Approximate Attention

Abhinav Bhatele, Daniel Nichols

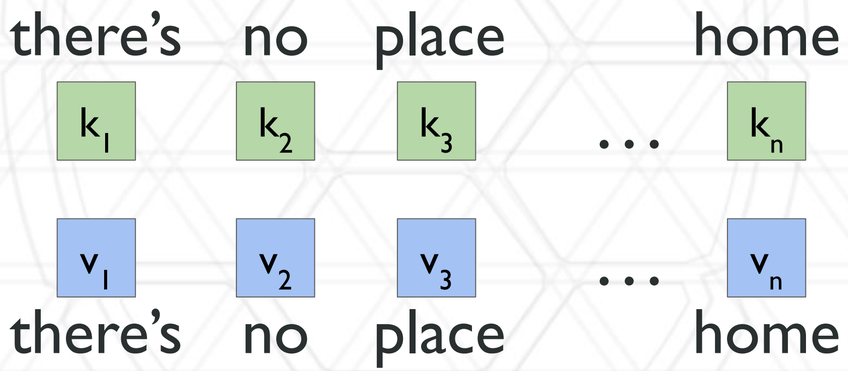
UNIVERSITY OF MARYLAND

Announcements

- Interim report for the project is due on ~~April 17~~ April 22
- Midterm is on April 10
- Office hours
  - Today 1-2pm IRB 5237
  - Wednesday 12-2pm Zoom (link on piazza)

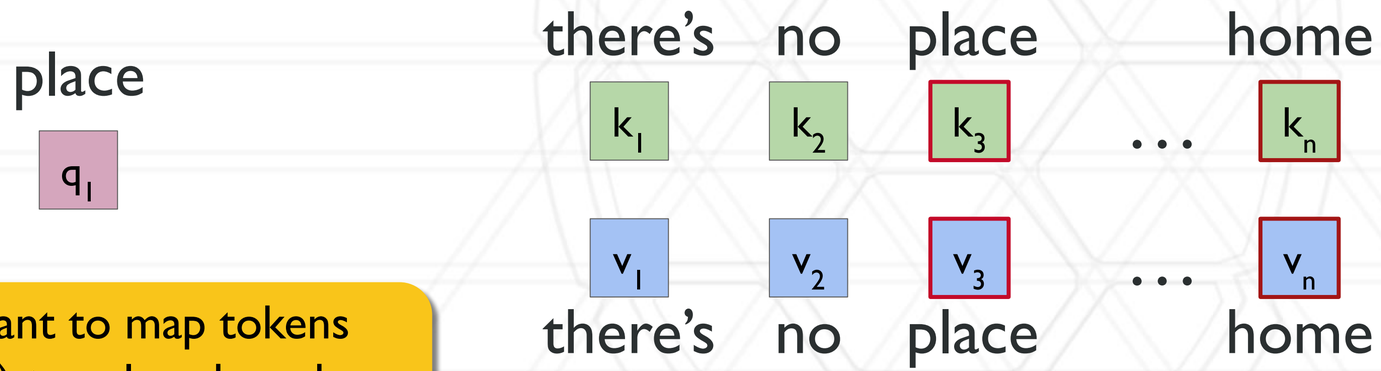
Attention Reminder

- Determine how much tokens should "attend" to other tokens
- Consider a hashmap of our tokens



Attention Reminder

- Determine how much tokens should "attend" to other tokens
- Consider a hashmap of our tokens



We want to map tokens (queries) to values based on the most similar keys

Attention Reminder

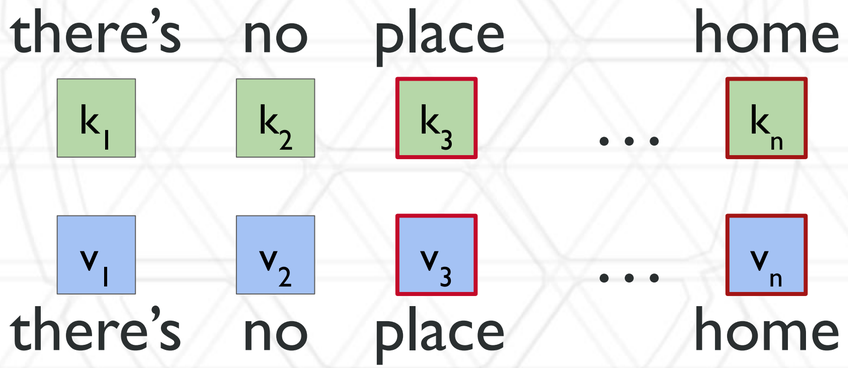
- Determine how much tokens should "attend" to other tokens
- Consider a hashmap of our tokens

We can answer this query with a scaled dot product of the similar values

place

q_1

$0.6*v_3+0.4*v_n$



But how much should we scale this dot product?

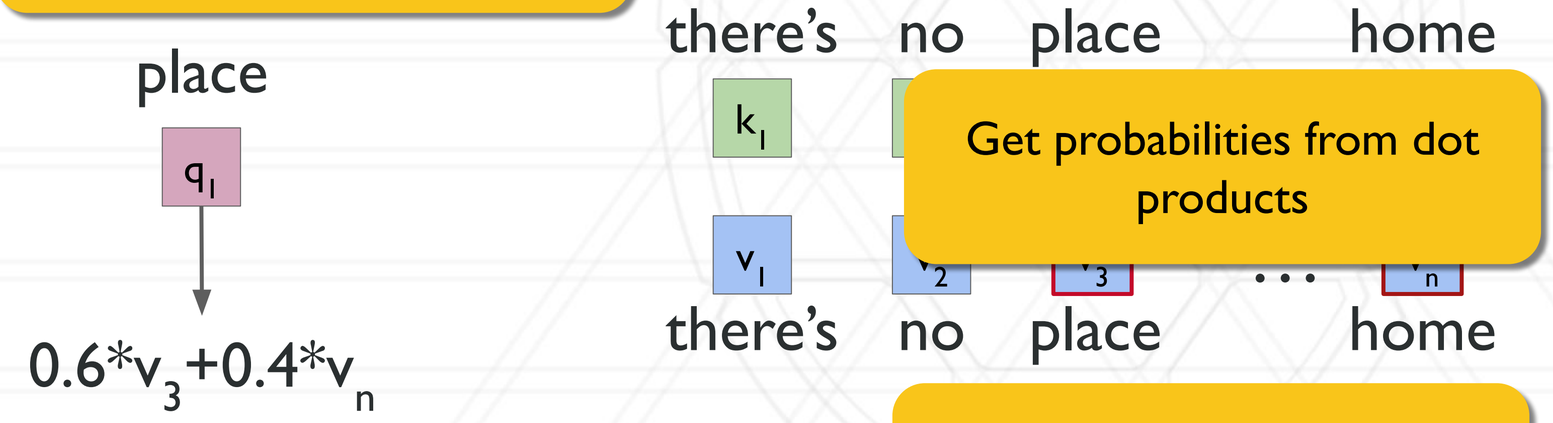
Attention Reminder

- Determine how much tokens should "attend" to other tokens
- Consider a hashmap of our tokens

We can answer this query with a scaled dot product of the similar values

Compute dot product of the query with all the keys

$$O = [q_i k_1, q_i k_2, \ldots, q_i k_n]$$



Get probabilities from dot products

$$P = \text{softmax}(O)$$

But how much should we scale this dot product?

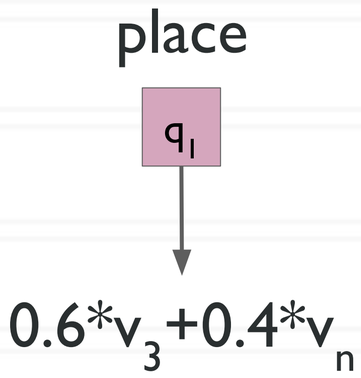
Scale the values based on probabilities

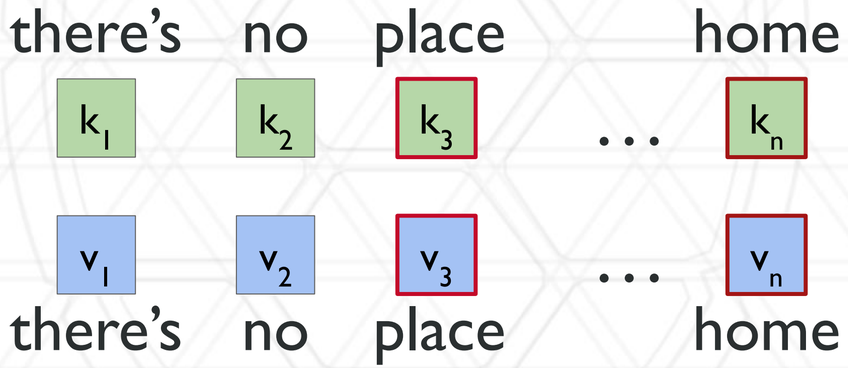
$$x_i = P[v_1, \ldots, v_n]^T$$

Attention Reminder

- Determine how much tokens should "attend" to other tokens
- Consider a hashmap of our tokens

$$O = [q_i k_1, q_i k_2, \ldots, q_i k_n]$$





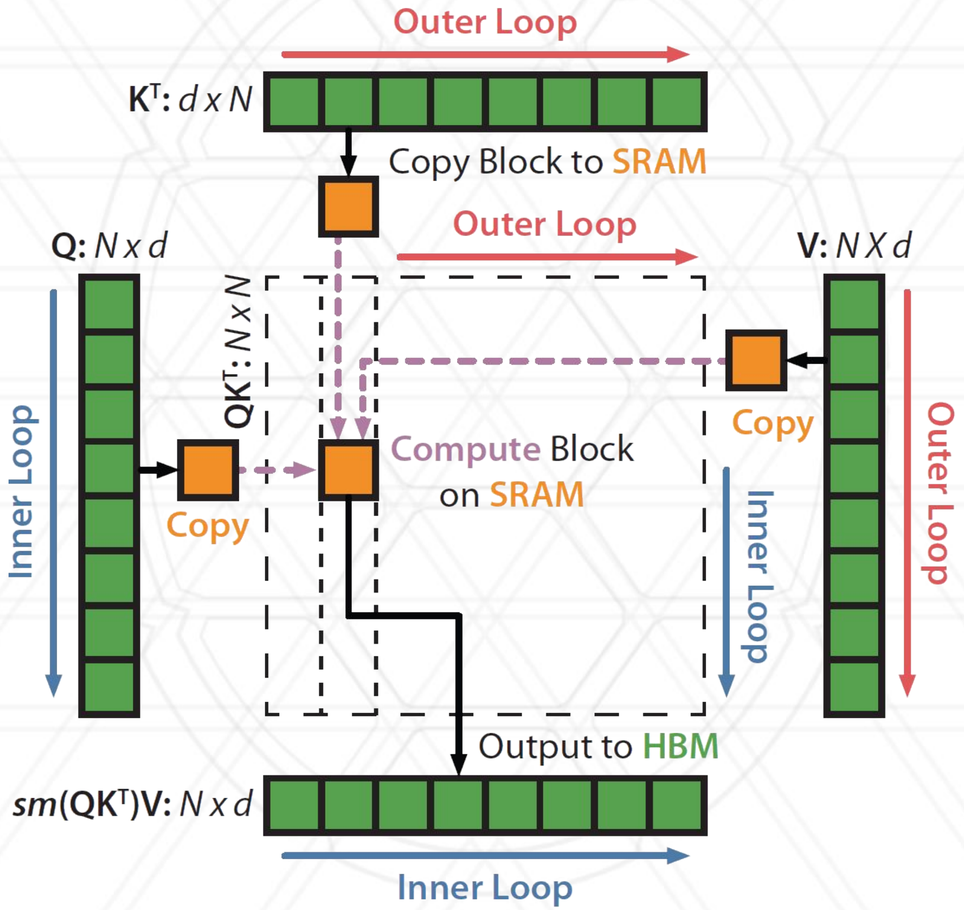
$$P = \text{softmax}(O)$$

This is $O(N^2)$ in the sequence length!

$$x_i = P[v_1, \ldots, v_n]^T$$

Addressing Quadratic Memory Constraints

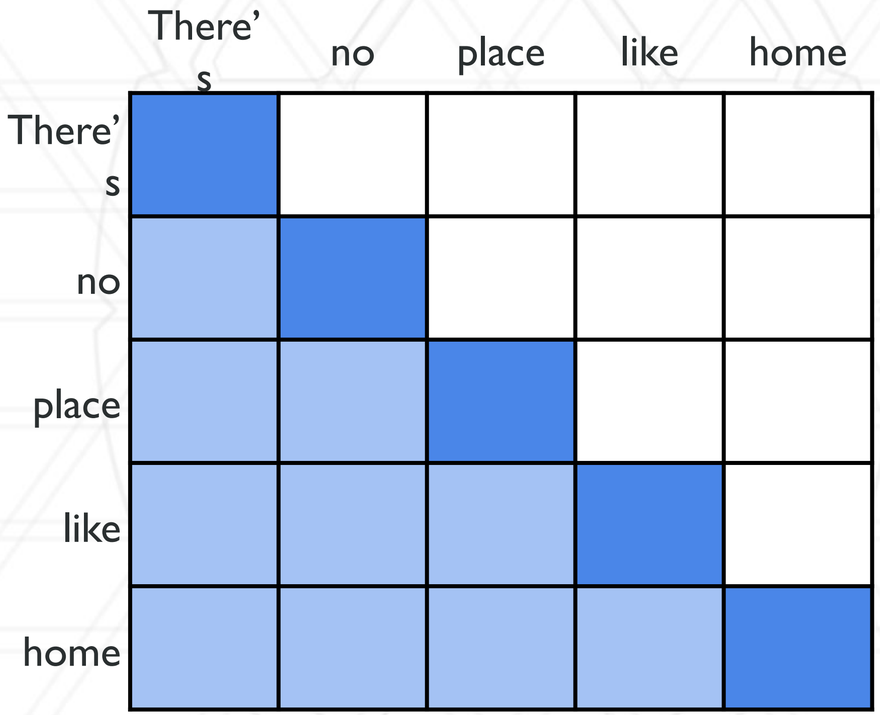
FlashAttention: We've seen methods to avoid the NxN space complexity



Reducing Complexity in Attention

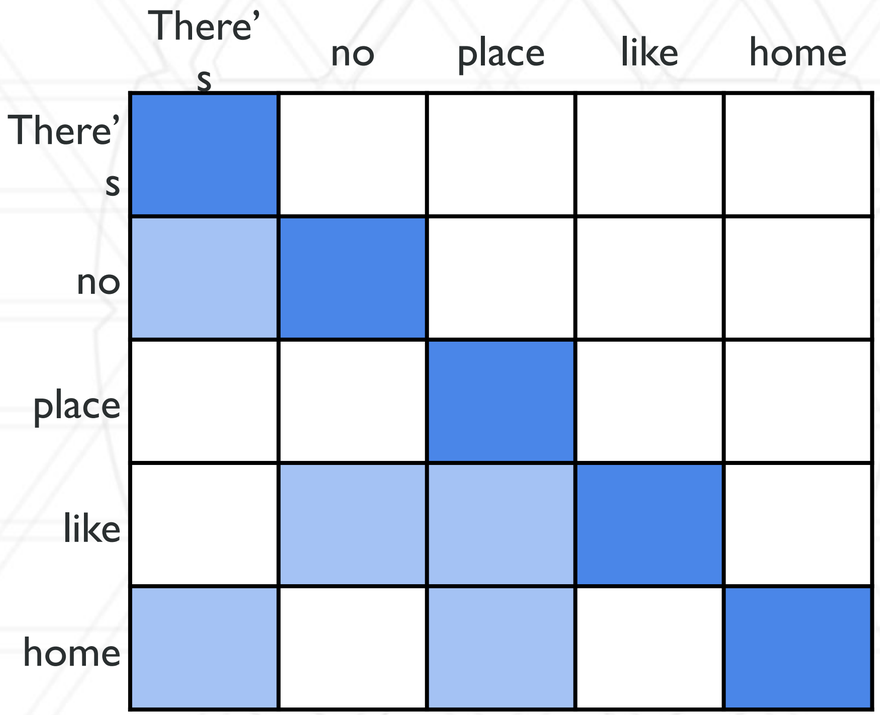
Reducing Complexity in Attention

We need to compute how much each token attends to the others.



Reducing Complexity with Sparsity

We need to compute how much each token attends to the others.



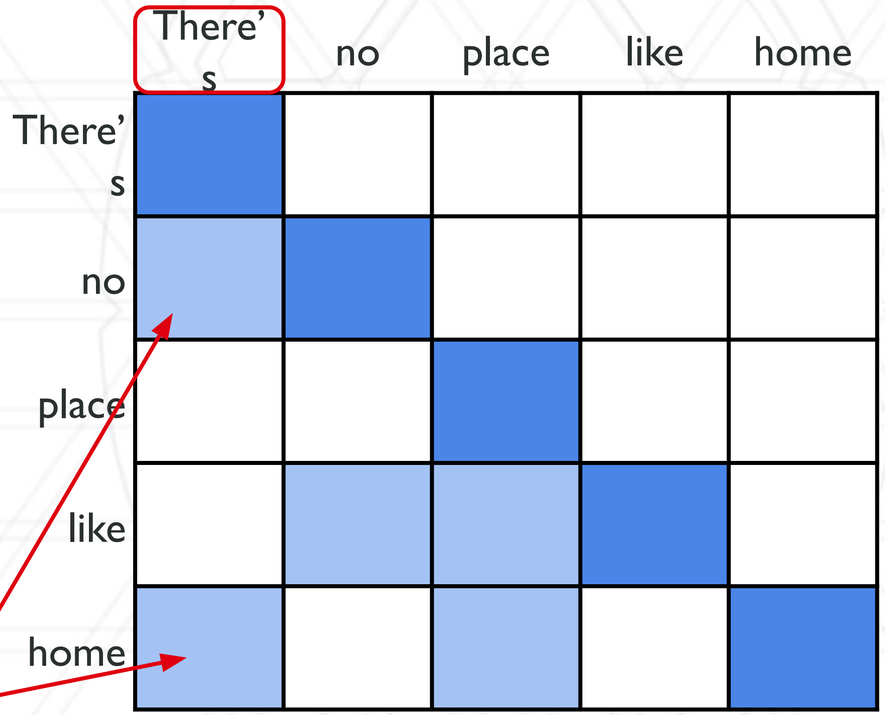
In practice, the attention matrix is *near* sparse

Reducing Complexity with Sparsity

We need to compute how much each token attends to the others.

So for a particular query…

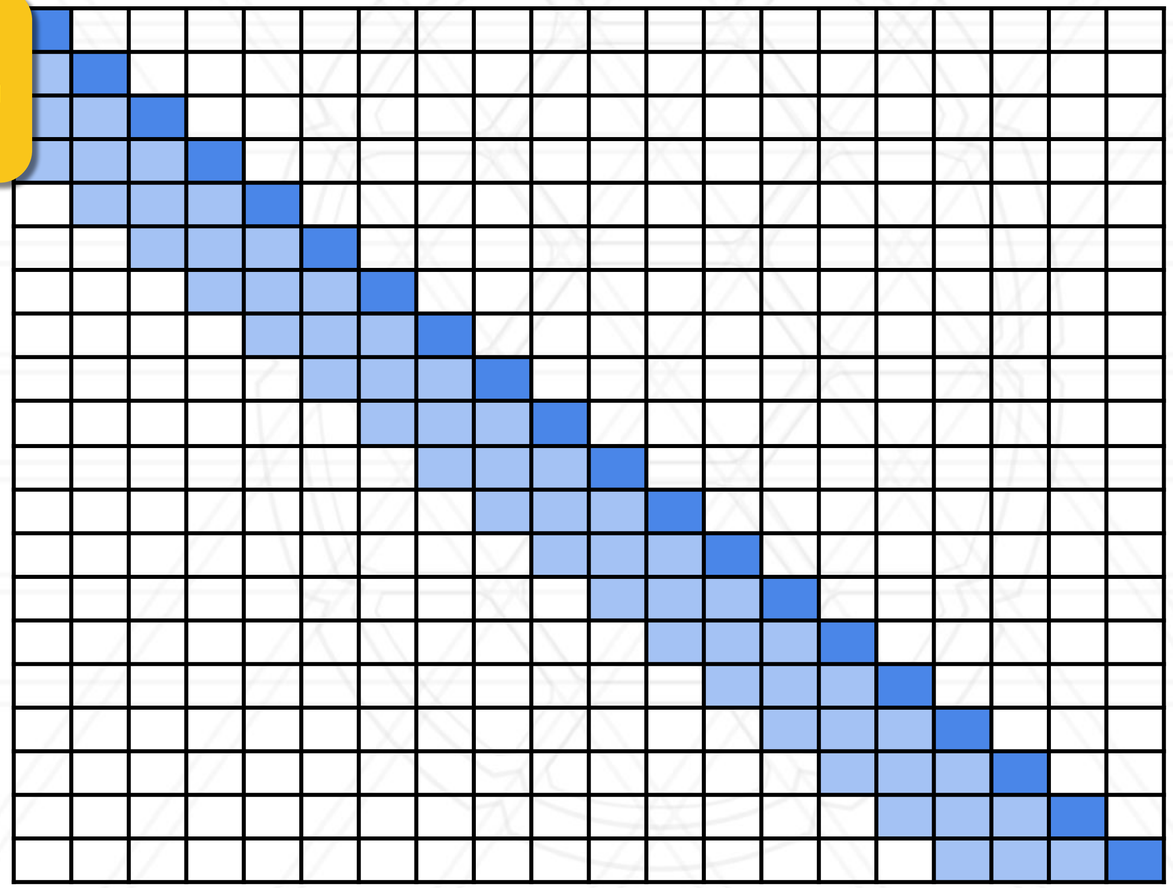
How do we determine what other tokens are important to attend to?



In practice, the attention matrix is *near* sparse

Reducing Complexity with Locality

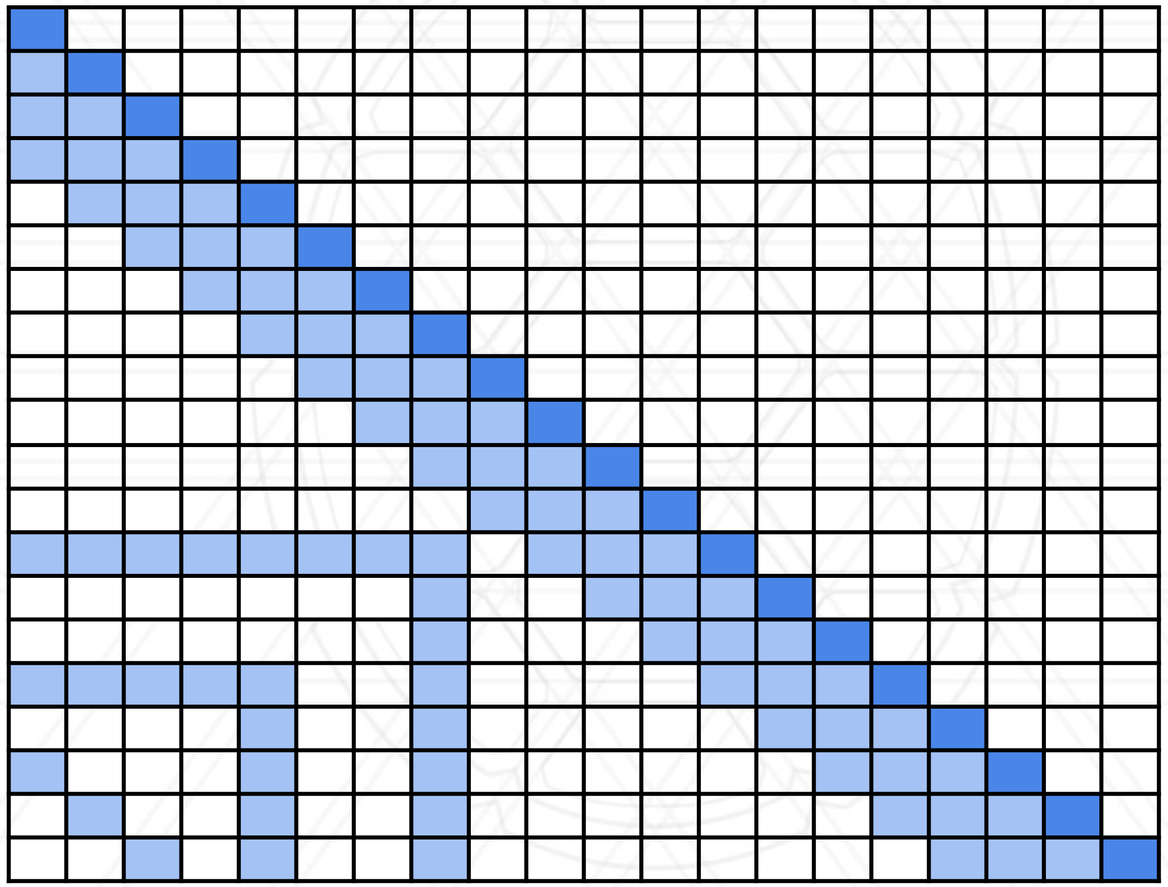
Often in language we care most about near words when interpreting something



We can attend only to a neighborhood of tokens

Reducing Complexity with Structured Sparsity

We can try to find some structured sparse format that…

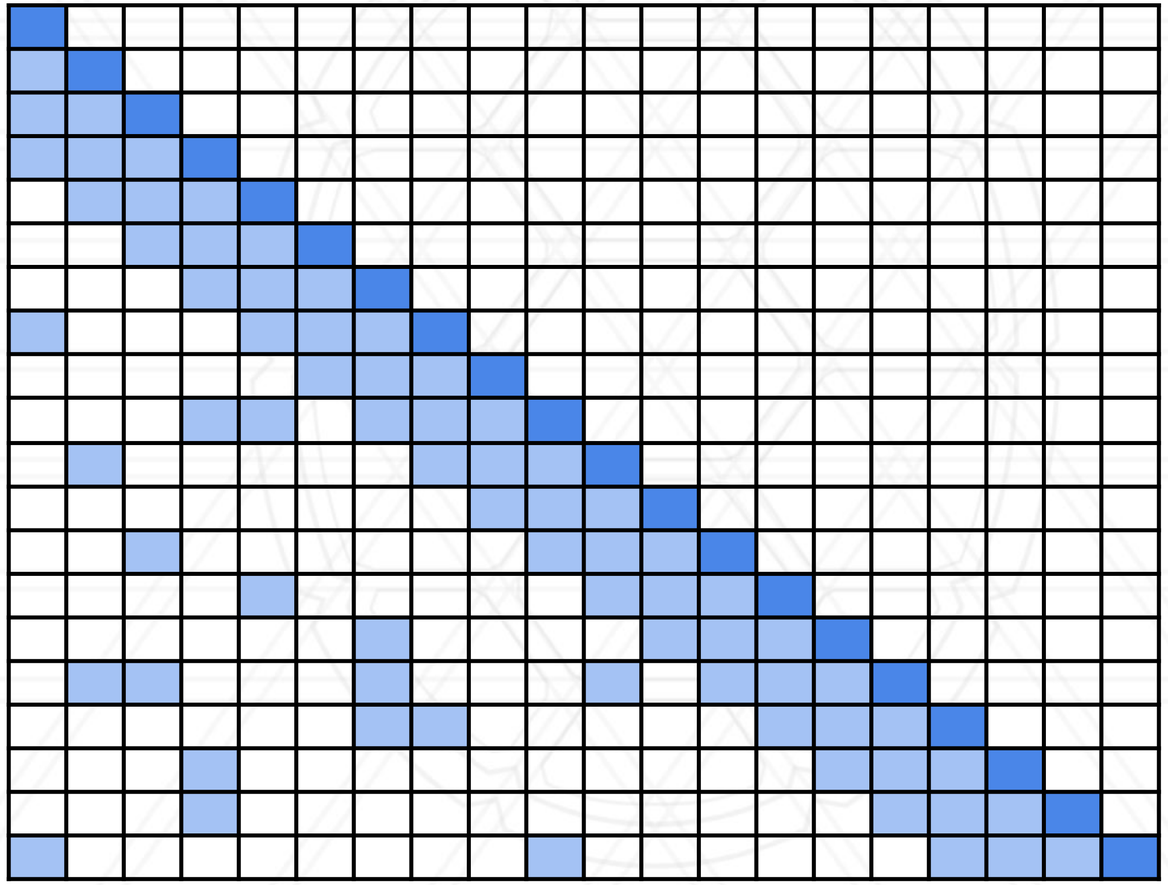


reduces compute and/or memory requirements…

and preserves accuracy.

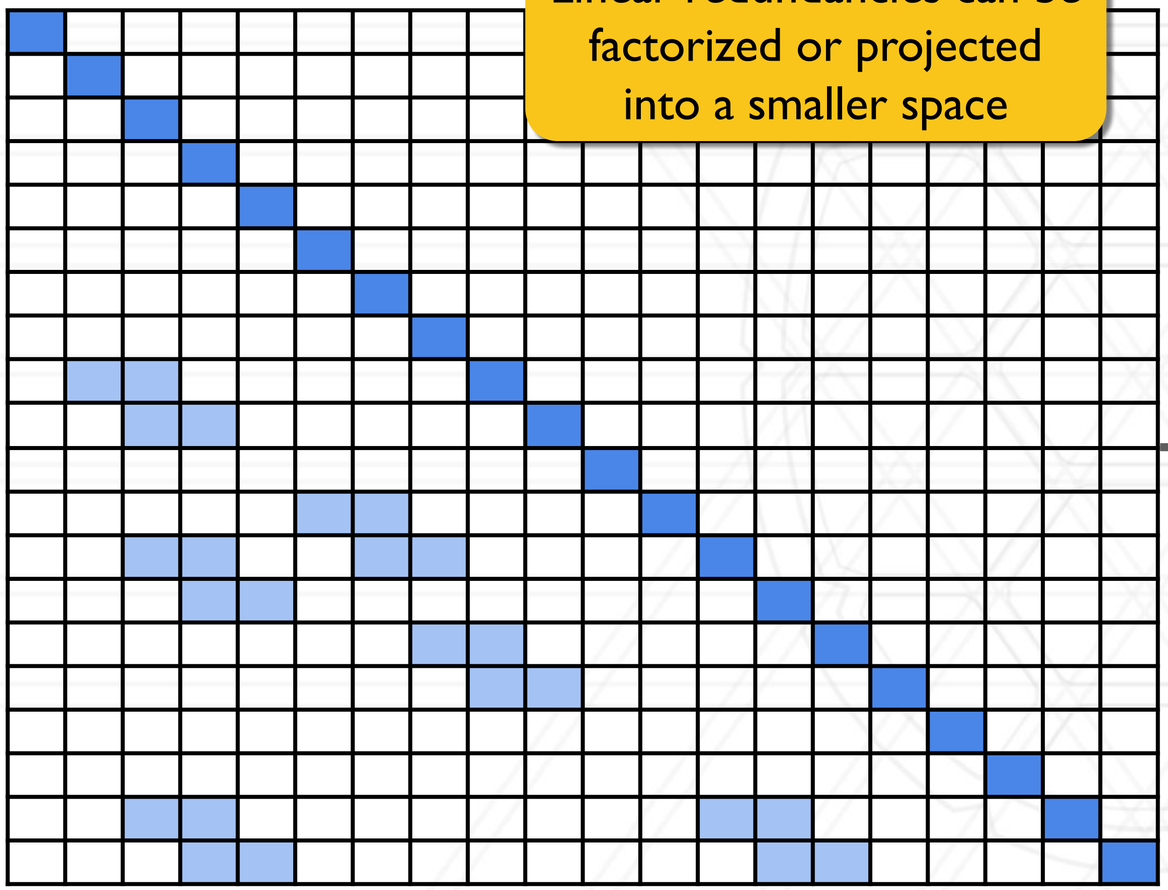
Reducing Complexity with Dynamic Sparsity

We can figure out the sparsity at runtime



Trade-off of better sparsity pattern vs more memory or compute

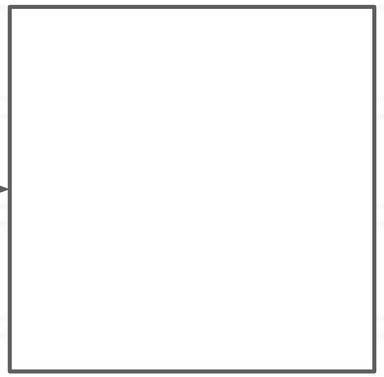
Reducing Complexity from Repeated Structure



Linear redundancies can be factorized or projected into a smaller space

Repeated patterns can be compressed

compression or dimensionality reduction



How does this help us?

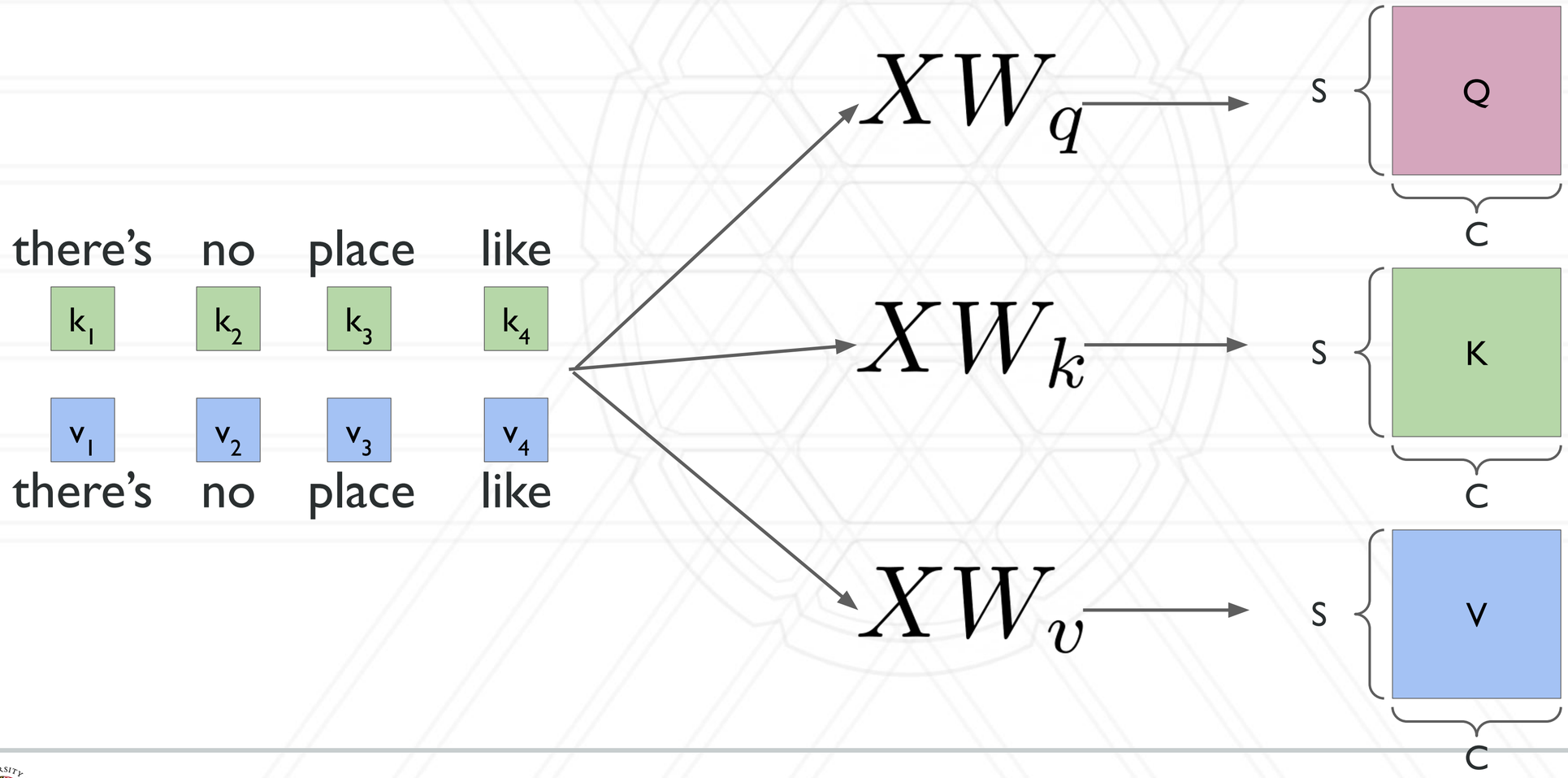
- We do not need to compute all of the attention values
  - Potentially sub-quadratic attention
- We do not need to store the whole KV-cache!
  - This can get quite large, becoming a memory bottleneck

Benchmarking Approximate Algorithms

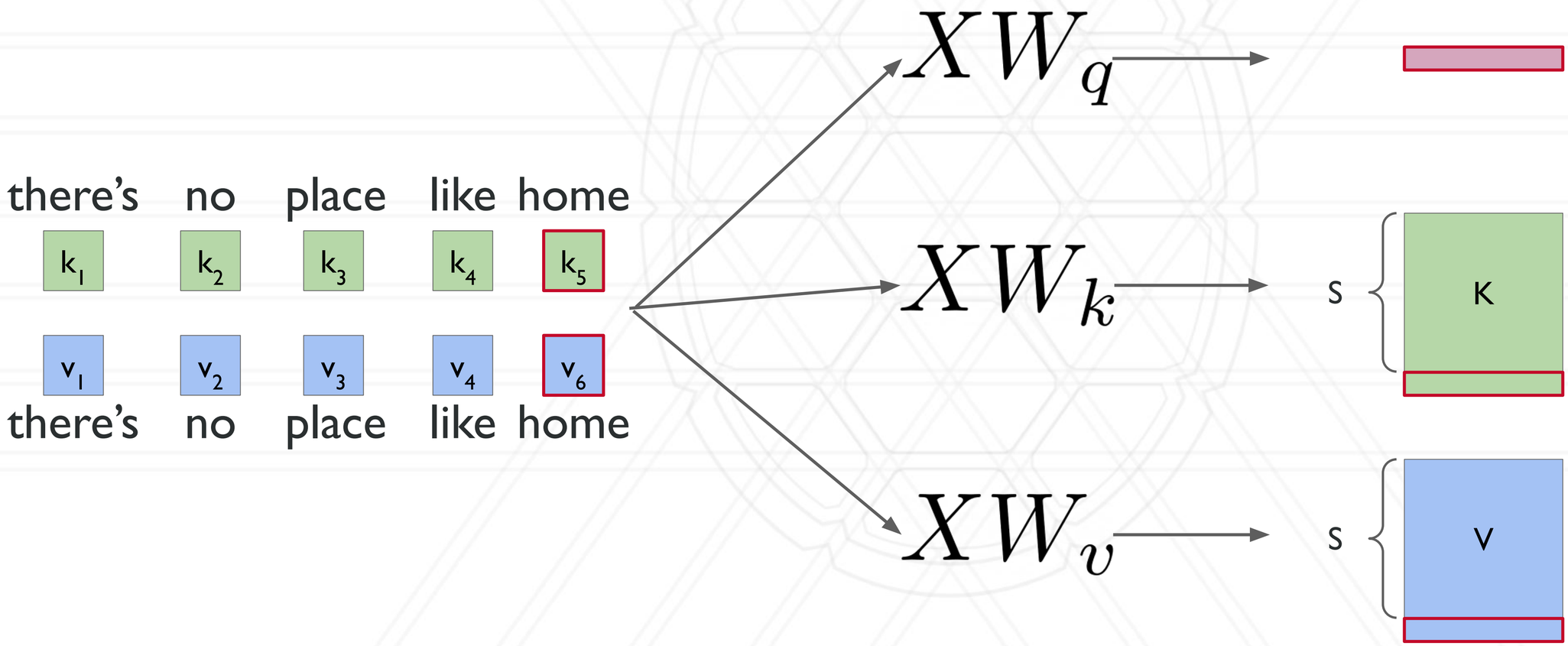
- Accuracy/correctness is crucial to benchmark!
- How do we measure accuracy of approximate attention?
  - perplexity of model
  - performance on downstream benchmarks

$$\text{perplexity}(X) = \exp\left\{-\frac{1}{S}\sum_{i}^{S}\log p_\theta\left(x_i|x_0, x_1, \ldots x_{i-1}\right)\right\}$$

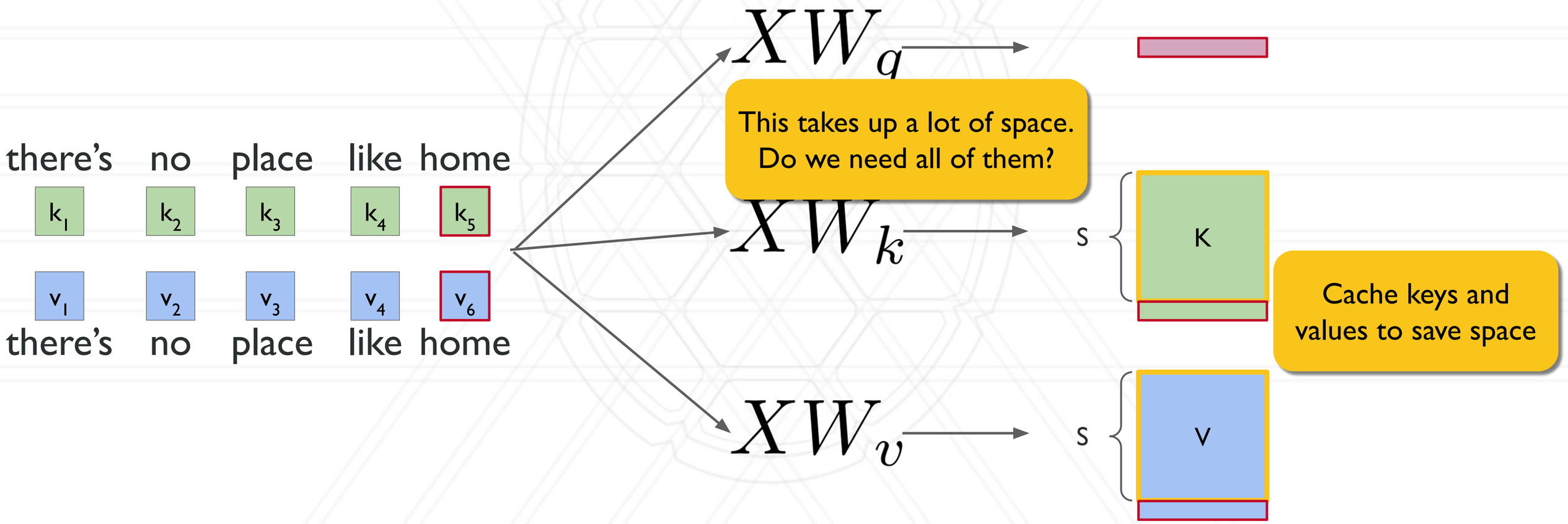
K-V Cache Eviction



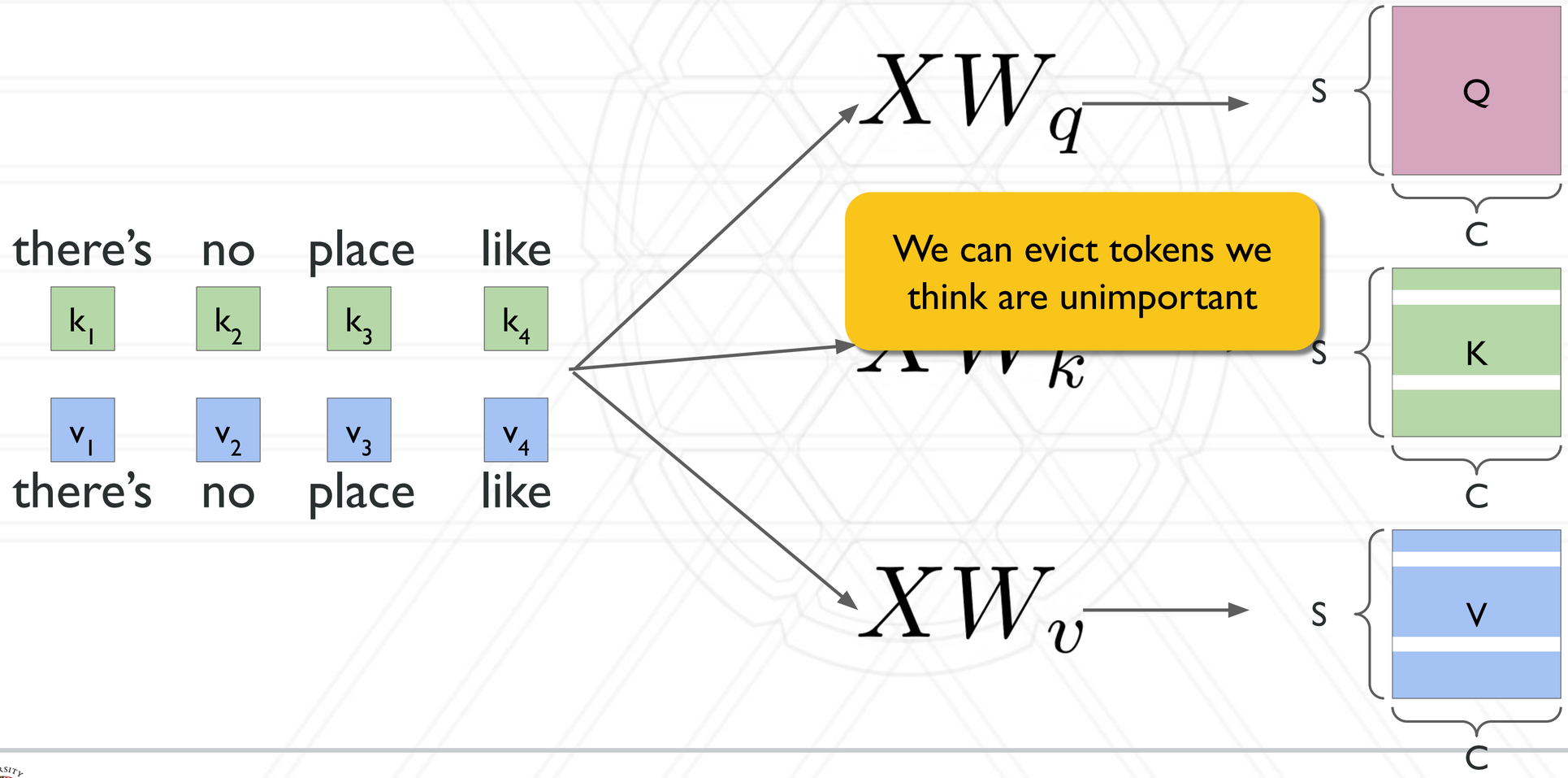
K-V Cache Eviction



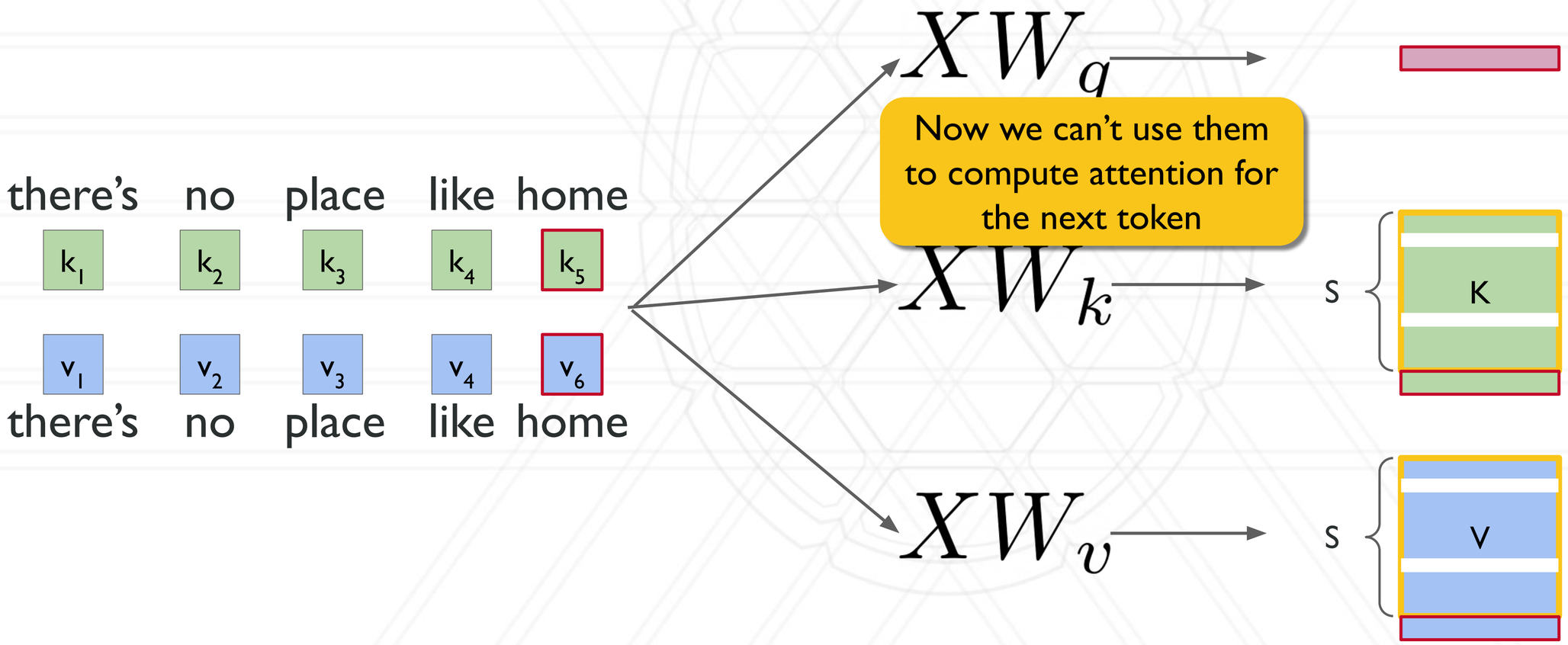
K-V Cache Eviction



K-V Cache Eviction



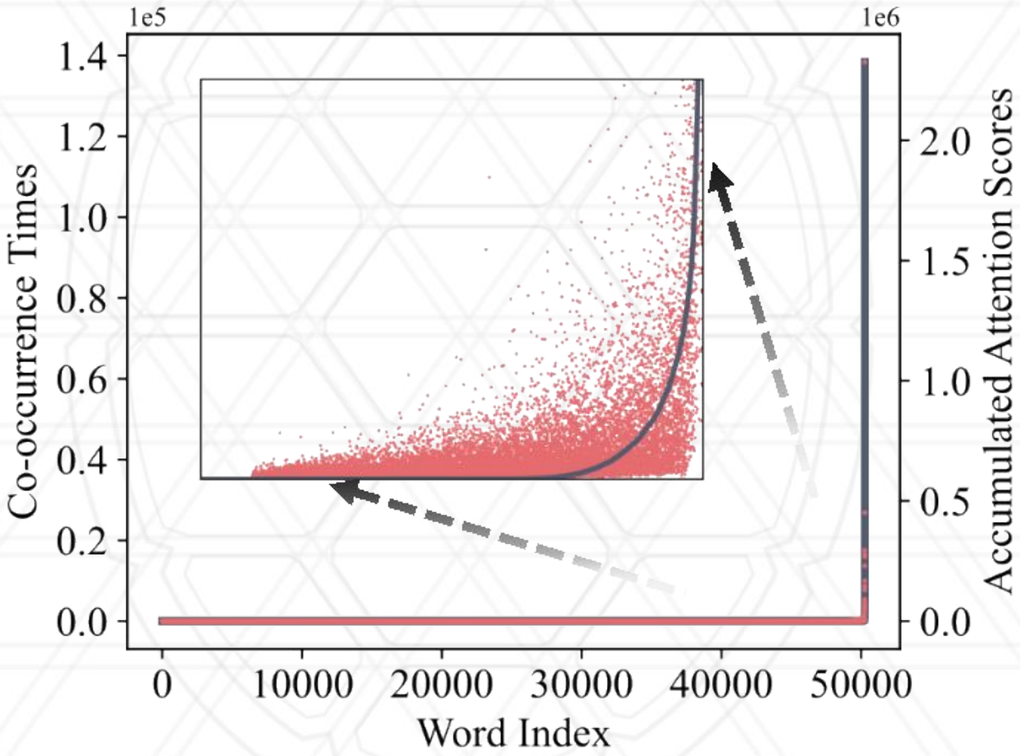
K-V Cache Eviction



H2O: Approximate Attention

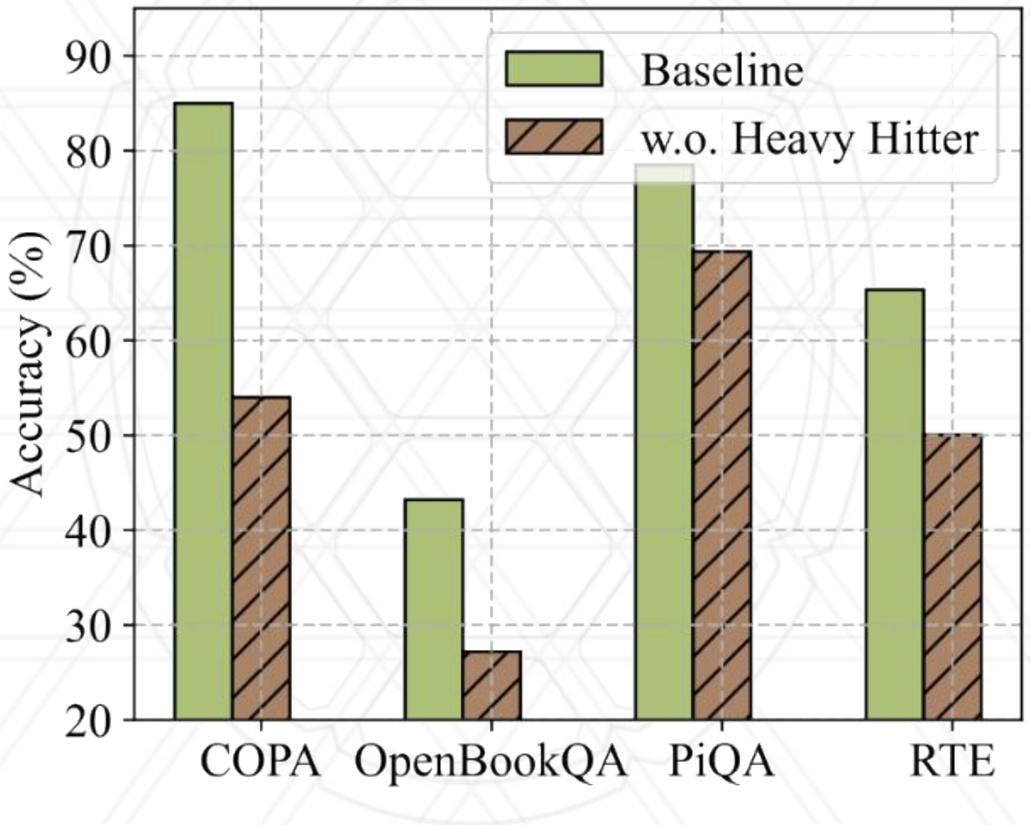
- How can we reduce the KV cache size without hurting accuracy?
  - 30B, 128 bs, 1024 sequence length needs ~180GB space for KV cache alone
- Not all attention values are needed, but how do we determine the most important?
- Across many popular LLMs attention is ~95% sparse

Attention Scores Follow A Power Law



A small subset of tokens are most influential for attention computation

Are the important tokens that important?



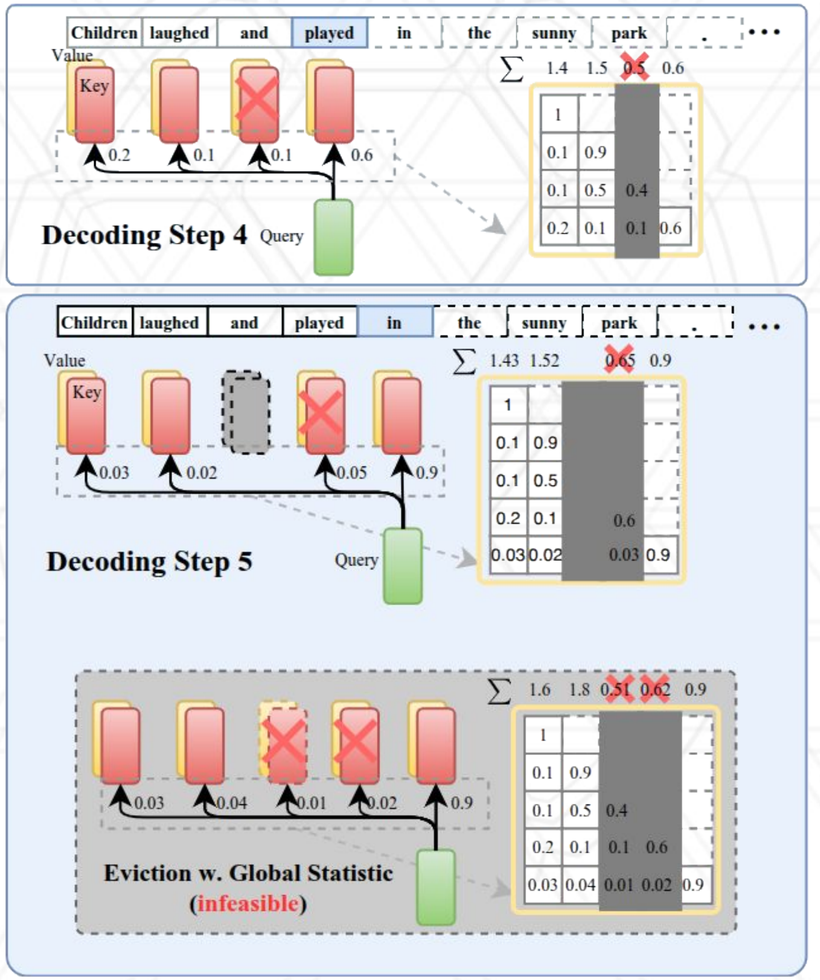
Removing the "heavy hitter" tokens leads to a significant drop in accuracy

Only Retain Important Tokens

- Experimental results suggest keeping heavy-hitters and recent tokens in KV cache is enough for high accuracy with big memory savings
- How do we determine heavy-hitters apriori?

Use Local Attention Scores to Approximate Global Attention

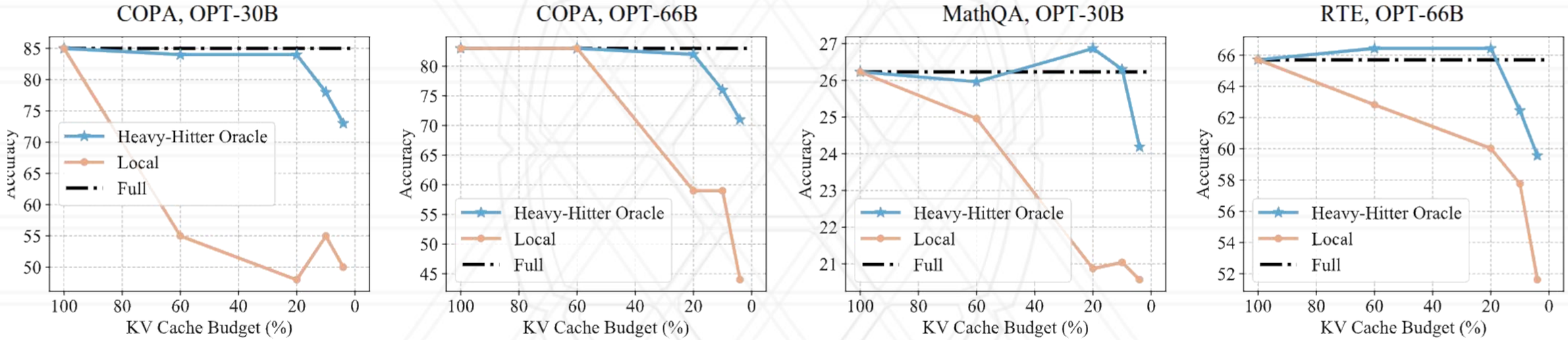
Use local attention steps during decoding phase to evict from KV cache



Sufficiently models the global attention scores

H2O Results: How does it do?

Across most of their experiments they can reduce KV cache to 20% size and get comparable results



Models "collapse" once an important token is evicted from K-V cache

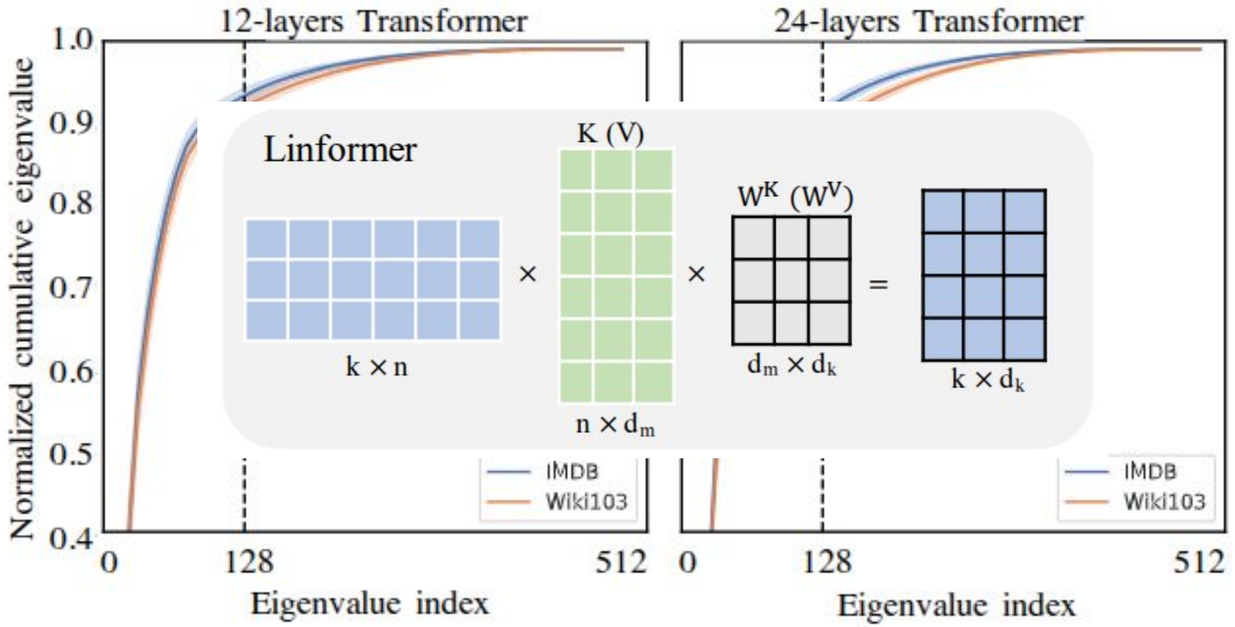
It actually helps accuracy in a number of benchmarks

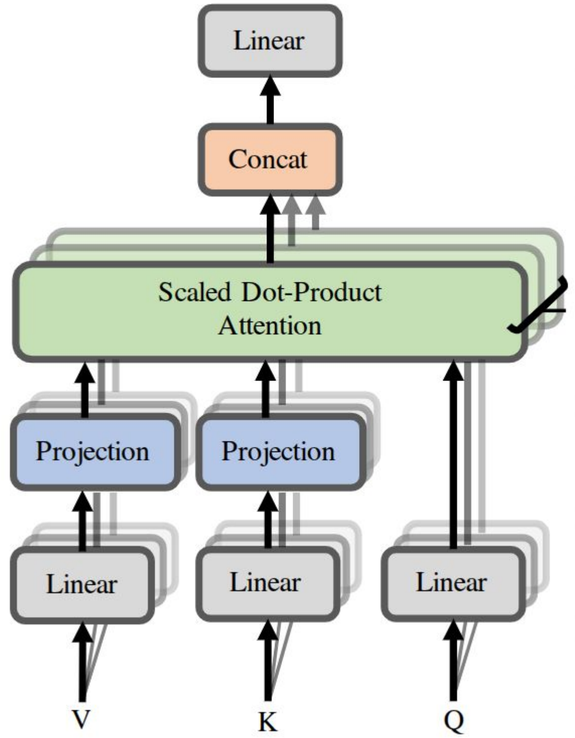
Combining with Other Methods

- Cache eviction policy can be combined with other types of sparse attention
- Get the benefits of other types of sparsity, while reducing KV cache size
- Generally improves all other methods

Beyond Sparsity

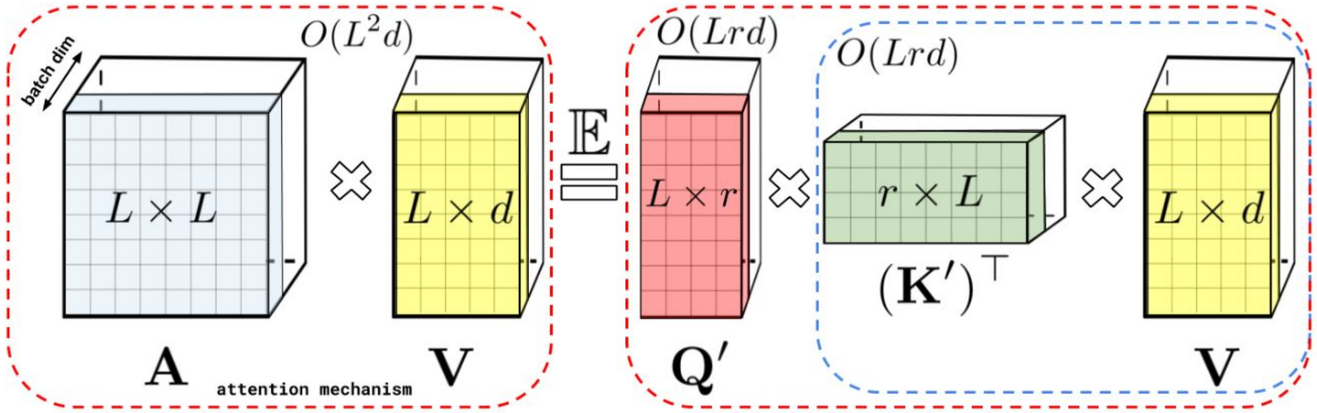
- Low rank approximations
  - "Linformer: Self-Attention with Linear Complexity" Wang et. al.





Beyond Sparsity

- Low rank approximations
- Kernel methods
  - "Rethinking Attention with Performers" Choromanski et. al.



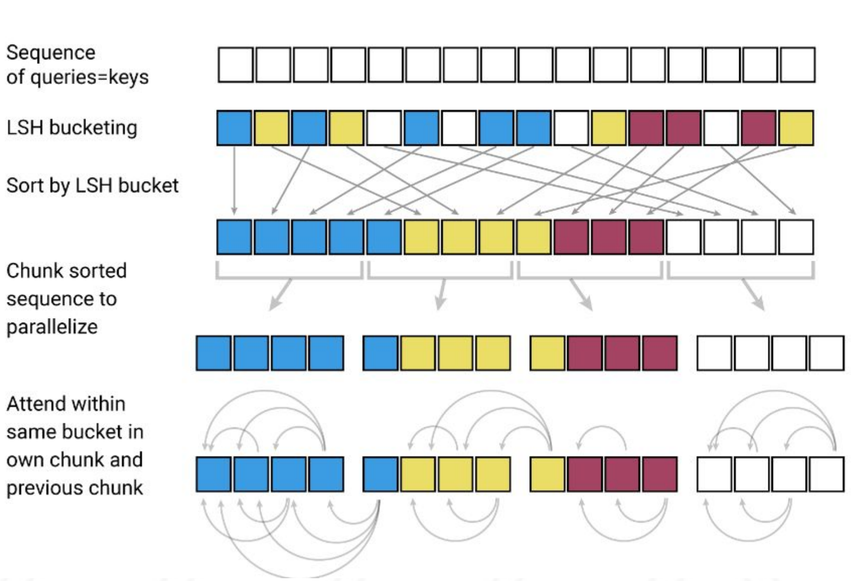
$$\mathrm{K}(\mathbf{x}, \mathbf{y}) = \mathbb{E}[\phi(\mathbf{x})^\top \phi(\mathbf{y})].$$

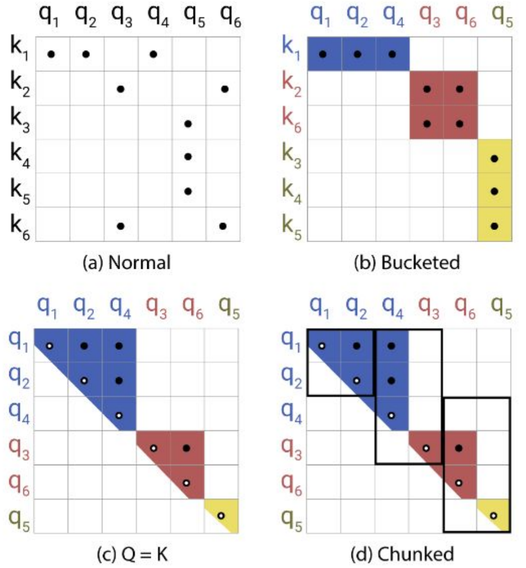
$$\widehat{\mathrm{Att}_{\leftrightarrow}}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \widehat{\mathbf{D}}^{-1}(\mathbf{Q}'((\mathbf{K}')^\top \mathbf{V})), \qquad \widehat{\mathbf{D}} = \mathrm{diag}(\mathbf{Q}'((\mathbf{K}')^\top \mathbf{1}_L)).$$

Beyond Sparsity

- Low rank approximations
- Kernel methods
- Hashing
  - "REFORMER: The Efficient Transformer" Kitaev et. al.

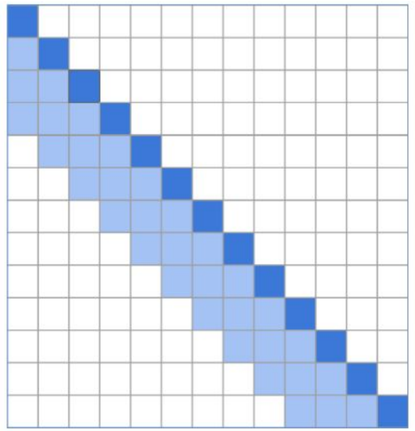
Use hashing to bucket similar tokens and compute dense attention within buckets



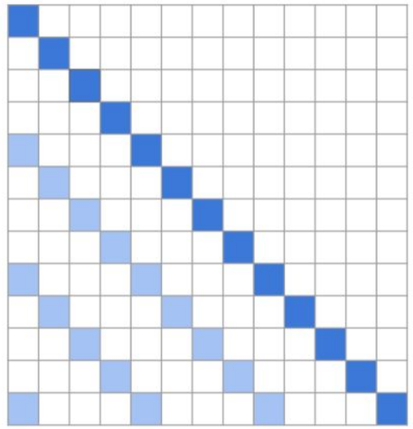


Beyond Sparsity

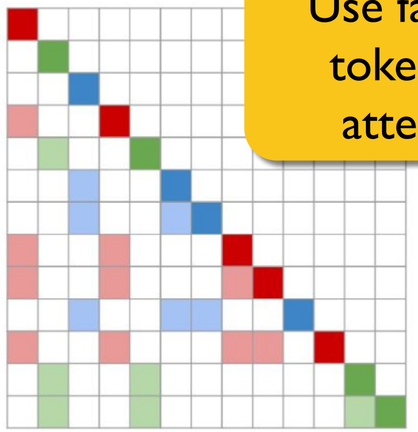
- Low rank approximations
- Kernel methods
- Hashing
- Clustering
  - "Efficient Content-Based Sparse Attention with Routing Transformers" Roy et. al.



(a) Local attention



(b) Strided attention

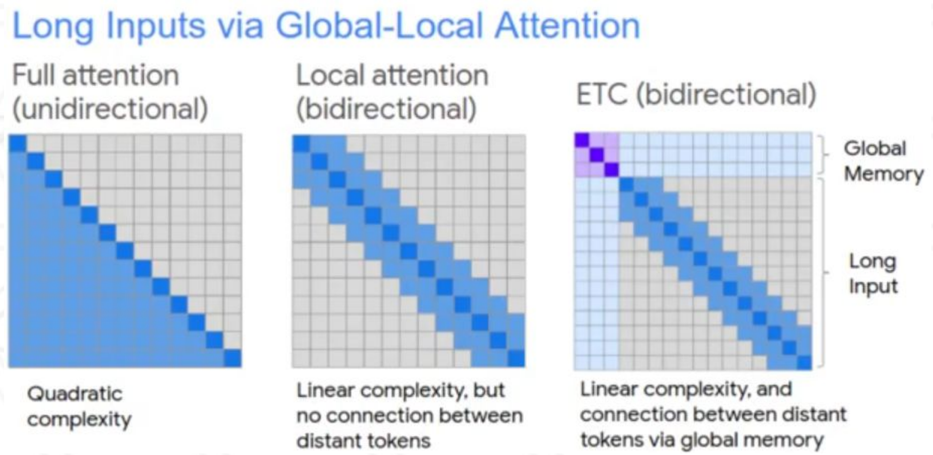


(c) Routing attention

Use fast K-Means to cluster tokens and only do dense attention within clusters

Beyond Sparsity

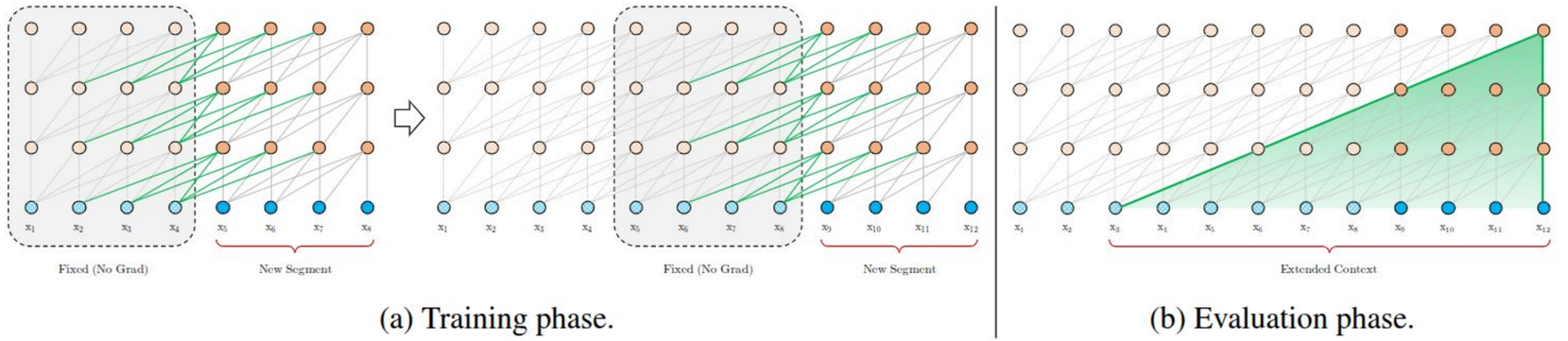
- Low rank approximations
- Kernel methods
- Hashing
- Clustering
- Sinks and Memory Augmentation
  - "ETC: Encoding Long and Structured Inputs in Transformers" Ainslie et. al.



Beyond Sparsity

- Low rank approximations
- Kernel methods
- Hashing
- Clustering
- Sinks and Memory Augmentation
- Recurrence
  - "Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context" Dai et. al.

Process in chunks and carry *some* information between chunks



(a) Training phase. (b) Evaluation phase.

Beyond Sparsity

- Low rank approximations
- Kernel methods
- Hashing
- Clustering
- Sinks and Memory Augmentation
- Recurrence
- Hybrid
  - H2O

UNIVERSITY OF
MARYLAND