

Why the Next Token Slows Down as the Chat Gets Longer

Sean Fraser

17 October 2026

Abstract

On a single machine, a local model can answer the first turns quickly and then slow down. The weights did not grow. The history did. Each new token reads keys and values already stored, so decode traffic grows with context even when the parameter count is fixed. This note writes that traffic down, shows how grouped-query attention and KV quantization change the slope, and separates those arithmetic facts from any claim about a particular card. Published FlashAttention speedups are training results. A 2 to 3 times speculative-decoding result on T5-XXL is not a promise for a long local chat. The useful comparison is the context length where cache reads overtake weight reads, because that is where a faster GPU and a shorter history stop being the same repair.

Introduction

The first reply is quick. After a few thousand tokens of history, the same model on the same GPU takes longer to produce the next token. Nothing was retrained. The prompt cache grew.

That is the subject of this note. Local and open-weight inference, the kind you can inspect and serve yourself, is often a batch of one sequence, or a handful. There is no crowd of other requests to hide the memory read. Decode becomes a tour of device memory: the weights, then a longer KV cache, then a small amount of arithmetic on the new token. A parameter count does not describe that tour.

My inference machine is one Threadripper PRO 7960X with one RTX PRO 6000. I am not reporting a timing run from it. The curves in this paper are calculated from a formula. They describe the shape of the bill. They do not report milliseconds on that card, because I did not measure those milliseconds here.

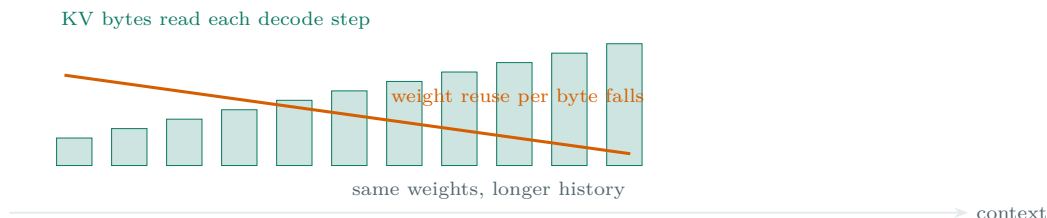


Figure 1: KV payload per request grows with cached context. The orange line is a sketch of weight reuse per byte of traffic, falling as the history lengthens, for a fixed parameter count. Calculated shape, not a profiler capture.

The Step That Rereads the Past

Autoregressive decoding conditions the next token on everything before it. After the first pass over the prompt, a practical implementation stores keys and values instead of recomputing every past hidden state. It still has to read them. Storage is not free attention.

For one request, the logical cache is

$$M(S) = 2LS h_{kv} d_h b_{kv}.$$

L is the layer count. S is the number of cached tokens. h_{kv} is the number of key-value heads. d_h is the head dimension. b_{kv} is bytes per element. The factor of 2 counts keys and values. A batch of B similar requests multiplies by B if nothing is shared. A shared system prompt should be counted once, plus each user’s suffix. If the engine does not share it, do not put that saving in the plan.

A decode step that streams all N weights at b_w bytes per parameter, and reads the cache once, moves about

$$D(S) = N b_w + B M(S)$$

bytes through the memory interface that is actually the bottleneck. On a single GPU that interface is the device memory bus, unless the working set has fallen out to host RAM. Offload changes the slope, usually for the worse, because host memory is not the device memory this formula assumes. The arithmetic below stays on device. Offload is a different machine.

Floating-point work on the new token does not grow with S in the dense projections. Attention work does. A roofline that ignores attention FLOPs can still show the right qualitative story at modest context: the step is memory traffic. At very long context the quadratic attention term can return.

FlashAttention’s point, in the 2022 paper, is that exact attention is often limited by IO between HBM and on-chip SRAM, not by a lack of FLOPs, and that tiling cuts those round trips without changing the math in exact arithmetic [Dao et al., 2022]. The paper’s wall-clock numbers are training runs: 15 percent on BERT-large at length 512 against an MLPerf record, 3 times on GPT-2 at length 1K, and 2.4 times on the long-range arena. I am not pasting those onto a local decode loop. The mechanism can travel. The stopwatch does not.

A Slope You Can Compute Before You Profile

Take a hypothetical local model, not a product SKU. $L = 32$, $h_q = 32$, $d_h = 128$. FP16 KV means $b_{kv} = 2$. Multi-head attention uses $h_{kv} = 32$. Grouped-query attention with 8 KV heads uses $h_{kv} = 8$. Multi-query attention uses 1. Bytes per token per request are

$$m = 2L h_{kv} d_h b_{kv}.$$

That is 512 KiB, 128 KiB, and 16 KiB. At 4096 cached tokens, one multi-head request holds 2 GiB of KV. The grouped-query request holds 512 MiB. The multi-query request holds 64 MiB. At 16384 tokens the multi-head figure is 8 GiB for one conversation, before weights. GiB here is 2^{30} bytes.

A card that also holds an 8-billion-parameter FP16 model, about 16 GB of weights, is already tight at that multi-head length if anything else is resident. Quantizing the weights to 4 bits changes the weight term. It does not change m unless you also quantize the cache.

This is why a long chat fails in a way a short benchmark does not catch. The model card fits. The conversation may not.

Grouped-query attention is not a runtime switch on a finished multi-head checkpoint. Ainslie and coauthors define it as an intermediate number of KV heads, between one and the query-head count. They also show a recipe for uptraining an existing multi-head checkpoint into multi-query attention with about 5 percent of the original pretraining compute. Uptrained grouped-query attention, in that paper, lands near multi-head quality with speed near multi-query attention [Ainslie et al., 2023]. If you did not uptrain, you do not have their quality result. You have a different model.

When Weights Stop Being the Larger Read

Define the ratio of weight bytes to KV bytes in one decode step:

$$\rho(S) = \frac{Nb_w}{BM(S)}.$$

While ρ is large, a second request amortizes the weights, and context length is a side issue. When ρ falls through 1, the history is the larger read. For the hypothetical 8-billion-parameter FP16 model, $Nb_w = 16 \times 10^9$ bytes. With grouped-query attention at 131072 bytes per token and batch 1,

$$S_{\text{eq}} = \frac{Nb_w}{m} = \frac{16 \times 10^9}{131072} \approx 122070$$

tokens. That is past a comfortable local window, so on this toy the weights still dominate a single 8192-token chat. Change the assumptions and the crossover moves. Four-bit weights cut Nb_w to about 4×10^9 bytes, and S_{eq} falls to about 30500 tokens. Eight concurrent sessions on the same grouped-query layout cut that again by 8, to a few thousand tokens. A 70-billion-parameter model at 4 bits is about 35 GB of weights, which raises S_{eq} unless several sessions share the card.

I am not recommending a window. The crossover moves when you change bits per weight, KV heads, or concurrency. One slogan that “8192 is fine” cannot survive all three changes. Write the three numbers down for the model you load. Then profile. If the profile disagrees, the formula missed a reread, a write, or a dequantizer. Believe the profile, and correct the formula. Do not discard the shape because one kernel did not match it.

A Table You Can Recompute

Same formula as the growth sketch. Hypothetical architecture, $L = 32$, $d_h = 128$, $b_{kv} = 2$, batch 1.

KV layout	Heads	Bytes per token	Cache at 4,096	Cache at 16,384
Multi-head	32	512 KiB	2 GiB	8 GiB
Grouped-query	8	128 KiB	512 MiB	2 GiB
Multi-query	1	16 KiB	64 MiB	256 MiB

If the engine reports a KV use far from this table, it is counting something else: an allocator reserve, a paged block tail, a duplicated head, or a bug. The gap is the measurement. The table is only the floor.

Quantization and Speculation Change Different Terms

AWQ is a weight-quantization method. It uses activation statistics to protect a small set of salient channels, scales those channels so it can avoid a mixed-precision hardware path, and does not need backpropagation on the calibration set. The authors report more than 3 times speedup over a Hugging Face FP16 implementation on desktop and mobile GPUs in TinyChat, and they show a 70-billion-parameter Llama-2 model running on a mobile GPU [Lin et al., 2023]. That is a weight-traffic and kernel result. It does not shrink $M(S)$. A long grouped-query cache on a quantized 70-billion-parameter model can still be the largest read in the step. Four-bit weights do not fix context unless the cache was quantized too.

KV quantization would change b_{kv} . I do not have a result in the sources for this draft that I will treat as a free halving of decode time. Halving b_{kv} halves the cache term in $D(S)$. Step time falls by that fraction only when the cache term dominated, and only when the kernel actually moves half the bytes. Both conditions are measurements.

Speculative decoding is the other lever people reach for when a local chat feels slow. Leviathan, Kalman, and Matias sample several draft tokens and verify them in one target pass. On T5-XXL they report a 2 to 3 times acceleration against the standard T5X implementation, with identical outputs, when the correction preserves the target distribution [Leviathan et al., 2022]. The acceptance rate decides whether you get that acceleration. If the draft model is weak on your domain, you pay for drafting and still emit about one token. If the step was already compute-bound, verification does not hide behind a memory stall that was not there.

A long context makes the target verification itself read a long cache. Speculation can still help. The draft model has to be cheap relative to that read, not merely smaller on a model card.

The constant-acceptance toy from the companion note still applies. For four proposals and a conditional acceptance probability of 0.8, expected tokens per verification cycle are 3.36. With draft cost at a tenth of a target step, the toy speedup is about 2.4 times. It is not the T5-XXL result, and it is not your chat.

Paging Does Not Shorten the History

PagedAttention stops the engine from reserving a contiguous maximum-length region for every request. It does not reduce the number of KV elements a length- S sequence must store if nothing is shared [Kwon et al., 2023b]. The June 2023 vLLM report is the place that states the waste figures I will use: 60 to 80 percent waste from fragmentation and over-reservation in the systems examined, and residual waste under 4 percent in the last block after paging [Kwon et al., 2023a]. Those figures are about reservation. They are not a shortening of S .

On a local engine the practical gain is that a 32768-capable allocation does not have to be touched on the second turn. The loss, if you are not watching, is that a long chat still allocates. The process grows until the machine starts paging to disk. Swapping a KV cache to disk is a decode-latency choice. It is usually a bad one. “It still runs” is not a latency result.

Prefix reuse is the case where paging’s sharing story matters on one machine. A fixed system prompt used by every session should occupy one cache, not one cache per tab. If the engine keys the cache on the raw string and a single date in the prompt changes, the lookup misses. Measure reuse. Do not assume the banner text is free the second time.

The Desk and the Cluster Are Different Machines

A serving deployment can amortize Nb_w by batching other people’s requests. On a desk, B is often 1. A second session is a choice you make against your own machine, not a traffic pattern you can assume. There is no other request to hide the weight read behind. Local decode can feel the context slope in a chat that a busy server would still treat as weight-dominated. “Just batch it” is useless advice for a single user’s next token.

Offload to CPU RAM is the other desk-specific trap. It lets a model load that does not fit in device memory. Every decode step that misses in device memory pays a bus that was not in $D(S)$. If the profiler shows the step waiting on host-to-device copies, the context-growth formula is no longer the whole story. You have added a second memory. Shorten the kept history, or buy device memory. Do not tune a kernel that is not on the critical path.

What to Time on a Desk

Pick one open-weight model and one build. Freeze the commit.

1. Generate from a fixed seed at temperature 0 if the sampler allows it. Do not mix sampler noise into the clock.
2. Sweep cached context from a few hundred tokens to the longest length you are willing to serve.
3. Record time per output token separately from time to first token. Time to first token includes the prompt pass. The complaint about a long chat is usually the later tokens.
4. Plot time per token against S . A flat line means you are not in the cache-read regime this note describes, or the timer is too coarse. A roughly linear rise should be compared with m divided by a measured memory bandwidth, not a datasheet peak.
5. Repeat with a second session that shares the system prompt, and without sharing, if the engine exposes that switch.
6. Repeat at 4-bit weights if you have a kernel. State whether KV precision changed.

A slope much steeper than m over measured bandwidth means extra reads, a host sync, or a scheduler tick. A slope much flatter means the cache is not being fully reread. Explain that before you treat it as a win.

One curve is a demonstration. Two curves that isolate Nb_w from m are a diagnosis. I did not run this sweep for the paper. The protocol is the part I will stand behind without a profiler screenshot.

Four Lengths, One Weight Term

The crossover is one number. The shape before it is the thing a desk chat actually sits on. Same toy as the rest of this note: 16×10^9 bytes of FP16 weights, grouped-query attention, 131072 bytes per token, batch 1. KV is in GiB, 2^{30} bytes. The ratio ρ does not depend on that unit choice.

Cached tokens	KV	ρ , weight bytes over KV bytes
512	0.062 GiB	238.42
4,096	0.500 GiB	29.80
8,192	1.000 GiB	14.90
16,384	2.000 GiB	7.45
32,768	4.000 GiB	3.73

At 8192 tokens the history is real, and it is still about a fifteenth of the weight read. A chat that slows there is not yet in the regime where cache bytes dominate this toy. Something else is on the step: a reread, a host copy, a dequantizer, or a timer that includes the prompt pass. That is why the sweep has to separate time to first token from time per later token.

The line is still rising. From 8192 to 32768 tokens the KV term grows by 3 GiB. The weight term does not. If a measured time-per-token curve is flat across that range, the engine is not moving this m on every step, or the timer cannot see it. If the curve rises much faster than m over a measured bandwidth, the formula is missing a read. Believe the curve, and write down the missing term. Do not throw out the table because the first kernel disagreed with it.

Where the Crossover Moves

$S_{eq} = Nb_w/(Bm)$, rounded to the nearest token, for the hypothetical head size used above. Four-bit weight bytes are half a byte per parameter. Seventy billion parameters at FP16 are 140×10^9 bytes. These are not product SKUs.

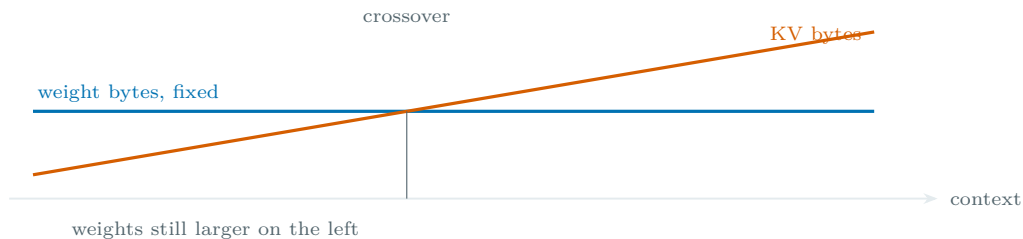


Figure 2: Weight bytes stay flat. KV bytes rise with cached context until the two meet. Left of the mark, weights are the larger read. Calculated shape for one toy, not a profile.

Case	Weight bytes	Layout	Batch	S_{eq}
8B, FP16	16×10^9	Grouped-query, 8 heads	1	122,070
8B, 4-bit	4×10^9	Grouped-query, 8 heads	1	30,518
8B, 4-bit	4×10^9	Grouped-query, 8 heads	8	3,815
8B, FP16	16×10^9	Multi-head, 32 heads	1	30,518

Case	Weight bytes	Layout	Batch	S_{eq}
8B, FP16	16×10^9	Multi-query, 1 head	1	976,562
70B, 4-bit	35×10^9	Grouped-query, 8 heads	1	267,029
70B, FP16	140×10^9	Grouped-query, 8 heads	1	1,068,115

Multi-query attention pushes the crossover out past any window you will keep on a desk. It does not do that for free. The quality result in Ainslie and coauthors is an uptrained model, not a flag [Ainslie et al., 2023]. Multi-head attention pulls the crossover in to about 30,518 tokens on the 8B FP16 toy, the same place 4-bit grouped-query attention lands for a single session. The two changes are not interchangeable. One changes the checkpoint. The other changes the weight bytes and needs a kernel.

A 70B FP16 crossover near 1,068,115 tokens is not a serving plan. The weights alone are 140×10^9 bytes. An assumed 80×10^9 byte GPU cannot hold them. The fit question comes first. The crossover is only meaningful on a placement that has already loaded the weights. I am not quoting a card. I am showing that a large S_{eq} can be unreachable for a reason the ratio does not contain.

A Shared Prompt Is One Cache or Eight

Take a 2048-token system prompt and eight sessions, grouped-query, 131072 bytes per token. The prompt is 0.25 GiB if the engine stores it once. It is 2 GiB if the engine stores it eight times. Add a 1024-token suffix on each session. The shared case is 1.25 GiB. The copied case is 3 GiB. The suffixes are not shareable. The prompt is shareable only if the cache key matches.

A one-token edit in the prompt is a miss if the key is the raw string. The 1.75 GiB difference then disappears, and the machine holds 3 GiB for text the user thinks of as one banner. Measure the engine’s reported reuse after a restart, and after a prompt edit. A warm process that still holds the old blocks is not a restart.

Paging does not create this saving. It allows it, if the block table can point two sequences at one physical block [Kwon et al., 2023b]. If the engine allocates a fresh block for every session, the logical formula and the resident bytes will disagree by about that 1.75 GiB in this example. The disagreement is the measurement.

What to Write Down on the Sweep

The protocol in the previous section is a list of actions. The sheet is the thing you keep. One row per completed sweep point. I did not fill a sheet for this paper.

1. Write the model revision, the build commit, the weight bits, and the KV bits.
2. Write S , whether the prefix was shared, and the batch.
3. Write time to first token separately from time per later token.
4. Write the engine’s reported KV bytes next to $M(S)$.
5. Repeat at a second S , changing nothing else, before you change weight bits.
6. Repeat at a second weight width only after the two context rows exist.

Two rows that differ only in S are the slope. Two rows that differ only in weight bits, at a short S where ρ is large, are the weight term. One row is a demonstration. A slope you cannot compare with m over a measured bandwidth is not yet a diagnosis.

Do not include the prompt pass in the per-token column. That pass is allowed to grow with S for a different reason. Mixing the two will make every model look cache-bound, including the ones this table says are not.

Offload Is a Second Bus

$D(S)$ assumes the weights and the cache are read from device memory. If the working set does not fit, the step pays a second bus. That bus is not in the crossover table. A 1 GiB cache copy at an assumed 16 GB/s takes 67 ms before the device can use those bytes. Sixteen gigabytes per second is a planning rate, not a measured copy on the Threadripper machine, and not a PCIe datasheet quote. Replace it with the rate the profiler shows.

The shape is enough to separate the cases. If the profiler shows the step waiting on host-to-device copies, shortening the kept history reduces the bytes on that bus. Buying a faster GPU does not, unless the new card also holds the working set. Quantizing the weights can make the model fit, and then the device-memory formula applies again. Quantizing the weights while the cache still misses does not remove the 67 ms sketch. The cache term has to fit, or be quantized, or be shortened.

Write the resident device memory next to the model size before you interpret a slow chat as context growth. If the process has already spilled, the four-length table is the wrong diagnosis. Fix the fit. Then sweep S .

The copy time is M/R at whatever rate the profiler records. The table uses three assumed rates so the slope is visible. It is not a measurement of the Threadripper machine.

Cache copied	8 GB/s	16 GB/s	32 GB/s
1 GiB	134 ms	67 ms	34 ms
2 GiB	268 ms	134 ms	67 ms
4 GiB	537 ms	268 ms	134 ms

A 4 GiB copy at 8 GB/s is half a second before useful device work. That is longer than a long local prefill in many chats, and it is still an assumption until you time the copy. If the measured rate is slower than 8 GB/s, the table is a lower bound on the damage, not an upper bound. Shorten the history or keep the cache on the device. Do not tune a kernel that is waiting on the bus.

Apply the same table to the grouped-query toy before you treat a slow chat as a weight problem. At 8192 tokens the cache is 1 GiB. At 32768 tokens it is 4 GiB. On the 16 GB/s planning rate those copies are 67 ms and 268 ms. The weight term did not change between those two lengths. If the measured step time grows by about that difference, and the profiler shows host copies, the history is leaving the device. If the measured step time does not grow, and the profiler shows no host copy, you are still on the device bus, and the four-length ρ table is the one to use.

A partial fit is the awkward case. Weights resident, cache spilling, is not the same as the whole model spilling. The copy table applies only to the bytes that miss. Write the miss size down. Do not multiply the whole $D(S)$ by a host-bus rate. That product will scare you, and it will be the wrong bill.

How to Read the Two Figures

The growth figure is the payload. The bars rise because each new cached token adds m bytes that the next decode step has to read. The falling line is weight reuse per byte of traffic, not a measured bandwidth. It falls because the weight term stays fixed while the KV term grows. Do not read the orange line as a latency curve. Latency is what the sweep is for.

The crossover figure is the same fact drawn the other way. Weight bytes are flat. KV bytes rise. They meet at S_{eq} for that toy. Left of the mark, a faster GPU and a shorter history are not the same repair: the larger read is still the weights. Right of the mark, shortening the history removes the larger term. A chat that feels slow at 8192 tokens, on the single-session FP16 grouped-query toy, is still left of the mark. Look for a reread, a host copy, or a prompt pass mixed into the timer before you treat that slowdown as cache dominance.

Use the figures in that order. Confirm which side of the mark the context you serve is on. Then decide whether the bytes are even on the device. The offload table is the second question. The figures do not answer it. A rising bar can be sitting in host memory. The profiler, not the drawing, says which bus moved the bytes.

Limits

The traffic model ignores activation traffic, KV writes, dequantization scales, and attention kernels that reread K and V. It assumes the weights stream once per step. A kernel that streams them twice will not match ρ . Expert-parallel models and sliding-window attention need their own m . A windowed model whose effective S is capped will flatten the curve. That flatten is a change in the model, not a change in bandwidth.

No energy figure is claimed. A slower decode on a plugged-in workstation is mostly a latency complaint. The same slope on battery power is an energy bill, and I have not metered one.

The hypothetical dimensions are not a released model’s configuration unless they happen to coincide. AI assistance was used to retrieve papers, check the arithmetic, and compile the PDF. This note is not peer reviewed.

Conclusion

On the only single-session FP16 grouped-query toy in this note, weights still dominate at 8192 tokens. Cache reads overtake weight reads near 122070 tokens. That crossover is where a faster GPU and a shorter history stop being the same repair. A chat that slows at a few thousand tokens is not yet that crossover, unless the weight bytes, the KV heads, or the batch are different from the toy. Count KV bytes per token. Count weight bytes. See which term the context length is about to overtake. Quantizing weights, sharing a prefix, and shortening the kept history move different terms. A faster GPU moves the bus under all of them. It does not tell you which term you are paying.

References

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints, 2023. URL <https://arxiv.org/abs/2305.13245>.

- Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness, 2022. URL <https://arxiv.org/abs/2205.14135>.
- Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Yu, Joey Gonzalez, Hao Zhang, and Ion Stoica. vllm: Easy, fast, and cheap llm serving with pagedattention. vLLM Blog, 20 June 2023. Retrieved 17 October 2026, 2023a. URL <https://blog.vllm.ai/2023/06/20/vllm.html>.
- Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention, 2023b. URL <https://arxiv.org/abs/2309.06180>.
- Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding, 2022. URL <https://arxiv.org/abs/2211.17192>.
- Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for llm compression and acceleration, 2023. URL <https://arxiv.org/abs/2306.00978>.