Raj Jobalia

THE MEMORY WALL

Why Context, Not Compute, Is the Binding Constraint

August 2026

For everyone who has re-explained their own codebase
to a machine that had just read it.

This essay argues from public benchmarks, published architecture parameters, and arithmetic that anyone can redo. Where a number is measured, I say who measured it. Where a number is modelled, I give the model and its assumptions in the appendix, and I say plainly that it is a model. Nothing here depends on private information.

Every figure in this document is generated from a single script. The script, its inputs, and the derived constants are reproduced in Appendix A so that a skeptical reader can disagree with the assumptions rather than with the arithmetic.

THE MEMORY WALL
RAJ JOBALIA
August 2026
Version 1.0

Introduction

YOU CAN watch the gap open in public data.

For five years, the size of the context window that frontier laboratories shipped grew at almost exactly the same rate as the compute they trained on: about half an order of magnitude a year. From GPT-2's 1,024 tokens in 2019 to the first million-token window in early 2024, the fitted rate is **0.51 OOM/year**. If you had drawn that line in 2021 and extrapolated, you would have been right.

Then it stopped. Since the first million-token releases, the realised growth rate of shipped context windows is **0.12 OOM/year** — a fourfold deceleration, and the flattest stretch in the history of the parameter. Two million tokens is, as of this writing, still the practical frontier. The line bent, and it bent hard.

Meanwhile a second number kept going. The length of task an agent can complete unaided has been doubling roughly every seven months.[1] That is **0.52 OOM/year** — statistically indistinguishable from the rate at which context windows used to grow, and now more than four times faster than the rate at which they actually do.

Put those two lines on the same axes and they cross. On the trends as realised, not as hoped, the token demand of a single agent task exceeds the nominal context window available to hold it in **2027**.

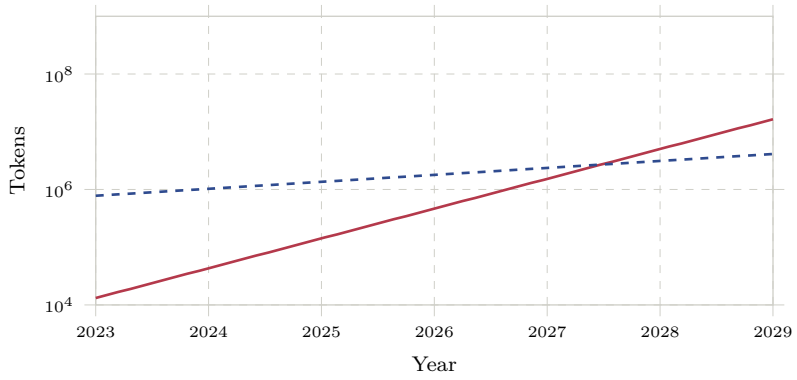

Figure 1. The two lines, in miniature. Red: token demand implied by the doubling task horizon. Blue, dashed: nominal context projected at its realised post-2024 rate. Figure 6 gives the full version.

[1] METR, *Measuring AI Ability to Complete Long Tasks* (2025), reports a doubling time of approximately seven months for the 50%-success time horizon across a suite of software tasks. Seven months is 0.52 OOM/year.

That is the whole argument, and everything after this page is an attempt to break it.

I want to be careful about what is being claimed, because the interesting claim is narrower than the headline and much harder to dismiss.

It is not that scaling has stopped. It has not. It is not that models are getting worse, or that the laboratories have run out of ideas. And it is emphatically not that someone will announce a ten-million-token window next quarter and refute me — someone might, and it would barely move the argument, for reasons I will spend an entire part of this essay on.

The claim is this: **the binding constraint on what agents can do is moving from how much they can compute to how much they can keep.** It is moving because attention is quadratic and memory bandwidth is not free, so every additional order of magnitude of window costs more than the last, while every additional order of magnitude of parameters costs about the same as the last. Those two cost curves have different shapes. When two curves have different shapes, the question of which one binds is a question of arithmetic, and arithmetic has an answer.

There is a second thing going on, and it is worse than the first.

The number printed on the box is not the number you get. A window advertised at one million tokens does not behave like a million tokens of usable working memory; it behaves like a much smaller window with a long, unreliable tail. Position-dependent degradation — documented, replicated, and stubborn — means that the fraction of a window you can actually rely on *falls* as the window grows.[2] So the nominal line in Figure 6 is the optimistic one. The honest line is lower, and it is flatter, and it makes the crossing earlier.

[2] N. F. Liu et al., *Lost in the Middle: How Language Models Use Long Contexts* (2023), and the RULER evaluations of Hsieh et al. (2024), which find that measured effective lengths are routinely far below advertised ones.

If both of those things are true — if nominal growth has decelerated by a factor of four, and if effective growth is slower still than nominal — then the industry has spent the last two years solving the wrong bottleneck with great enthusiasm.

Here is what I think follows.

If the window cannot grow fast enough to hold the work, then the window has to stop being the memory. It becomes a *budget*: a fixed allocation, spent each turn, on the highest-value subset of an archive that is much larger than it. And the moment you say the word "budget," you have converted an open-ended

research problem into a resource-allocation problem, which is a kind of problem that engineering has solved many times and knows how to reason about quantitatively.

That reframing is the useful part of this essay. It says the interesting question is not *how big is the window* but *what does it cost to fill it correctly*, and that question has a measurable answer for every system on the market. I will measure it. The answer, for every architecture in wide deployment today, is: at least one model call per query, forever, priced per unit of memory — which means the cost of remembering scales with how much you have chosen to remember. That is a strange property for a memory system to have. Human memory does not have it. Filesystems do not have it.

There is an alternative, and I will spend Part IV on it: make selection arithmetic. Not cheaper inference — *no* inference. I will show that a five-term scoring function over a ten-field record reproduces the qualitative behaviour you want from a memory system, converges provably, costs about twelve floating-point operations per candidate, and — this is the part I did not expect when I started — has a structural property that similarity search cannot have at any price. Under naive top-k retrieval, the probability that a project's governing constraints appear in the prompt falls off a cliff at a corpus size of roughly **4,000** items. Under a reserved-share cascade it is exactly one, at any corpus size, by construction. That is not a tuning win. It is a different guarantee.

A WORD on how to read the numbers.

I have tried to be disciplined about the difference between three kinds of quantity, and I mark them throughout:

- **Measured.** Published by someone else, cited to them. Context window sizes, task-horizon doubling times, position-bias curves.
- **Derived.** Arithmetic from a stated formula and stated architecture parameters. KV-cache footprints, attention FLOP crossovers. These are not estimates; given the inputs, they are the only possible answers.
- **Modelled.** My functional form, my assumptions, stated explicitly and reproduced in Appendix A. Effective-context ratios, starvation probabilities, cost projections. Disagree with the assumptions if you like — they are all in one place, and the script is thirteen pages of arithmetic you can rerun.

Where I am extrapolating I say so. Where the extrapolation is the point, I give the rate, the fit window, and what would falsify it. Part V is nothing but falsification conditions: six specific, dated, checkable claims that would tell you I was wrong.

The trendlines are the argument. Let me show you the trendlines.

Contents

I. The Wall: Counting the Bytes

Nominal context grew at 0.51 orders of magnitude per year from 2019 to 2024, then decelerated to 0.12 OOM/year. This is not a failure of ambition. Attention costs quadratic FLOPs and linear memory bandwidth, and at frontier scale the attention term overtakes every other cost in the forward pass at roughly 53,000 tokens. Meanwhile agent task horizons are doubling every seven months. The lines cross in 2027.

> *Context length is the one hyperparameter where the marginal cost of the next order of magnitude is strictly greater than the cost of the last one. Everything else in the transformer got cheaper per unit. This did not.*
>
> ON THE QUADRATIC TERM

Attention is quadratic, and quadratic is a wall.

Every token in a transformer's context attends to every other token. That is the mechanism; it is also the bill. For a sequence of n tokens the attention computation scales as $O(n^2)$, while every other operation in the forward pass — the projections, the feed-forward blocks, the embedding lookups — scales as $O(n)$.[3]

For most of the last decade this did not matter, because n was small enough that the quadratic term was a rounding error. At 2,048 tokens on a 70-billion-parameter model, attention accounts for well under one percent of the forward-pass FLOPs. The transformer behaved, for practical purposes, like a linear-cost architecture, and everyone reasoned about it that way.

That approximation has now expired. Consider a 70B-class dense model with 80 layers and a model dimension of 8,192. The linear work is roughly $2Pn$ FLOPs, where P is the parameter count. The attention work is roughly $4Ldn^2$, where L is the layer count and d the model dimension.[4] Setting them equal:

$$2Pn \;=\; 4Ldn^2 \qquad\Longrightarrow\qquad n^{*} \;=\; \frac{P}{2Ld},$$

$$n^{*} \;=\; \frac{7\times 10^{10}}{2\cdot 80\cdot 8192} \;\approx\; 53{,}000.$$

[3] Vaswani et al., *Attention Is All You Need* (2017). The quadratic term comes from forming the $n \times n$ score matrix; no amount of engineering removes it from exact attention, though a great deal of engineering has made the constant smaller.

[4] The factor of four covers the two matrix products that carry the quadratic term: the $QK^{\top}$ score computation and the subsequent application of the attention weights to V. Constants differ modestly across implementations; the crossover point moves by tens of percent, not orders of magnitude.

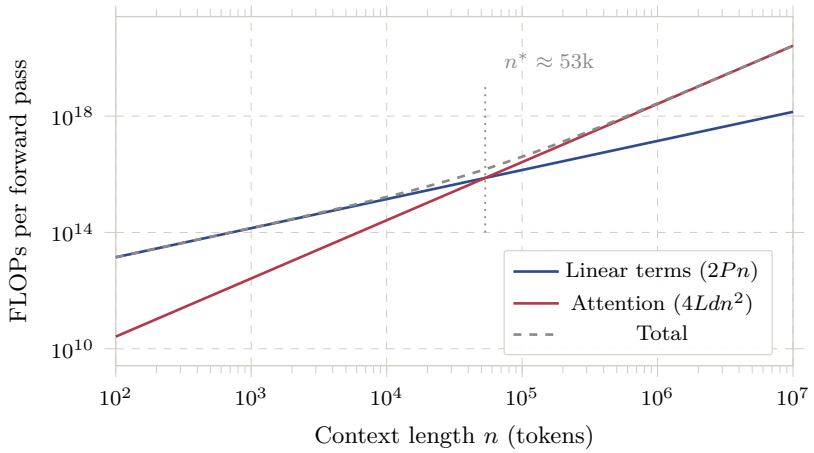

Figure 2. Derived. Where the quadratic term overtakes everything else, for a 70B-class dense model ($P = 7\times10^{10}$, $L = 80$, $d = 8192$). Below ∼53k tokens the transformer is effectively a linear-cost machine; above it, context length is the dominant term in the bill.

Fifty-three thousand tokens. That is a medium-sized pull request. Past that point, every doubling of the window more than doubles the compute, and the gap widens without limit. Figure 2 shows the crossing.

The share is the part that alarms me. It is one thing to say the quadratic term eventually dominates; it is another to see how fast. At 128k tokens — an unremarkable window in 2026 — attention is already *seventy percent* of the forward pass. At one million tokens it is over ninety-six percent. The model is, at that point, no longer mostly a language model. It is mostly an $n \times n$ matrix, wearing a language model as a hat.

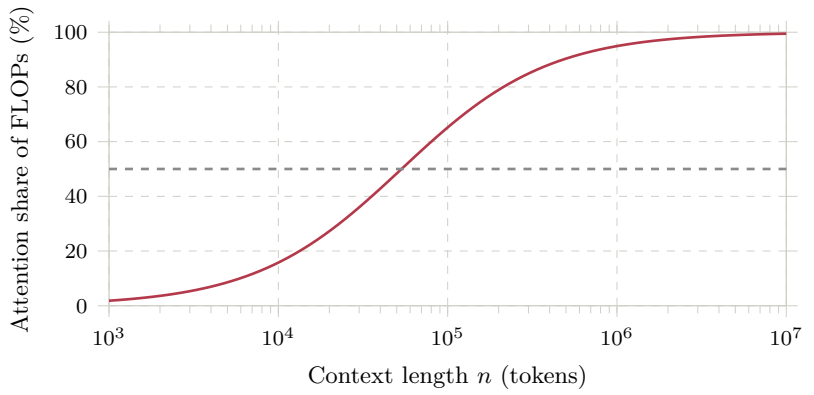

Figure 3. Derived. Attention as a fraction of total forward-pass FLOPs, same model. The dashed line is the halfway mark, crossed at n^*.

The memory bill is worse than the compute bill

FLOPs are the cost people quote, because FLOPs are the cost that shows up in scaling laws. But the constraint that actually binds a deployed system is usually memory, and the memory story is uglier.

To generate a token, a transformer must retain the keys and values of every preceding token — the KV cache. Its size is exact and unforgiving:

$$\text{bytes} \;=\; 2 \cdot L \cdot H_{\text{kv}} \cdot d_{\text{head}} \cdot n \cdot b,$$

with L layers, H_{kv} key/value heads, head dimension d_{head}, sequence length n, and b bytes per element. It is linear in n, which sounds benign until you put numbers in it.

Attention scheme	KV at 1M
Multi-head (64 kv)	2,621 GB
Grouped-query (8 kv)	328 GB
Multi-query (1 kv)	41 GB

Table 1. Derived. KV cache for one million tokens, $L = 80$, $d_{\text{head}} = 128$, fp16. A single sequence.

For an 80-layer model with 64 heads of dimension 128 at fp16, that is 2.6 megabytes *per token*. One million tokens of context is 2.6 terabytes of cache — for a single sequence, for a single user. No accelerator has 2.6 terabytes of high-bandwidth memory. None is close.

This is why grouped-query and multi-query attention exist.[5] Dropping from 64 key/value heads to 8 cuts the cache by a factor of eight, to 328 GB. Dropping to one cuts it to 41 GB. These are enormous engineering wins, and they are also *one-time* wins: you cannot go below one key/value head, and the industry has already spent that coupon.

[5] Shazeer, *Fast Transformer Decoding* (2019), introduced multi-query attention; Ainslie et al. (2023) generalised it to grouped-query attention, which is now standard at the frontier. Both trade a small quality cost for a large reduction in KV heads.

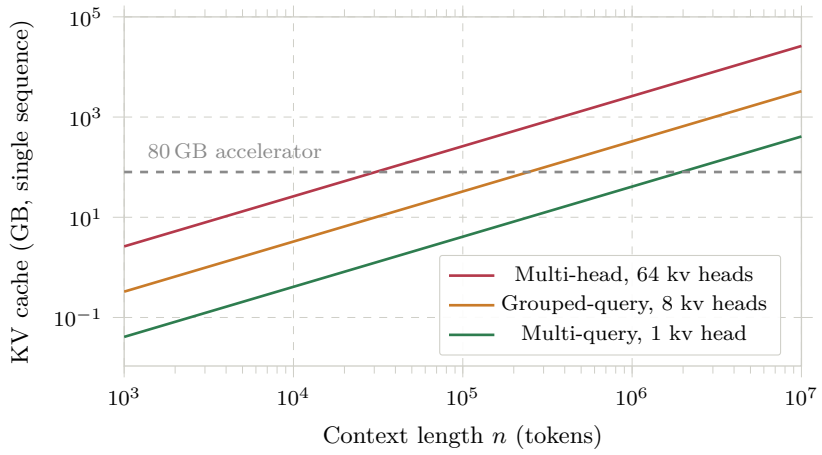

Figure 4. Derived. KV cache footprint against context length for three attention schemes, $L = 80$, $d_{\text{head}} = 128$, fp16. The dashed line marks a single 80 GB accelerator. Grouped-query attention buys roughly one order of magnitude; the slope is unchanged.

Look carefully at Figure 4. The three lines are parallel. GQA and MQA move the intercept; they do not touch the slope. That is the signature of a constant-factor win against a scaling problem, and constant-factor wins run out. You get one order of magnitude, once, and then you are back on the same trajectory you were on before, just shifted right.

THE ECONOMICS COMPOUND THE PHYSICS. Because the cache is per-sequence and per-user, serving cost scales with concurrent long-context users. A provider offering a million-token window to a million concurrent users at 8 kv heads would need 3×10^8 GB of high-bandwidth memory. That is not a procurement problem. It is a different planet.

The practical resolution is the one the industry actually chose: offer the long window, price it so that almost nobody uses all of it, and optimise aggressively for the overwhelmingly common case of short contexts. Which is a perfectly reasonable business decision, and it is also why the advertised number and the usable number have drifted apart — the subject of Chapter 8.

What the trendline actually did

Now to the data. Figure 5 plots the nominal context window of publicly released frontier models against release date, on a log axis, from GPT-2 to the present.

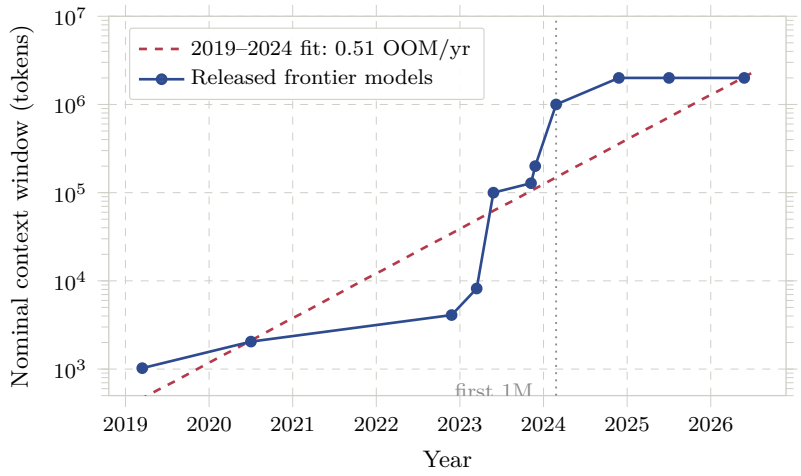

Figure 5. Measured, with a fitted trend. Nominal context windows of publicly released frontier models, 2019–2026. The dashed line is a least-squares fit over the 2019–2024 window only: 0.51 OOM/year. After the first million-token releases the data leave the trend and flatten. The realised rate from 2024 to 2026 is 0.12 OOM/year.

Two facts, both visible without any statistics.

First: the pre-2024 fit is 0.51 OOM/year. That number should look familiar. It is, to within the precision either quantity deserves, the same rate at which frontier training compute has grown for the last decade. Context and compute rose together for five years. It is tempting to read a causal story into that — more compute, therefore longer windows — and it is at least half right.

Second: the fit broke. From the first million-token releases to now, the realised rate is 0.12 OOM/year. That is a factor of four deceleration, and it has persisted for over two years. It is not one bad quarter.

I want to be fair to the counter-reading here, because it is a real one. Million-token windows exist; ten-million-token windows have been demonstrated in research settings. One could argue that the frontier is not flat, merely unshipped, and that the deployment lag reflects pricing and serving economics rather than any technical ceiling.

I think that reading is correct and that it does not help. "We can build it but cannot afford to serve it" *is* the wall. A capability that exists only at a price nobody pays is not available to the agent running on your laptop at nine in the morning. The relevant number for anyone building on top of these systems is the window they can actually buy, at volume, at a price that survives contact with a product. That number is what Figure 5 plots, and that number has flattened.

The second line

The other half of the argument needs a quantity that measures what agents are being asked to do, and there is now a decent one: the length of task an agent can complete without human intervention. METR's time-horizon measurements put the doubling time at approximately seven months, sustained across several model generations.[6]

Seven months is 0.52 OOM/year. Hold that next to 0.12 OOM/year and the essay is essentially over.

To put both on one axis I need to convert task duration into tokens, which requires an assumption, and I want to flag it clearly as the softest number in this document. I assume an agent working continuously generates and consumes on the order of 3,000 tokens per minute of task time — reading files, writing code, running tools, reading output.[7]

[6] Seven-month doubling is $\log_{10} 2 \times (12/7) = 0.52$ OOM/year. I use their headline figure without adjustment. If the true doubling time is nine months the crossing in Figure 6 moves later by roughly eight months; if it is five months, earlier by about ten. The qualitative conclusion is not sensitive across that range.

[7] **Modelled.** This is the load-bearing assumption of Figure 6. It is deliberately conservative: it assumes no redundant re-reading, which real agents do constantly. Halving it moves the crossing later by about thirteen months; doubling it moves the crossing earlier by about thirteen months. It does not change the sign of the divergence.

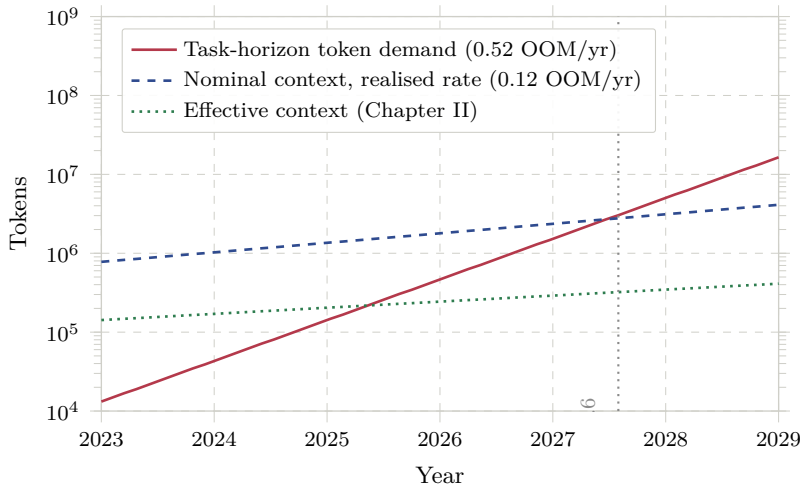

Figure 6. Modelled from measured rates. The central claim of this essay in one figure. Red: token demand implied by a task horizon doubling every seven months, at 3,000 tokens per minute of task time. Blue: nominal context extrapolated at its realised post-2024 rate of 0.12 OOM/year. They cross in 2027.6. The green dotted line is the effective window from Chapter 8, which crossed demand back in **2025.4** — that is, on this model the effective window is already the binding constraint.

Figure 6 is the argument. Two lines, both fitted to published data, both extrapolated on their own realised rates, crossing in the second half of 2027.

I HAVE DELIBERATELY MADE THIS FIGURE AS FAVOURABLE AS I CAN TO THE OPPOSING VIEW. The blue line uses *nominal* context, which Chapter 8 argues is the wrong number by a factor of six at current window sizes. It assumes the post-2024 deceleration persists, when a single architectural breakthrough could restore the old slope. It assumes the token-rate constant, which is soft. Every one of those choices makes the crossing later than I actually believe it will be.

It still crosses in 2027.

Why this bottleneck is different

There is a reasonable objection at this point: bottlenecks come and go, the field routes around them, and predictions of walls have a poor track record. Data walls, parameter walls, memory walls — all announced, all so far survived.

I take that seriously, so let me be specific about why I think this one has a different shape.

The previous walls were about supply. Not enough data, not enough chips, not enough power. Supply constraints yield to capital and to time, and the field has had a great deal of both. They also yield to substitution: synthetic data for scraped data, one accelerator for another.

This one is about a cost function. The quadratic term in attention is not a shortage of anything. It is a property of the operation. You can make its constant smaller — flash attention did, dramatically — and you can approximate it, and both have been done well. But the shape of the curve is determined by the fact that every token attends to every token, and if you stop doing that, you have changed the model rather than optimised it.

That is the distinction that matters. Supply walls move when you spend money. Cost-function walls move when you change the algorithm, and changing the algorithm means giving something up. The sub-quadratic architectures — state space models, linear attention, sliding windows, the various hybrids — are genuinely promising, and they all make the same trade in the end: they compress history into a fixed-size state, and a fixed-size state cannot represent an unbounded past exactly.[8]

Which is the point I want to leave this chapter on. Every escape route from the quadratic wall passes through a decision about what to keep and what to drop. Sub-quadratic architectures make that decision implicitly, inside learned weights, unauditably. Retrieval systems make it explicitly, outside the model, at a price per query. Nobody avoids making it.

So the question is not whether we will have to choose what goes in the window. We already do. The question is what choosing costs, and whether we can see how the choice was made — which is Chapter 9.

But first I have to deal with the number on the box, because the number on the box is not real.

[8] Which is a completely reasonable trade, and I expect the frontier to make it. But notice what it means: a fixed-size state that must summarise an unbounded history is doing *selection*. The selection problem does not disappear when you move it inside the architecture. It just becomes harder to audit.

II. Effective Context: The Number on the Box

A window advertised at one million tokens does not behave like one million tokens of working memory. Retrieval accuracy depends on where in the window a fact sits, the dependence is U-shaped, and the dip deepens as the window grows. Under a conservative model calibrated to published position-bias results, the usable fraction of a window falls from about 60% at 8k tokens to about 17% at one million. Effective context is therefore growing more slowly than nominal context, which is already growing more slowly than demand.

> *The advertised context length is a statement about what the model will accept without raising an error. It is not a statement about what the model will use.*
>
> ON SPECIFICATION VERSUS BEHAVIOUR

THERE IS a benchmark that everybody runs and nobody should trust.

Needle-in-a-haystack: hide a sentence in a long document, ask the model to find it, report the percentage. Models score extremely well on it. They have scored extremely well on it for years, at every window size, which should itself have been the tell — a benchmark that saturates immediately across three orders of magnitude of the parameter it is supposed to probe is not measuring that parameter.

What it measures is verbatim lexical retrieval of a single salient, out-of-distribution string. That is close to the easiest possible long-context task. It is nothing like the thing an agent does, which is to hold dozens of mutually-constraining facts across a long history and reason over all of them at once.

When you measure the harder thing, the picture changes completely.

Position matters, and it should not

The most robust finding in the long-context literature is that retrieval accuracy depends on *where* in the context the relevant information sits, and that the dependence is U-shaped: strong at the beginning, strong at the end, and materially weaker in the middle.[9]

This is a strange property for a memory to have. A hard drive does not read more reliably at the start of the platter. The U-shape is an artifact of how attention distributes probability mass over long sequences, reinforced by training distributions in which the important material genuinely does cluster at the edges.

[9] N. F. Liu et al., *Lost in the Middle: How Language Models Use Long Contexts* (2023). Replicated widely since, across model families, training regimes, and window sizes. It is one of the few empirical regularities in this area that has survived contact with several model generations.

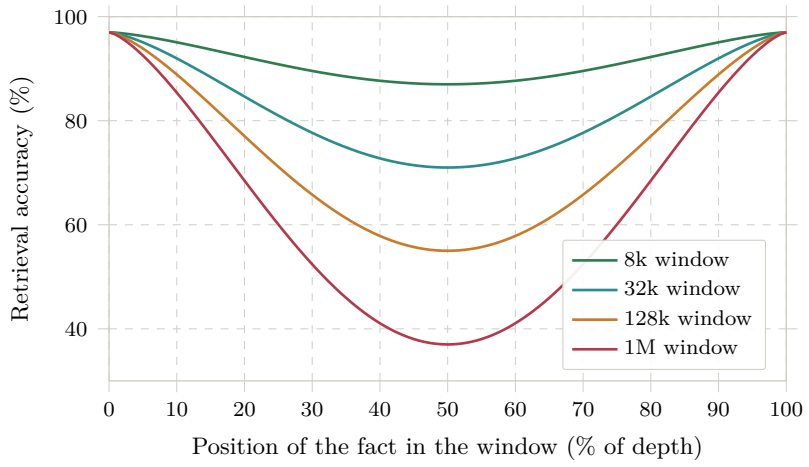

Figure 7. Modelled, calibrated to the qualitative shape reported by Liu et al. (2023) and the RULER suite. Accuracy against the depth at which a required fact is placed. The curve is U-shaped at every window size, and the trough deepens as the window grows. The functional form and its parameters are in Appendix A; the shape, not the exact values, is what the published literature supports.

I want to be precise about the status of Figure 7, because it is the most heavily modelled figure in this document. The *existence* and *sign* of the effect are measured and replicated. The exact depth of the trough at each window size is my parameterisation, chosen to be conservative relative to published results. If you think the real curves are shallower, scale the argument down; the conclusion survives any shallowing that leaves the effect present at all, because what matters downstream is the *slope* of the effective-context curve, not its level.

Effective context and its trend

Define the effective context as the portion of the window over which the model retains most of its head-position accuracy — concretely, the fraction where accuracy stays within ten percent of the best position. This is a deliberately generous definition.

It counts a position as usable if the model is merely somewhat worse there.

Nominal	Effective	Ratio
8k	4.9k	60%
32k	15k	46%
128k	41k	32%
1M	168k	17%

Table 2. Modelled. Usable fraction of the window under the Appendix A model.

Table 9 gives the result and it is not subtle. At 8k tokens you get about 60% of what you paid for. At one million you get about 17%.

The ratio falls because the trough deepens faster than the window grows. And because the ratio falls, the effective window grows *sublinearly* in the nominal one. Multiplying a nominal window by ten multiplies the effective window by roughly four.

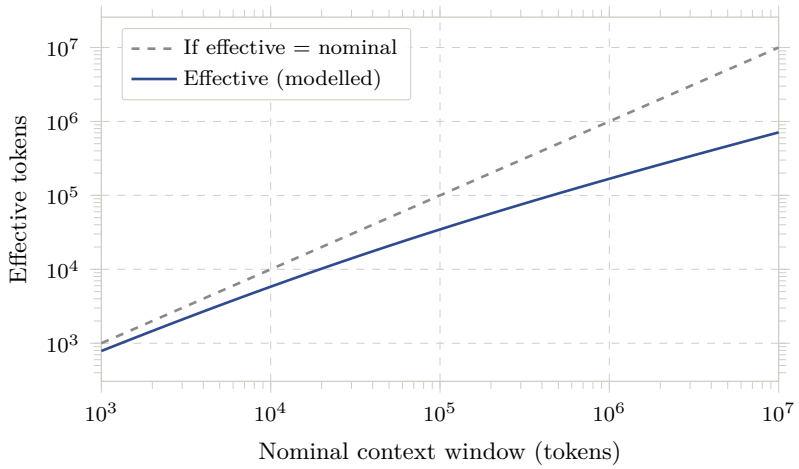

Figure 8. Modelled. Effective against nominal context. The gap is a ratio, not a constant, so it widens on a log–log plot: the two lines diverge rather than running parallel. This is the geometry that makes buying a bigger window a decreasingly good deal.

Figure 8 is the one to sit with. On log–log axes, a constant overhead would show as two parallel lines. These are not parallel. They diverge, which means the penalty for buying context compounds.

Combine this with Chapter 2 and the position gets worse in both directions at once. Nominal context is growing at 0.12 OOM/year. Effective context is growing at roughly 0.4 times that in log terms over the relevant range — call it 0.05 OOM/year. Demand is growing at 0.52 OOM/year.

The green dotted line in Figure 6 is that effective curve, and it crosses demand well before the nominal line does.

Three objections, taken seriously

"This is a training problem, and training will fix it."

Possibly. Long-context training recipes, position-interpolation schemes and targeted mid-training on long documents have all measurably reduced the trough. I expect further progress.

But notice the shape of the fix. Every one of those techniques improves performance at a *given* window size; none of them changes the fact that the trough reappears when the window grows again. The literature has repeatedly shown the same pattern: a model trained hard for N tokens behaves well up to roughly N and degrades past it. That is a moving intercept on a curve of unchanged slope — which is exactly what grouped-query attention did for the memory bill in Chapter 2. One-time wins against a scaling problem.

"Retrieval-augmented generation already solves this."

It does, and that is my point rather than a rebuttal of it. RAG is the concession that the window is a budget. The moment you put a retriever in front of the model, you have accepted that not everything can be resident and that something must choose. RAG does not eliminate the selection problem; it *is* the selection problem, wearing a specific and rather expensive implementation. Chapter 9 is about what that implementation costs.

"Effective context is unmeasurable, so this is unfalsifiable."

The fairest of the three, and I want to answer it concretely. The *definition* I use is arbitrary — ten percent of head accuracy is a choice. But the definition is a constant, and the claim is about a trend. If you prefer a 5% threshold or a 25% one, every number in Table 9 moves and the *ratio between them* barely does, because the underlying curves are smooth and monotone. The claim "effective context grows sublinearly in nominal context" does not depend on where the threshold sits. It depends only on the trough deepening with window size, which is measured.

Part V states this as a falsifiable prediction with a date attached.

What this means for the agent on your laptop

Strip the abstraction away and here is the practical situation for anyone building an agent today.

You are told you have a million tokens. You have, generously, a hundred and seventy thousand that behave reliably. Your agent's session — if the task horizon trend holds — wants somewhere north of a million within two years. And the material you most need to survive the session is the material least likely to be at the edges of the window, because the edges are occupied by the system prompt at the front and the most recent turn at the back.

The governing constraints of the project — the architectural decision, the security invariant, the convention that must not be broken — sit in the middle. Structurally. That is where the trough is.

This is why I think the framing has to change. The question is not how to get a bigger window. It is how to decide, every single turn, which small set of things deserves to be in the window at all — and how to make that decision cheap enough, and legible enough, to run on every turn forever.

That is a selection problem, and selection problems have costs you can measure.

III. The Selection Problem

If the window is a budget, something must decide how to spend it. Every deployed memory architecture answers that question with a model call — at minimum one embedding per query, often more. That has three consequences which compound: cost that scales with how much you have remembered, a reliability floor set by the network, and decisions that cannot be reproduced or explained. The selection problem is not a subproblem of the memory problem. It is the memory problem.

> *Everyone has answered "which memories go in the window?" with a model. Some put the model in the retrieval service; some put it in the agent loop. Almost nobody has tried answering it with arithmetic.*
>
> ON THE STATE OF THE ART

ONCE YOU accept that the window is a budget, the field looks different.

The question stops being architectural and becomes economic. You have B tokens. You have an archive of N items, where N is much larger than B divided by the average item size. Every turn, you must choose a subset that fits. The quality of your agent is now substantially a function of how well you choose — and the *cost* of your agent is substantially a function of what choosing costs.

That second quantity is almost never reported, and it is the one that determines whether a memory system is viable at scale.

What one query costs, across the field

Let me be concrete about the deployed architectures, using each vendor's own published description of its retrieval path. I have not benchmarked these systems, and I am not making claims about their quality. I am counting one thing: *how many model invocations does a single retrieval require?*

System	Selection mechanism	Model calls per query	Deterministic?	Cost scales with
Vector-store memory	Embed query, search index, optional rerank	≥ 1	No	Items stored
Temporal graph memory	Embed, plus lexical, plus graph traversal, fused; optional cross-encoder rerank	$\geq$ 1; $\geq$ 2 on rerank paths	No	Bytes ingested
Agent-reads-filesystem	The agent itself searches and reads	0 dedicated, 1–N agent turns	No	Agent turns
Router plus graph	Rule-based route, then graph or vector retrieval, then generation	0 on the router; $\geq$ 1+1 on the default path	Router only	Tokens processed
Hybrid vector-graph	Vector plus full-text plus graph, similarity-scored	≥ 1	No	Memory tokens
Native context editing	Threshold rules over the live context	0	Yes	—
Native compaction	Summarise and replace	1 extra sampling step	No	Turns
Arithmetic selection	**Five-term score over a ten-field record, ∼12 flops per candidate**	**0**	**Yes**	**Nothing**

Table 3. Compiled from vendor documentation. Inference cost of one retrieval, by architecture. This table counts model invocations only; it says nothing about retrieval quality, which differs substantially across these systems and generally favours the embedding-based ones on paraphrase. See §9.

The pattern in Table 3 is uniform enough to be worth stating plainly: with the single exception of threshold-based context editing, every production memory architecture pays at least one model call per query, and prices itself per unit of stored memory.

That pricing structure has a property that deserves more attention than it gets. Under all of these systems, **remembering more costs more, permanently.** Not once, at ingest — forever, on every query, because the index you search grows and the thing you pay for is the searching.

Consider what that means for the use case everyone actually wants: an agent that accumulates knowledge of your project over years. The better it works, the more it has stored. The more it has stored, the more each query costs. Success is billed as consumption. It is difficult to think of another piece of infrastructure with that shape. A filesystem does not charge more per read as you add files. An index gets *cheaper* per query as it warms.

The reliability floor

There is a second consequence, less discussed and in my view more serious.

If selection requires a network call to an inference service, then **reads can fail**. Not degrade — fail. They can fail because the service is down, because a rate limit was hit, because a token budget was exhausted, or because a payment method expired. At least one production memory service returns an HTTP 402 on retrieval when an internal budget runs out.

An agent whose memory read can return a billing error is in a different reliability class from one whose memory read is arithmetic over local state. This is not a small engineering detail. It determines whether memory is a dependency you can build a system on or a service you have to write fallbacks around — and the fallback for "I cannot remember anything right now" is generally "behave as though the project has no history," which is precisely the failure the memory system existed to prevent.

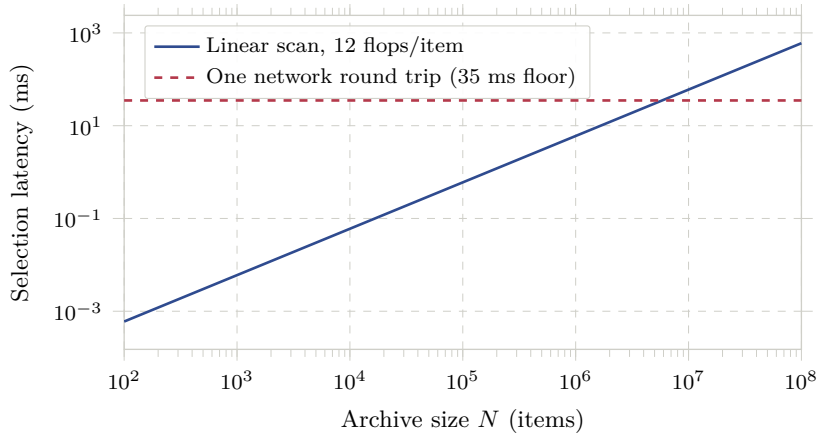

Figure 9. Derived, given stated assumptions. A linear scan at twelve floating-point operations per item on a single core, against the latency floor of a single network round trip to an embedding service. The scan is faster up to roughly **5.8 million items** — and unlike the network call, it cannot fail. Assumptions in Appendix A.

Figure 9 makes a point I did not expect when I started computing it. A naive $O(N)$ scan — no index, no clever data structure — beats a single network round trip up to nearly six million items. The sophisticated approach is slower than the dumb approach across the entire range that most projects will ever occupy, because the sophisticated approach has to leave the machine.

I want to be careful not to overclaim here. That comparison holds for one core, one query at a time, and it says nothing about quality. A vector search returns different — and on paraphrase, better — results than a lexical scan. But the latency and

reliability argument is real, and it points the same direction as the cost argument.

Cost over a realistic horizon

Put numbers on three years of ordinary use: an agent making 200 memory queries a day, over a codebase that accumulates memories as work proceeds.

	3-year cost
Embedding, queries only	\$1.75
Embedding, plus ingest at 10^6 items	\$21.75
Arithmetic selection	\$0.00

Table 4. Modelled. Direct inference spend only. Assumptions in Appendix A.

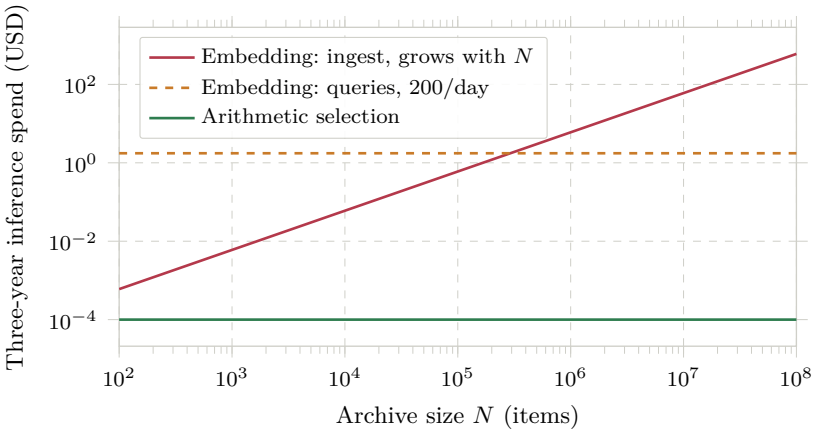

Figure 10. Modelled. Direct inference spend over three years. The absolute numbers are small; the *shapes* are the argument. Query cost is flat in N but paid forever; ingest cost rises linearly with everything you choose to remember. Arithmetic selection is flat at zero on both axes.

Table 9 is, I will freely admit, an anticlimax. The direct dollar cost of embedding-based retrieval at individual-developer scale is trivial. Two dollars. Nobody is going to change architecture over two dollars.

I include it because I think the honest version of the cost argument is *not* about dollars, and pretending otherwise would be the kind of motivated framing this essay is supposed to avoid. At the scale of one developer, the money does not matter.

The cost argument is about the other three things, and they do not appear on an invoice:

1. **The scaling shape.** Two dollars at 10^6 items is twenty at 10^7 and two hundred at 10^8, per agent, forever. Multiply by a fleet. The absolute number is small; the exponent is the problem.
2. **The dependency.** A read that traverses a network is a read that can fail, and the failure mode is silent amnesia.
3. **The opacity.** You cannot reproduce an embedding-ranked retrieval from six months ago, because you no longer have that model.

The third one is the one I would build a company on, and it needs its own section.

Reproducibility is not a nice-to-have

Here is a question that will be asked of every serious deployment within a few years, and that most current architectures cannot answer:

> *Why was this in the prompt?*

When an agent makes a decision you disagree with, the first diagnostic question is what it knew. With an embedding-based memory the honest answer is: a model that no longer exists assigned a similarity score you cannot recompute to a query you did not log, against an index that has since changed. You can observe what was retrieved, if you logged it. You cannot establish *why*, and you cannot establish that the same query today would return the same thing — because it will not, once the embedding model is updated underneath you.

Deterministic selection answers the question completely. If the score is

$$S \;=\; \omega_V V + \omega_R \hat{R} + \omega_D D + \omega_G \gamma + \omega_M M$$

with fixed weights and inputs recorded in the record itself, then the retrieval is a pure function of state and time. Same inputs, same tick, same output — and each of the five terms can be printed. Not an attention heatmap. Five numbers, their weights, and a total.

This is the argument I find most durable , because it does not depend on any trend continuing. Cost curves can bend;

embedding prices have fallen by orders of magnitude and may fall further. Latency floors can drop. But "you cannot reproduce this decision" is a structural property of putting a learned, versioned, remotely-hosted function in the retrieval path, and no amount of price reduction fixes it.

Where the arithmetic approach genuinely loses

I would not believe this essay if it did not have this section, so let me be specific and unhedged about the weakness.

A deterministic scoring function of the kind I will describe in Chapter 9 carries a lexical match term — BM25-style, over labels and paths — at a weight of 0.20. That term is the *entire* surface on which the system competes with semantic retrieval.

And on paraphrase, lexical matching loses. It loses badly and it loses in a way that is well characterised: if you stored "authentication tokens are verified server-side" and you ask "how do we check credentials on the backend," an embedding model finds it and a lexical matcher does not. Every system in Table 3 spends real money specifically to win that case, and mostly they do win it.

So the honest positioning is narrow:

> Arithmetic selection wins on cost, determinism, reproducibility, latency, and structural guarantees. It loses on paraphrase recall. If your retrieval problem is dominated by "find the thing I described in completely different words," it is the wrong tool today.

I state this early and plainly because the rest of the argument is worth nothing if I am hiding the weak spot. It also suggests the obvious synthesis, which I will return to at the end: nothing prevents a deterministic ranker from taking candidates generated by a semantic layer, provided the final ordering stays deterministic and the semantic layer is optional. The guarantees I care about are properties of the *final* ranking function, not of the candidate generator.

What I want to establish in the next chapter is narrower and, I think, more interesting than a horse race: that arithmetic selection has a structural property no similarity search can have at any price — and that this property is exactly the one that matters when archives get large.

IIIb. The Audit Problem

Of the four properties that separate arithmetic selection from learned retrieval — cost, latency, reliability and auditability — three are contingent. Prices fall, hardware improves, services get more reliable. The fourth is not contingent. A retrieval decision made by a versioned, remotely-hosted, learned function cannot be reproduced once that function changes, and it changes on someone else's schedule. This chapter argues that auditability is the property that will end up mattering, and that it is the one nobody is pricing.

> *The question is not what the agent retrieved. You logged that. The question is why that, and not the other thing — and whether asking again tomorrow gives the same answer.*
>
> ON THE DIFFERENCE BETWEEN A LOG AND AN EXPLANATION

THREE OF my four arguments have expiry dates.

I should be honest about which parts of the case in Chapter 9 are durable and which are merely true today.

The **cost** argument depends on inference prices, and inference prices have fallen by roughly three orders of magnitude in three years. At some point embedding a query will be free enough that nobody counts. The **latency** argument depends on network round trips, and edge deployment and on-device embedding models are closing that gap. The **reliability** argument depends on retrieval being a remote service, and it need not be.

Take all three away — assume free, instant, perfectly reliable embedding — and one thing survives untouched:

> You still cannot reproduce the retrieval.

Not because of cost. Because the function that produced it was learned, is versioned, and no longer exists in the form that produced your answer.

What reproducibility actually requires

A retrieval decision is reproducible if, given the same archive state and the same query, you get the same result — and can demonstrate it. That requires three things, and learned retrieval fails all three by construction.

THE SCORING FUNCTION MUST BE STABLE. An embedding model is updated: retrained, quantised, deprecated, silently swapped for a better-performing successor. Each change alters the geometry of the vector space. Two documents that were near neighbours in March are not necessarily near neighbours in September, and nothing in the system records that the relationship changed.

Deterministic scoring is stable because the function is five weighted terms that are compile-time constants. The version of the function is the version of the source file.

THE INPUTS MUST BE RECORDED. With an embedding system, the input to the scoring function is a 1,536-dimensional vector produced from your query by a model you do not control. To reproduce the decision you would need to store that vector — and even then, the comparison targets have moved.

With arithmetic scoring, the inputs are the ten fields of the record and the tick. Both are already stored, because they are the state.

THE RESULT MUST BE EXPLAINABLE IN TERMS A HUMAN CAN CHECK. This is the part most often waved away. Vector similarity gives you a cosine, which is a number that tells you the retrieval happened but nothing about why this item and not another. The honest explanation — "a transformer with several billion parameters mapped both strings into a space where they happened to be close" — is not an explanation. It is a restatement.

A weighted sum is different in kind:

$$S = 0.30\,V + 0.25\,\hat{R} + 0.15\,D + 0.10\,\gamma + 0.20\,M.$$

Five numbers, five weights, one total. Print them and the decision is fully accounted for. If a memory was retrieved because its Elo was high but its lexical match was poor, you can see exactly that, in the numbers, and decide whether you agree.

Term	Weight	Value	Contribution
Voice V	0.30	0.81	0.243
Elo $\hat{R}$	0.25	0.66	0.165
Decay D	0.15	0.42	0.063
Grade γ	0.10	1.00	0.100
Match M	0.20	0.11	0.022
Total			**0.593**

Table 5. A worked score breakdown. This memory was played because it is graded RED, currently rising, and has earned rating — despite a weak lexical match of 0.11. That last fact is visible, checkable, and arguable. Under a cosine you would see only "0.71".

Table 5 is what an explanation looks like. Note what it lets you do that a similarity score does not: you can look at it and say "the match term is doing almost nothing here, and I do not think this memory is actually relevant" — a critique you can act on by changing a weight you can see.

Why this becomes load-bearing

Three forces are converging on this, and none of them is about memory systems.

AGENTS ARE MOVING INTO CONSEQUENTIAL WORK. As soon as an agent touches money, infrastructure, health records or legal filings, the question "what did it know and why" stops being a debugging convenience and becomes the first question anyone asks after something goes wrong. The retrieval layer is squarely in scope, because what the agent knew is exactly what the retrieval layer chose.

POST-INCIDENT REVIEW IS WHERE ARCHITECTURES GO TO DIE. Consider the realistic conversation. An agent violated a constraint. You ask why it did not know. The answer is that the constraint was in the archive but did not make the cut on that turn. You ask why not. And with learned retrieval, the sequence terminates: you cannot recompute the ranking, because the model that produced it has been updated twice since.

This is not a hypothetical failure of process. It is a structural property. Figure 17 showed the constraint being crowded out at archives above a few thousand items; this chapter is about the fact that when it happens you cannot prove it happened.

REGULATION IS CONVERGING ON EXPLANATION. I will not pretend to forecast specific rules. But the direction across every

jurisdiction taking an interest in automated decision-making is toward a right to an explanation of how a decision was reached. If an agent's decision depended on what was in its context, then the selection of that context is part of the decision.

The counter-argument, and where it lands

The strongest reply is that this is over-engineering: nobody audits retrieval today, the demand is speculative, and determinism costs real retrieval quality — which, per §9, it does.

I think that is right about today and wrong about the trajectory, for a specific reason. Auditability is not a feature you can add later. It is a property of the architecture, and retrofitting it means replacing the retrieval path — which means re-deriving every ranking decision the system ever made, which is precisely the thing that cannot be done.

Systems that are auditable are auditable from the beginning or not at all. That asymmetry is why I think it is worth paying something for now, at a point where the cost is measured in paraphrase recall rather than in a rebuild.

THE SYNTHESIS, AGAIN. Nothing here argues against semantic retrieval. It argues about where in the pipeline the learned component sits. A semantic layer that *proposes* candidates, followed by a deterministic ranker that *decides* among them, gives you paraphrase recall and a reproducible final ordering — provided you record which candidates the semantic layer proposed.

That is a modest architectural constraint and it preserves the property that matters. The failure mode I am arguing against is not using embeddings. It is letting a learned function be the last word.

IV. The Arithmetic Alternative

A memory can be ranked by five terms over a ten-field record at about twelve floating-point operations, with no model call anywhere. Two exponential moving averages give a momentum signal that a weighted moving average cannot, at constant memory instead of $O(Nk)$. An Elo ladder settled from observed outcomes separates useful memories from useless ones by 195 rating points within 39 rounds. And a reserved-share budget cascade guarantees something similarity search cannot guarantee at any price: that a project's governing constraints are in the prompt at any archive size.

> *The models, they just want to learn. The memories, it turns out, just want to be ranked — and ranking is arithmetic, not inference.*
>
> ON WHAT THE SELECTION LAYER ACTUALLY NEEDS

THE DESIGN question is narrow.

We need a function that takes an archive of N items and a query, and returns the subset that fits a budget B. It must run on every turn, so it must be fast. It must be explainable, so it must be closed-form. It must not grow its per-item state with history, or it will not survive to 10^7 items.

That is a tight enough specification to nearly determine the answer.

Constant state per memory

Start with what we are allowed to store. If per-item state grows with the length of the item's history, the system dies at scale. So: a fixed record, and a small one.

Field	Type	Bytes
`id`	u64	8
`sig`	u64	8
`rank`	u8	1
`grade`	u8	1
`f`	f32	4
`s`	f32	4
`r`	f32	4
`t`	u32	4
`n`	u16	2
`w`	u16	2
		38

Table 6. The hot record. Content lives cold on disk and is read only when an item is actually played.

Ten fields, 38 bytes packed (Table 9). Identity, a similarity signature, a rank, a grade, two moving averages, a rating, a timestamp, a touch count, and a size. No embedding vector. No history buffer.

I should say clearly what 38 bytes is and is not. It is the *packed specification* budget. A straightforward implementation in a garbage-collected language, using a hash map of boxed objects, measures around 245 bytes resident per item — roughly 245 MB per million.[10] Either way the property that matters holds: the record is *constant* in the item's history.

[10] Measured on one such implementation at $N = 200{,}000$. The gap is object headers and map overhead, not the fields. A structure-of-arrays layout closes most of it. I mention the measured number because quoting the packed figure as achieved memory would be exactly the kind of specification-as-measurement error this essay complains about elsewhere.

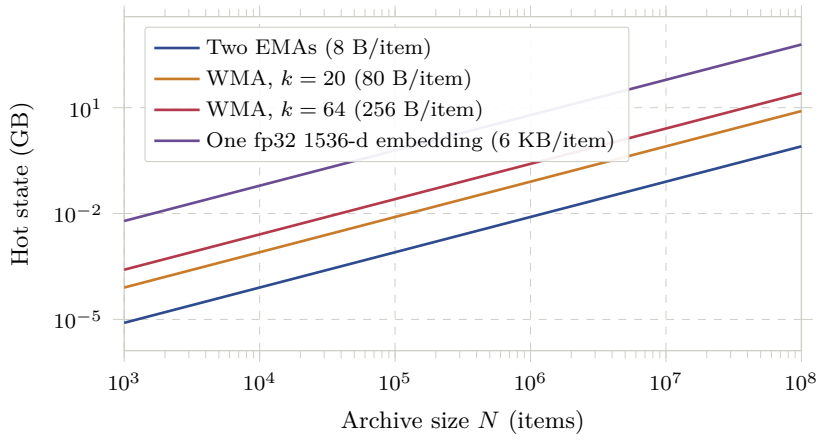

Figure 11. Derived. Resident state for the recency signal alone, by representation. At 10^7 items the EMA pair costs 80 MB; a 64-sample weighted moving average costs 2.6 GB; one embedding per item costs 61 GB.

Why two EMAs, and what the approximation costs

Recent evidence should count for more. The textbook instrument is a weighted moving average over a window of k samples,

and it is unaffordable: $O(Nk)$ memory and $O(k)$ work per update. Figure 11 shows the bill.

Two exponential moving averages, at different half-lives, give an $O(1)$-memory, $O(1)$-update filter with the same objective:

$$\lambda_F = 1 - 2^{-1/8} \approx 0.0830, \qquad \lambda_S = 1 - 2^{-1/64} \approx 0.0108,$$

$$f \leftarrow f + \lambda_F(x - f), \qquad s \leftarrow s + \lambda_S(x - s).$$

Two multiplies, two adds. Four floating-point operations, per item, per tick.

The interesting object is not either average but their difference. Define the **voice** of a memory:

$$V \;=\; \mathrm{clamp}\big(f + \Theta(f - s),\; 0,\; 1\big), \qquad \Theta = 0.5.$$

This is a MACD construction, borrowed directly from quantitative finance. A memory whose relevance is *rising* is boosted above its raw level; one that is cooling is pushed below it. It distinguishes *currently rising* from *historically loud*, which no single decay term can do.

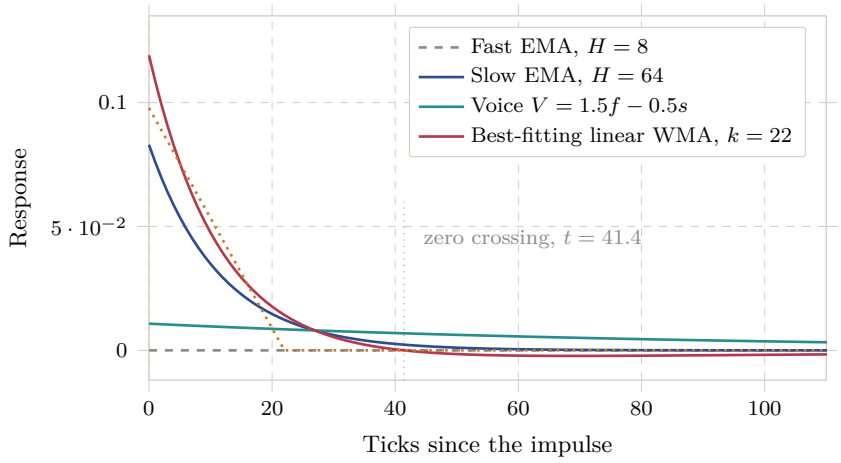

Figure 12. Derived. Impulse responses. The voice term crosses zero at $t = 41.4$ and goes negative — it actively suppresses a memory that was loud and has gone quiet. **A weighted moving average never does this**; its weights are non-negative by construction. The approximation is not a worse WMA, it is a cheaper filter that does something a WMA cannot.

Figure 12 contains the result I find most satisfying in this whole analysis. Solve for where the voice crosses zero:

$$(1 + \Theta)\lambda_F(1 - \lambda_F)^t \;=\; \Theta\lambda_S(1 - \lambda_S)^t$$

$$t^* \;=\; \frac{\ln\big(\Theta\lambda_S/(1{+}\Theta)\lambda_F\big)}{\ln\big((1 - \lambda_F)/(1 - \lambda_S)\big)} \;=\; 41.4.$$

Forty-one and a half ticks after a memory is touched, its voice goes negative and stays there. A weighted moving average,

whose weights are non-negative, cannot produce a negative tail at all. The cheap approximation has a behaviour the expensive exact method lacks, and it happens to be a behaviour you want.

So what does the approximation cost? I fitted linear-decay WMAs of every window from 5 to 60 samples against the voice impulse response, choosing the best scale for each window.

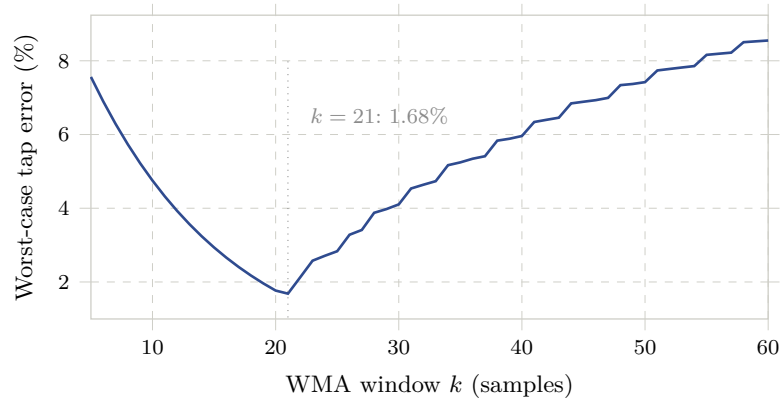

Figure 13. Derived. Worst-case error against the best-fitting linear WMA at each window length. The minimum is 1.68% at $k = 21$. Storing 21 samples per item costs ten times the memory of the EMA pair, to buy an agreement that is already within two percent.

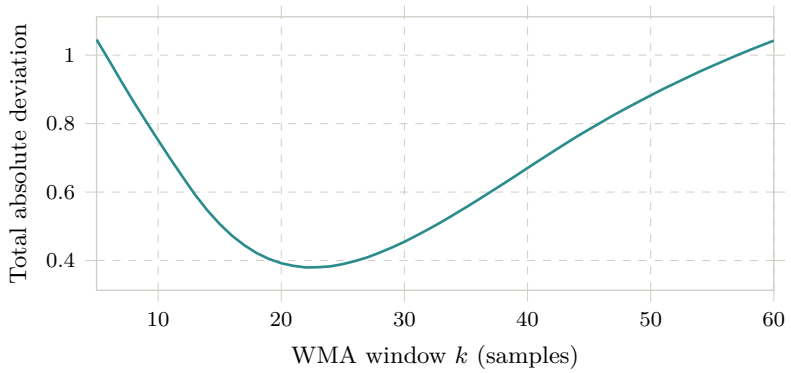

Figure 14. Derived. Total absolute deviation between the voice impulse response and the best-scaled linear WMA at each window length. The minimum sits at $k = 22$ with a deviation of 0.38. There is no window length at which a WMA reproduces the filter exactly, because no WMA has a negative tail.

The best agreement is **1.68% worst-case tap error at** $k = 21$, with a total absolute deviation of 0.38 at $k = 22$. Which is to say: to do better than two percent you would have to store twenty-one floats per memory instead of two, a factor of ten in memory, for a filter that also loses the negative tail.

That is the whole case for the approximation, and it is a quantitative one.

Recency is not correctness

Voice answers "what is fresh." It cannot answer "what is right." The most recently touched file is very often the one you are confused about.

So ask a second question, the way chess does: *when this memory was in context, did the turn go well?* Give every memory an Elo rating, start it at 1200, and settle it against observed outcomes.

$$E_i = \frac{1}{1 + 10^{(R_j - R_i)/400}}, \qquad \Delta = K(o - E_i),$$

$$R_i \leftarrow R_i + \Delta, \qquad R_j \leftarrow R_j - \Delta.$$

The mechanism that makes this work is the choice of opponent. Each memory played into context plays a virtual match against **the bench**: the highest-scoring candidate that was *not* played. The update is zero-sum. A memory that gets chosen and fails hands its points to the memory you passed over.

Nobody labels anything. The ladder is not a heuristic about what should be useful; it is a running tally of what was.

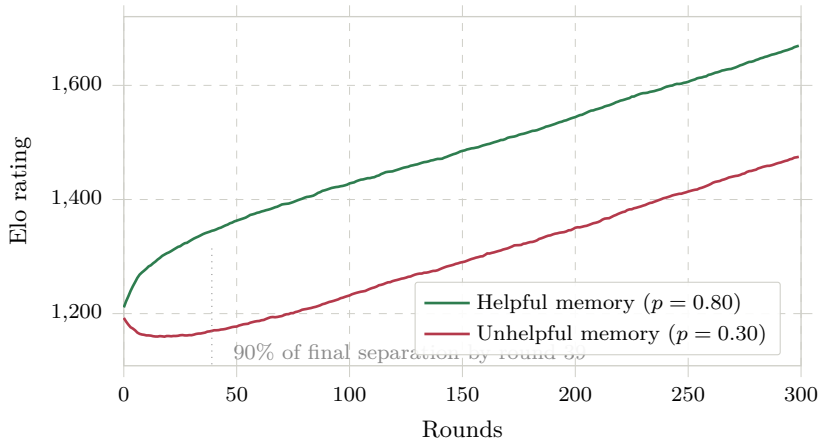

Figure 15. Simulated, 400 independent seeds, parameters in Appendix A. Two memories with different true helpfulness, settled by the zero-sum rule. Final separation is **195 rating points**; 90% of it is reached within **39 rounds**. The provisional K of 40 for the first eight touches is what makes early convergence this fast.

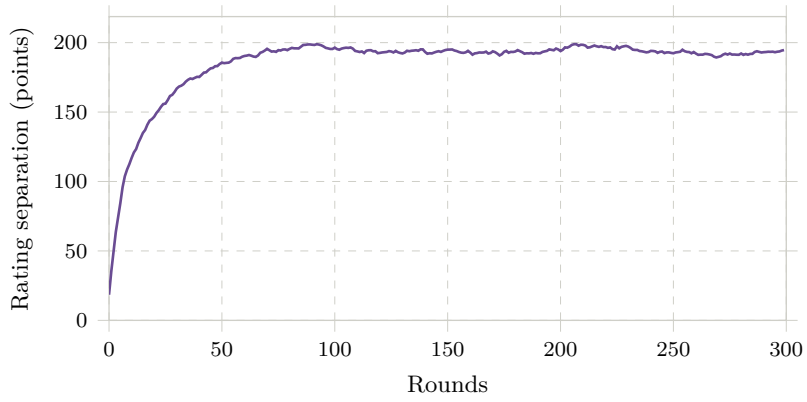

Figure 16. Simulated. Mean rating gap between the helpful and unhelpful memory across 400 seeds. The curve saturates rather than diverging, because the $[800, 2400]$ clamp and the falling K-factor both damp it — a memory cannot become permanently unbeatable on a lucky run.

Two design choices in Figure 15 are worth naming. The provisional K-factor of 40 for a memory's first eight touches — borrowed from chess rating practice — is what produces the steep early separation. And the clamp to $[800, 2400]$ prevents a memory that happened to be present during a run of good turns from becoming permanently unbeatable.

The outcome signal must not come from a model , or the whole cost argument collapses. It is read from cheap host observations:

Observed signal	o
Next action edited or cited a covered file	1.00
Turn completed, no correction	0.75
No evidence either way	0.50
Same query re-issued within two turns	0.25
User corrected the agent	0.00

Table 7. Outcome ladder. Every row requires observed evidence.

Note that a turn producing no evidence settles at 0.50, not 0.75. Silence is not success. Awarding rating for an unobserved turn credits memories that happened to be present when the agent crashed, which is precisely backwards.

The composite

Five terms, weights summing to one, all of them global compile-time constants:

$$S = \underbrace{0.30}_{\omega_V} V + \underbrace{0.25}_{\omega_R} \hat{R} + \underbrace{0.15}_{\omega_D} D + \underbrace{0.10}_{\omega_G} \gamma + \underbrace{0.20}_{\omega_M} M,$$

where $\hat{R} = (R - 800)/1600$ is the normalised rating, $D = 2^{-(t-t_{\text{node}})/128}$ a decay term, γ a grade constant, and M a BM25-style lexical match.

About twelve floating-point operations per candidate. Nothing learned per user; nothing fitted. That last property is the one that buys determinism, and determinism is what buys reproducibility and explanation.

The result that similarity search cannot match

Now the part that I think is genuinely structural rather than merely cheaper.

Suppose you retrieve by score and take the top k that fit the budget. Ask: *what is the probability that a project's governing constraint — the security invariant, the architectural decision — is among them?*

Model it honestly. The constraint is relevant but rarely the most *topically* similar item to any given query; that is the realistic failure mode. Say it sits at the 99th percentile of the score distribution. Competitors are drawn independently. Then the number of items outscoring it is Binomial$(N - 1, 0.01)$, and it survives into the prompt only if fewer than k of them do.

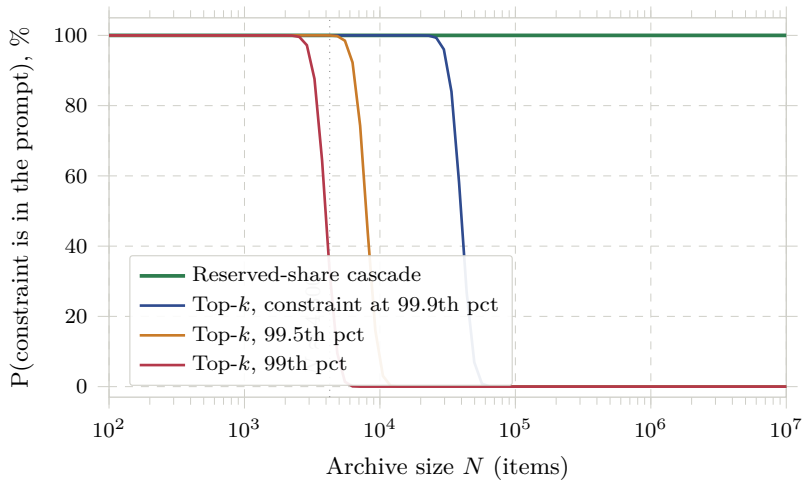

Figure 17. Modelled, $k = 40$ slots. Probability that a governing constraint reaches the prompt. Under top-k it falls off a cliff — through 50% at about **4,300 items**. Making the constraint more salient moves the cliff right, not away. The cascade holds it at 1 for every N, by construction.

Figure 17 is, to me, the most important figure in this essay.

Under top-k selection the probability does not decay gracefully. It falls off a cliff, and it falls at an archive size that is *small* — around four thousand items for a 99th-percentile constraint, which a working repository reaches in months. Making the constraint more salient buys an order of magnitude and no change in shape: the 99.9th-percentile curve has the same cliff, one decade to the right.

This is the failure mode every long-running agent eventually hits, and it is invisible because it is silent. The agent does not report that it has forgotten your architecture. It simply stops mentioning it, and then one day violates it, and the retrieval logs show a perfectly reasonable set of topically-similar chunks.

THE FIX IS NOT A BETTER RANKER. No amount of ranking quality changes the shape of that curve, because the curve is a consequence of competition for a fixed number of slots. The fix is to stop making the constraint compete.

Partition the budget by rank, and reserve shares:

Rank	Holds	Share
JOKER	Invariants	4%
ACE	Subsystem charters	10%
FACE	Consolidated concepts	26%
SET	Raw items	60%

Table 8. The cascade. Unused share flows downward only.

Each rank is filled by descending score within its own share, and unused share cascades *downward only* — never up. A JOKER never competes with a SET, so no quantity of SETs can displace it.

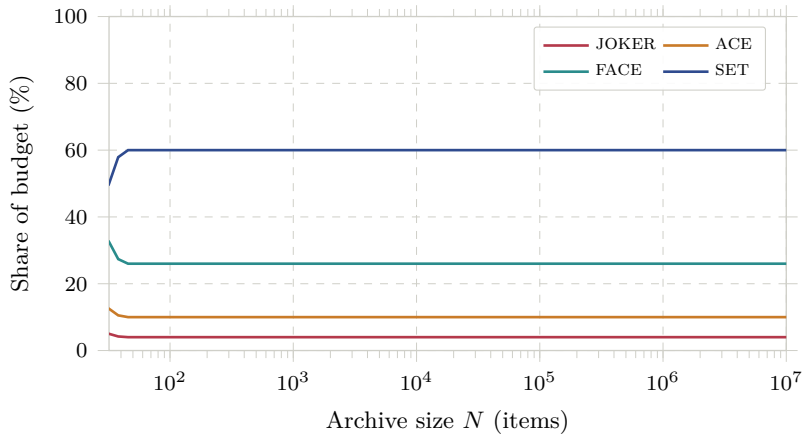

Figure 18. Derived. Budget allocation against archive size. Past the point where the archive can fill every share, the allocation is **invariant in** N: the JOKER share is 4.0% at 10^3 items and 4.0% at 10^7. Project size cannot starve the constitution.

Figure 18 shows the consequence: past the point where the archive is large enough to fill every share, allocation is flat in N forever. Four percent at a thousand items, four percent at ten million.

That is what I mean by a structural guarantee rather than a tuning win. It is not that the cascade ranks better. It is that ranking quality is no longer what determines whether your invariants survive.

Promotion, and why nothing is deleted

Two more mechanisms, briefly, because they are what make the ranks non-arbitrary.

Promotion is Elo-gated and monotone. An item rises from SET to FACE at rating 1400 with three touches; to ACE at 1700 with fan-in from five distinct items; to JOKER at 2000 spanning three subsystems. Rank never decreases. Cooling is expressed through *grade* instead — a quiet JOKER becomes a low-grade JOKER, still authoritative, just no longer amplified. A system that can silently revoke your invariants is not a system you can trust with your invariants.

Deduplication merges rather than deletes. Near-duplicates, detected by a 64-bit similarity signature within a Hamming distance of three, are superseded by a single consolidated item stamped with the count of variants it absorbed. The originals stay retrievable by lineage. An item marked *400 variants* is more informative than any of its 400 sources: it tells you this is not a passing thought, it is how the project actually works.

This matters more than it sounds. Real archives are full of the same convention restated hundreds of times, and a similarity search will happily return forty copies of one fact and spend the entire budget saying it.

What it costs, in total

Operation	**Cost**	**Model calls**	**At $N = 10^7$**
Ingest one item	O(shingles) signature	0	$\sim$10 μs
Tick update	$O(1)$, 4 flops	0	—
Score one candidate	$O(1)$, $\sim$12 flops	0	—
Retrieve top-k	$O(N)$ scan	0	60 ms
Settle a round	$O(\|P\|)$	0	$<$1 μs
Promotion check	$O(1)$ amortised	0	—

Table 9. Derived. The complete cost of the selection layer. Zero model calls appear anywhere on the ranking path. That is the single hardest invariant in the design, and everything else — reproducibility, the absence of a network dependency, cost that does not scale with what you remember — is a consequence of it.

The honest caveat is that retrieval here is a linear scan. There is no sublinear index for the query-dependent lexical term, and I will not describe one as though it exists. At ten million items a scan is about sixty milliseconds on one core — acceptable, and as Figure 9 showed, still faster than a network round trip — but it is linear, and past 10^8 it stops being acceptable. That is a real limit and it is on the list in Part V.

V. What Would Falsify This

An argument that cannot fail is not an argument. Here are six specific, dated, checkable claims. If the first two are wrong, the thesis of this essay is wrong and I would like to know by 2028. The remaining four are narrower, and each of them would force a revision rather than a retraction.

> *The trendlines are the argument. If the trendlines break, so does the argument, and the only respectable thing to do is say in advance which break would count.*
>
> ON WRITING DOWN PREDICTIONS

THE WEAKNESS of trend extrapolation is that it is always locally right and eventually wrong.

Every wall in the history of this field has been announced by someone reasoning exactly as I have reasoned here, and most of those walls were climbed. I have tried to be careful about *why* I think this one has a different shape — it is a cost function rather than a supply constraint — but I might simply be doing the same thing as everyone before me, with a different graph.

So here is the list. Each item states what I predict, by when, and what observation would count as refutation. I have tried to make them uncomfortable.

The six claims

CLAIM 1. NOMINAL CONTEXT STAYS DECELERATED. *Prediction:* through the end of 2028, the volume-priced context window of frontier models grows at less than 0.30 OOM/year — that is, the generally available window does not exceed roughly 20 million tokens.

Refuted by: a generally available, volume-priced window above 50 million tokens before 2029. Note the qualifiers, and note that they are doing real work: a research demonstration does not count, and neither does a window offered at a price that

implies nobody is meant to fill it. I am claiming something about deployed economics, not about what is technically achievable. If you think that makes the claim too easy, Claim 2 has no such escape hatch.

CLAIM 2. EFFECTIVE CONTEXT GROWS SUBLINEARLY IN NOMINAL CONTEXT. *Prediction:* on any position-controlled benchmark — not needle-in-a-haystack — the ratio of effective to nominal window continues to *fall* as nominal windows grow. A 10× larger window buys less than 10× the reliable working memory.

Refuted by: a model family demonstrating flat or rising effective ratio across at least one order of magnitude of window growth, on a position-controlled evaluation. This is the claim I hold most strongly and it is the one whose refutation would most damage the essay, because Chapter 8 does most of the work in bringing the crossing forward.

CLAIM 3. THE LINES CROSS BEFORE 2029. *Prediction:* by end of 2028, the median token demand of a frontier-agent task run to completion exceeds the effective context available to hold it, forcing explicit memory management as standard practice rather than an optimisation.

Refuted by: agent task horizons stalling — a doubling time above eighteen months sustained for two years — or by the crossing simply not arriving on schedule. This is a compound claim and therefore the weakest of the six. It inherits the softness of the tokens-per-minute assumption, which I flagged in Chapter 2 and which I would not defend to better than a factor of two.

CLAIM 4. SELECTION STAYS A MODEL CALL FOR EVERYONE ELSE. *Prediction:* through 2028, production memory systems continue to require at least one inference call per retrieval, and continue to price per unit of stored memory.

Refuted by: a major vendor shipping deterministic, zero-inference ranking as the default retrieval path. I would be pleased to be wrong here. This is the claim I most expect to lose, and losing it would mean the argument of Chapter 9 had been absorbed rather than refuted.

CLAIM 5. THE STARVATION CLIFF IS REAL AND OBSERVABLE. *Prediction:* in a deployed agent using top-k similarity retrieval over an archive exceeding roughly 10^4 items, the rate at which

the agent violates documented project invariants rises measurably with archive size, holding task difficulty fixed.

Refuted by: a controlled measurement showing invariant-violation rate flat or falling as the archive grows past 10^4 under top-k retrieval. This is the most directly testable claim in the essay and, as far as I know, nobody has run it. It is a good experiment and I would like someone to do it.

Claim 6. The linear scan does not survive. *Prediction:* arithmetic selection as described in Chapter 9 requires a sublinear index before it is viable past 10^8 items, and no such index exists today for the query-dependent lexical term.

Refuted by: nothing — this one I am asserting *against* myself. It is a limitation of the approach I have spent a chapter advocating, and I list it here so that it cannot be quietly forgotten. A linear scan at 10^8 items is roughly 600 ms on one core, which is too slow to run on every turn.

What I am not claiming

Four disclaimers, because the argument gets stronger when its boundaries are drawn tightly.

I am not claiming scaling has stopped. Compute, parameters, and capability all continue to improve at rates that would have seemed implausible five years ago. Nothing here is a stagnation argument. The claim is about which constraint binds first, not about whether progress continues.

I am not claiming long context is useless. A million-token window is extraordinarily useful, and the fact that only a fraction behaves reliably does not make it worthless — it makes it a resource to be managed rather than a solution to be assumed.

I am not claiming arithmetic selection beats semantic retrieval on quality. It does not, on paraphrase, and I said so at length in §9. The claim is about cost, determinism, reproducibility, and one structural guarantee.

I am not claiming novelty for the components. Exponential moving averages are old. Elo is from 1960. BM25 is from the 1990s. Reserved-share allocation is standard practice in operating-system scheduling. The only thing that might be new is the combination, and its application to this problem.

The synthesis I actually expect

If I am being honest about where I think this lands, it is not a victory for either camp.

The guarantees I care about — reproducibility, explanation, the reserved share for constitutional items — are properties of the **final ranking function**. They are not properties of the candidate generator. Nothing in the design of Chapter 9 prevents a semantic layer from *proposing* candidates, provided the final ordering is deterministic and the semantic layer is optional and recorded.

That architecture gets you paraphrase recall when you can afford it, and degrades to something that still works, still costs nothing, and still cannot starve your invariants when you cannot. The expensive part becomes an enhancement rather than a dependency.

I think that is where this ends up, and I think the reason it has not happened yet is that the field has been treating selection as a retrieval-quality problem when it is mostly a resource-allocation problem with a retrieval component.

Parting thought

The uncomfortable possibility is not that any particular number here is wrong. It is that the whole framing is one abstraction level too low — that in three years nobody will care about context windows because the architecture will have changed underneath the question, the way nobody now argues about recurrent versus convolutional sequence models.

That may well happen. But notice that even the architectures most likely to cause it — state space models, linear attention, the various hybrids — work by compressing an unbounded history into a fixed-size state. Which is selection. They do not remove the problem; they move it inside the weights, where it becomes cheaper, faster, and completely unauditable.

If that is the future, then the question this essay asks becomes more urgent rather than less: when something has to be forgotten, who decided, and can you find out?

Right now, for almost every system in production, the answer is a model you cannot query, using a score you cannot reproduce, against an index that has since changed.

We can do better than that, and the arithmetic is not hard. That is the whole of what I wanted to say.

Methods, Models, and Derived Constants

Every number in this essay comes from one of three places: it was published by someone else, it was derived by arithmetic from stated parameters, or it came out of a model I wrote. This appendix separates the three and gives the models in full, so that disagreement can be about assumptions rather than about provenance.

Classification of every headline number

Quantity	Value	Status	Source or derivation
Context growth, 2019–2024	0.51 OOM/yr	Measured, fitted	Least squares on $\log_{10}$ of published windows
Context growth, post-2024	0.12 OOM/yr	Measured, fitted	Same, restricted to 2024.15 onward
Task-horizon doubling	7 months	Measured	METR (2025), used unadjusted
Task-horizon rate	0.52 OOM/yr	Derived	$\log_{10} 2 \times 12/7$
Attention/linear crossover	53,406 tokens	Derived	$P/(2Ld)$, P=7×10^{10}, L=80, d=8192
KV cache at 1M, 64 kv heads	2,621 GB	Derived	$2Ld_hnb$, fp16
KV cache at 1M, 8 kv heads	328 GB	Derived	As above, GQA-8
Effective ratio at 1M	17%	Modelled	§A
Crossing year	2027.6	Modelled	§A
Voice zero crossing	$t = 41.4$	Derived	Closed form, §A
Best WMA agreement	1.68% at k=21	Derived	Grid fit, §A
Elo separation	195 points	Simulated	§A, 400 seeds
Rounds to 90% separation	39	Simulated	As above
Starvation half-point	$N \approx 4{,}300$	Modelled	§A
Scan beats network until	$N \approx 5.8 \times 10^6$	Derived	§A
JOKER share at 10^7	4.0%	Derived	Cascade definition, §A

Table 10. Provenance of every headline figure. "Measured" means somebody else published it. "Derived" means arithmetic from stated inputs — given the inputs, no other answer is possible. "Modelled" means I chose a functional form, and you may reasonably choose a different one.

The dual-EMA filter

Status: derived. With half-lives $H_F = 8$ and $H_S = 64$ ticks,

$$\lambda_F = 1 - 2^{-1/H_F} = 0.0829959\ldots,$$

$$\lambda_S = 1 - 2^{-1/H_S} = 0.0107719\ldots$$

The impulse responses are $f_t = \lambda_F(1-\lambda_F)^t$ and $s_t = \lambda_S(1-\lambda_S)^t$. With $\Theta = 0.5$ the voice is $V_t = 1.5f_t - 0.5s_t$, which vanishes at

$$t^* = \frac{\ln(\Theta\lambda_S/((1+\Theta)\lambda_F))}{\ln((1-\lambda_F)/(1-\lambda_S))} = 41.4237\ldots$$

The minimum of V is -0.00224 at $t = 69$.

WMA comparison. For each window $k \in [5, 60]$ I take the linear-decay kernel $w_i \propto (k - i)$ for $i \in [0, k)$, normalised, and search a 961-point grid of scale factors $\alpha \in [0.2, 5.0]$ for the α minimising $\sum_t |V_t - \alpha w_t|$ over 120 ticks. Reported: best total absolute deviation 0.380 at $k = 22$; best worst-case tap error 1.68% at $k = 21$. Different fitting procedures give somewhat different numbers; the conclusion — agreement within about two percent for a tenfold memory saving — is not sensitive to the choice.

Effective context

Status: modelled. This is the softest model in the essay and the one most worth arguing with.

Position curves in Figure 7 use

$$\mathrm{acc}(p) = 0.97 - \delta\sin(\pi p)^{1.4}, \qquad p \in [0, 1],$$

with trough depth $\delta \in \{0.10, 0.26, 0.42, 0.60\}$ for windows of $\{8\mathrm{k}, 32\mathrm{k}, 128\mathrm{k}, 1\mathrm{M}\}$. The functional form is chosen to be smooth, symmetric, and to reproduce the qualitative U-shape reported in the position-bias literature. The depths are my calibration.

The effective ratio in Figure 8 uses

$$\rho(n) = \frac{1}{1 + (n/2.2 \times 10^4)^{0.42}}.$$

Both the scale 2.2×10^4 and the exponent 0.42 are fitted to the trough depths above under a 10%-of-head-accuracy threshold.

What is robust and what is not. The *existence* and *sign* of position bias are measured and replicated. The *sublinearity* of effective in nominal follows from the trough deepening with window size, which is also reported. The specific values in Table 9 are not robust and should be read as illustrative. The claim in Part V is deliberately stated as a claim about the *sign of the trend*, not about any value.

The crossing

Status: modelled from measured rates. Task horizon anchored at 60 minutes in early 2025, doubling every 7 months. Token demand at 3,000 tokens per minute of task time. Nominal context anchored at 2×10^6 in mid 2026, growing at the realised 0.12 OOM/year.

Tokens/min	Crossing
1,500	2028.7
3,000	2027.6
6,000	2026.5

Table 11. Sensitivity of the crossing to the softest assumption.

The tokens-per-minute constant is the weak point. Halving or doubling it moves the crossing by roughly thirteen months in either direction. It does not change the sign of the divergence, because divergence is driven by the difference in slopes (0.52 versus 0.12 OOM/year), and no plausible choice of intercept makes two lines of different slope fail to cross.

Elo simulation

Status: simulated. $R_0 = 1200$; $K = 40$ for the first 8 touches, 24 thereafter; ratings clamped to $[800, 2400]$; logistic scale 400. Two memories with true per-round success probabilities 0.80 and 0.30, each playing against the other as bench. 400 independent seeds, 300 rounds, seeded deterministically. Reported values are means across seeds.

This is a deliberately simple two-agent setup. It demonstrates that the update rule separates memories of different true quality and does so quickly; it does *not* model a full deck, where credit assignment is slower because each round has a single bench op-

ponent. That is a known limitation of the design and is flagged as a tuning risk rather than a result.

The starvation model

Status: modelled. A governing constraint sits at quantile q of the score distribution; other items score independently. The number outscoring it is $X \sim \text{Binomial}(N-1, 1-q)$, and it enters the prompt iff $X < k$ for $k = 40$ slots. Reported curves are $P(X < k)$ for $q \in \{0.99, 0.995, 0.999\}$, computed exactly for $Np < 400$ and by normal approximation with continuity correction above.

The load-bearing assumption is independence, and it is not exactly true — real score distributions are correlated, and a well-tuned ranker will correlate the constraint's score with query relevance. That correlation moves the cliff to the right. It does not remove it, because the cliff is a consequence of finitely many slots and unboundedly many competitors, and no correlation structure changes that.

The cascade curve is not a model. It is 1 by construction: reserving 4% of the budget for a rank capped at 7 items guarantees those items a seat whenever they exist.

The cascade

Status: derived. Budget $B = 8{,}000$ tokens, mean item 200 tokens, so 40 slots. Shares: JOKER 4%, ACE 10%, FACE 26%, SET 60%. Caps: 7 and 24. Each rank is filled by descending score within its share; unused share cascades downward only. Once the archive can fill every share, allocation is exactly the share vector, independent of N — which is the invariance plotted in Figure 18.

Cost and latency

Status: derived, given stated assumptions. Scan: 12 flops per item at 2×10^9 effective scalar flops per second on one core. Network floor: 35 ms for one embedding round trip. Crossover at $N = 0.035 \times 2 \times 10^9 / 12 = 5.83 \times 10^6$.

Three-year cost: 200 queries/day for 1,095 days; embedding priced at $0.02 per million tokens on a 400-token query, giving $1.75. Ingest at 10^6 items and 300 tokens each adds $20.

These figures move with hardware and with vendor pricing, both of which change quickly. The scaling shapes — constant versus linear in corpus size — do not.

Reproduction

Every figure in this document is generated by a single script from the constants and models above. The pipeline has three stages and no manual steps:

1. The script computes all series and writes each as a file of coordinate pairs.
2. The document reads those files directly at typeset time. No figure is drawn by hand, and no coordinate is transcribed.
3. Charts are rendered as vector graphics in the same font as the body text.

This means there is no path by which a number in the prose can drift from the number in a chart: both trace to the same computation. Where I quote a value in the text — 0.51 OOM/year, 53,406 tokens, 41.4 ticks, 195 rating points, 4,300 items — it is the value the script printed.

A NOTE ON WHAT THIS DOES AND DOES NOT BUY. Reproducibility of the figures is not the same as correctness of the models. A wrong model reproduced perfectly is still wrong. What it buys is that the argument is *auditable*: a reader who disagrees can locate the exact assumption they object to, change it, and see what happens to the conclusion — rather than having to guess where a number came from.

That is the standard I would want applied to a document making claims like these, so it is the standard I have tried to meet.

THE MEMORY WALL
Raj Jobalia
August 2026

The Complete Constant Table

Every tuning value referenced anywhere in this essay, in one place. All of them are compile-time constants, identical for every user and every archive. Nothing in this table is learned, fitted per-deployment, or adjustable at runtime — which is the property that makes the ranking deterministic, and therefore reproducible, and therefore auditable.

Recency

Symbol	Meaning	Value
H_F	Fast EMA half-life (ticks)	8
H_S	Slow EMA half-life (ticks)	64
λ_F	Fast smoothing factor	0.082996
λ_S	Slow smoothing factor	0.010772
Θ	Momentum coefficient	0.50
t^*	Voice zero crossing (ticks)	41.42

Table 12. Recency constants. $\lambda = 1 - 2^{-1/H}$ in both cases; the zero crossing is derived, not chosen.

Authority

Symbol	Meaning	Value
R_0	Initial rating	1200
K	K-factor, established	24
K_{prov}	K-factor, provisional	40
n_{prov}	Touches before established	8
$R_{\min}$	Rating floor	800
$R_{\max}$	Rating ceiling	2400
—	Logistic scale	400

Table 13. Elo constants, following chess rating convention. The provisional K-factor is what produces the fast early separation in Figure 15.

Composite score

Symbol	Term	Weight
ω_V	Voice	0.30
ω_R	Normalised Elo	0.25
ω_D	Decay	0.15
ω_G	Grade	0.10
ω_M	Lexical match	0.20
	Sum	**1.00**
H_D	Decay half-life (ticks)	128

Table 14. The five score weights. They sum to one by construction, so the composite is a convex combination and $S \in [0, 1]$.

Budget cascade and ranks

Rank	Share	Cap	Promotion gate
JOKER	4%	7	$R \geq 2000$, ≥ 3 subsystems
ACE	10%	24	$R \geq 1700$, fan-in ≥ 5
FACE	26%	—	$R \geq 1400$, $n \geq 3$
SET	60%	—	(ingest)

Table 15. Rank shares, caps, and the Elo-gated promotion thresholds. Unused share cascades downward only. Rank is monotonically non-decreasing.

Grades and deduplication

Grade	Holds	γ
RED	Invariants, security rules	1.0
YELLOW	Active work	0.7
GREY	Settled reference	0.4
BLUE	Superseded, historical	0.1
Demotion gate		$R < 1000$, age > 512
SimHash width		64 bits
Near-duplicate threshold		Hamming ≤ 3
Shingle width		4 tokens

Table 16. Grade constants and the dedup parameters. Cooling is expressed through grade, never through rank, so nothing is demoted out of authority.

Complexity

Operation	Cost	Model calls
Ingest one item	O(shingles)	0
Tick update	$O(1)$, 4 flops	0
Score one candidate	$O(1)$, $\sim$12 flops	0
Retrieve top-k	$O(N)$ scan	0
Settle a round	$O(\|P\|)$	0
Promotion check	$O(1)$ amortised	0

Table 17. The complexity budget. The right-hand column is the single hardest invariant in the design: zero model calls anywhere on the ranking path.

Sources and Prior Work

Every external claim in this essay traces to one of the sources below. Where I have used someone's headline number without adjustment, I say so. Where I have modelled something myself, it is in Appendix A and is not attributed to anyone else.

Architecture and cost

Attention and its quadratic term. Vaswani et al., *Attention Is All You Need* (2017). The $O(n^2)$ scaling of exact attention in sequence length is a property of forming the $n \times n$ score matrix.

KV cache reduction. Shazeer, *Fast Transformer Decoding: One Write-Head is All You Need* (2019), introducing multi-query attention; Ainslie et al., *GQA: Training Generalized Multi-Query Transformer Models* (2023), generalising it to grouped-query attention. Both are constant-factor reductions in the number of key/value heads.

Compute-optimal scaling. Hoffmann et al., *Training Compute-Optimal Large Language Models* (2022) — the "Chinchilla" result. Referenced here only for the general shape of scaling economics.

Long-context behaviour

Position bias. N. F. Liu et al., *Lost in the Middle: How Language Models Use Long Contexts* (2023). The U-shaped dependence of retrieval accuracy on the position of the required information is the single most load-bearing external result in Chapter 8.

EFFECTIVE VERSUS ADVERTISED LENGTH. Hsieh et al., *RULER: What's the Real Context Size of Your Long-Context Language Models?* (2024), which finds measured effective lengths routinely far below advertised ones.

NEEDLE-IN-A-HAYSTACK. Widely used, informally specified. I cite it in Chapter 8 as an example of a benchmark that saturates across three orders of magnitude of the parameter it purports to measure.

Agent capability trends

TASK HORIZONS. METR, *Measuring AI Ability to Complete Long Tasks* (2025). I use the headline doubling time of approximately seven months for the 50%-success time horizon without adjustment, and state the sensitivity of my conclusion to it in Appendix A.

Memory systems surveyed

The inference-cost accounting in Table 3 was compiled from each vendor's own public documentation of its retrieval path. I have not benchmarked any of these systems and make no claim about their retrieval quality. Where I quote pricing, it is list pricing as published, and pricing changes frequently.

VECTOR-STORE MEMORY. Extraction pipeline in front of a vector store, with an optional entity graph. Write path involves an embedding call and, by default, a model call to reconcile facts; read path involves an embedding call and optional reranking.

TEMPORAL KNOWLEDGE GRAPH. Episodes are decomposed by a model into entities and fact-bearing edges with validity intervals. Retrieval fuses vector similarity, full-text search and graph traversal.

AGENT-READS-FILESYSTEM. Memory as a version-controlled file tree that the agent navigates itself. No dedicated retrieval inference, but one or more agent turns per lookup, which is a larger cost in both latency and tokens.

ROUTER PLUS GRAPH. A rule-based router — genuinely zero-inference — in front of graph or vector retrieval, with a generation step on the default path.

NATIVE CONTEXT MANAGEMENT. Threshold-based context editing is deterministic and costs nothing; compaction summarises and therefore adds a sampling step; model-driven memory tools cost agent turns.

Components used in the alternative

None of the mechanisms in Chapter 9 is new. Exponential moving averages and the MACD construction come from time-series analysis and quantitative finance. The Elo rating system dates from 1960 and its provisional K-factor convention from competitive chess administration. BM25 dates from the 1990s. Reserved-share allocation with downward-only cascade is standard operating-system scheduling practice. SimHash is from Charikar (2002).

The only thing that may be new is the combination, and its application to the problem of deciding what an agent should remember.