

Express Language Modeling



Albert Gong
Cornell



Annabelle Carrell
Cambridge

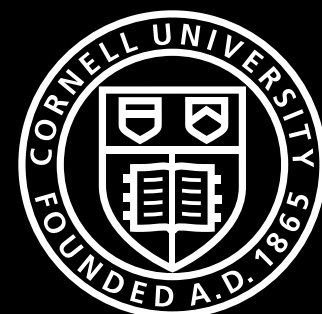


Raaz Dwivedi
Cornell

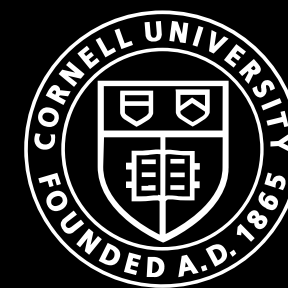


Lester Mackey
MSR

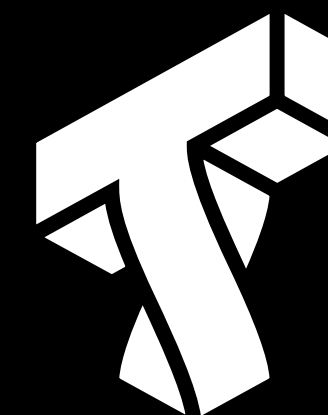
github.com/microsoft/thinformer



Cornell University



**CORNELL
TECH**



arxiv.org/abs/2606.10944



Scaled dot-product attention

Data: n queries, keys, values

Goal: compute $\text{Softmax}(Q^\top K / \sqrt{d}) V$

Equivalently, for each query $\mathbf{q}$, compute
$$\frac{\frac{1}{n} \sum_{i=1}^n \exp(\langle \mathbf{q}, \mathbf{k}_i \rangle / \sqrt{d}) \mathbf{v}_i}{\frac{1}{n} \sum_{i=1}^n \exp(\langle \mathbf{q}, \mathbf{k}_i \rangle / \sqrt{d})}$$

Computational cost: $\Theta(n^2 d)$

Storage cost: $\mathcal{O}(nd)$

Prior work: HyperAttention (Han et al. '24) · BalanceKV (Kochetkova et al. '25) · Thinformer (Carrell et al. '25) · FlashAttention 2 (Dao '24) · SnapKV (Li et al. '24) · StreamingLLM (Xiao et al. '24) · PyramidKV (Cai et al. '25) · ExpectedAttention (Devoto et al. '25) · KeyDiff (Park et al. '25) · Knorm (Devoto et al. '24)

Our Strategy: Replace $\{(k_i, v_i)\}_{i=1}^n$ with $\{(k'_i, v'_i)\}_{i=1}^{n_{\text{out}}}$

Data: n queries, keys, values

Goal: compute $\text{Softmax}(Q^\top K / \sqrt{d}) V$

Equivalently, for each query q , compute
$$\frac{\frac{1}{n_{\text{out}}} \sum_{i=1}^{n_{\text{out}}} \exp(\langle q, k'_i \rangle / \sqrt{d}) v'_i}{\frac{1}{n_{\text{out}}} \sum_{i=1}^{n_{\text{out}}} \exp(\langle q, k'_i \rangle / \sqrt{d})}$$

Computational cost: ~~$\Theta(n^2 d)$~~ $\mathcal{O}(n n_{\text{out}} d)$ + extra cost of compression

Storage cost: ~~$\mathcal{O}(nd)$~~ $\mathcal{O}(n_{\text{out}} d)$ + extra space for compression

Need numerator to satisfy:

$$\frac{1}{n} \sum_{i=1}^n \exp(\langle q, k_i \rangle / \sqrt{d}) v_i \approx \frac{1}{n_{\text{out}}} \sum_{i=1}^{n_{\text{out}}} \exp(\langle q, k'_i \rangle / \sqrt{d}) v'_i$$

Need denominator to satisfy:

$$\frac{1}{n} \sum_{i=1}^n \exp(\langle q, k_i \rangle / \sqrt{d}) \approx \frac{1}{n_{\text{out}}} \sum_{i=1}^{n_{\text{out}}} \exp(\langle q, k'_i \rangle / \sqrt{d})$$

How to choose $\{(k'_i, v'_i)\}_{i=1}^{n_{\text{out}}}$?
This is where thinning comes in.



```
from thinformer import ThinformerAttention
attention_layer = ThinformerAttention()
attn_output, attn_output_weights = attention_layer(query, key, value)
```

Runtime: $\mathcal{O}(n_{\text{out}}n)$ (exact attention: $\Theta(n^2d)$)

Error: $\mathcal{O}(\sqrt{\log n_{\text{out}} \log(n_{\text{out}} \log(n/n_{\text{out}}))} / n_{\text{out}})$

State-of-the-art for unweighted coresets attention approximation

Thinning

Input: $\mathcal{S}_{in} = \{x_i\}_{i=1}^n$ **Output:** $\mathcal{S}_{out} = \{x'_i\}_{i=1}^{n_{out}}$

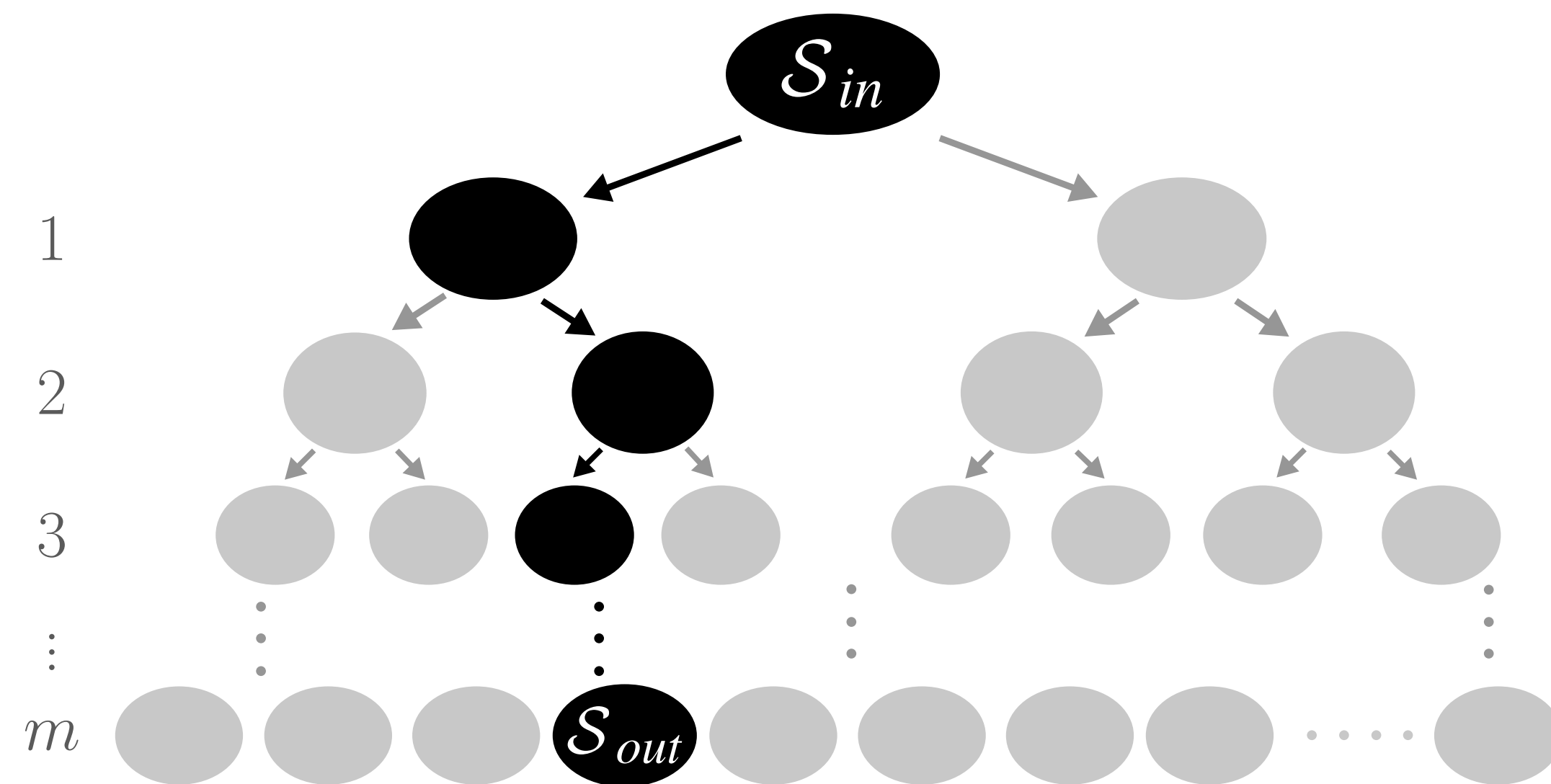
$$\left| \frac{1}{n} \sum_{i=1}^n f(x_i) - \frac{1}{n_{out}} \sum_{i=1}^{n_{out}} f(x'_i) \right| < \text{small}^*$$

* for f in a function class

Typical integration error for uniform subsampling is $\Theta(s^{-1/2})$: need $s = 10000$ for 1% error

Can we do better?

Recursive halving via self-balanced walk

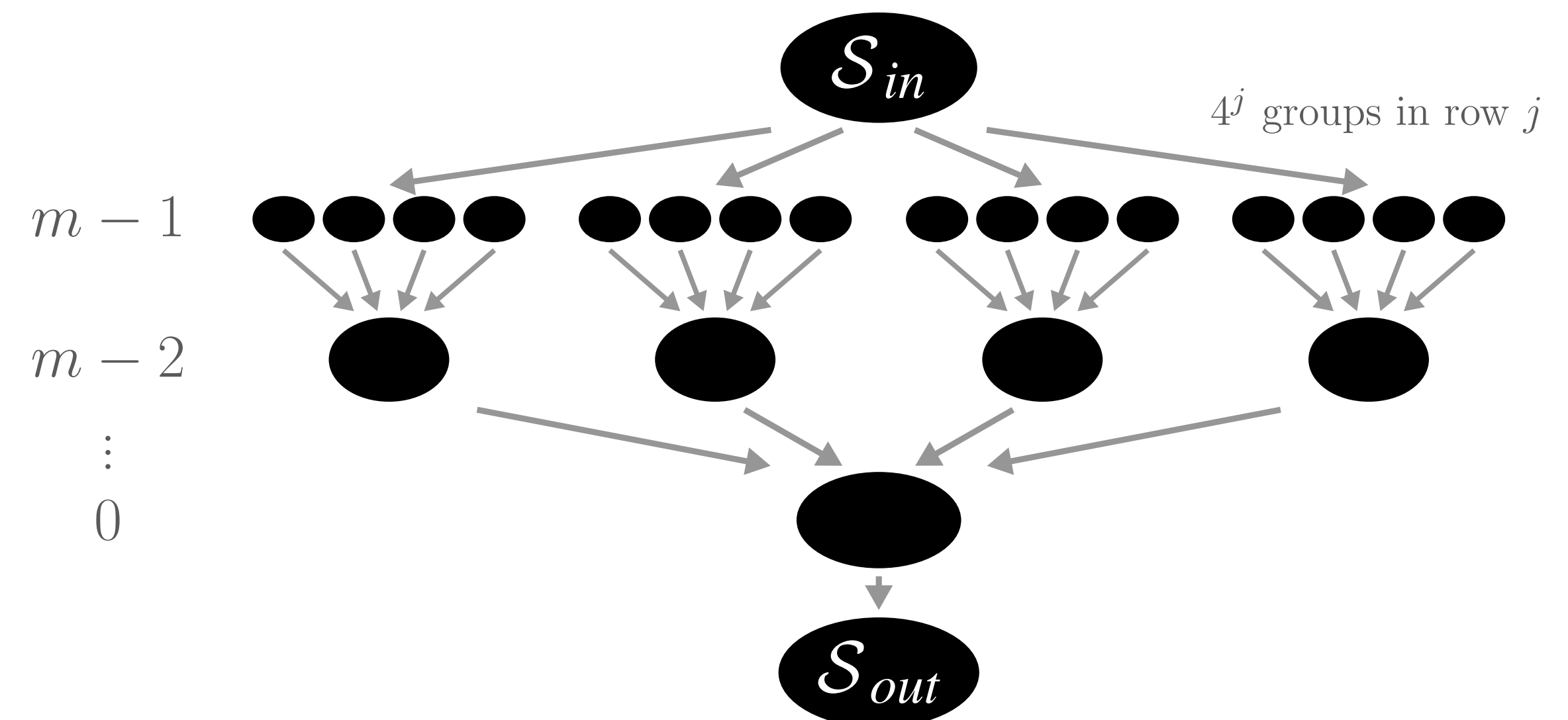


Error: $\mathcal{O}(\sqrt{\log n_{out}} / n_{out})$

Runtime: $\mathcal{O}(n^2)$

Kernel Thinning (Dwivedi & Mackey, '21)

Split into groups



Error: $\mathcal{O}(\sqrt{\log(n/n_{out})} \log n / n_{out})$

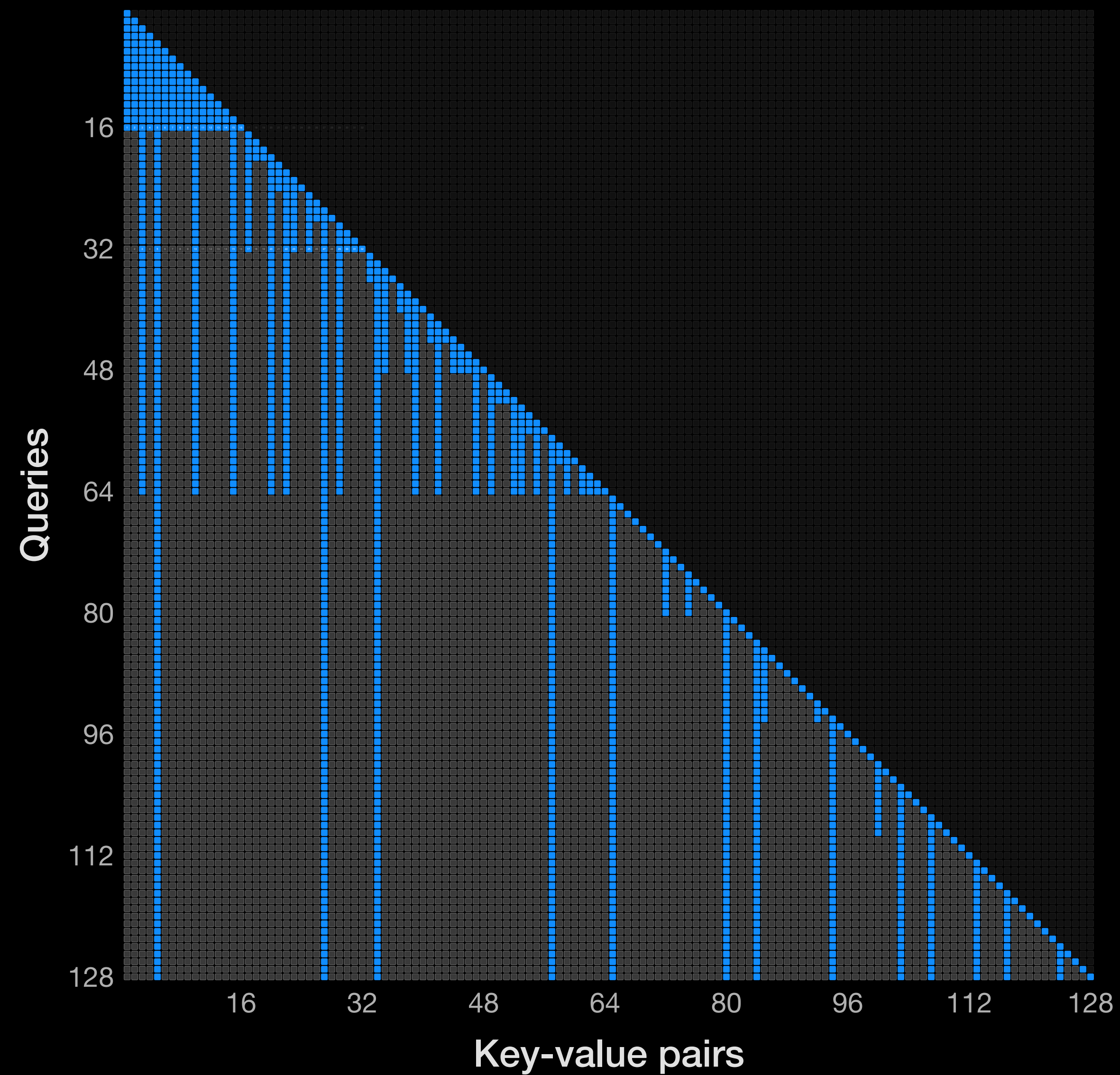
Runtime: $\mathcal{O}(n_{out}^2 \log(n/n_{out}))$

Compress (Shetty et al., '22)

What does Thinformer miss?



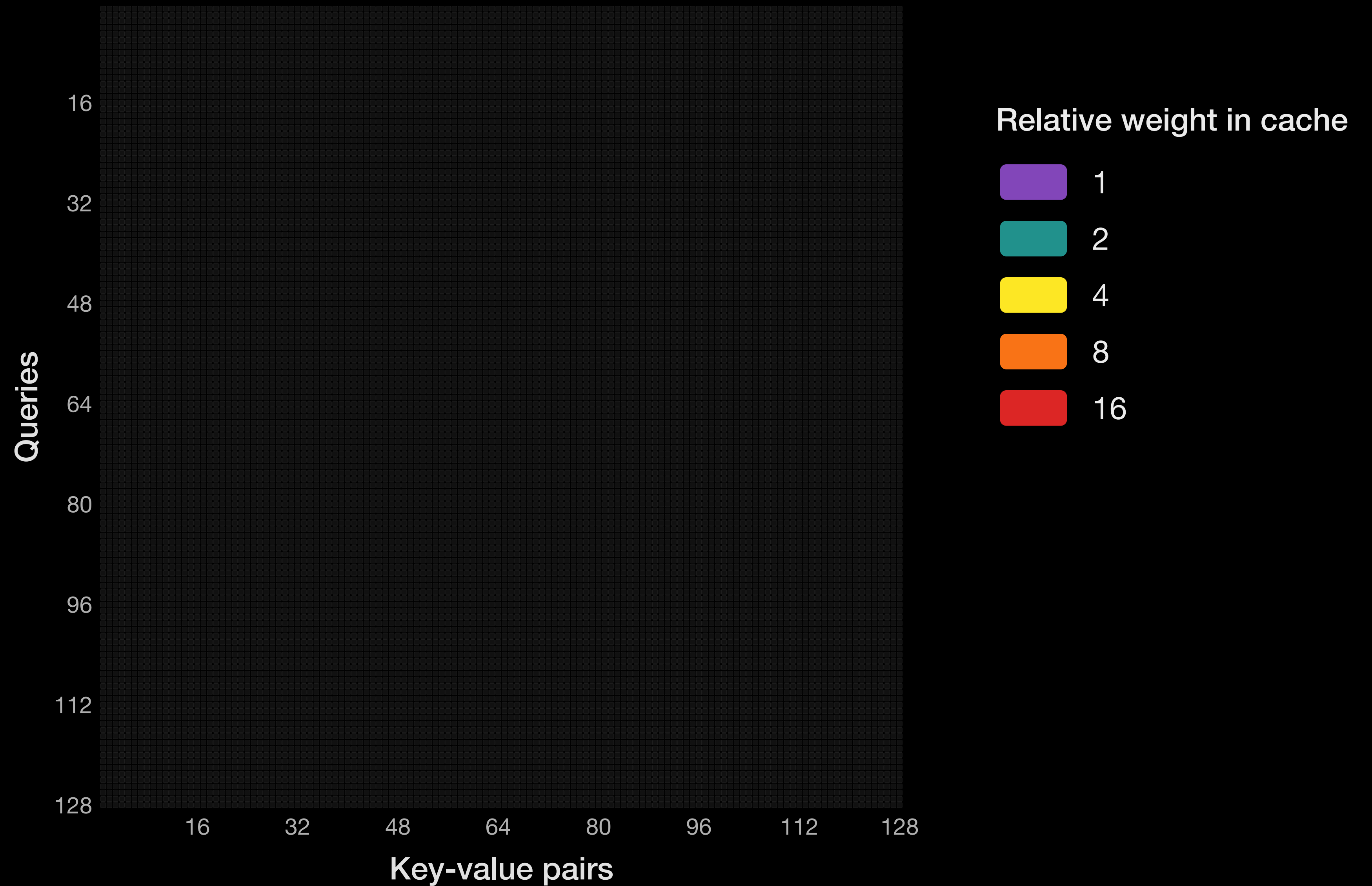
Thinformer Express



Thinformer Express

Round by round

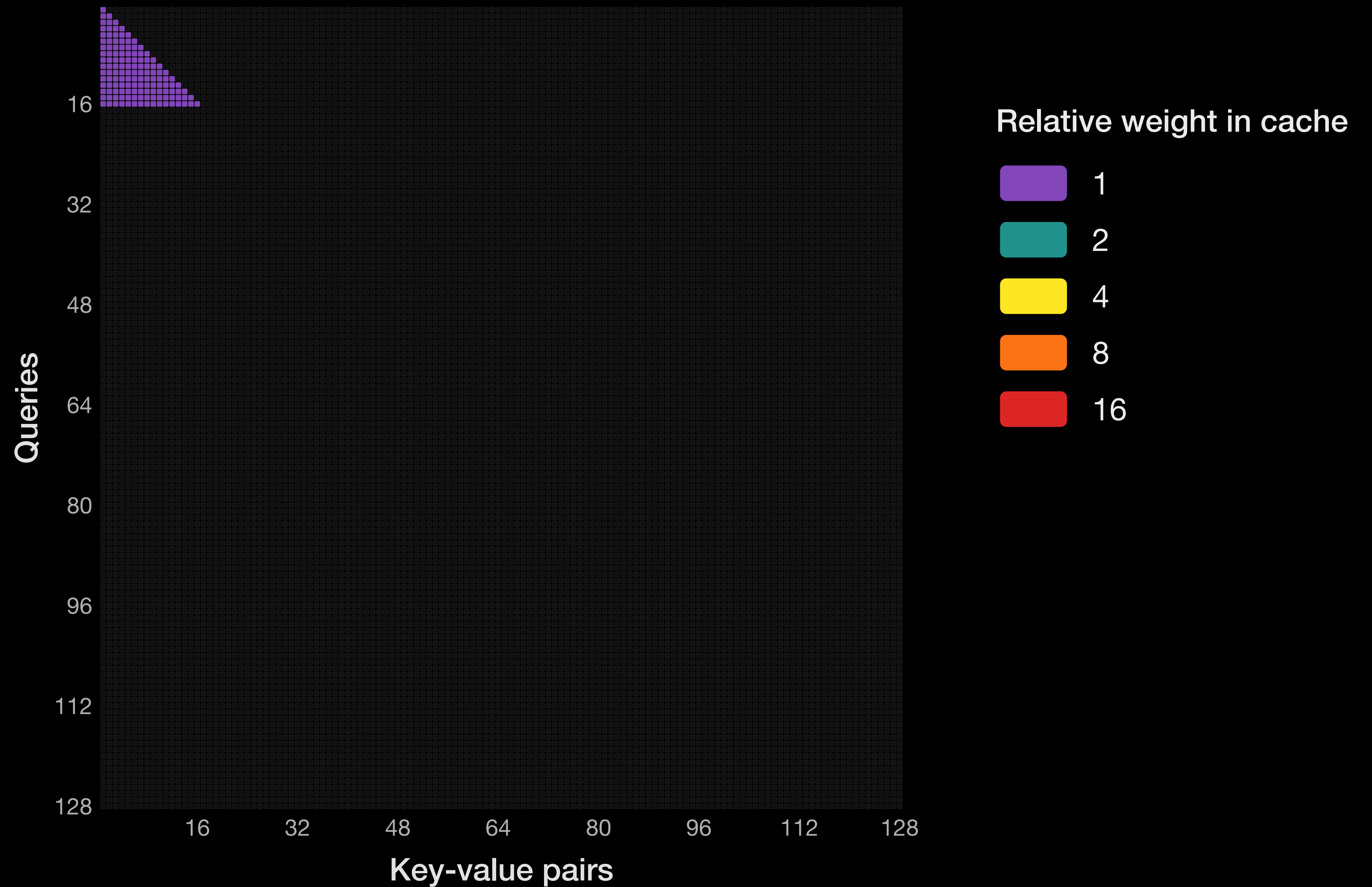
Each row is the cache
one query sees



Thinformer Express

Round m = 0

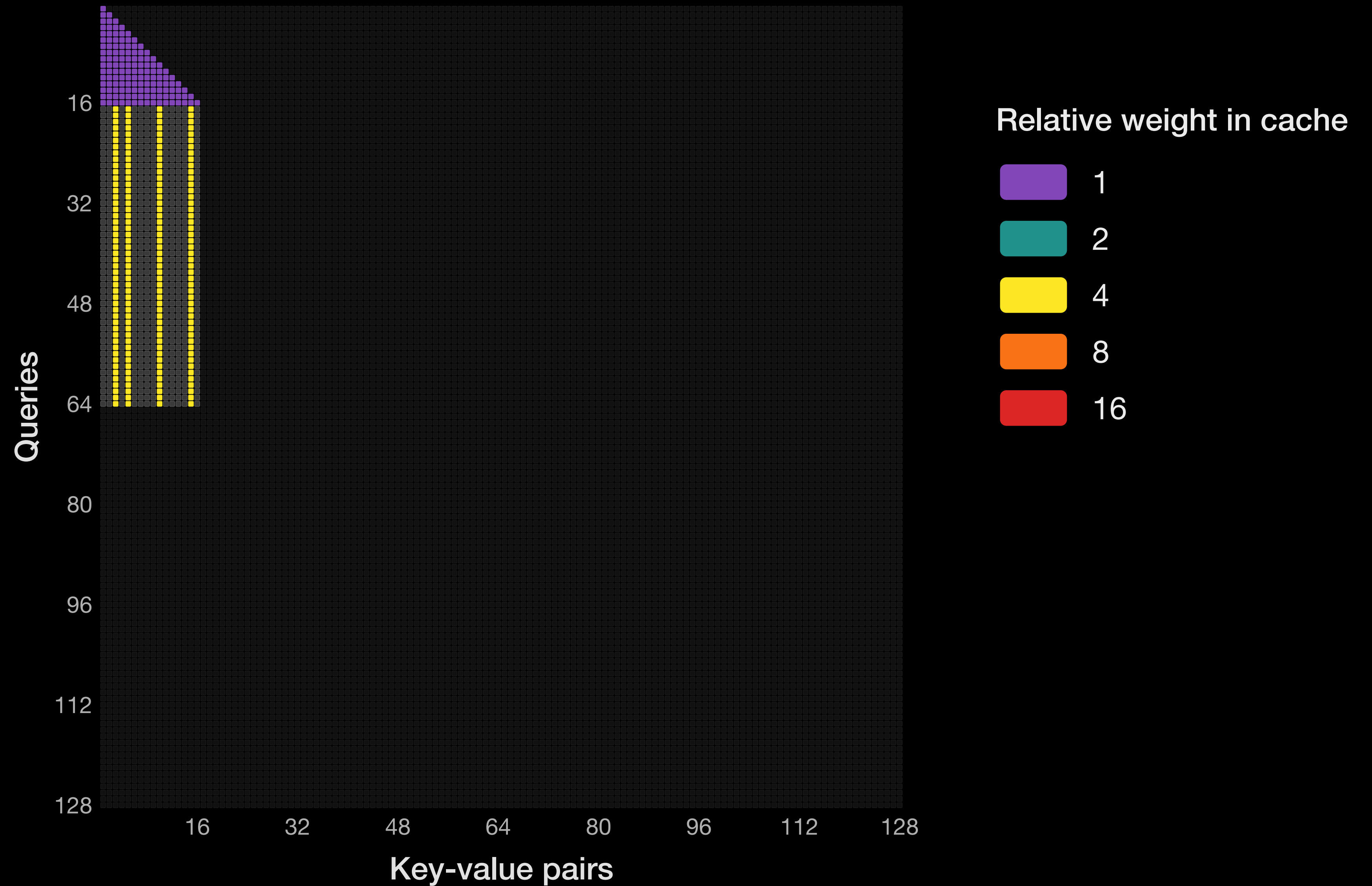
Exact phase: keep all
16 tokens (weight 1)



Thinformer Express

Halve twice

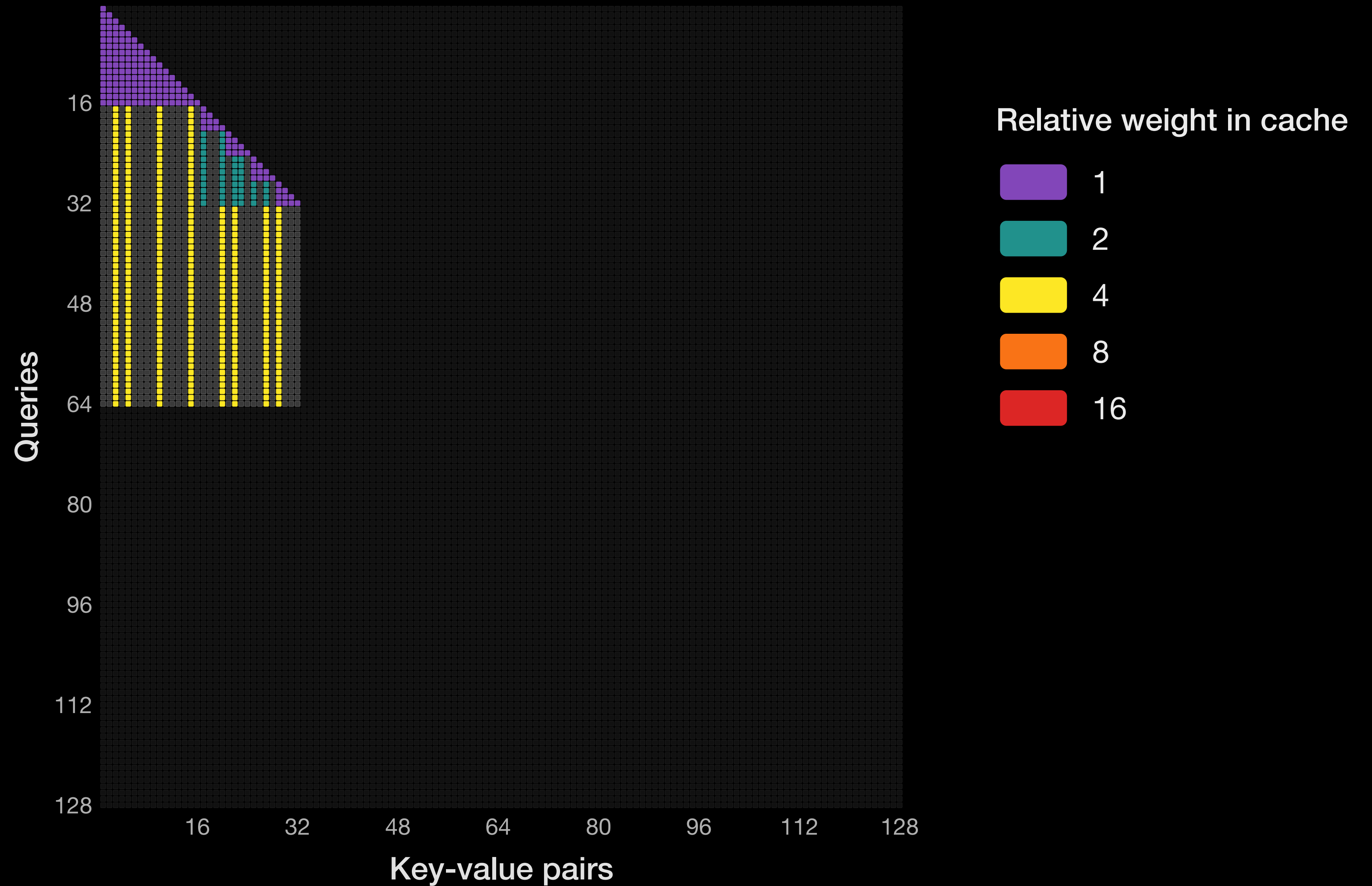
16 tokens $\rightarrow$ 4 tokens
of weight 4; m: 0 $\rightarrow$ 2



Thinformer Express

Round m = 2

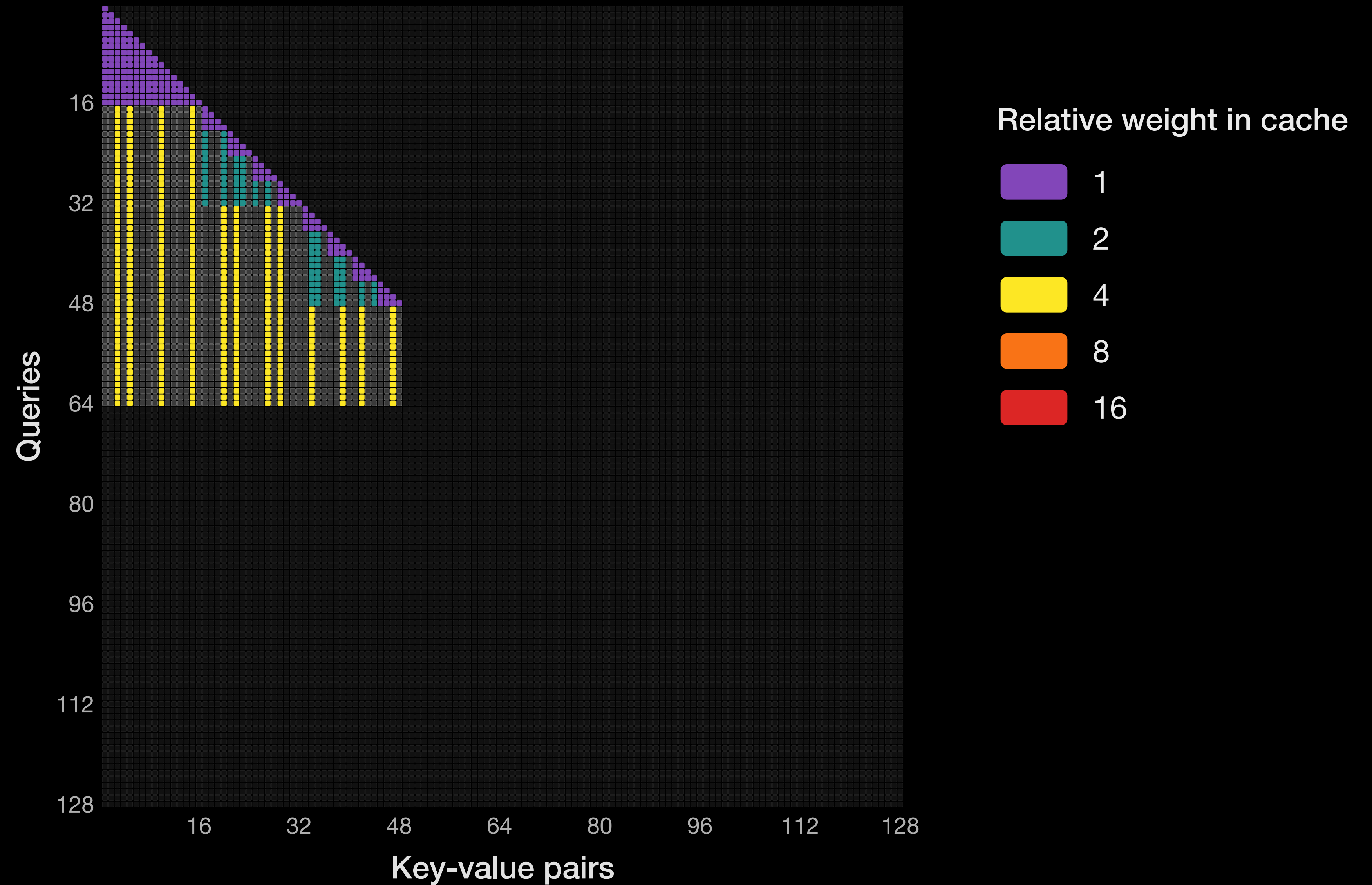
Compress2 on tokens 17–32
→ 4 tokens of weight 4



Thinformer Express

Round $m = 2$

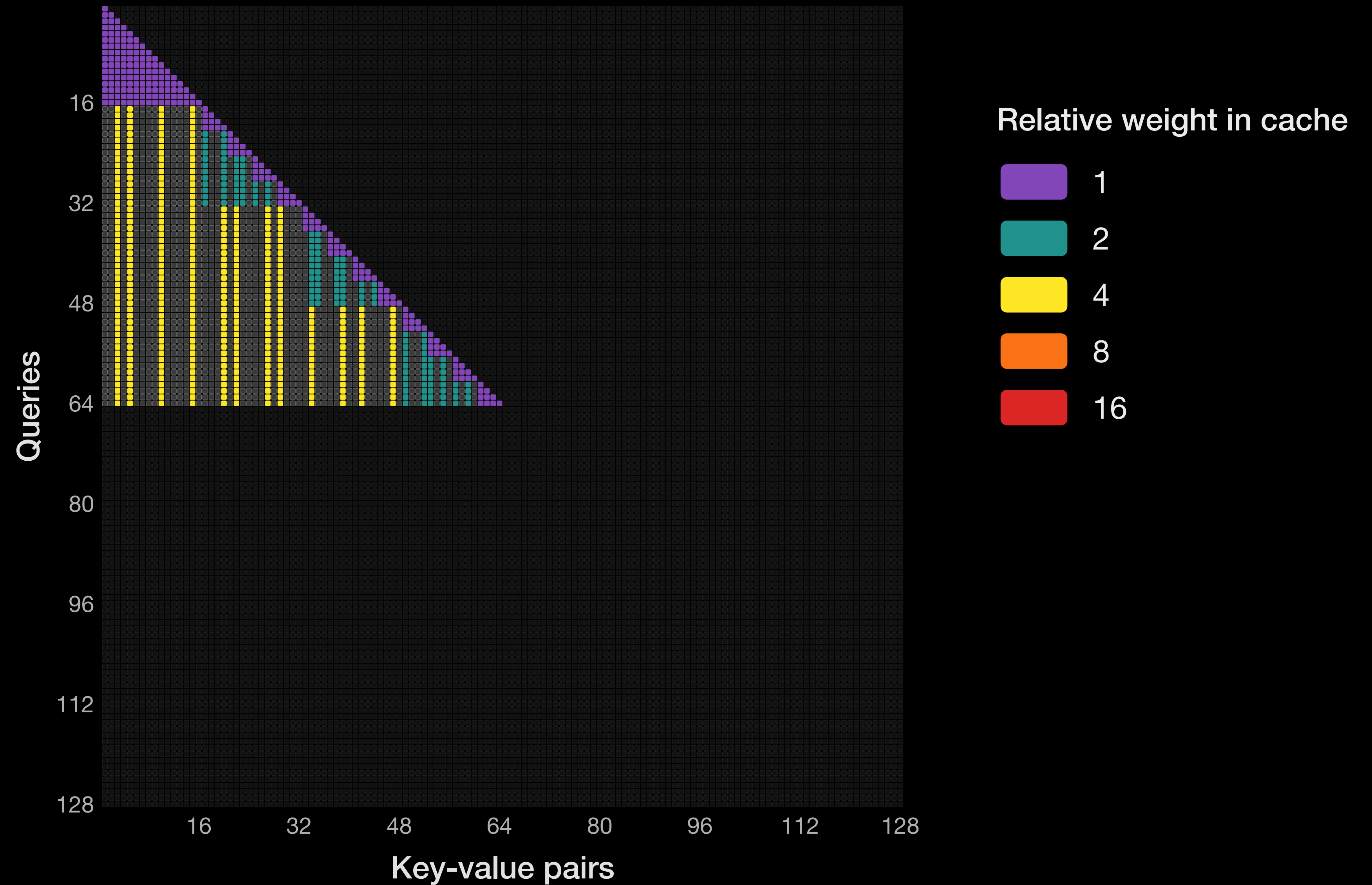
Compress2 on tokens 33–48
→ 4 tokens of weight 4



Thinformer Express

Round $m = 2$

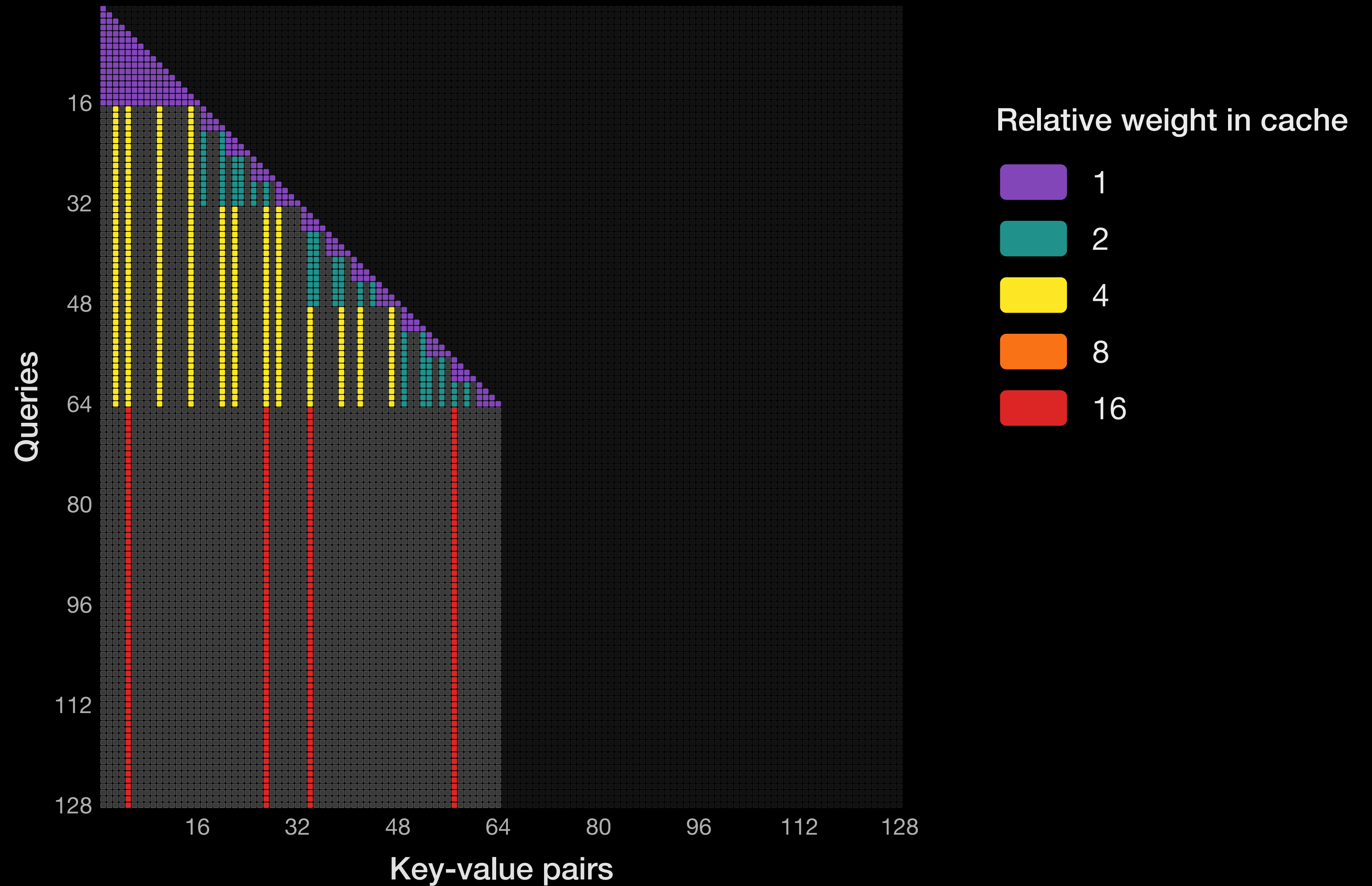
Compress2 on tokens 49–64
→ 4 tokens of weight 4



Thinformer Express

Halve twice

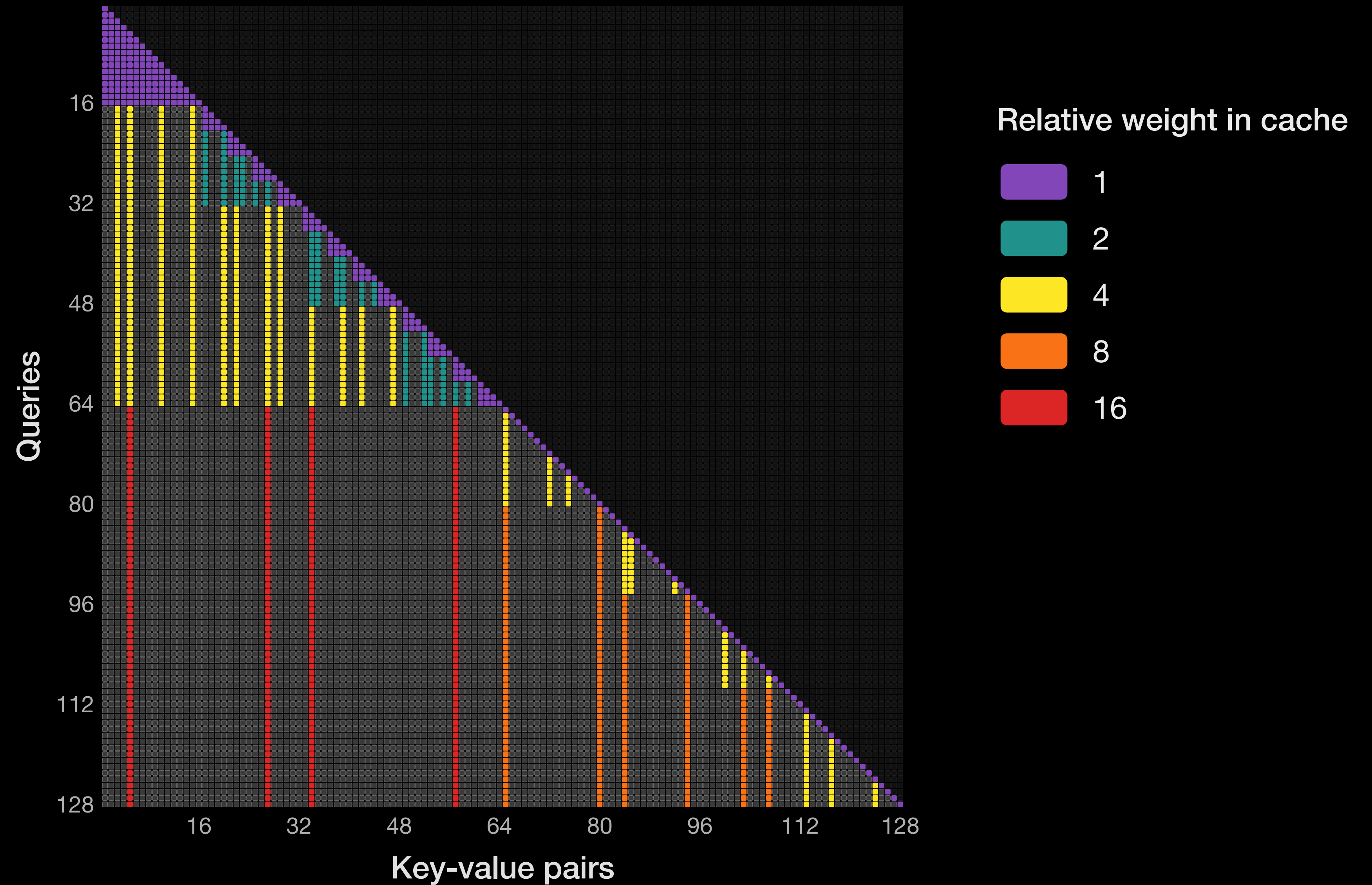
16 tokens $\rightarrow$ 4 tokens
of weight 16; m: 2 $\rightarrow$ 4



Thinformer Express

Round m = 4

Compress2 on 64 tokens
(tokens 65–128)



Thinformer Express achieves state-of-the-art guarantees

Approximation error. Let

$R_n \triangleq \max_{i \in [n]} \max(\|\mathbf{q}_i\|_2, \|\mathbf{k}_i\|_2)$ and $\mathbf{V}_n$ be the value matrix.

Ours:

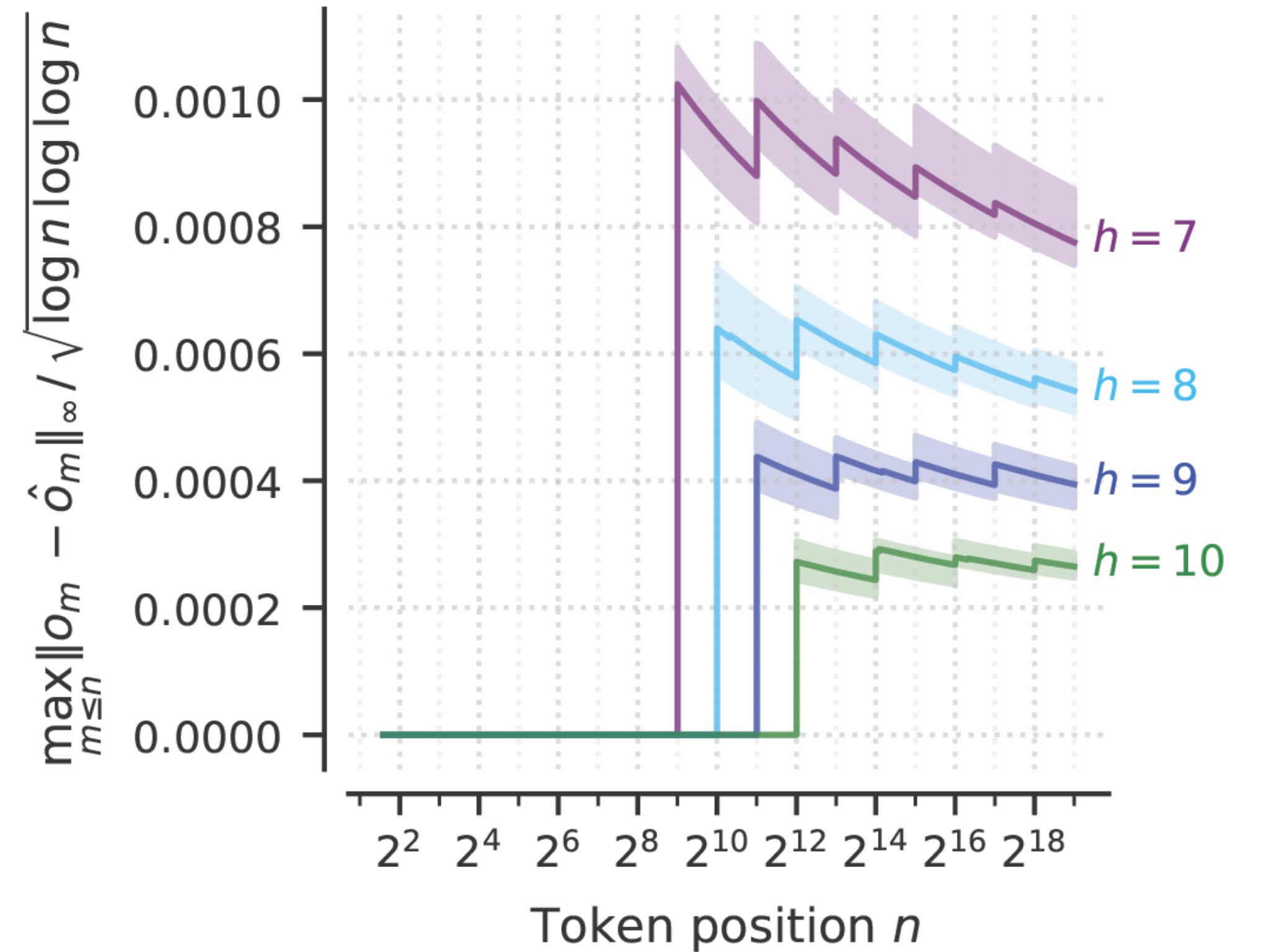
$$\max_{j \in [n]} \|\hat{\mathbf{o}}_{\text{exp},j} - \mathbf{o}_j\|_\infty \leq O\left(\frac{\exp(2R_n^2) \log n \sqrt{d \log(dn)}}{n^a} \cdot \|\mathbf{V}_n\|_{\max}\right)$$

HyperAttention (Han et al. '23):

$$\|\hat{\mathbf{o}}_{\text{hyp},n} - \mathbf{o}_n\|_\infty = O\left(\frac{\exp(\frac{17}{3}R_n^2) (\log n)^{\frac{1}{6}}}{n^{a/6}} \cdot \|\mathbf{V}_n\|_{\text{op}}\right)$$

Lower bound (Liberty et al. '26):

$$\Omega\left(\frac{\exp(R_n^2/\sqrt{d}) \cdot \|\mathbf{V}_n\|_{\max}}{n^a}\right)$$



Thinformer Express achieves state-of-the-art guarantees

Approximation error. Let

$R_n \triangleq \max_{i \in [n]} \max(\|q_i\|_2, \|k_i\|_2)$ and V_n be the value matrix.

Ours:

$$\max_{j \in [n]} \|\hat{o}_{\text{exp},j} - o_j\|_\infty \leq O\left(\frac{\exp(2R_n^2) \log n \sqrt{d \log(dn)}}{n^a} \cdot \|V_n\|_{\max}\right)$$

HyperAttention (Han et al. '23):

$$\|\hat{o}_{\text{hyp},n} - o_n\|_\infty = O\left(\frac{\exp(\frac{17}{3}R_n^2) (\log n)^{\frac{1}{6}}}{n^{a/6}} \cdot \|V_n\|_{\text{op}}\right)$$

Lower bound (Liberty et al. '26):

$$\Omega\left(\frac{\exp(R_n^2/\sqrt{d}) \cdot \|V_n\|_{\max}}{n^a}\right)$$

See Thm. 3 in paper for refined bound

Compression time.

Ours:

$$O(d n_{\text{out}}^2 \log(n_{\text{out}}) \log(n/n_{\text{out}}))$$

BalanceKV (Kochetkova et al. '25):

$$\Theta(d n n_{\text{out}} \log(n/n_{\text{out}}))$$

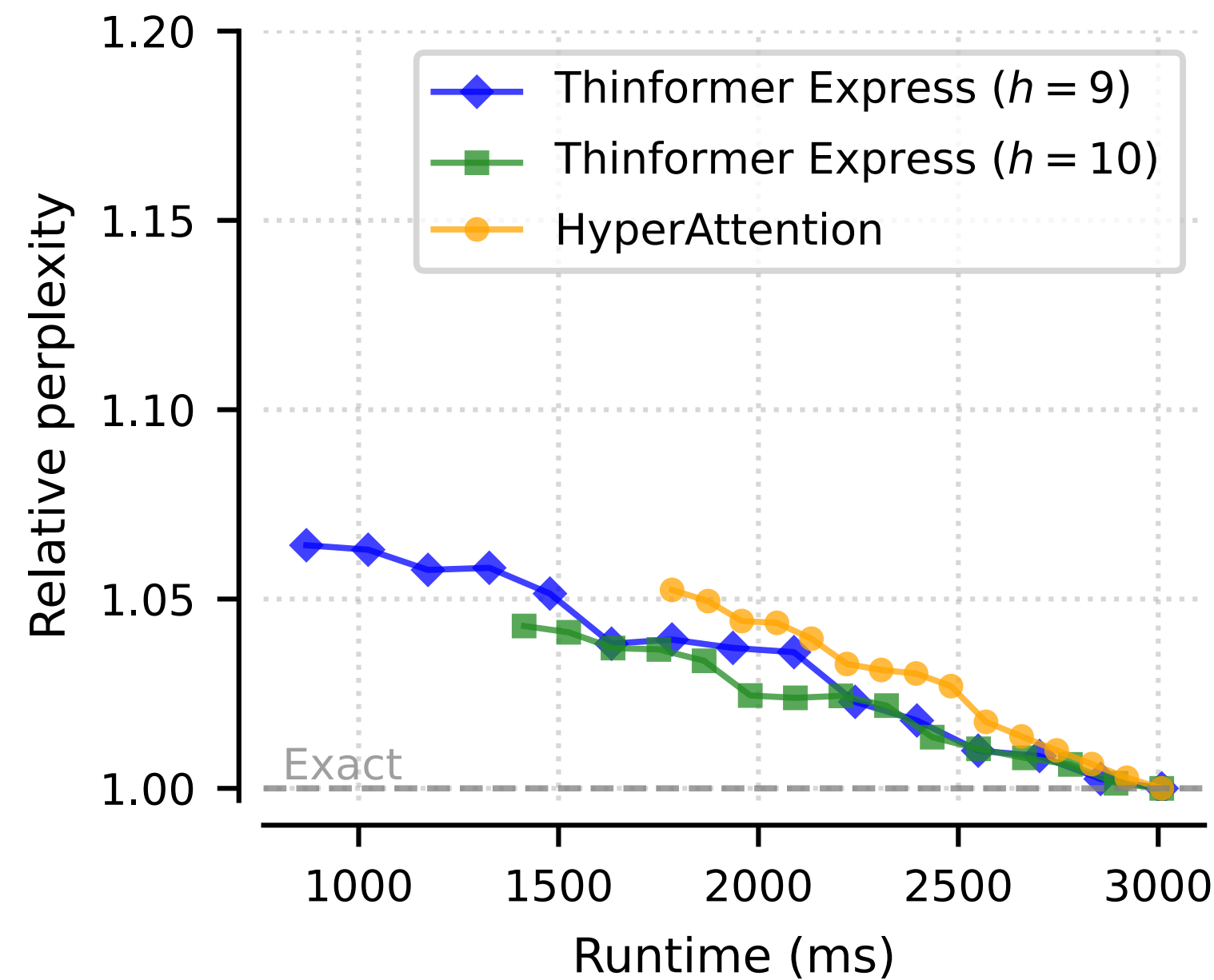
Cache size.

$$\text{Ours: } O(d n_{\text{out}})$$

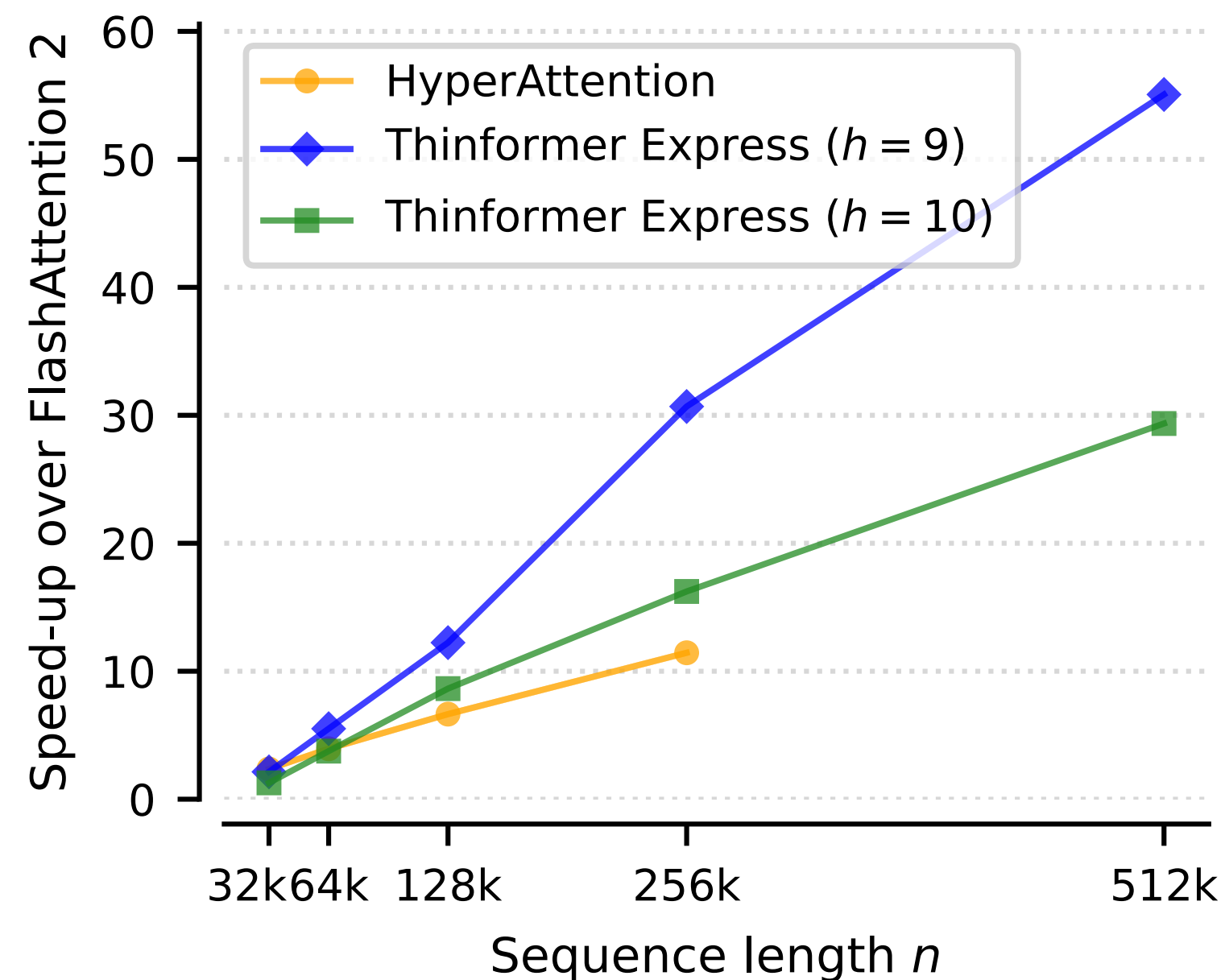
BalanceKV (Kochetkova et al. '25):

$$\Theta(d n_{\text{out}} \log(n/n_{\text{out}}))$$

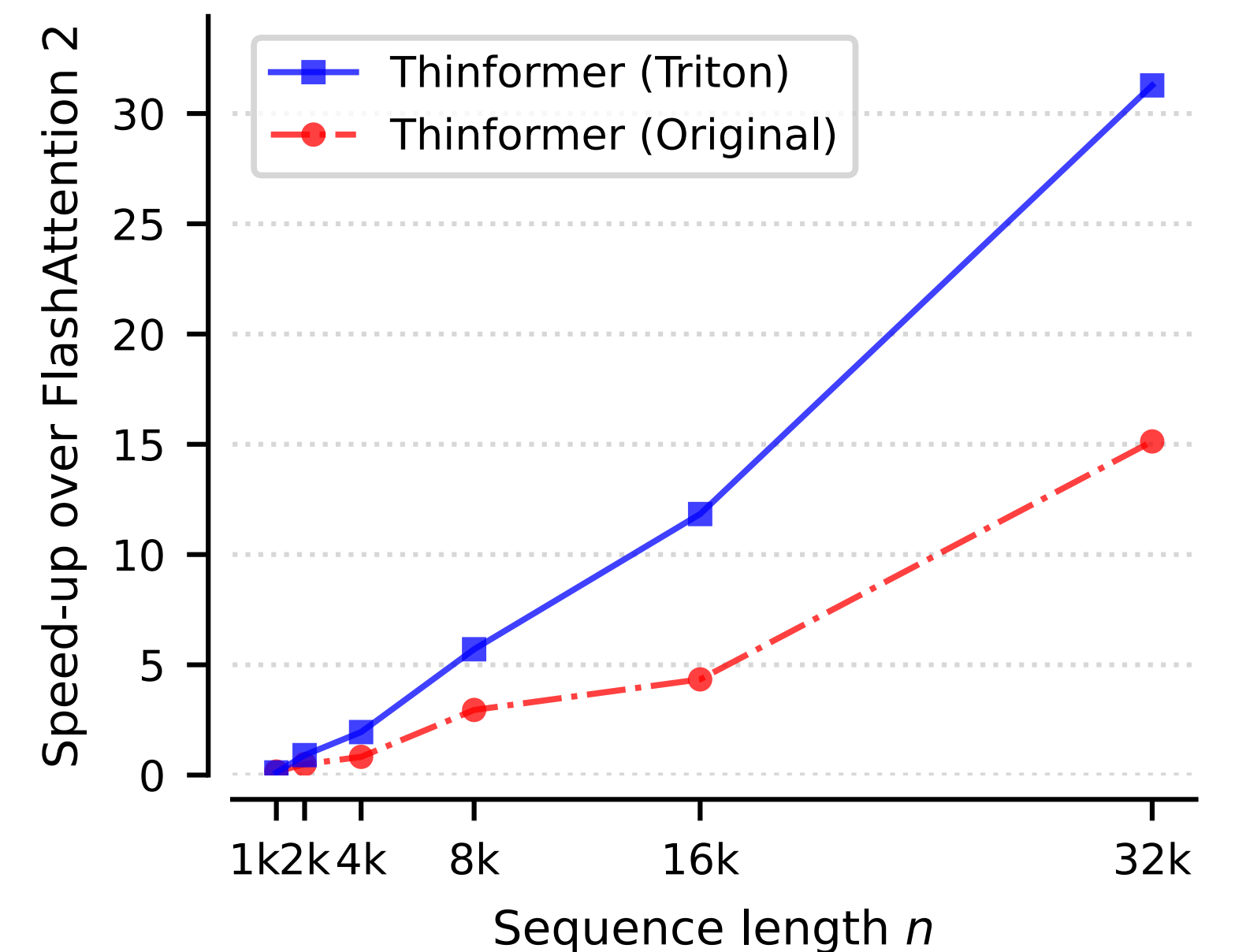
Bottleneck #1: long-context prefill



Thinformer Express achieves a better accuracy-vs-runtime trade-off than HyperAttention

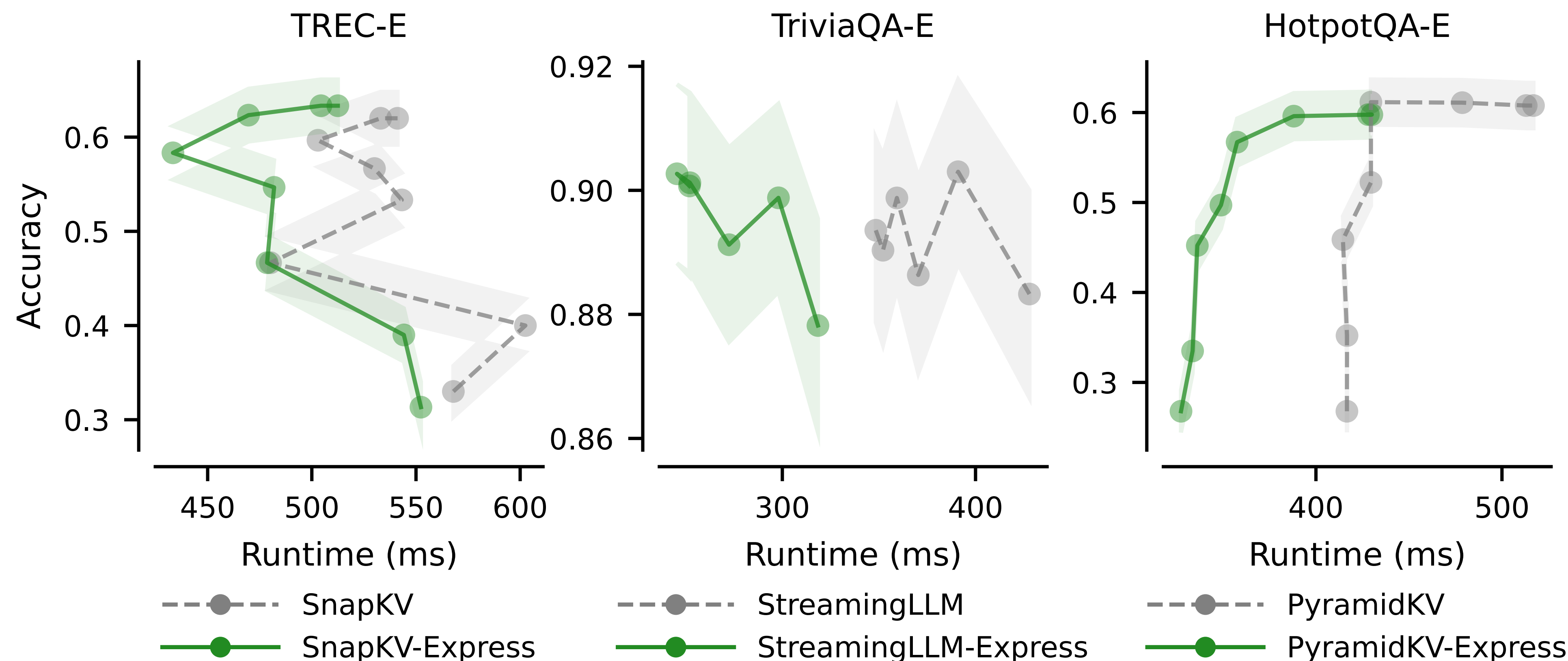


Thinformer Express achieves a **55×** speed-up over FlashAttention 2 at sequence length 512k



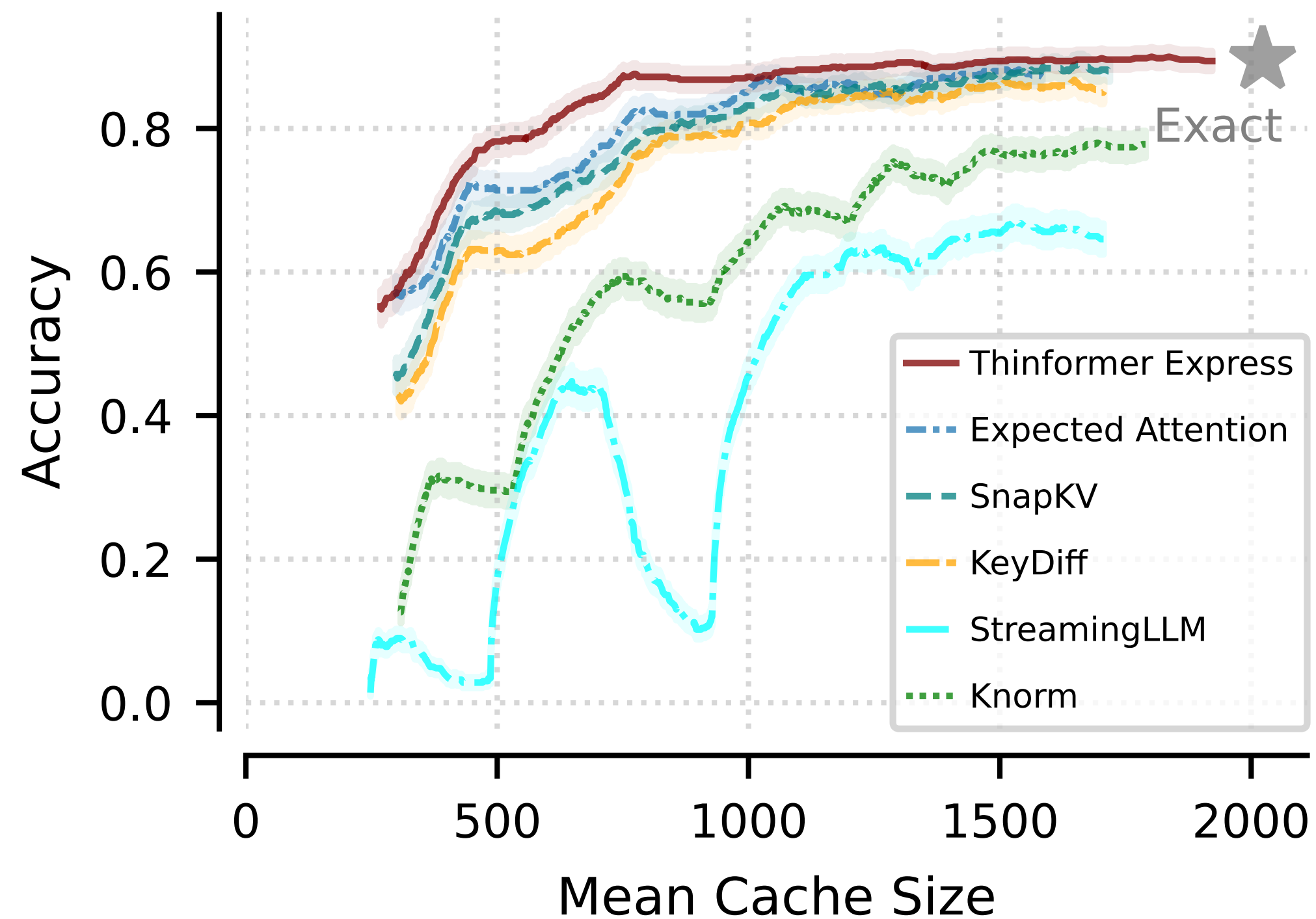
Thinformer (Triton) is **2×** faster than the prior Thinformer (torch.compile)

Bottleneck #2: KV cache compression

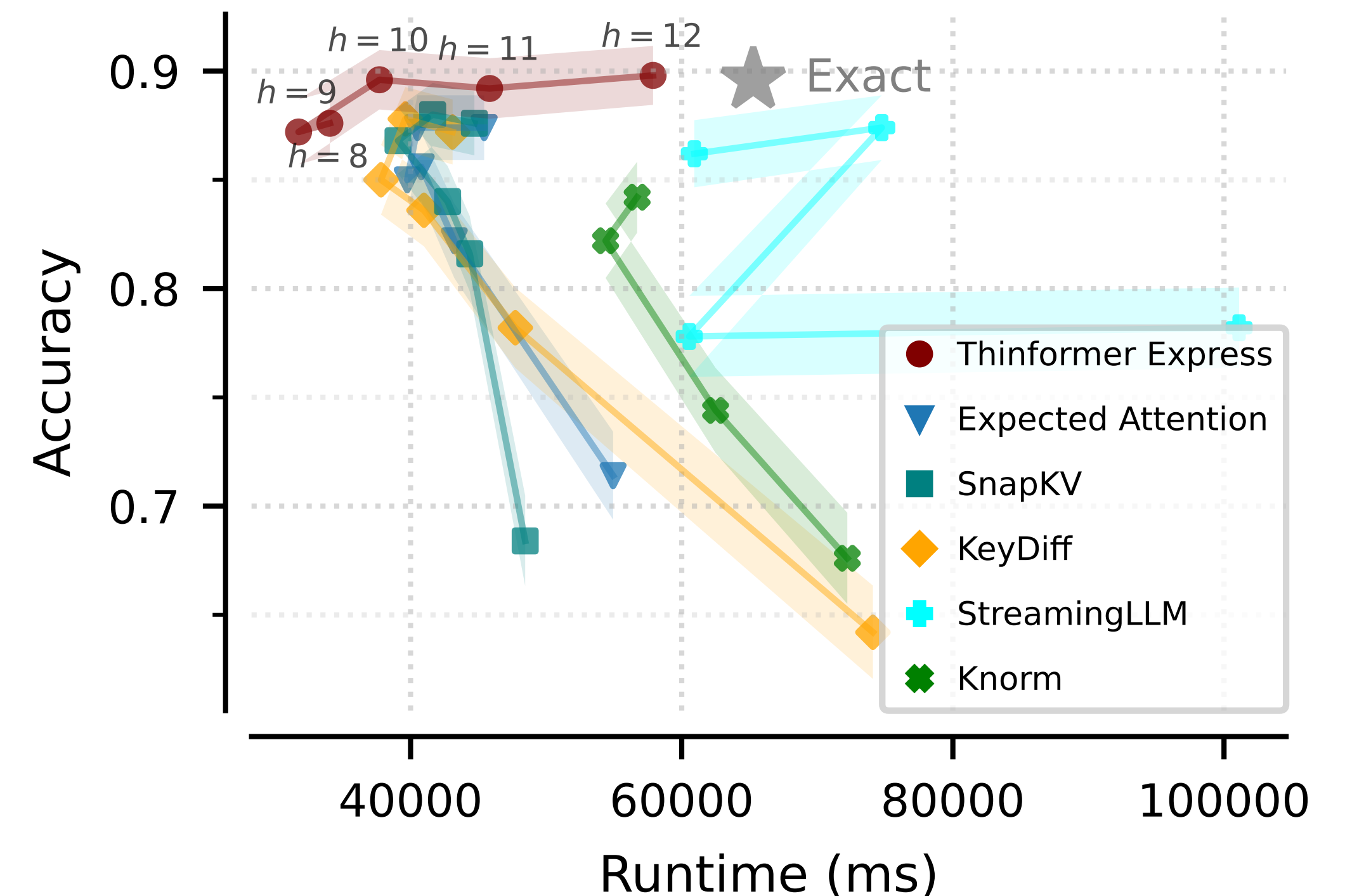


Thinformer Express preserves KV cache compression quality while substantially reducing total attention runtime

Bottleneck #3: long-form decoding



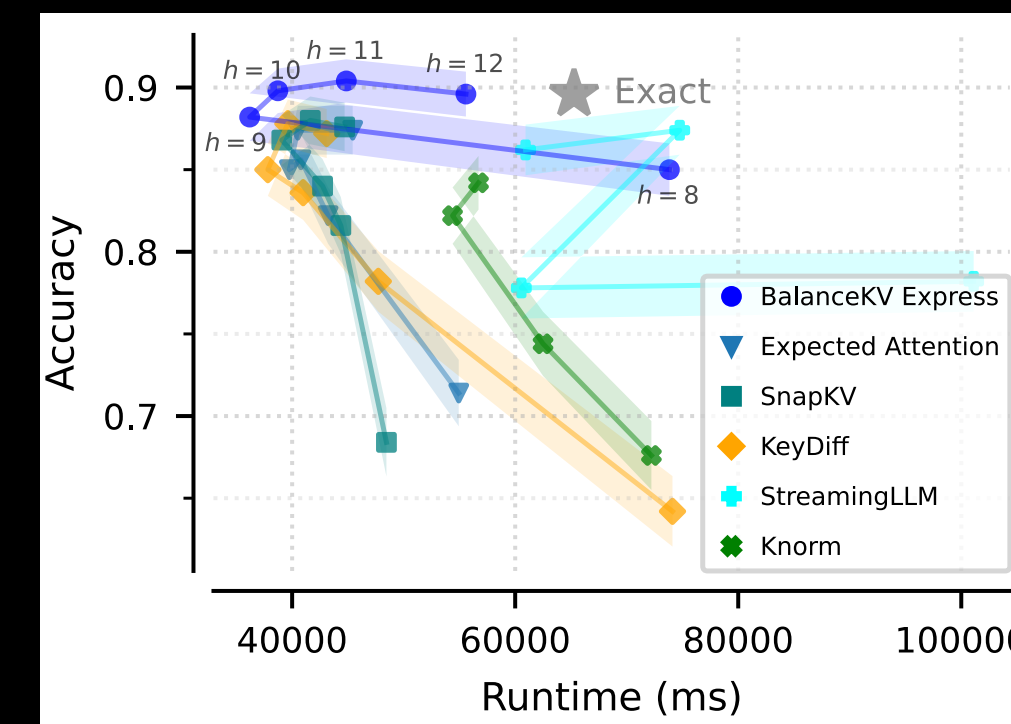
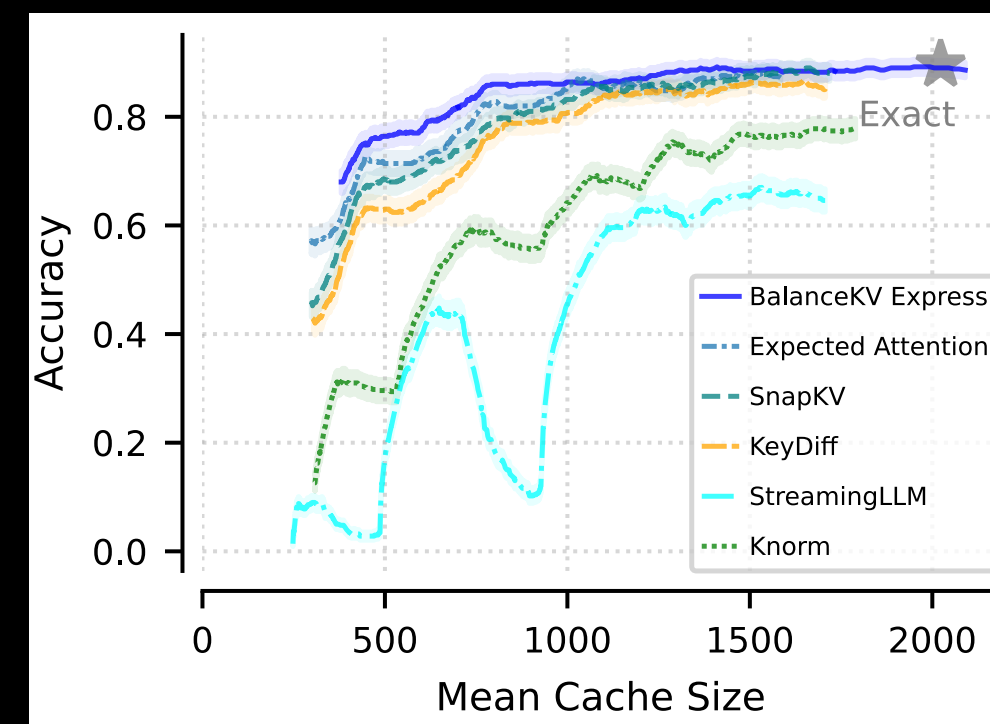
Thinformer Express retains **99.6%** the accuracy of exact attention with only **64%** percent of the cache size



Thinformer matches the accuracy of exact attention with only **58%** percent of the computation time

Future work

- **Other languages, domains, and modalities:** Express delivers state-of-the-art guarantees and real-world performance, but only on datasets that we evaluated
- **Other halving algorithms:** any sub-Gaussian halving algorithm plugs in, even ones without known guarantees (e.g., BalanceKV)



- **Faster GPU kernels:** warp specialization, the Tensor Memory Accelerator, persistent kernels, and FP8, as in FlashAttention 2 and 3

```
from thinformer.triton import ThinformerAttention
attention_layer = ThinformerAttention()
attn_output, attn_output_weights = attention_layer(query, key, value)
attn_output, attn_output_weights = attention_layer(query, key, value, causal=True)
```

Thank you!