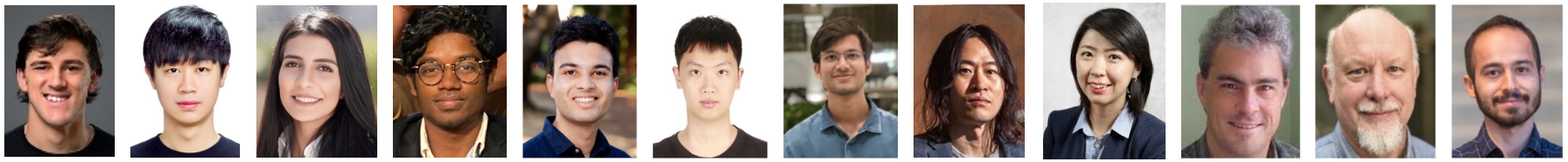
Accelerating Long Context Length LLM Inference



Coleman Hooper [1]*, Sehoon Kim[1]*, Hiva Mohammadzadeh[1], Monishwaran Maheswaran[1], Aditya Tomar, Haocheng Xi, Rishabh Tiwari,

June Paik[2], Sophia Shao[1], Michael W. Mahoney[1], Kurt Keutzer[1], **Amir Gholami[1]**

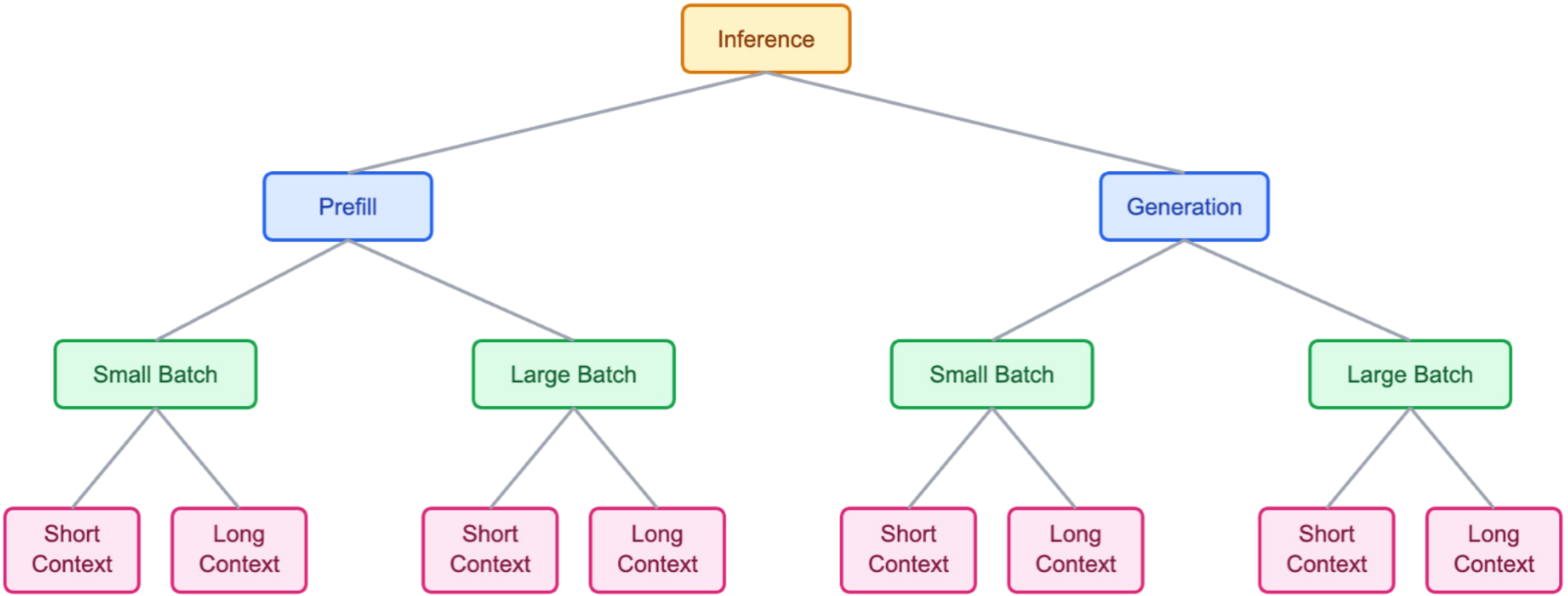
[1] University of California Berkeley

[2] FuriosaAI

Agenda

- **Inference bottleneck analysis for Transformers**
- Squeezed Attention: Algorithm
  - Selective Attention
  - Hierarchical Attention
- Results
- Kernel Implementation and Benchmarking
- Conclusion

Multi-Regime Analysis



Arithmetic Intensity

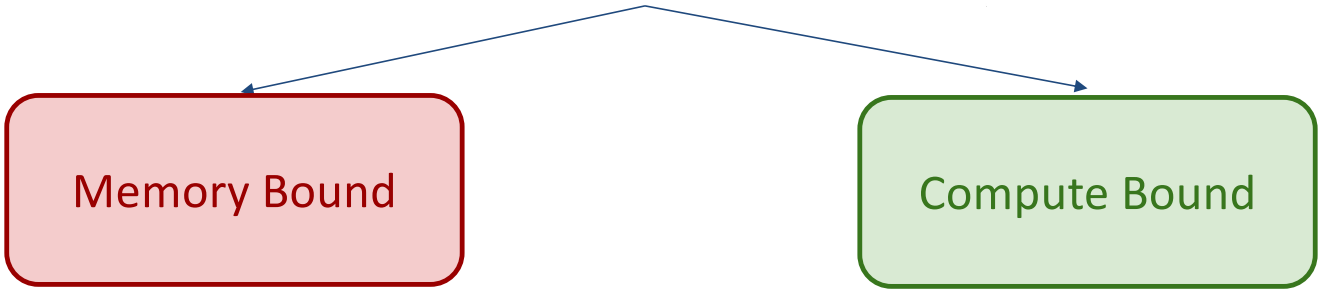
Arithmetic Intensity: number of floating point operations (FLOPs) that can be performed per byte loaded from memory, or memory operations (MOPs)

$$\text{Arithmetic Intensity} = \frac{\#\,\text{FLOPs}}{\#\,\text{MOPs}}$$

Arithmetic Intensity

Arithmetic Intensity: number of floating point operations (FLOPs) that can be performed per byte loaded from memory, or memory operations (MOPs)

$$\text{Arithmetic Intensity} = \frac{\#\text{ FLOPs}}{\#\text{ MOPs}}$$



- Limited by hardware's peak memory bandwidth (GB/s)
- Benefit from techniques that optimize memory load-store operations (e.g., quantization)

- Limited by hardware's peak FLOP/s (FLOPs per second)
- Benefit from algorithmic improvements that reduce computational complexity (e.g., subquadratic attention)

Fine-Grained Analysis

Categorize all Transformer operations for a **finer-grained analysis**

Linear
Wq, Wk, Wv, Wo, FFN

Attention
QK^T & attn_weights * V

Aggregate
Linear + Attention + LayerNorm, Softmax, Activation

1. Asymptotic analysis of arithmetic intensity for linear, attention, and aggregate operations
2. Visualize analysis using analytical roofline model for a practical inference setting

Asymptotic Analysis of Arithmetic Intensity for Prefill and Decoding

Table 1. Asymptotic analysis of arithmetic intensity for linear, attention, and aggregate operations under prefill and decoding for batch size B, sequence length S_L, hidden dimension d, and generation length of k tokens.

Prefill

	Linear	Attention	Aggregate
FLOPs	$\mathcal{O}(B \cdot S_L \cdot d^2)$	$\mathcal{O}(B \cdot {S_L}^2 \cdot d)$	$\mathcal{O}(B \cdot S_L \cdot d^2) + \mathcal{O}(B \cdot {S_L}^2 \cdot d)$
MOPs	$\underbrace{\mathcal{O}(B \cdot S_L \cdot d)}_{\text{activations}} + \underbrace{\mathcal{O}(d^2)}_{\text{weights}}$	$\underbrace{\mathcal{O}(B \cdot S_L)}_{\text{flash-attn scores}} + \underbrace{\mathcal{O}(B \cdot S_L \cdot d)}_{\text{activations } \{Q, C_K, C_V\}}$	$\mathcal{O}(B \cdot S_L \cdot d) + \mathcal{O}(d^2)$
Arithmetic Intensity	$\approx \begin{cases} \mathcal{O}(B \cdot S_L), & S_L \ll d \\ \mathcal{O}(d), & S_L \gg d \end{cases}$	$\approx \begin{cases} \mathcal{O}(S_L), & S_L \ll d \\ \mathcal{O}(S_L), & S_L \gg d \end{cases}$	$\approx \begin{cases} \mathcal{O}(B \cdot S_L), & S_L \ll d \\ \mathcal{O}(S_L), & S_L \gg d \end{cases}$

Decode

	Linear	Attention	Aggregate
FLOPs	$\mathcal{O}(k \cdot B \cdot d^2)$	$\mathcal{O}(k \cdot B \cdot S_L \cdot d)$	$\mathcal{O}(k \cdot B \cdot d^2) + \mathcal{O}(k \cdot B \cdot S_L \cdot d)$
MOPs	$\underbrace{\mathcal{O}(k \cdot B \cdot d)}_{\text{activations}} + \underbrace{\mathcal{O}(k \cdot d^2)}_{\text{weights}}$	$\underbrace{\mathcal{O}(k \cdot B \cdot S_L)}_{\text{attention scores}} + \underbrace{\mathcal{O}(k \cdot B \cdot S_L \cdot d)}_{\text{activations } \{C_K, C_V\}}$	$\mathcal{O}(k \cdot d^2) + \mathcal{O}(k \cdot B \cdot S_L \cdot d)$
Arithmetic Intensity	$\approx \begin{cases} \mathcal{O}(B), & S_L \ll d \\ \mathcal{O}(B), & S_L \gg d \end{cases}$	$\approx \begin{cases} \mathcal{O}(1), & S_L \ll d \\ \mathcal{O}(1), & S_L \gg d \end{cases}$	$\approx \begin{cases} \mathcal{O}(B), & S_L \ll d \\ \mathcal{O}(1), & S_L \gg d \end{cases}$

Tiwari R*, Xi H*, Tomar A*, Hooper C, Kim S, Horton M, Najibi M, Mahoney MW, Keutzer K, Gholami A., QuantSpec: Self-Speculative Decoding with Hierarchical Quantized KV Cache. arXiv:2502.10424.

Asymptotic Analysis of Arithmetic Intensity for Prefill and Decoding

Prefill Arithmetic Intensity

$$\underbrace{\begin{cases}\mathcal{O}(B \cdot S_L), & S_L \ll d \\ \mathcal{O}(S_L), & S_L \gg d\end{cases}}_{\text{prefill}} \gg \underbrace{\begin{cases}\mathcal{O}(B), & S_L \ll d \\ \mathcal{O}(1), & S_L \gg d\end{cases}}_{\text{decode}}.$$

Decode Arithmetic Intensity

Asymptotic Analysis of Arithmetic Intensity for Prefill and Decoding

Prefill Arithmetic Intensity

$$\underbrace{\begin{cases} \mathcal{O}(B \cdot S_L), & S_L \ll d \\ \mathcal{O}(S_L), & S_L \gg d \end{cases}}_{\text{prefill}} \gg \underbrace{\begin{cases} \mathcal{O}(B), & S_L \ll d \\ \mathcal{O}(1), & S_L \gg d \end{cases}}_{\text{decode}}.$$

Decode Arithmetic Intensity

?

Compute Bound

Memory Bound

Asymptotic Analysis of Arithmetic Intensity for Prefill and Decoding

Prefill Arithmetic Intensity

$$\underbrace{\begin{cases} \mathcal{O}(B \cdot S_L), & S_L \ll d \\ \mathcal{O}(S_L), & S_L \gg d \end{cases}}_{\text{prefill}} \gg \underbrace{\begin{cases} \mathcal{O}(B), & S_L \ll d \\ \mathcal{O}(1), & S_L \gg d \end{cases}}_{\text{decode}}.$$

Decode Arithmetic Intensity

Observations:

- AI of prefill scales with SL but not decoding
- Larger batch size only helps with short SL
- **Batching does not help** when dealing with long SL

In decoding every sequence in the batch undergoes self-attention separately and therefore cannot benefit from batching in the same way linear layers do.

Analytical Roofline Model

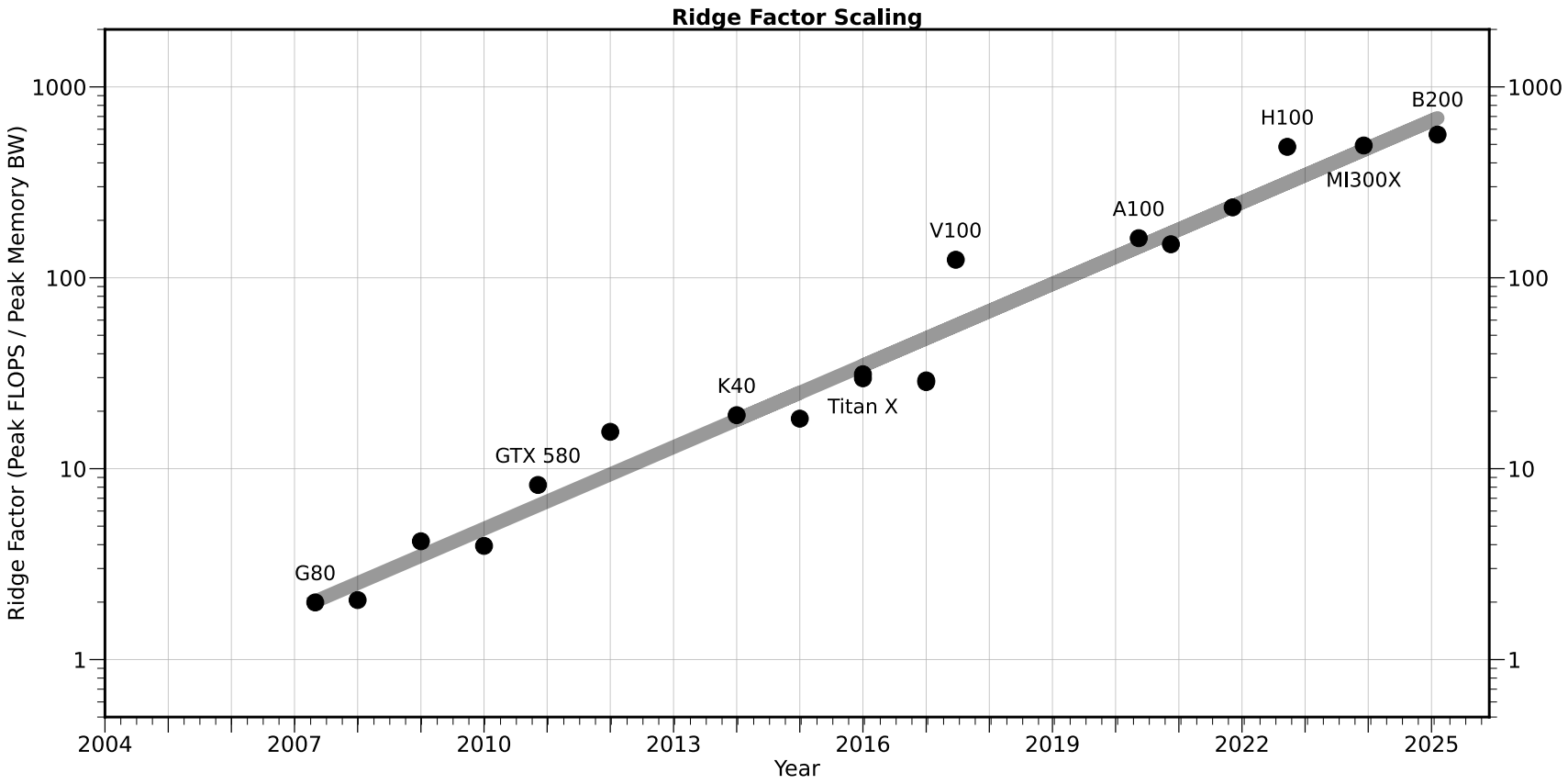
Ridge point: $$\frac{\text{peak compute performance (FLOP/s)}}{\text{peak memory-BW (GB/s)}}$$

Memory-Bound **<** **Ridge point** **<** **Compute-Bound**

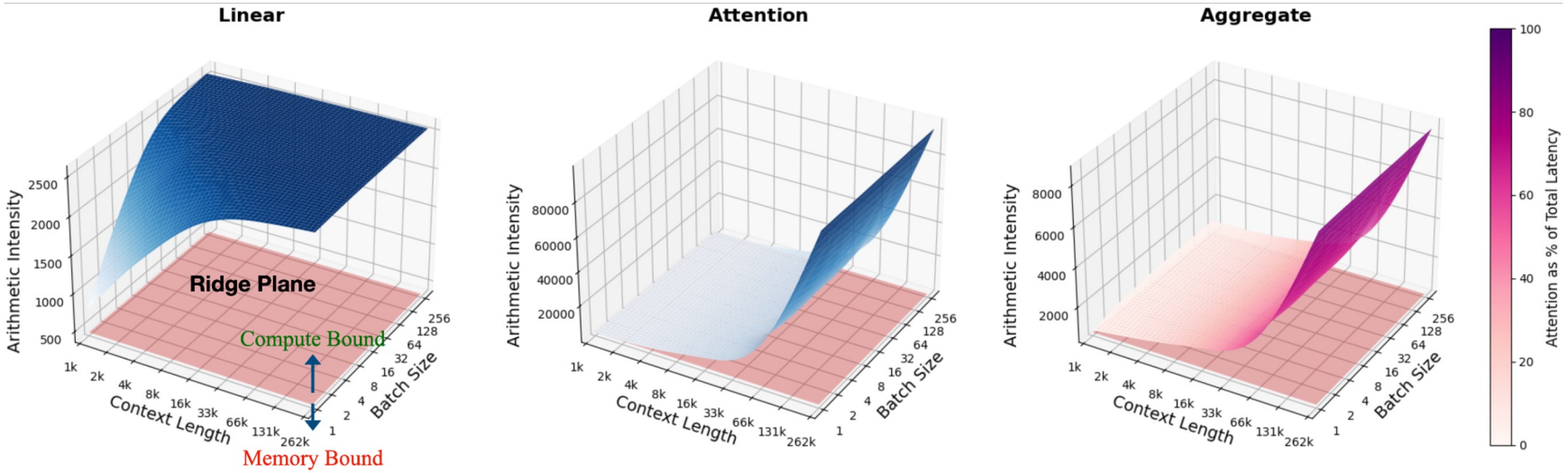
Ridge point (A100) = 161
Ridge point (A6000) = 403
Ridge point (H100) = 485

Ridge Point Scaling Trend

Memory-Bound < **Ridge point** < **Compute-Bound**



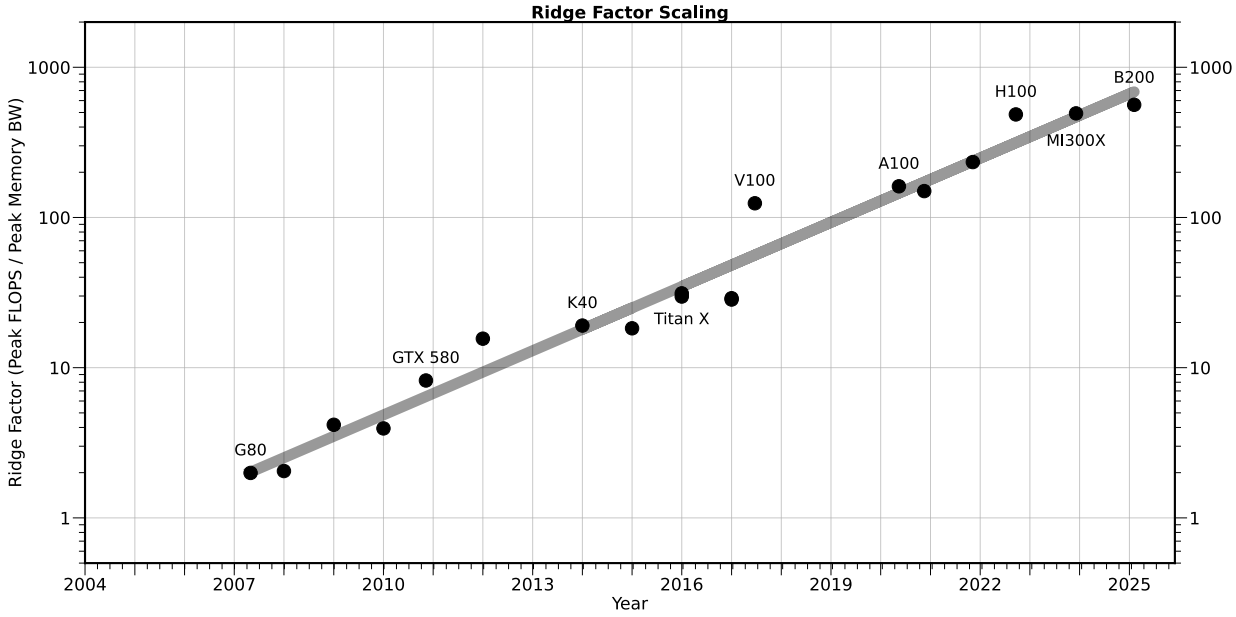
Prefill (Llama-2-7B, NVIDIA A6000)

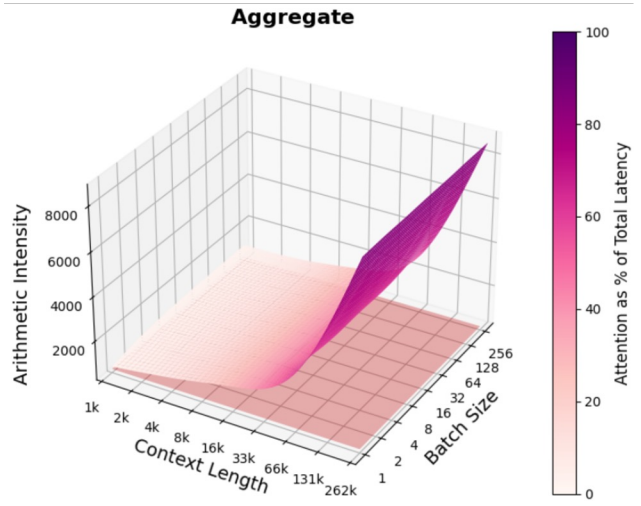


During prefill, all regimes lie above the ridge plane and thus are **compute-bound**.

Tiwari R*, Xi H*, Tomar A*, Hooper C, Kim S, Horton M, Najibi M, Mahoney MW, Keutzer K, Gholami A. QuantSpec: Self-Speculative Decoding with Hierarchical Quantized KV Cache. arXiv:2502.10424.

Prefill (Llama-2-7B, NVIDIA A6000)

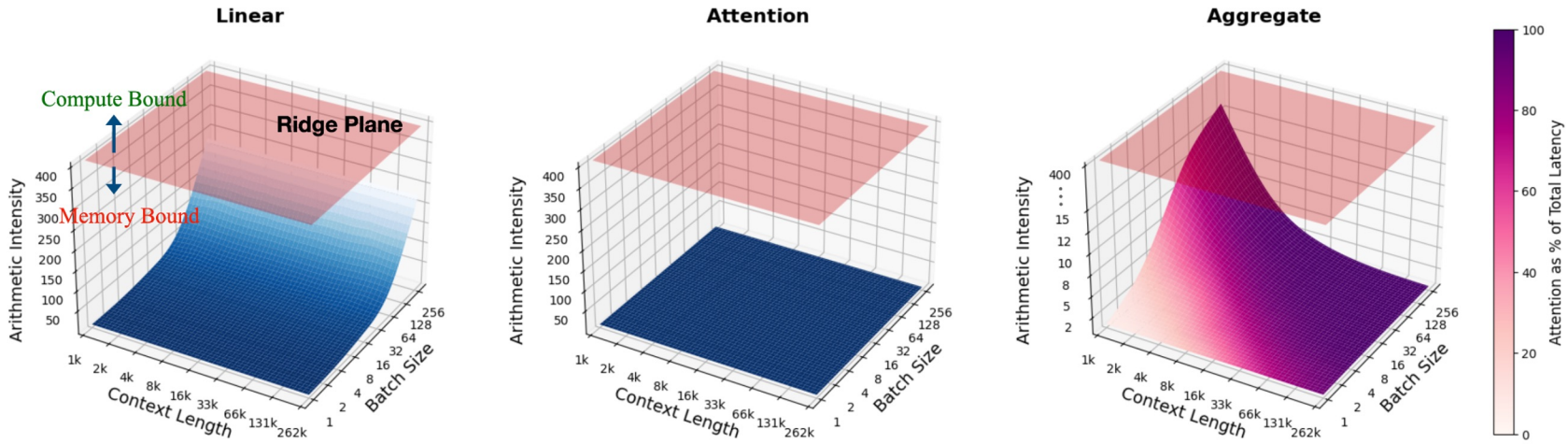




During prefill, all regimes lie above the ridge plane and thus are **compute-bound**.

Tiwari R*, Xi H*, Tomar A*, Hooper C, Kim S, Horton M, Najibi M, Mahoney MW, Keutzer K, Gholami A. QuantSpec: Self-Speculative Decoding with Hierarchical Quantized KV Cache. arXiv:2502.10424.

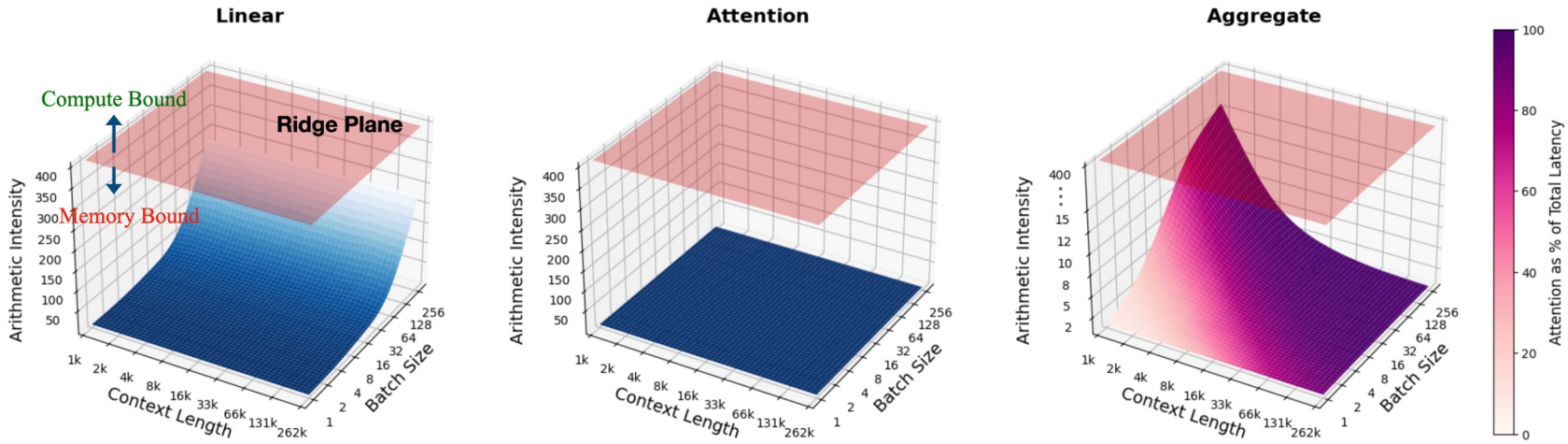
Decode (Llama-2-7B, NVIDIA A6000)



Observations:

- Small batch + short context regime
  - Memory ops for the linear projections dominate
- All other regimes **including long context**:
  - Attention dominates due to the expensive load-store operations for the **large KV cache**

Decode (Llama-2-7B, NVIDIA A6000)



Observations:

- Small batch + short context regime
  - Memory ops for the linear projections dominate
- All other regimes **including long context**:
  - Attention dominates due to the expensive load-store operations for the **large KV cache**

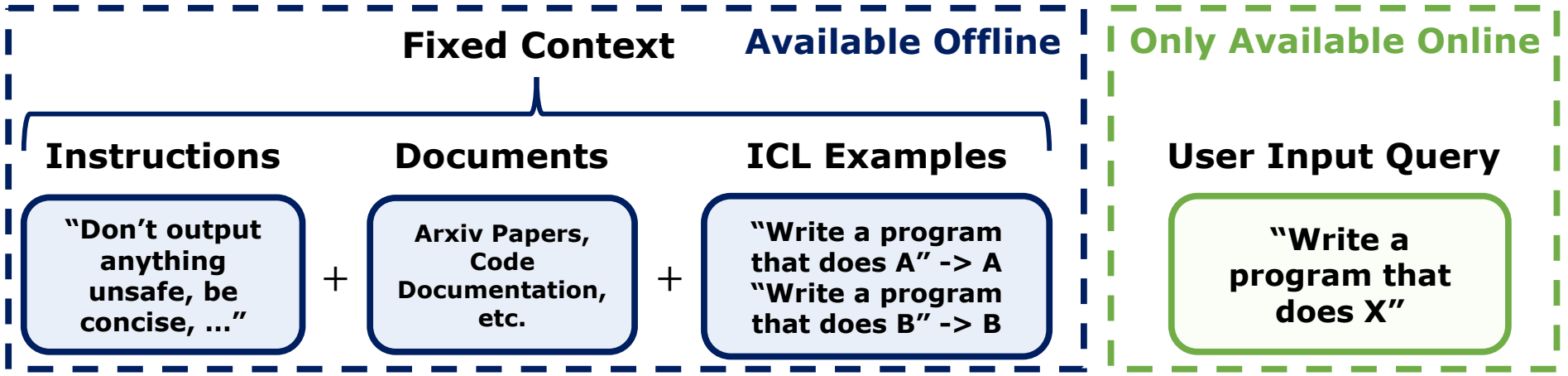
=> Need to find a way to reduce KV Cache size

Agenda

- Inference bottleneck analysis for Transformers
- **Squeezed Attention: Algorithm**
  - Selective Attention
  - Hierarchical Attention
- Results
- Kernel Implementation and Benchmarking
- Conclusion

Fixed-Context LLM Applications

- In emerging LLM applications, many input prompts are concatenated with long *fixed context* (a portion of the input prompt fixed across user queries)
  - Contains system instructions (e.g. “don’t output anything unsafe”)
  - Contains documents or documentation
  - May also contain few-shot in-context examples for the target task



Idea: Accelerating Attention to the Fixed Context

- Performing the computation for attention to the fixed context is expensive, and it limits what information you can include to personalize your LLM
- **However, the fixed context tokens are fixed for successive user queries**

  💡 Idea: Can we **preprocess the fixed context offline** to reduce its overhead during inference?

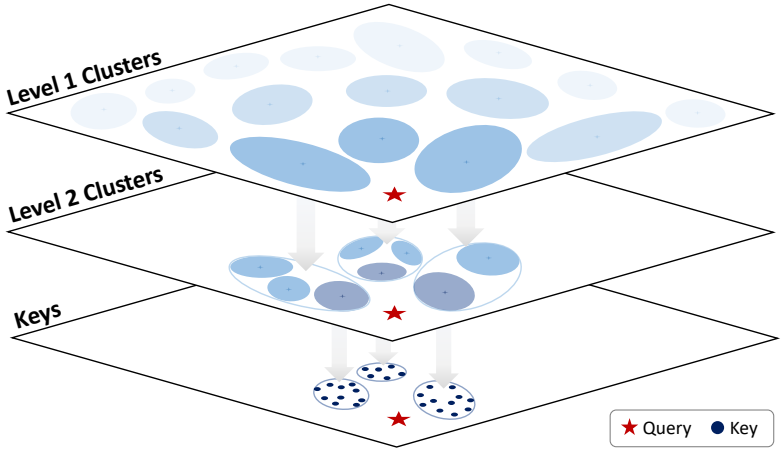
  💡 The overhead here is loading KV Cache

  💡 If we could quickly find important KV Cache values and only load them, that would result in both efficient and accurate inference

Challenge here is that important KV Cache depends on the user query

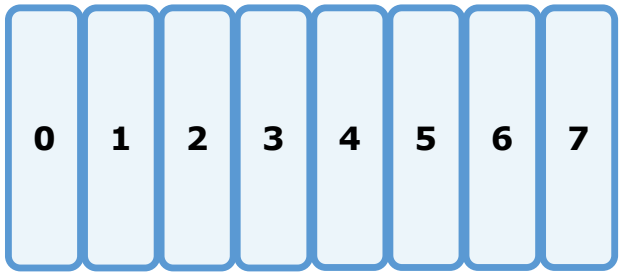
Approach: Query-Aware Sparse Attention

- What keys are important depends on the query
- If we could preprocess the KV Cache so we can quickly filter out and zoom in to the important values we can only load them

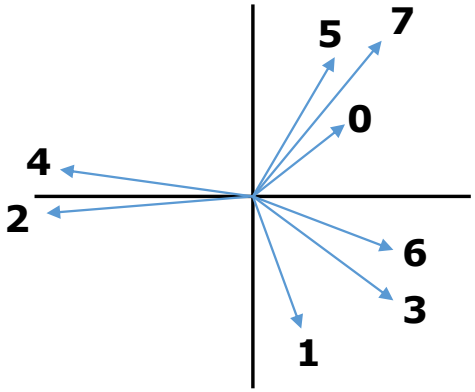


Fixed Context Keys

- Fixed Context keys shown below (as well as a visualization of the directions that they point in embedding space)

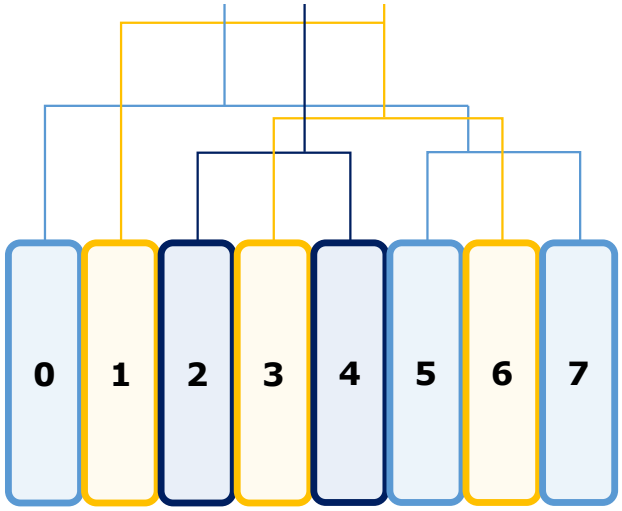


Fixed Context Keys

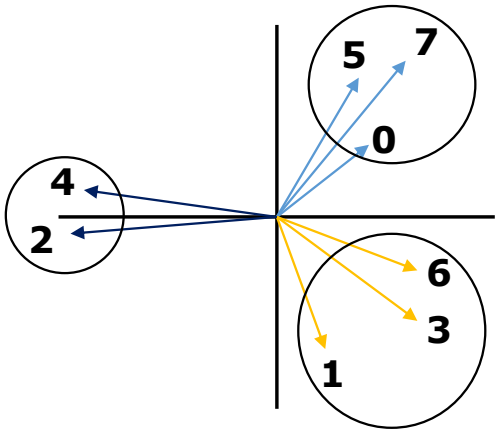


Cluster Keys Offline

- Use clustering based on cosine similarity to group together keys which point in a similar direction
  - **Idea is that keys which point in the same direction will have correlated dot products with query vectors**

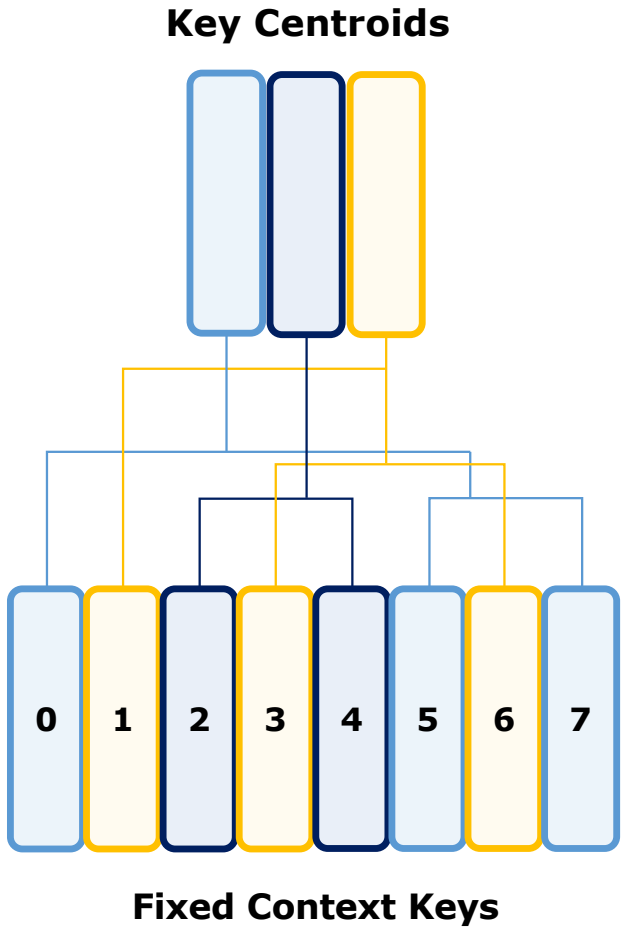


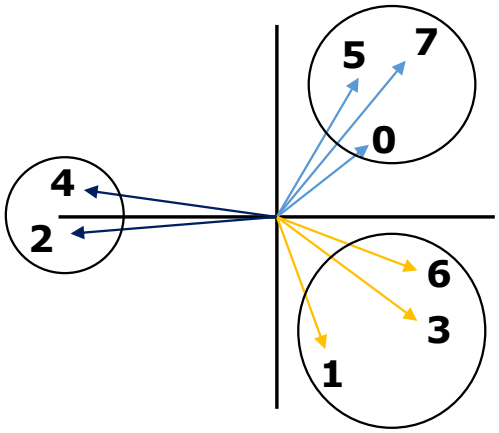
Fixed Context Keys



Cluster Keys Offline

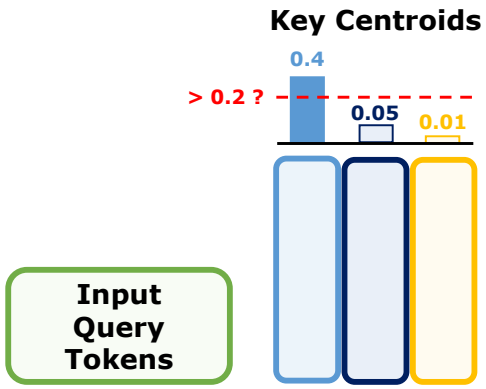
- Clustering based on cosine similarity to group together keys
- Compute a centroid for each cluster
  - We can compare with the centroid to tell us if the keys in that cluster will have high attention scores





Comparing with Centroids

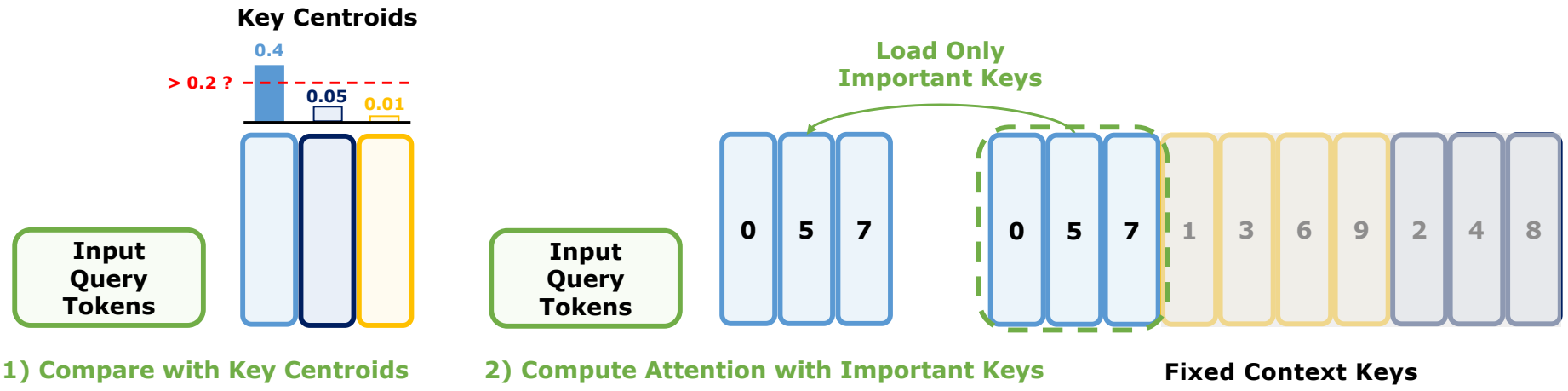
- First, compare the input query tokens with each of the key centroids



1) Compare with Key Centroids

Only Loading Important Keys

- First, compare the input query tokens with each of the key centroids
- Use the centroid score to estimate the expected attention score for each token
- Compare with threshold (after SoftMax) to decide whether to retain each key

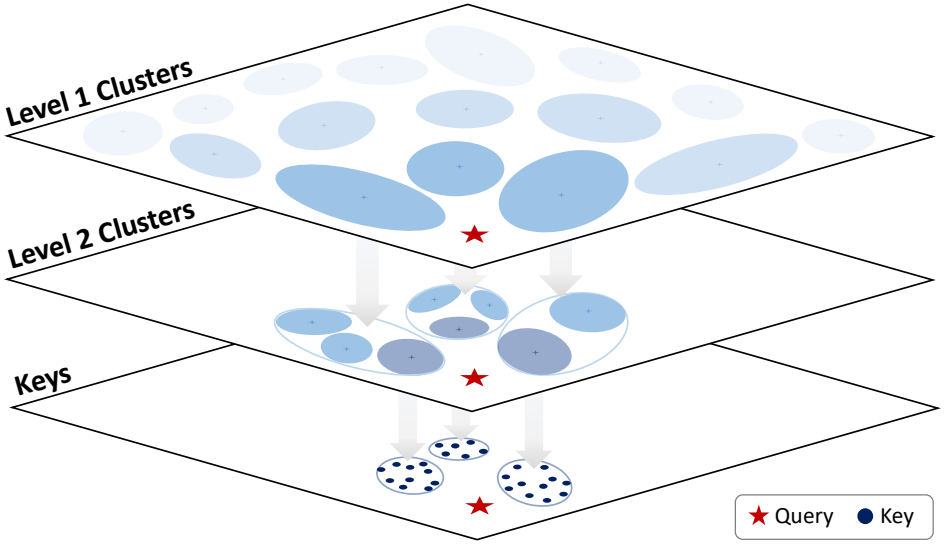


Agenda

- Inference bottleneck analysis for Transformers
- **Squeezed Attention: Algorithm**
  - Selective Attention
  - **Hierarchical Attention**
- Results
- Kernel Implementation and Benchmarking
- Conclusion

Extending to Hierarchical Centroid Lookup

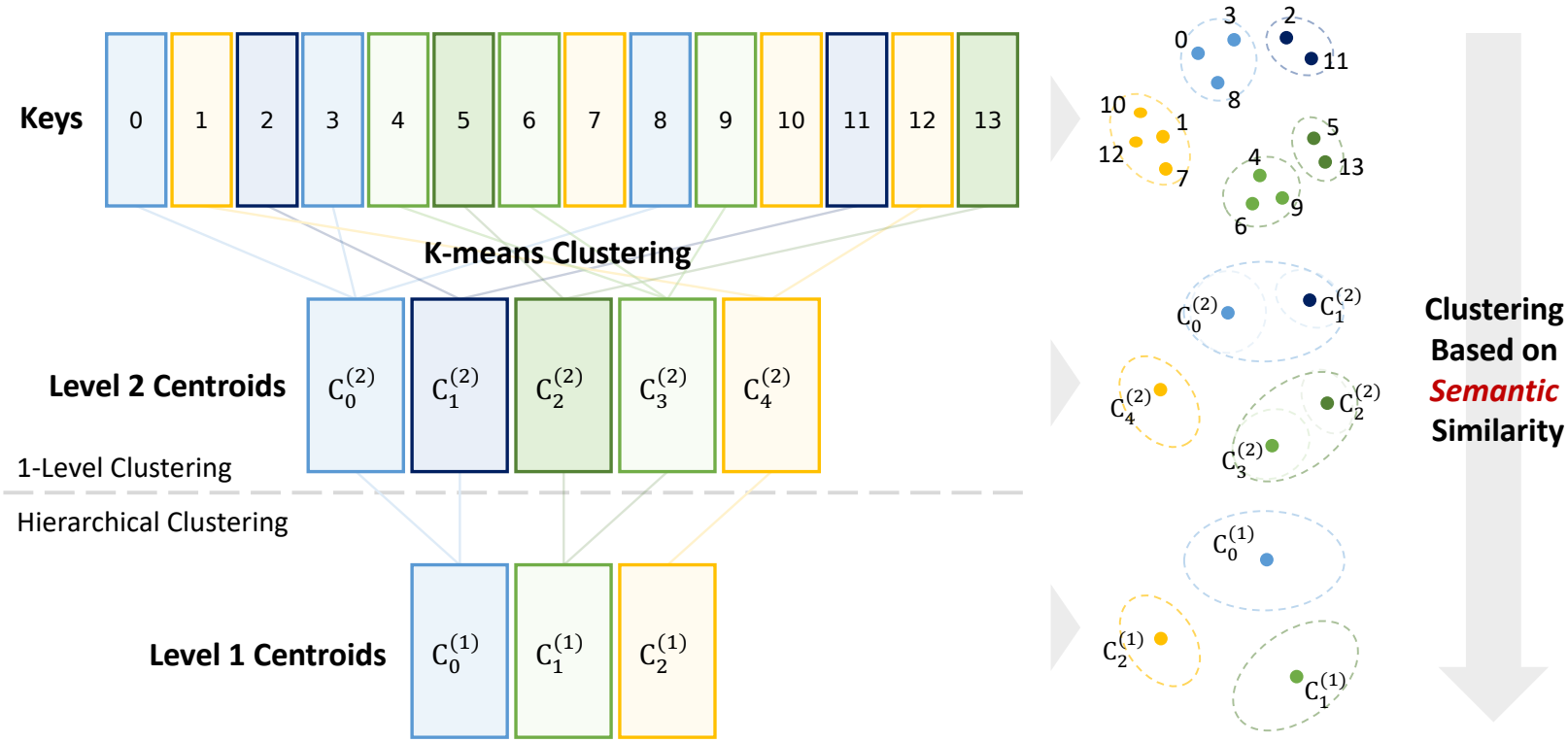
- Hierarchical lookup can improve the resolution of the clusters, **while still being efficient by only comparing with a subset of the later clusters**
- If we use c' clusters at each level and retrieve a subset of these clusters at each step, we need $\boldsymbol{O(\log(N))}$ hierarchical levels to reduce the keys to the desired count $\boldsymbol{k}$
- **The complexity becomes $\boldsymbol{O(c'\log(N) + k)}$**



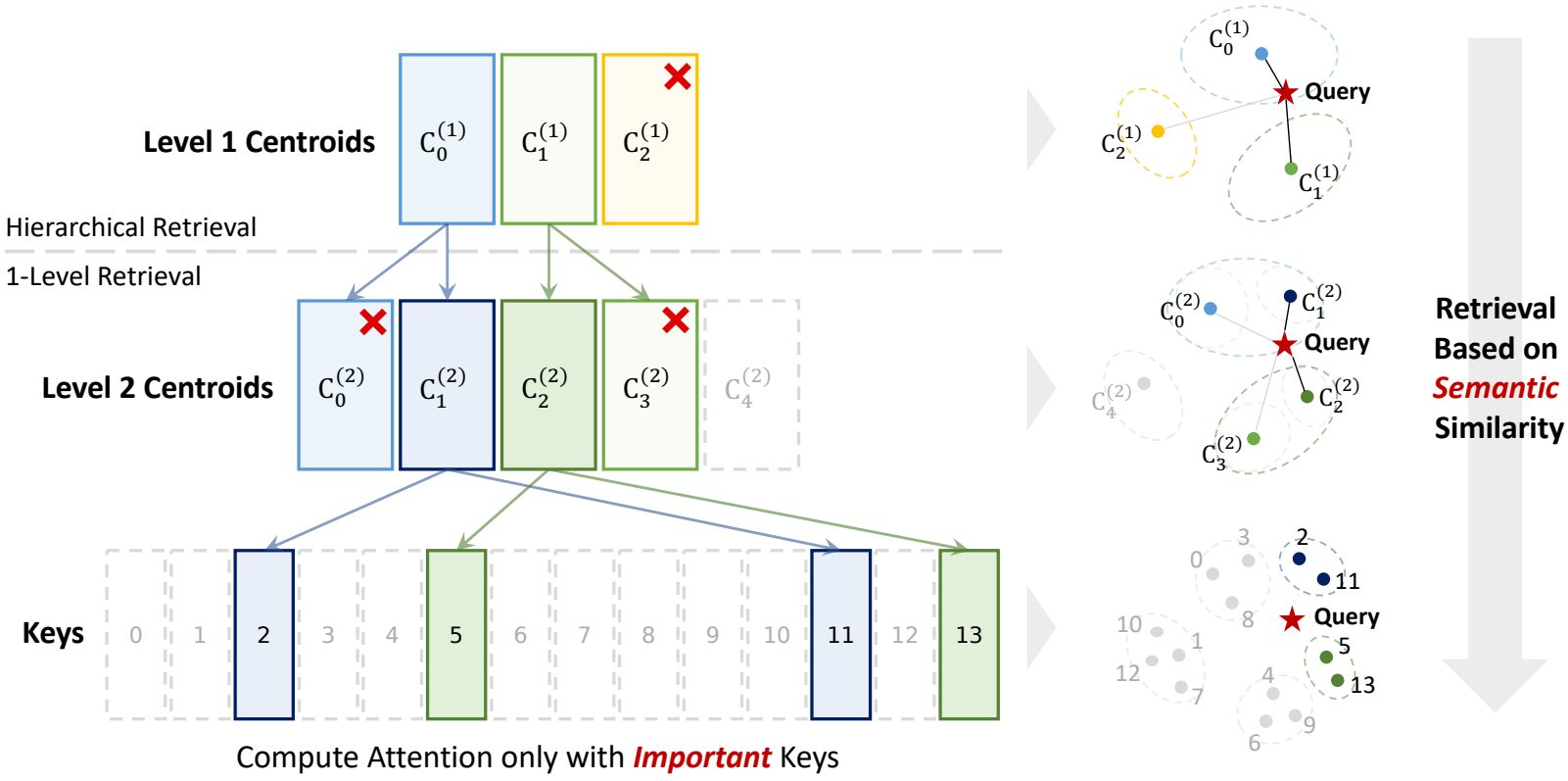
Approach	Complexity
Standard Attention	$O(N)$
SqueezedAttention (1-Level)	$O(c + k)$
SqueezedAttention (Hierarchical)	$\boldsymbol{O(c'\log(N) + k)}$

Complexity Analysis for SqueezedAttention

Full Method (Offline)



Full Method (Online)



Compute Attention only with ***Important*** Keys

Agenda

- Inference bottleneck analysis for Transformers
- Squeezed Attention: Algorithm
  - Selective Attention
  - Hierarchical Attention
- **Results**
- Kernel Implementation and Benchmarking
- Conclusion

LongBench Results

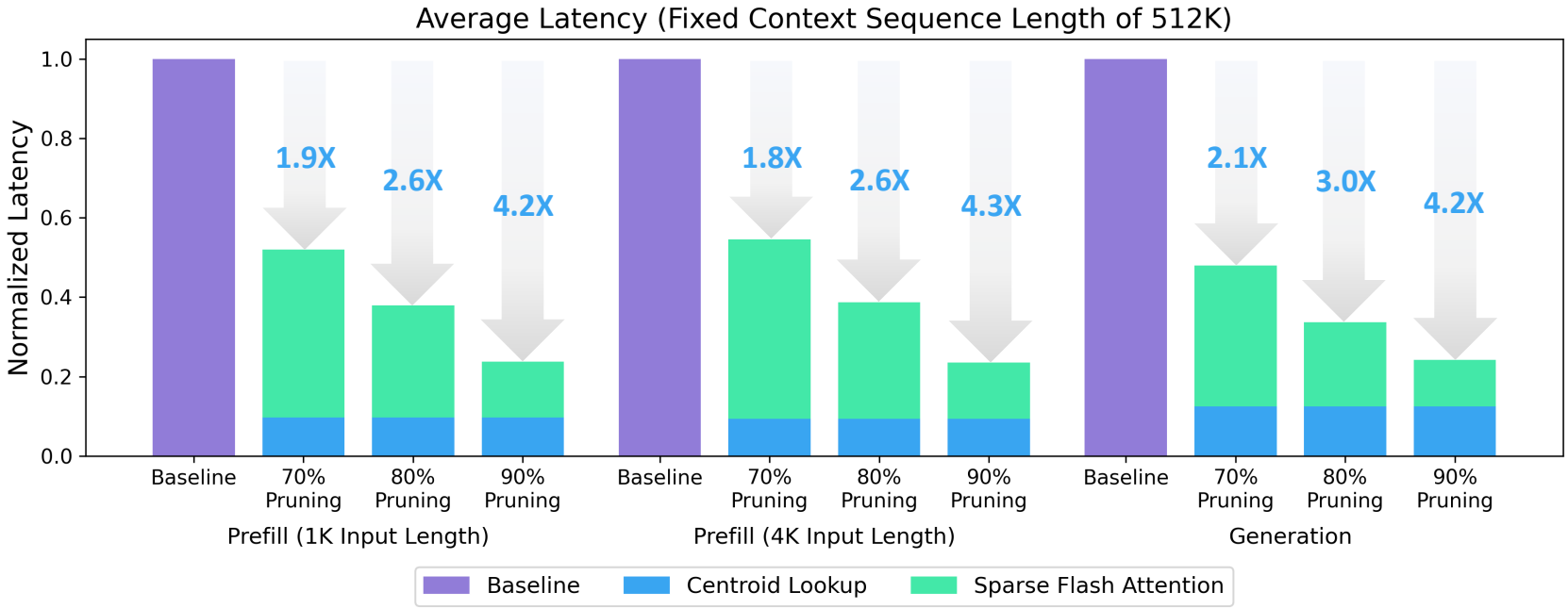
- Accuracy result shown for different LongBench tasks (LLaMA-2-7B-32K model)
  - QUEST is shown as baseline (groups tokens sequentially and then performs approximate sparse attention)
  - We attain higher accuracy than the baselines with significantly lower KV cache budget, and our hierarchical method also maintains comparable accuracy

Config	Budget	Single-Document QA			Multi-Document QA			Summarization			Few-shot Learning			Code		Avg.	
		NQA	Qasper	MFQA	Hotpot	2Wiki	Musique	GRep	QMSum	MNews	TREC	TQA	SSum	RBench	LCC	w/o SSum	All
All KV	1	17.91	11.12	33.87	12.45	11.95	6.54	29.37	16.93	21.58	71.50	87.96	43.87	61.45	59.14	33.98	34.69
Squeeze-70%	0.325	18.55	11.78	34.33	12.31	12.31	6.26	29.50	16.90	20.76	69.00	87.96	43.90	61.29	59.53	33.88	34.60
QUEST	0.168	20.42	9.72	29.46	11.45	9.75	5.46	27.06	17.20	21.83	68.50	86.36	-	61.93	59.38	32.96	-
Squeeze-90%	0.125	18.15	14.39	32.38	11.84	11.70	6.45	29.06	16.93	21.66	70.00	87.43	45.15	58.79	59.37	**33.70**	34.52
H-Squeeze-90%	0.112	17.41	14.23	32.71	11.99	11.38	6.68	29.14	16.97	20.41	68.00	87.37	44.85	58.94	59.61	33.45	34.26

LongBench accuracy for different % of tokens pruned, using 5% centroids (one centroid per 20 tokens on average)

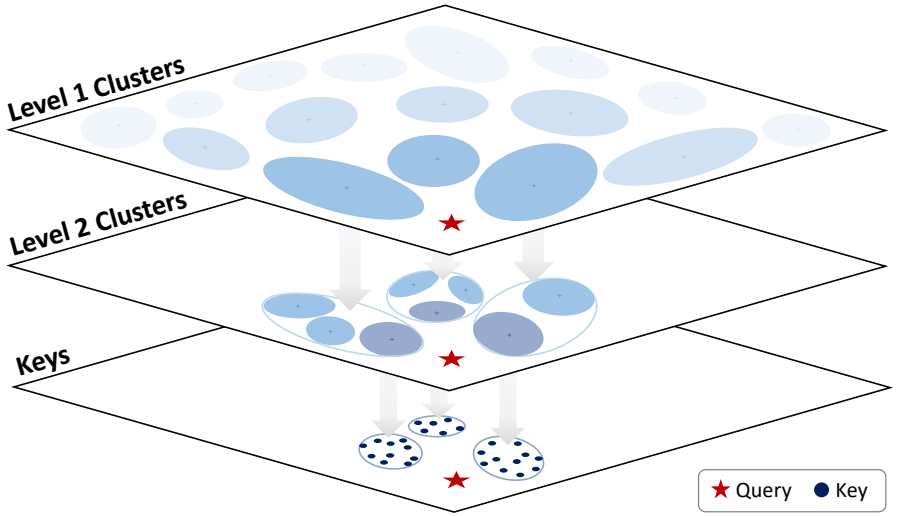
Kernel Implementation

- We design custom Triton kernels for the centroid lookup and sparse FlashAttention computations, and benchmark these against the FlashAttention kernel
  - We benchmark these on H100 NVL platform, using 512K Fixed Context length
  - **Results show (normalized) speedups relative to FlashAttention using our method**



Summary

- **Asymptotic Analysis for Transformer Inference**
- **Fast attention to the fixed context** in the input prompt
- Hierarchical method also **reduces the overhead of the centroid comparison**
- Significant speedups (**4.3x/4.2x for prefill/generation**) with minimal accuracy degradation



Thank you!

- **QuantSpec paper:** https://arxiv.org/pdf/2502.10424
- **Squeezed Attention Paper:** https://arxiv.org/pdf/2411.09688
- **Squeezed Attention Code:** https://github.com/SqueezeAILab/SqueezedAttention