

The problem

Long context means a huge KV cache, so you have to throw some of it away. There are competing rules for choosing what to drop.

But every method ships its rule welded to its own kernel. When one beats another, you cannot tell whether the rule was smarter or the code was just faster.

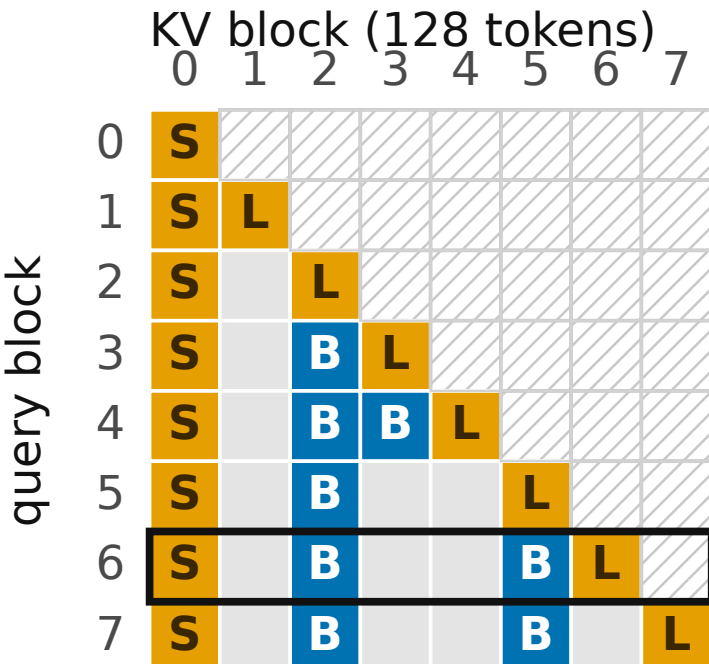
The move

Split the method in two: the SIGNAL that decides which blocks matter, and the KERNEL that drops the rest. Published methods weld them together. We fix the kernel and the budget, and swap only the signal.

A signal is just a rule for scoring blocks -- keep the most recent, the most attended-to, the best query match. Same code, same allowance, so any accuracy gap IS the rule.

A compiled KV working set

What each query keeps



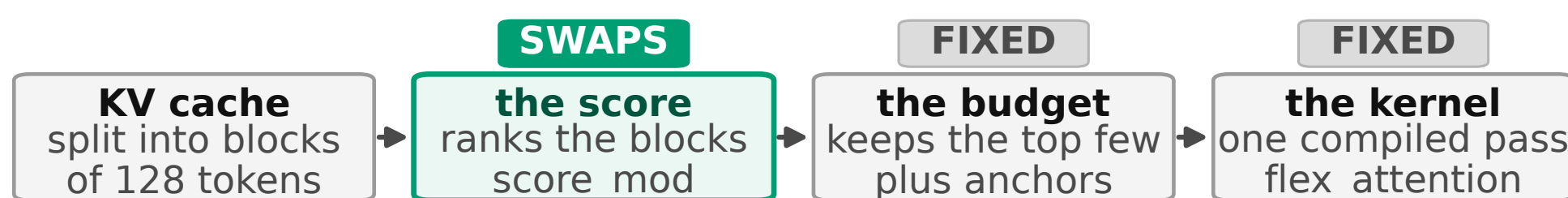
Reading row 6 (boxed)
Sees 7 blocks. 25% allows 2.
Keeps 4: first, newest, +2.

How to read the grid

- S** first block -- always kept
- L** most recent block -- always kept
- B** kept because it scored highest
- dropped to stay inside the budget
- in the future -- cannot be used

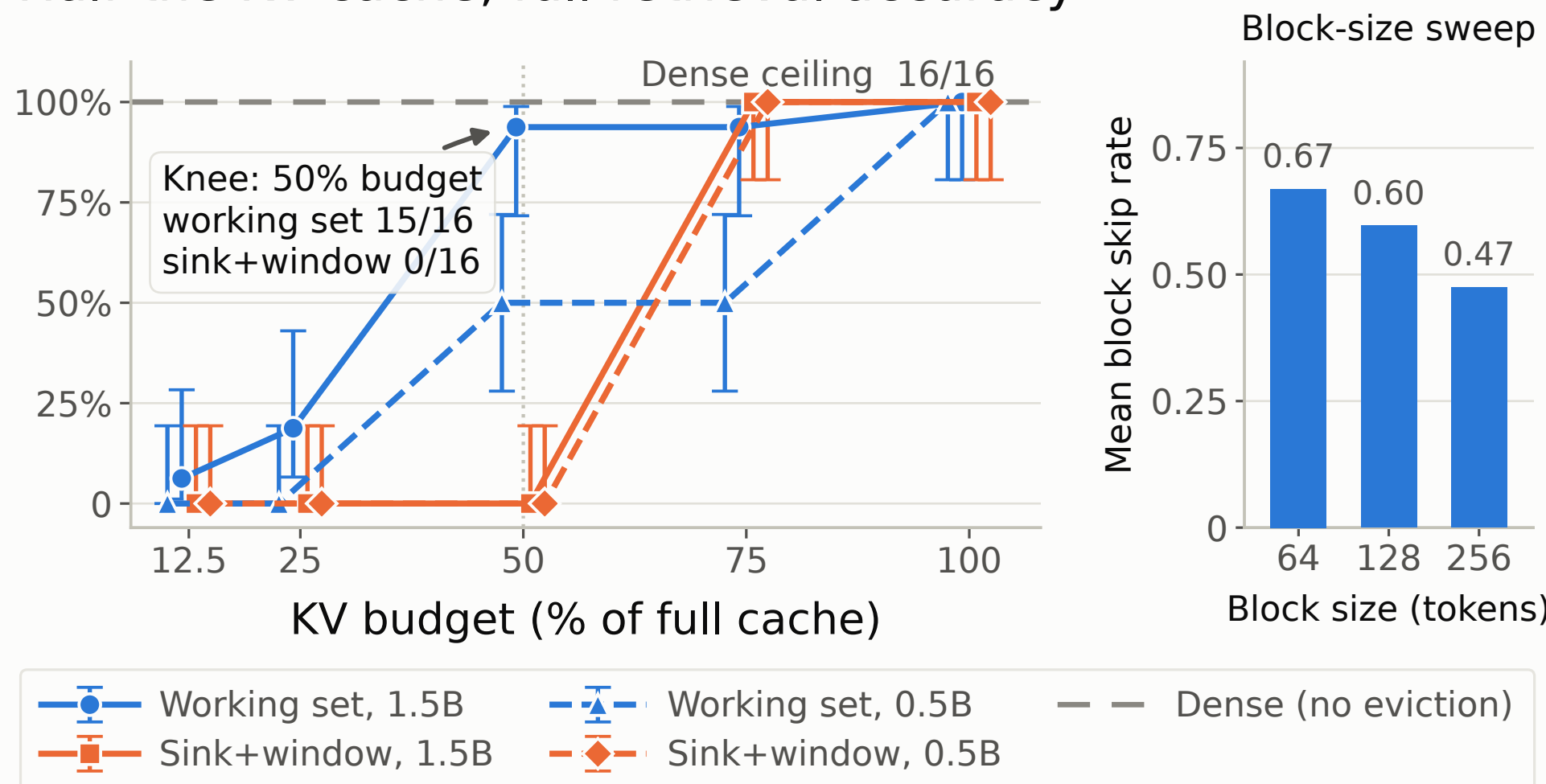
The score is the only swappable part
most recent most attended
best match both combined
position-corrected

Budget and kernel stay FIXED.
Only the score swaps, so it is
a controlled variable.



How much cache can you drop?

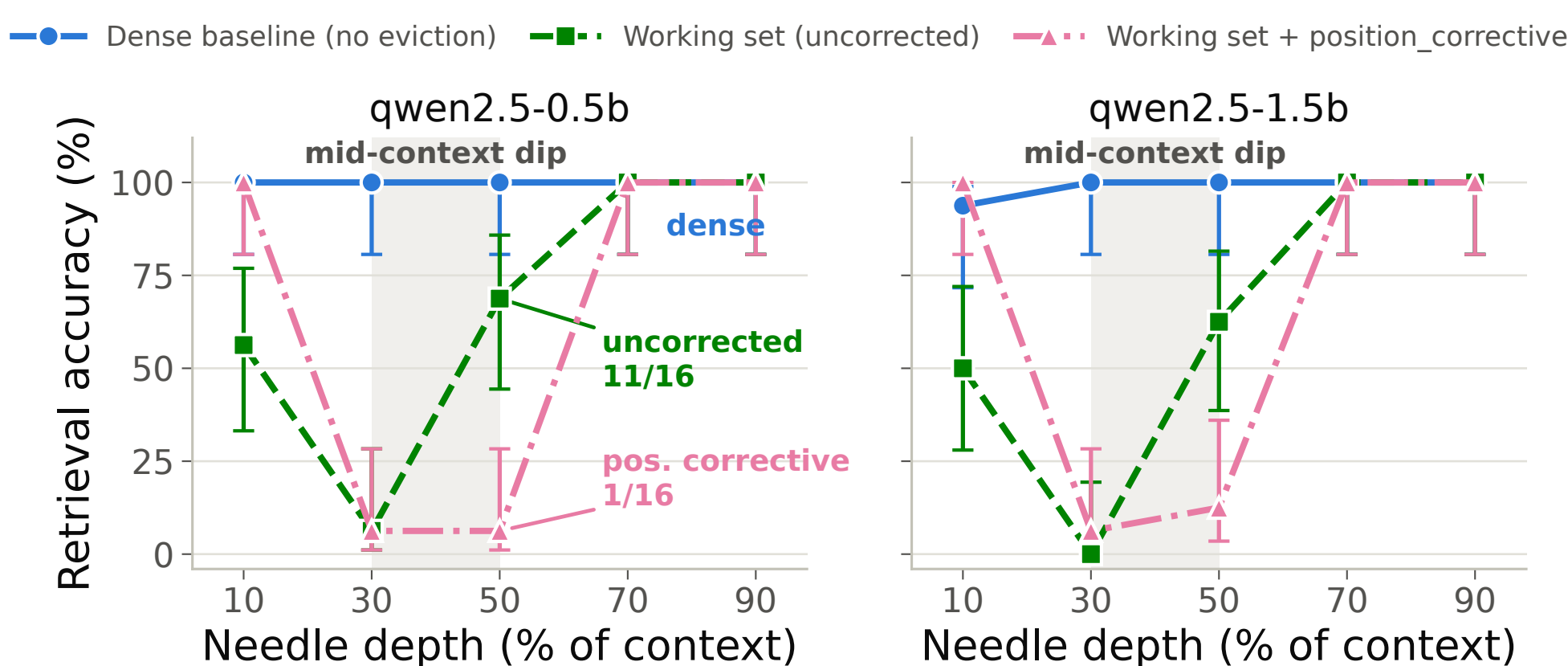
Half the KV cache, full retrieval accuracy



Find-the-fact task, 8K context. Accuracy is flat down to a 50% budget, then falls off a cliff. At 50% the scored working set answers 15 of 16 where a fixed recent-plus-first-tokens baseline answers 0 of 16.

Where it fails, shown not buried

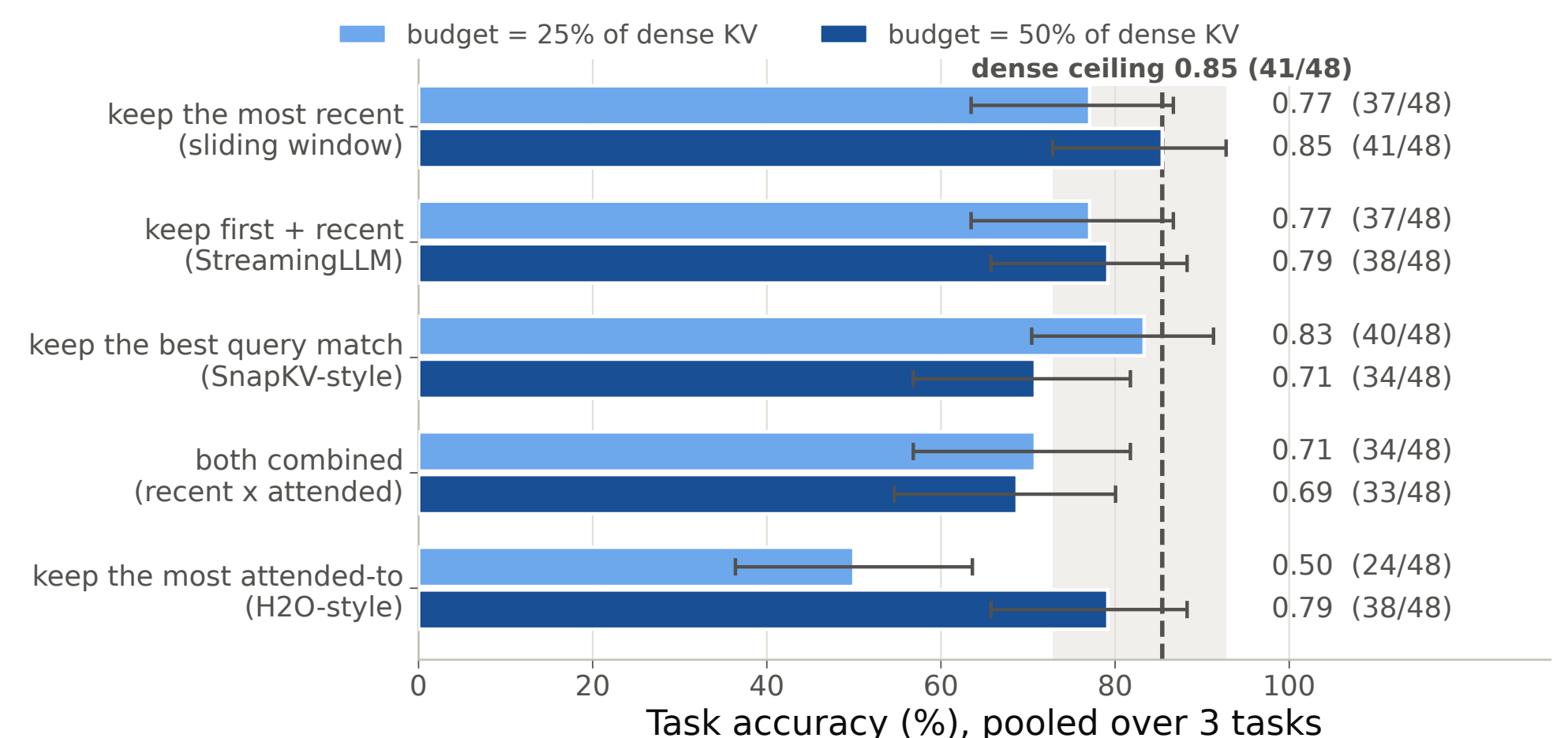
Mid-context recall (lost-in-the-middle)



Mid-context breaks it: 1 of 16 at 30% depth, against 16 of 16 for dense. A position-aware fix restored the edges, not the middle.

Do the rival rules differ?

Interchangeable importance signals at equal budget, same compiled kernel

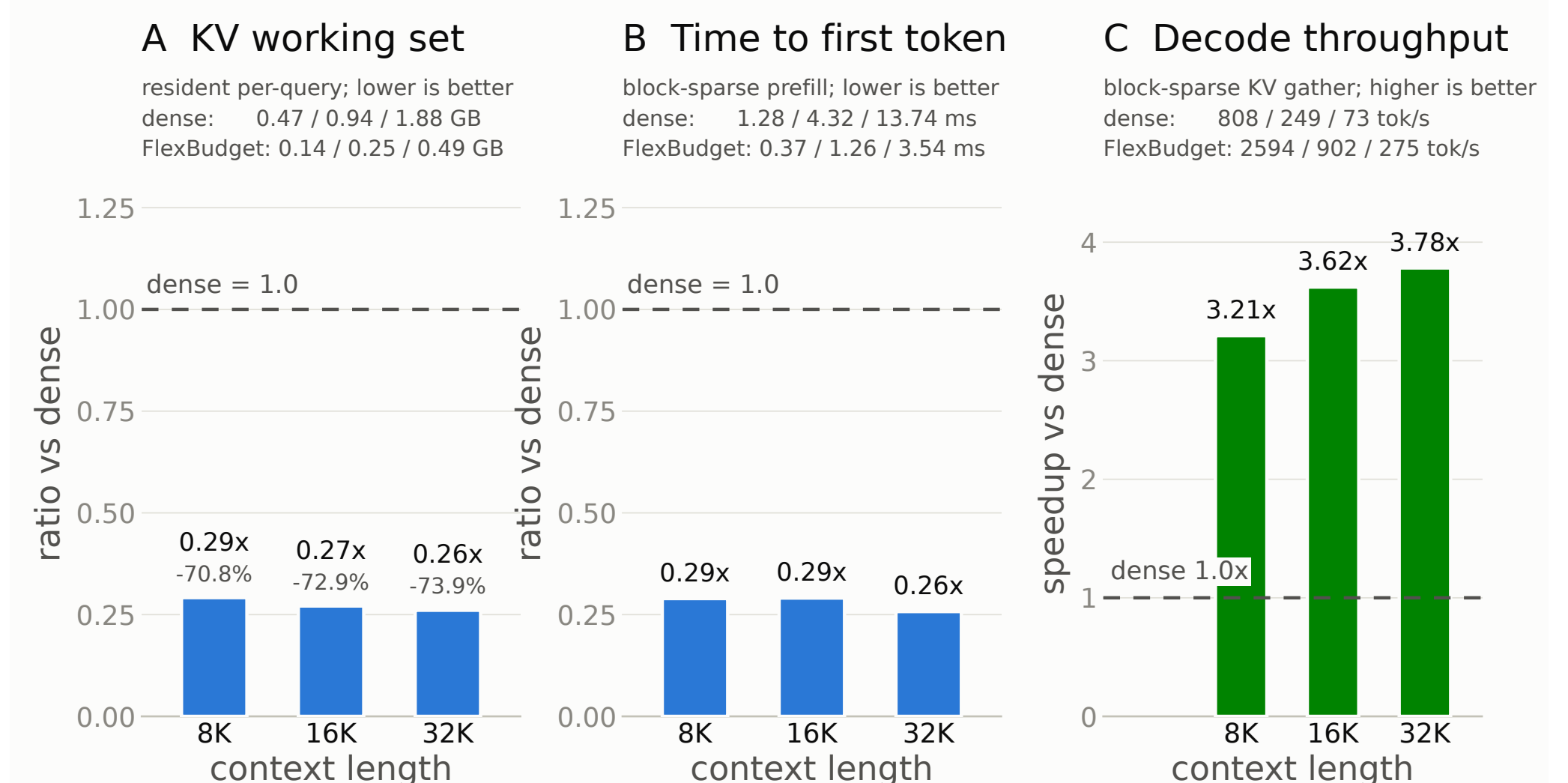


Five rules, one budget, one kernel. The error bars OVERLAP: at equal budget the rules are interchangeable. This is NOT a claim that any one of them wins.

What it buys, in this one setting

Measured on one AMD Instinct MI300X

Qwen2.5-7B, budget 0.25, block size 128



Takeaways

- Which blocks to drop becomes a controlled variable.
- At equal budget the published rules are hard to tell apart.
- Half the cache is enough -- until the answer is mid-context.
- Stock PyTorch: swap in your own rule.

What this does NOT claim

- Pilot scale: 16 trials per cell, one seed. Variance unmeasured.
- In the budget sweep two rules accidentally shared a code path, so those arms are not independent. Fixed in code, not yet re-run.
- No claim beyond one GPU, two model sizes and these tasks.

Reproducibility

- Core PyTorch only: flex_attention + torch.compile. No custom kernels, no retraining.
- One AMD Instinct MI300X, ROCm 7.2.1, torch 2.10.
- At full budget the mask matches dense attention to within 0.016 on all three head-dim legs, against an independent torch SDPA reference (tolerance 0.05).
- Every number traces to a result file.