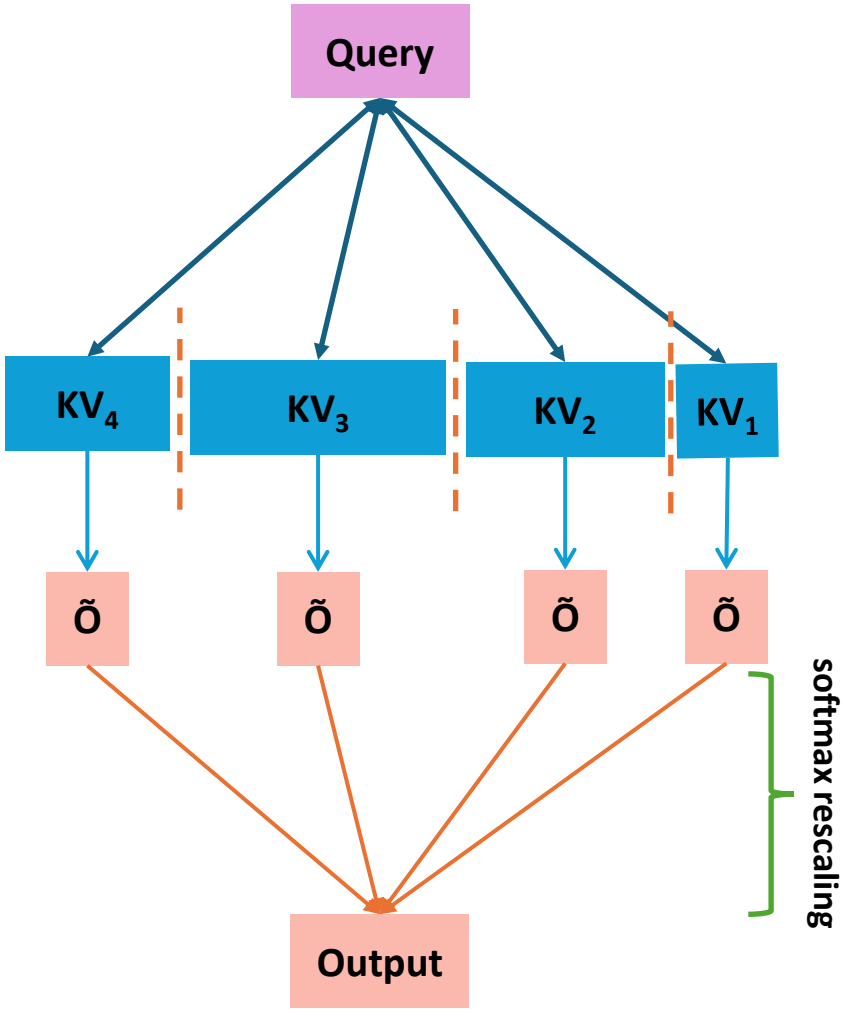
LeanAttention: Enabling StreamK decomposition



In LeanAttention's decomposition:

- Some heads get split into unequal portions
- To ***correctly reduce*** their partial outputs, the reductive operator must be **associative** in nature.
- The reductive operator in LeanAttention is termed **Softmax Re-scaling**.

LeanAttention: Enabling StreamK decomposition



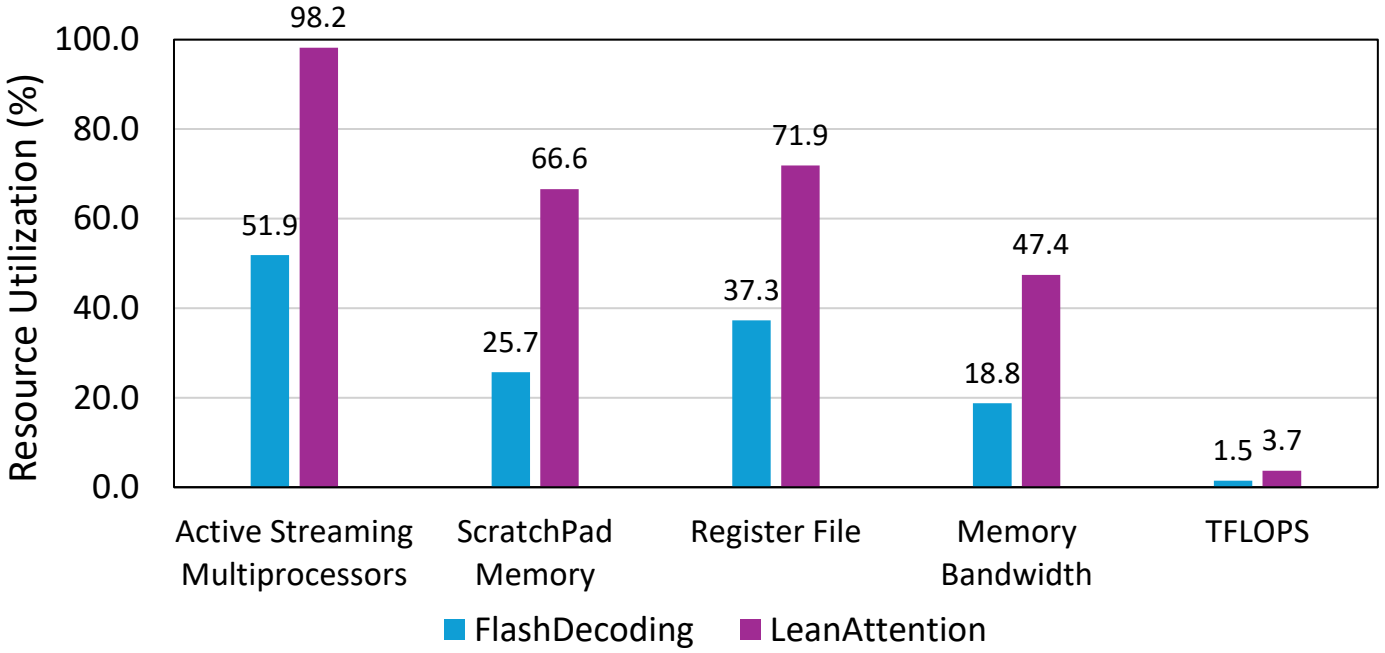
In LeanAttention's decomposition:

- Some heads get split into unequal portions
- To ***correctly reduce*** their partial outputs, the reductive operator must be **associative** in nature.
- The reductive operator in LeanAttention is termed **Softmax Re-scaling**.
- We utilize the ***associativity of Softmax Re-scaling*** to enable StreamK partitioning in LeanAttention ***without accuracy loss***.

Roadmap

- LLM Inference 101
- Challenges with current attention mechanisms
- LeanAttention's Design
- **Evaluation Results**

Effect on GPU Occupancy in Decode

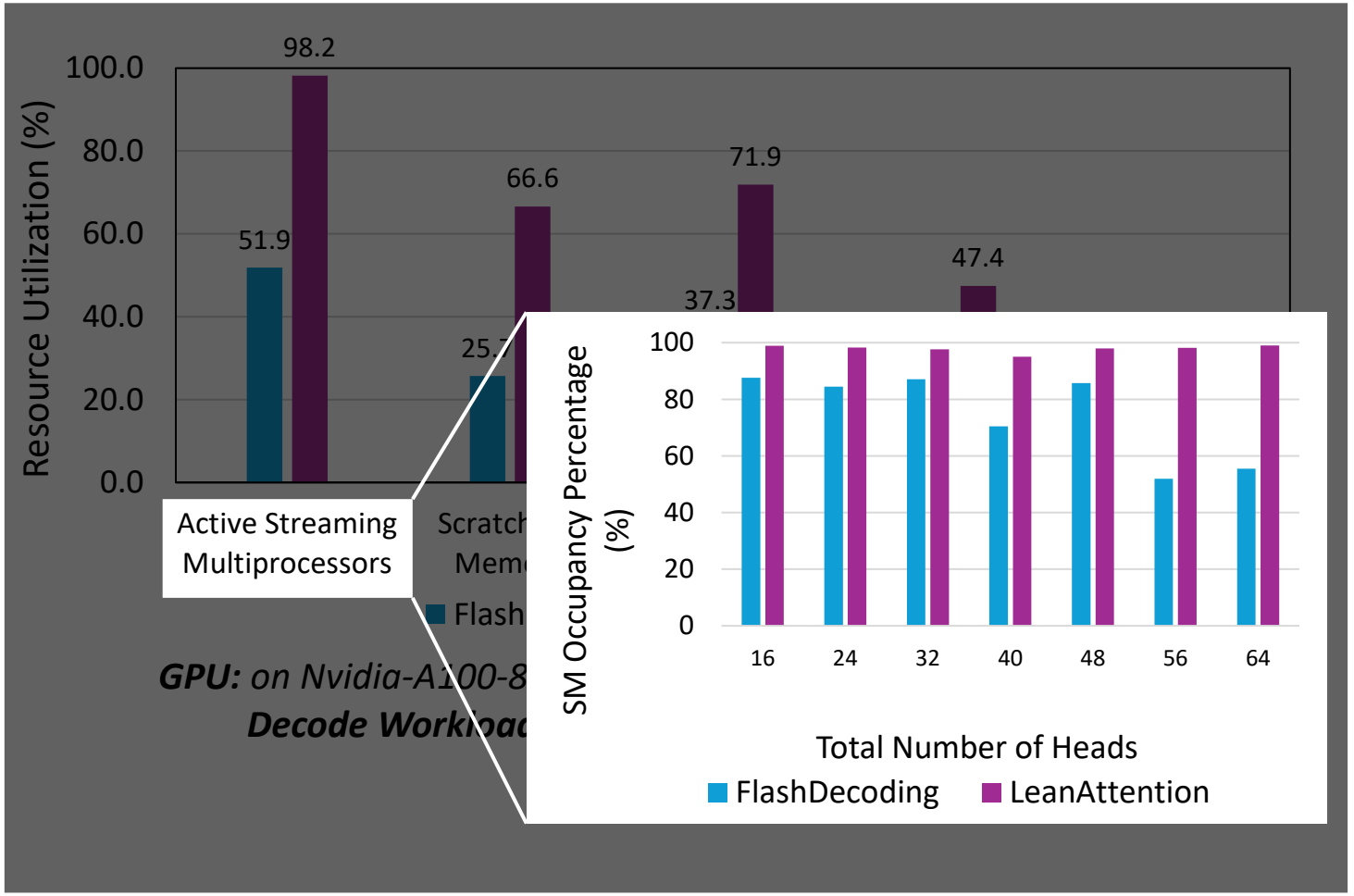


GPU: *on Nvidia-A100-80GB GPU (Measured on Nsight Compute)*
Decode Workload: *56 heads, single batch, 6k context*

FlashDecoding:
quantization inefficiencies = **imbalanced loads** on GPU → partially occupied SMs.

LeanAttention occupies ***all SMs*** available in the system.

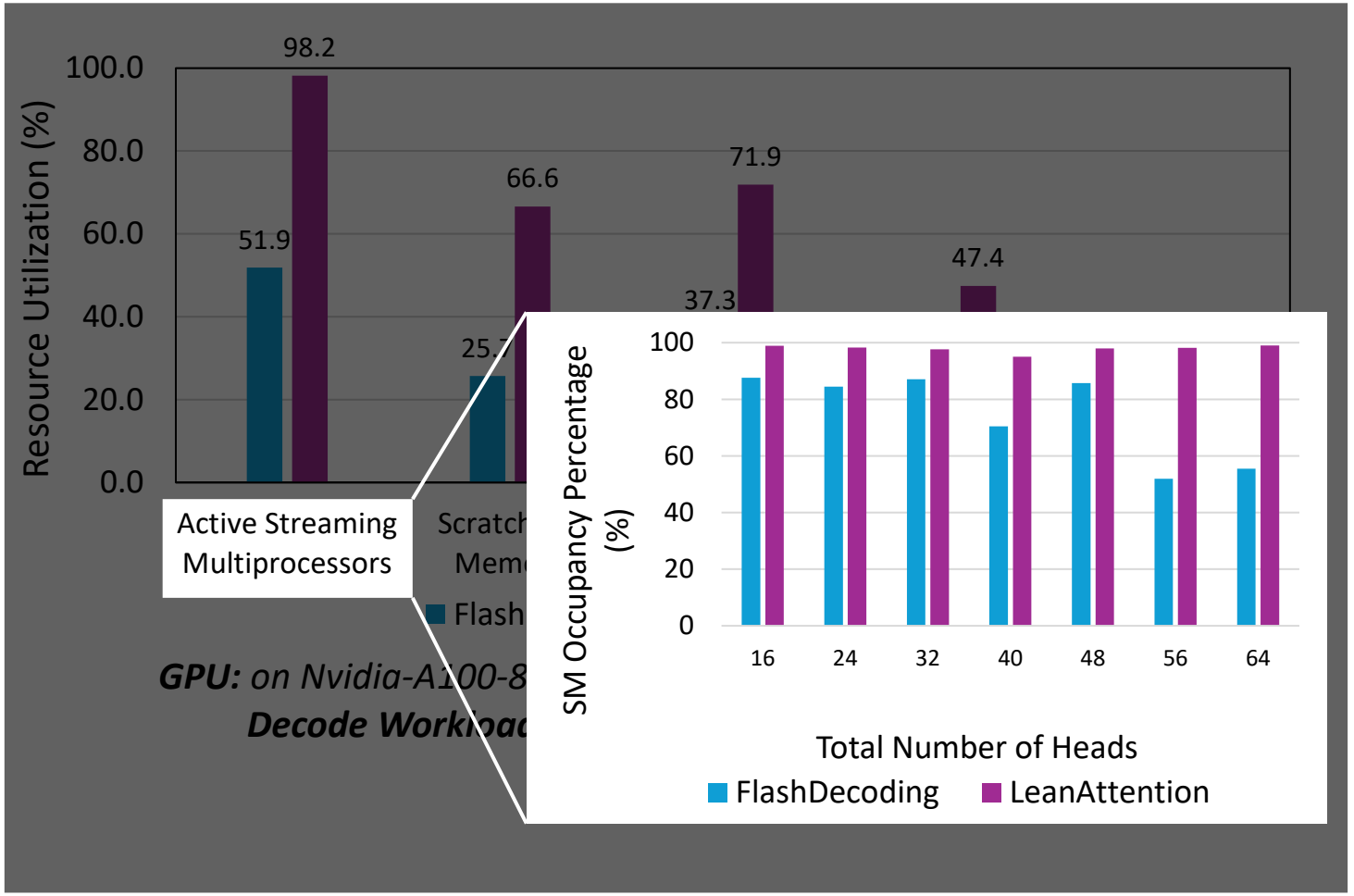
Effect on GPU Occupancy in Decode



FlashDecoding:
quantization inefficiencies = **imbalanced loads** on GPU → partially occupied SMs.

LeanAttention occupies ***all SMs*** available in the system.

Effect on GPU Occupancy in Decode

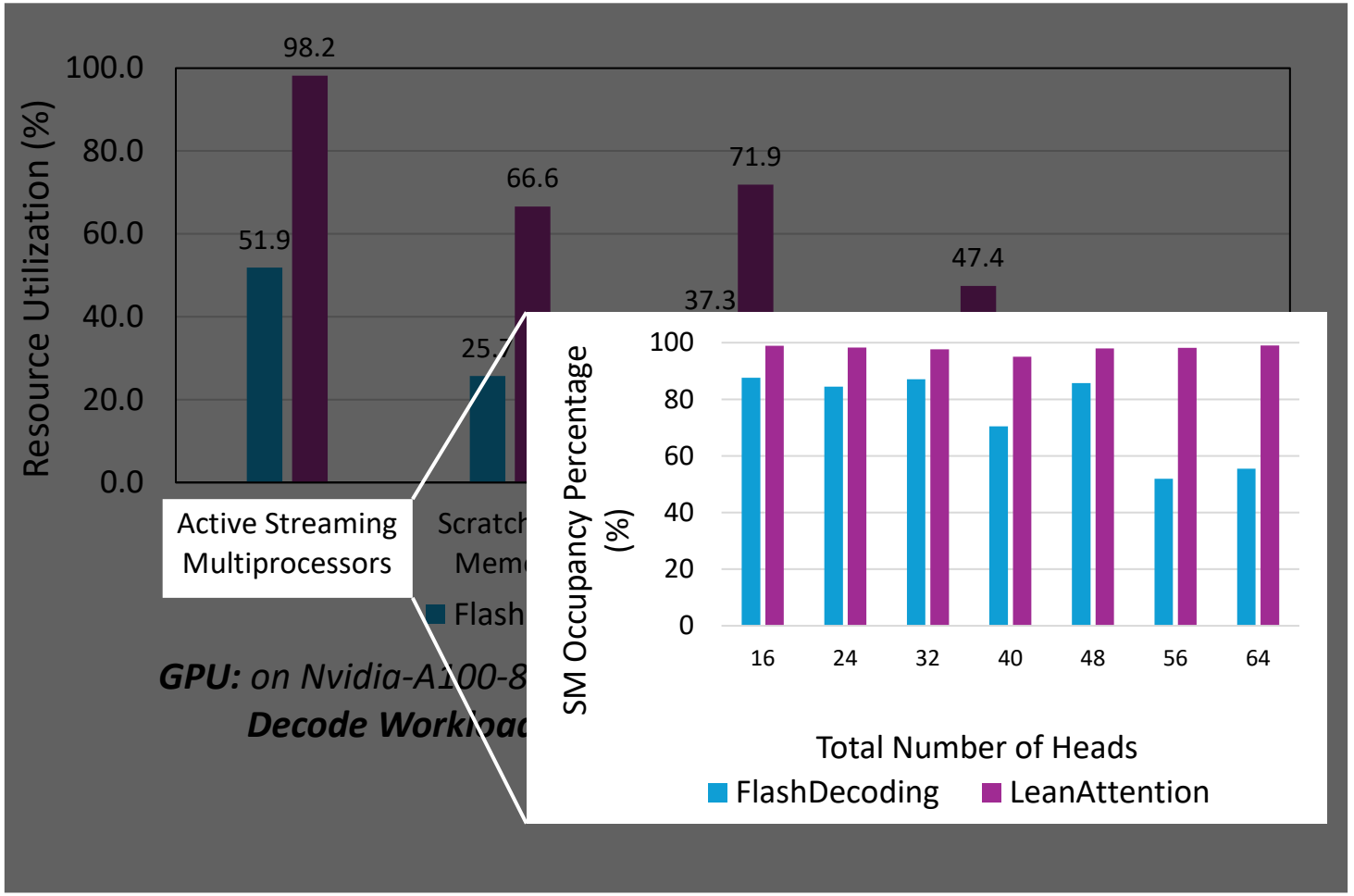


FlashDecoding:
quantization inefficiencies = **imbalanced loads** on GPU → partially occupied SMs.

GPU Occupancy is ***contingent on workload*** dimensions.

LeanAttention occupies ***all SMs*** available in the system.

Effect on GPU Occupancy in Decode



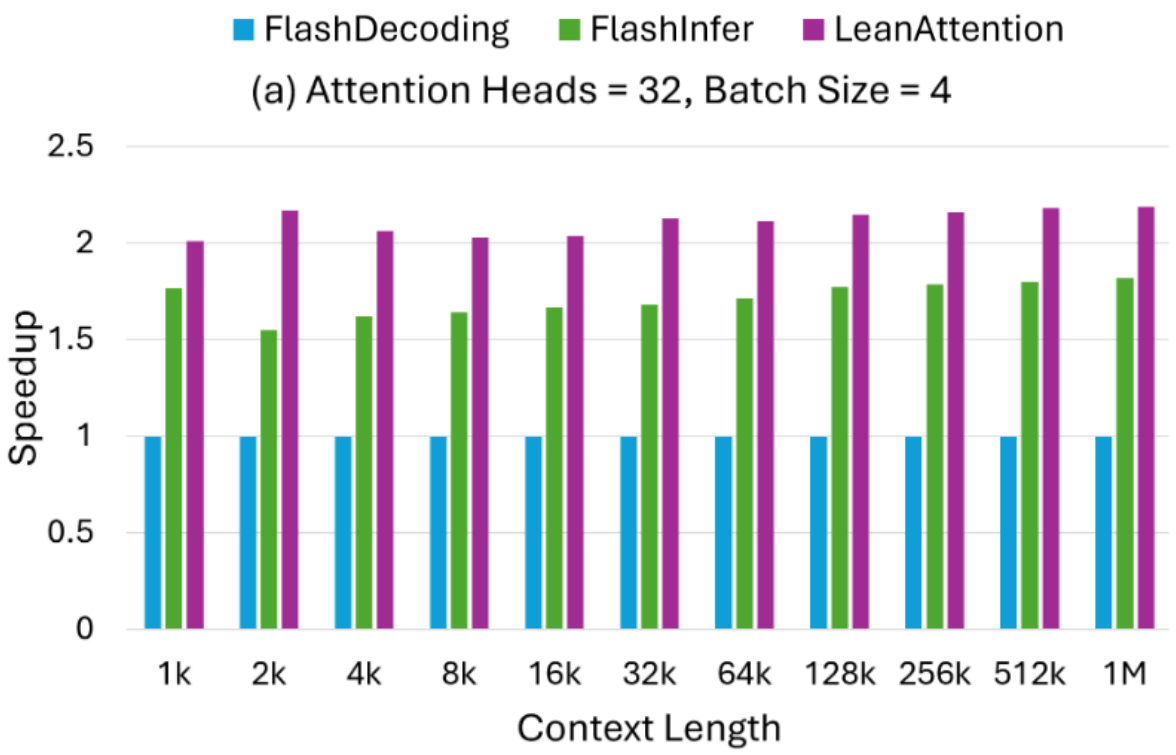
FlashDecoding:
quantization inefficiencies = **imbalanced loads** on GPU → partially occupied SMs.

GPU Occupancy is ***contingent on workload*** dimensions.

LeanAttention occupies ***all SMs*** available in the system for ***all*** workload sizes.

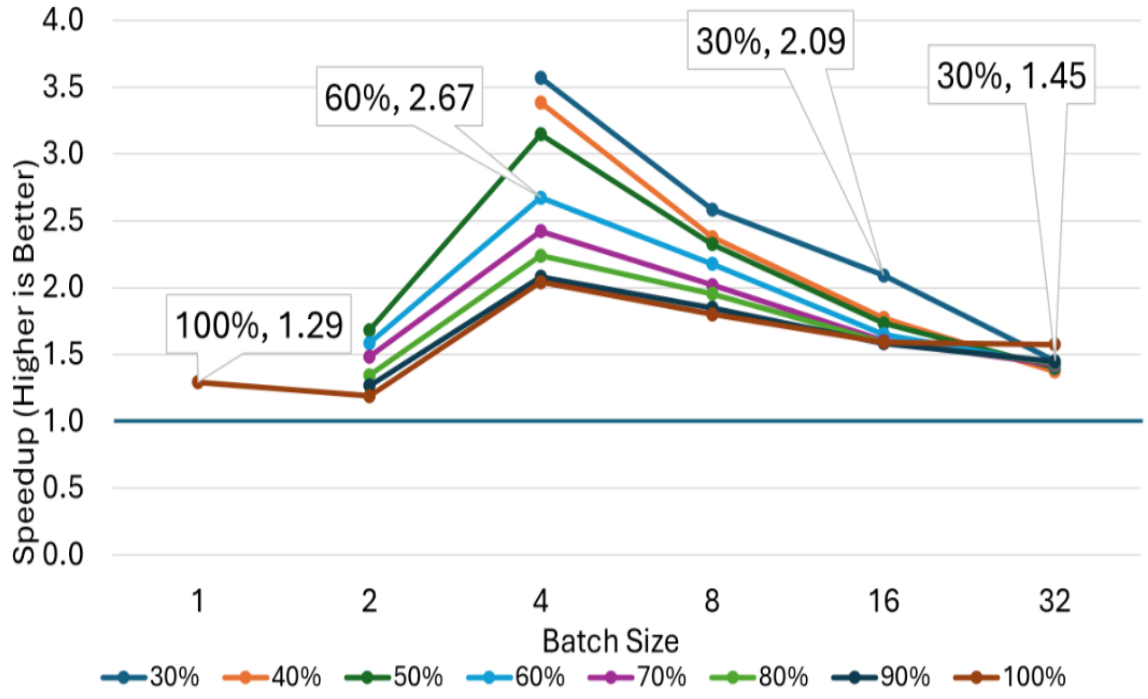
Evaluation Results

Non-Ragged Decode



2.18x speedup for 32k or higher context lengths

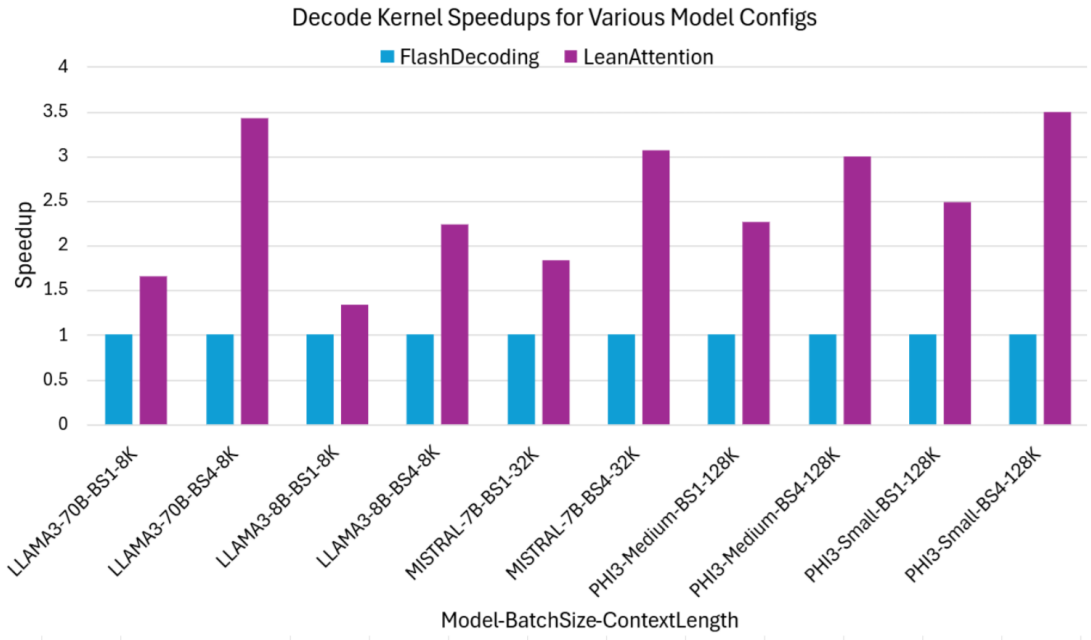
Ragged Batching



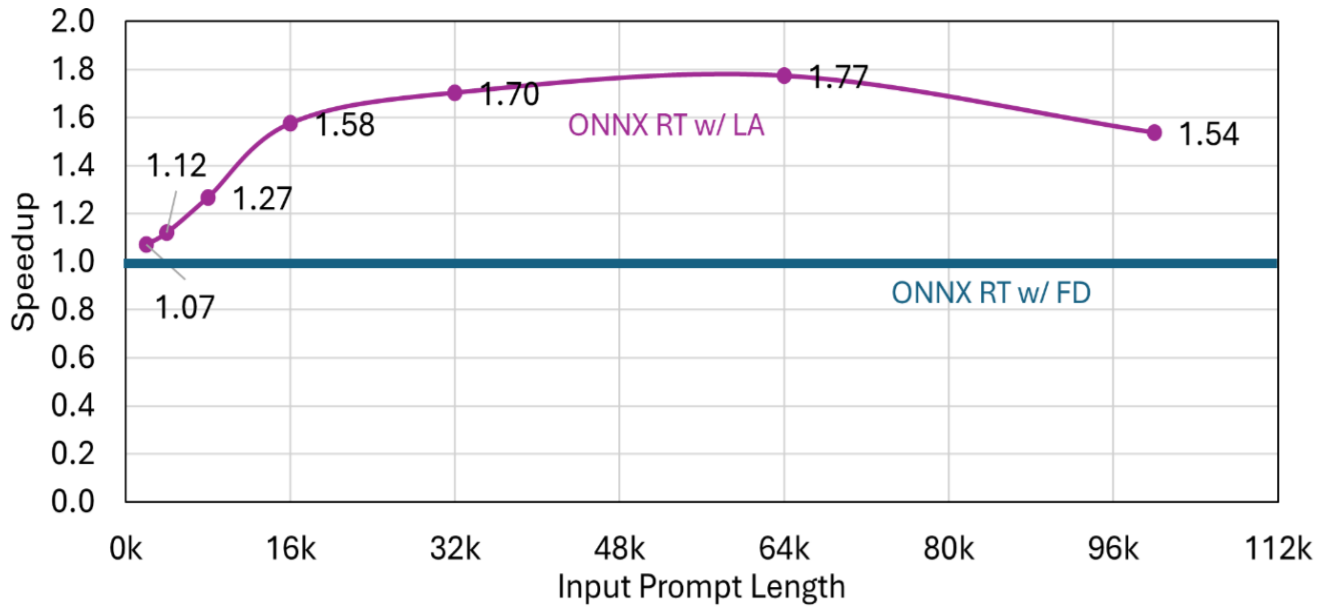
Speedup for different ragged batching cases

Batch context ratio(%) = avg context / max context

Evaluation Results



Deliver speedups in Attention Execution across Models



End-to-End speedups for Phi-3 Medium Model

LeanAttention always performs either exactly the same or better than FlashAttention, FlashDecoding

Publicly available on ONNXRuntime !

Summary

- Self-attention is the heart of transformer-based models; however it is a **time consuming** operation particularly during **Decode Phase** and **Ragged Batching.**

- State-of-the-art GPU execution techniques such as **FlashAttention, FlashDecode**, etc., **fail to utilize the GPU cores** (SMs) efficiently leading to increased latencies and costs.

- We propose **Lean Attention**, which **equally distributes the work** to all SMs of the GPUs by leveraging techniques originally developed for simple matrix multiplications.

- To do this, we first identify that the **softmax rescaling operation** within attention is **associative** similar to an **addition operation** in matrix multiplication and then leveraging **"stream-k" decomposition** of work.

- **Lean Attention** can deliver more than **2.18x speedup** over state-of-the art GPU execution techniques.

Thank you!

https://arxiv.org/abs/2405.10480