StaleBench: A Benchmark for Answer Freshness in Retrieval-Augmented Generation

Karan Singh
karansingh25822@gmail.com
ORCID: 0009-0000-0920-2379

June 2026

Abstract

A retrieval-augmented generation (RAG) system can store a new fact in its index yet still return the old answer. Index-level freshness checks miss this, because they ask whether the new document is stored, not whether the answer changed. We present StaleBench, an open-source benchmark that measures answer freshness: after a fact changes, how long a system takes to return the new value (catch-up latency) and how often it still returns the old one (recovery rate). Answers are scored by exact match against a controlled ground truth, with no language model as a judge. Across ten open models from three families (2024 to 2026) and sparse, dense, and reranked retrieval, about half of all answers remain stale even under immediate re-indexing, and the cause is document position rather than retrieval. Placing the newest document last fully resolves this for the most capable models but backfires on others, so the fix must be verified per system. StaleBench applies to any RAG system whose documents can be changed and refreshed.

1 Introduction

Retrieval-augmented generation (RAG) joins a retriever and a language model [1]. The retriever finds documents for a query, and the model reads them and writes an answer. RAG is often used because it can stay current. When a fact changes in the world, a team can add a new document, and the system is expected to give the new answer.

In practice this does not always happen. A system can add the new document to its index in less than a second and still answer with the old value. The reason is that adding a document and using it are not the same step. The index can be correct while the answer is wrong.

Most freshness tools cannot see this, because they check the index, not the answer. By an index measure the system looks fresh, but the user reads the answer, and if the answer is old the system is not fresh in any way that matters. This paper measures freshness at the answer. We make four contributions.

- **A metric.** We define *catch-up latency*, the number of steps from a fact changing until the answer becomes correct and stays correct, and the related *recovery rate*.
- **A benchmark and tool.** We build StaleBench, which controls the documents, the clock, and the true answers, scores by exact match, and runs against any RAG system through a small two-method interface.

- **A finding.** Across ten models from three families and across sparse, dense, and reranked retrieval, about half of answers stay stale even under immediate refresh. The cause is on the model side, not the retrieval side: it is the position of the documents in the context.
- **A fix, and its limit.** Placing the newest document last helps most models and fully removes the staleness for the most capable ones. But it backfires on some models that anchor on the first document instead. Because the fix is model-dependent, a benchmark is needed to know whether it helps a given system. This is what StaleBench provides.

2 Problem definition

Setup. Time moves in discrete steps $t = 0, 1, 2, \ldots$. The world holds a set of facts. Each fact f has a question q_f and a value. At a known step, called the change tick c_f, the value of fact f changes from an old value to a new value. We use an append model: the old document stays in the corpus and a new document is added. So after the change the corpus holds both the old and the new document for that fact. The system must pick the new one.

Refresh policies. A RAG system does not always rebuild its index at every step. We model three common policies:

- **never**: the index is built once at the start and never updated.
- **batch**: the index is rebuilt on a fixed period (for example every few steps).
- **immediate**: the index is rebuilt at the same step the fact changes.

Catch-up latency. After the change tick, the question q_f is asked at each step. The answer is correct at step t if it contains the new value and not the old one. To avoid counting a single lucky step, we use an anti-flicker window w: the answer must be correct for w steps in a row before the fact counts as recovered. The catch-up latency L_f is the number of steps from the change tick until the first such stable-correct step. A fact that does not recover within a horizon of H steps is marked as not recovered.

Recovery rate. Over a set of facts, the recovery rate is the share that recover inside the horizon. A higher rate means a fresher system. Because each fact is a yes-or-no event, the rate is reported with a Wilson 95% confidence interval [2], which is stable for rates near 0 and 1.

The idea of measuring staleness against a clock is common in distributed systems, where reads can return old data for a bounded time [3]. StaleBench brings the same view to RAG, but at the level of the model's answer.

3 Related work

Outdated information in RAG. HoH [4] is a static question-answer dataset that measures how outdated documents lower answer quality when old and new content are both present. It shows the problem is real, but it does not use a clock, refresh policies, or a time-to-recover measure. StaleBench adds the time view and the policies, and it is a tool the reader can run on their own system.

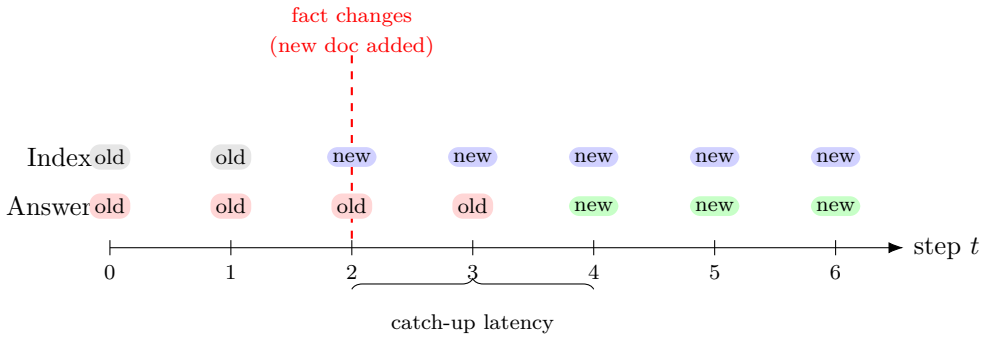

Figure 1: The freshness gap. The index holds the new document right after the fact changes, so an index-level check reports the system as fresh. The answer keeps the old value for several more steps. StaleBench measures this gap as catch-up latency.

Freshness of the benchmark itself. DRAGOn [5] keeps a RAG benchmark fresh by making new versions of the test data, so models cannot have seen them in training. It scores answers with a language model judge. Its goal is to keep the test fair over time, not to measure how long a system stays stale. StaleBench has the opposite focus: a fixed, controlled test that measures the system's own delay, with exact-match scoring.

Knowledge conflict and position. Earlier work shows that models do not always follow the retrieved text, and that the place of a document in the context changes how much the model uses it. Models can keep a memorized answer instead of the retrieved one [6, 7], and they use the middle of a long context less than the start and end [8]. We build on this. Our contribution is to connect this position effect to freshness over time, to measure it with a clear metric, and to give a simple fix.

In short, prior work shows that outdated documents hurt quality and that position matters. To our knowledge, no prior open-source tool measures answer-level catch-up latency across refresh policies and ships as a black-box benchmark for any RAG system. This is the gap StaleBench fills.

4 The StaleBench benchmark

Controlled world. StaleBench builds the corpus, so it knows the true answer at every step. Each fact has one short value, such as a name or a place. Because the value is known and exact, an answer is scored by exact match. The scorer matches whole words, so a value like "Park" is not matched inside "Parkinson". This means StaleBench does not need a second language model to judge answers, which removes a common source of noise and cost.

Black-box interface. A system under test only needs two methods. The `index` method takes a set of documents and builds or refreshes the retrieval index. The `answer` method takes a query and returns a string. Any system that can do these two things can be measured, no matter what retriever or model is inside. This makes StaleBench work with closed and open systems alike, as long as their documents can be changed.

Stability. Language model output changes from run to run. StaleBench runs many trials and reports the mean rate with a Wilson confidence interval, so the numbers stay stable. The benchmark also ships a validation test: a system that is always fresh must score higher than a system that is

Table 1: Models tested, with sizes and release dates. They span three families (Qwen, Llama, Gemma) and two years of open model releases, from 2024 to 2026.

Family	Sizes tested	Vendor	Released
Qwen2.5	1.5B, 3B, 7B	Alibaba	September 2024
Llama-3.1	8B	Meta	July 2024
Llama-3.2	1B, 3B	Meta	September 2024
Gemma-2	2B, 9B	Google	June 2024
Gemma-4	E2B, E4B	Google	April 2026

always stale. If the metric cannot tell these two apart, the metric is broken. This test passes in the released version.

5 Experimental setup

Models. We test ten open instruction-tuned (chat) models from three families and a range of sizes: Qwen2.5 at 1.5B, 3B, and 7B [9]; Llama-3.2-1B, Llama-3.2-3B, and Llama-3.1-8B [10]; and Gemma-2 at 2B and 9B plus two 2026 Gemma-4 models [11]. All run text-only, with reasoning turned off, through a local endpoint that follows the OpenAI chat format. We excluded one 2026 reasoning model (Qwen3.5) because its reasoning could not be disabled in the serving stack and it did not produce a usable short answer within a practical token budget. Table 1 lists the models with their release dates; together they span two years of open releases, from mid-2024 to 2026.

Retrievers. We cover the common retrieval choices. The sparse retriever uses TF-IDF or BM25. The dense retriever turns each document into a vector with an embedding model and ranks by cosine similarity; we try four embedders of different sizes (all-MiniLM-L6, bge-small, bge-large [12], and nomic-embed). An optional cross-encoder reranker, which re-scores the retrieved documents, is also tested. Retrieval returns the top $k = 3$ documents by default.

Scenario and counts. Each run uses 24 facts with single-value questions, the append corpus model of Section 2, and anti-flicker window $w = 2$. Answers are decoded at temperature 0. The headline numbers use 6 trials per cell ($n = 144$ fact-level events); the broad component sweep uses 3 trials per cell ($n = 72$). At these sizes a 95% confidence interval is roughly ± 0.08 ($n = 144$) or ± 0.12 ($n = 72$), so differences smaller than that are noise.

Competence control. A weak model could score low simply because it cannot follow the task, which would look like staleness. To rule this out, we add a control: present *only* the new document (no old or new conflict) and check whether the model answers correctly. If a model passes this control but still fails the full benchmark, its low score is genuine staleness, not task failure.

6 Results

6.1 About half of all answers stay stale, even under immediate refresh

Table 2 shows the recovery rate for all ten models under the three refresh policies, using the TF-IDF retriever. The pattern is the same for every model. The `never` policy recovers almost nothing: this is the sanity check, since without refresh the new value never enters the index. The `immediate`

Table 2: Recovery rate by model and refresh policy (TF-IDF retriever, $n = 144$). Higher is fresher. The 95% confidence interval is shown for the immediate policy; never and batch are point estimates.

Model	never	batch	immediate [95% CI]
Qwen2.5-1.5B	0.00	0.50	0.50 [0.42, 0.58]
Qwen2.5-3B	0.00	0.67	0.62 [0.54, 0.70]
Qwen2.5-7B	0.00	0.58	0.51 [0.43, 0.59]
Llama-3.2-1B	0.00	0.33	0.33 [0.26, 0.41]
Llama-3.2-3B	0.00	0.54	0.50 [0.42, 0.58]
Llama-3.1-8B	0.00	0.58	0.54 [0.46, 0.62]
Gemma-2-2B	0.00	0.50	0.50 [0.42, 0.58]
Gemma-2-9B	0.00	0.50	0.46 [0.38, 0.54]
Gemma-4-E2B	0.00	0.58	0.54 [0.46, 0.62]
Gemma-4-E4B	0.00	0.50	0.50 [0.42, 0.58]

policy is best and `batch` is close behind. But even under `immediate` refresh the recovery rate is only about 0.50 (range 0.33 to 0.62). In plain words: about half of all answers still give the old value right after a fact changes, even though the new document is already indexed. A fresh index does not give a fresh answer. This holds across three families, sizes from 1.5B to 9B, and two model generations.

Recovery rate asks *whether* the answer catches up. Catch-up latency asks *how fast*. These are different, and the refresh policy controls the speed. Figure 2 shows that under immediate refresh a recovered answer catches up in well under one step on average, while under batch it takes one to two steps, and under never it never does. So the policy governs the latency, while whether the answer ever catches up at all is governed by the model.

6.2 The problem is in the model, not the retriever

A natural worry is that the retriever simply fails to fetch the new document. It does not. Table 3 compares three retrievers (TF-IDF, BM25, dense). Recovery moves only within noise across them, for every model. The choice of embedder makes no real difference either: on the three mid-size models, four embedders (all-MiniLM, bge-small, bge-large, nomic) all land between 0.46 and 0.62. Adding a reranker does not help (it is slightly worse, around 0.42, because the still-relevant old document is also ranked high).

So the new document is retrieved; the model simply does not use it. The staleness is on the model side: it is about how the model reads the documents, not which documents it is given. This matches work showing that models do not always follow the retrieved text and that a document's position in the context changes how much it is used [8, 7].

The last column of Table 3 is the competence control (Section 5): the share correct when only the new document is present. Nine of ten models score 1.00, so their staleness under conflict is genuine, not an inability to do the task. The exception is Llama-3.2-1B at 0.62; its low recovery is partly weakness, so its numbers should be read with care.

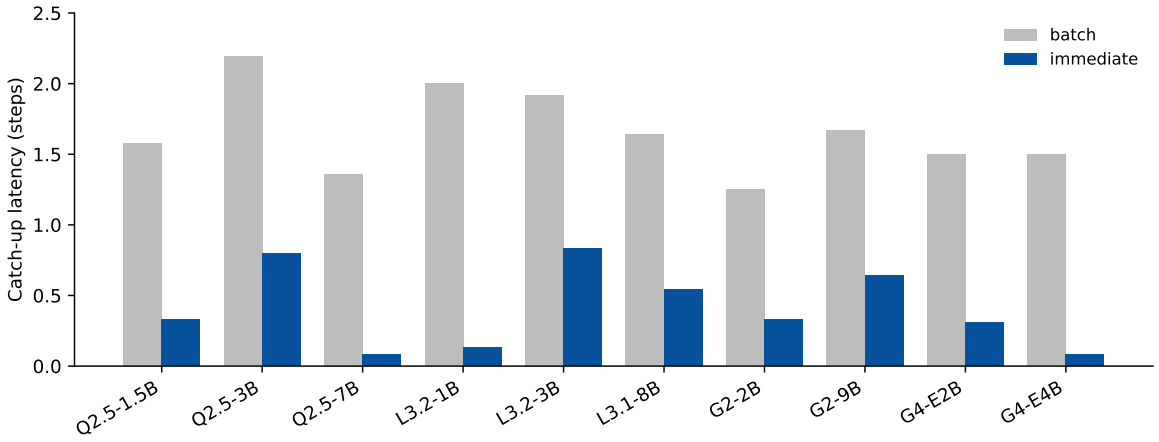

Figure 2: Catch-up latency (mean steps among recovered facts, TF-IDF, $n = 144$). The refresh policy sets the speed: immediate is well under one step, batch is one to two steps, and never is infinite (not shown). This is separate from the recovery rate, which sets whether the answer catches up at all.

6.3 A fix that depends on the model: put the newest document last

If the model is sensitive to document order, then changing the order should change the answer. We sort the retrieved documents so the newest one comes last. Figure 3 and Table 4 show the effect, and it is not the same for every model:

- For the most capable models the fix is dramatic: Llama-3.1-8B goes from 0.54 to 1.00 (perfect), and Gemma-2-9B from 0.46 to 0.92.
- For most other models it helps clearly, for example Gemma-4-E2B from 0.54 to 0.79 and Qwen2.5-3B from 0.62 to 0.75.
- But for some models it backfires: Qwen2.5-1.5B collapses from 0.50 to 0.04, and Gemma-4-E4B drops from 0.50 to 0.33.

The fix is therefore powerful but model-dependent: it can fully solve the problem or make it much worse, and which way it goes is not predictable from model size or family alone. This is the practical reason a benchmark is needed: a team cannot assume the fix helps, it has to measure it on its own model.

6.4 Why the fix backfires: some models read the first document

To see what happens when the fix backfires, we examine the actual answers of the smallest model, Qwen2.5-1.5B, on six changed facts (Figure 4). With the natural retriever order it gets four of six right. With the newest document placed last, it answers with the old value on all six: every fact it had gotten right now flips to the old value.

This is a primacy bias: the model anchors on the first document in the context, so putting the new document last makes it read the old document first and stay with it. The capable models behave the opposite way: they weight the last document, so newest-last makes them pick the new value. This

Table 3: Immediate recovery by retriever (naive order, $n = 72$) and the competence control (share correct when only the new document is present). The retriever does not change the result; staleness is genuine wherever competence is 1.00.

Model	TF-IDF	BM25	dense	competence
Qwen2.5-1.5B	0.50	0.62	0.46	1.00
Qwen2.5-3B	0.62	0.62	0.42	1.00
Qwen2.5-7B	0.51	0.42	0.50	1.00
Llama-3.2-1B	0.33	0.46	0.33	0.62
Llama-3.2-3B	0.54	0.50	0.46	1.00
Llama-3.1-8B	0.54	0.46	0.58	1.00
Gemma-2-2B	0.50	0.62	0.54	1.00
Gemma-2-9B	0.46	0.46	0.62	1.00
Gemma-4-E2B	0.54	0.62	0.42	1.00
Gemma-4-E4B	0.50	0.54	0.33	1.00

Table 4: Immediate recovery with naive order vs recency order (newest document last), TF-IDF, $n = 144$. "pp" is percentage points.

Model	naive	recency	change
Qwen2.5-1.5B	0.50	0.04	−46 pp (backfire)
Qwen2.5-3B	0.62	0.75	+13 pp
Qwen2.5-7B	0.51	0.54	+3 pp
Llama-3.2-1B	0.33	0.54	+21 pp
Llama-3.2-3B	0.50	0.62	+12 pp
Llama-3.1-8B	0.54	1.00	+46 pp
Gemma-2-2B	0.50	0.67	+17 pp
Gemma-2-9B	0.46	0.92	+46 pp
Gemma-4-E2B	0.54	0.79	+25 pp
Gemma-4-E4B	0.50	0.33	−17 pp (backfire)

split matches the U-shaped position bias reported for language models, where the start (primacy) and the end (recency) of the context receive the most weight [8]. The new point here is that which end dominates differs by model, which is exactly why one fixed ordering cannot help every system.

6.5 Newer models are not fresher; size predicts the fix, not the baseline

The 2024 and 2026 Gemma models let us ask whether newer models are fresher. They are not. Gemma-4 (2026) matches Gemma-2 (2024) at baseline (about 0.50 each), and Gemma-4 actually responds worse to the fix. Model size within a family also does not predict baseline freshness in a clean way: Qwen2.5 at 1.5B, 3B, and 7B gives 0.50, 0.62, and 0.51. What size does predict is how well the fix works: the larger, more capable models (Llama-3.1-8B, Gemma-2-9B) are the ones the recency fix helps most. Earlier sweeps over sampling temperature, the number of retrieved documents, and the anti-flicker window left the baseline within a narrow band, so the result is not an artifact of a single setting.

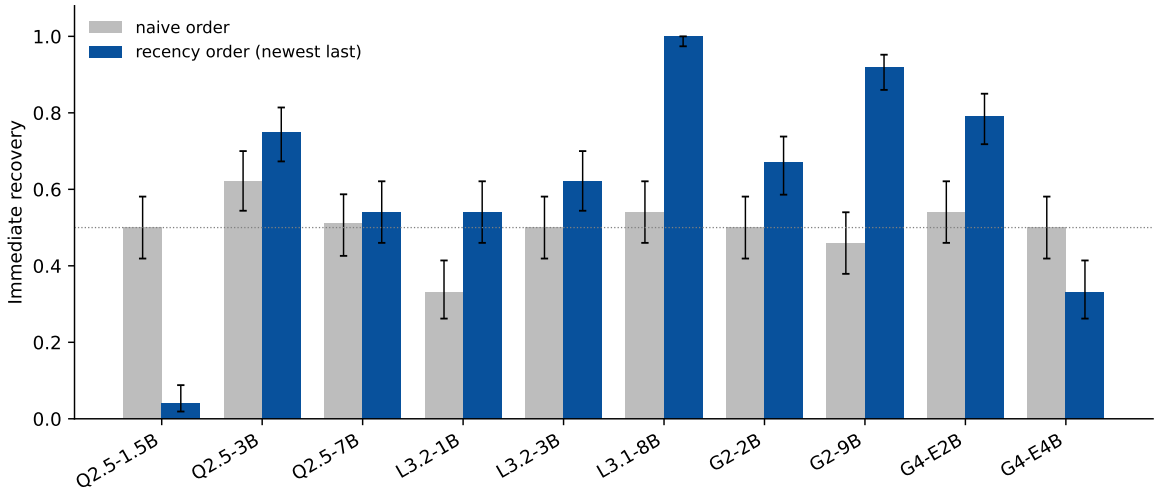

Figure 3: The recency-order fix (placing the newest document last) is model-dependent. Grey is naive order, blue is recency order; immediate policy, $n = 144$, with Wilson 95% confidence intervals. The fix solves the problem for the most capable models (Llama-3.1-8B reaches 1.00, Gemma-2-9B 0.92) but backfires on others (Qwen2.5-1.5B falls to 0.04).

7 Discussion and limitations

What this means for RAG builders. A fast index is not enough. A team that wants fresh answers should measure the answer, not only the index. Reordering retrieved documents so the newest is last is cheap to try, and for some models it removes the staleness entirely. But it is not a safe default: on some models it makes things worse. So the practical advice is to measure it on your own model rather than assume it, which is what StaleBench is for.

Limitations. The numbers come from controlled, synthetic facts with single-value questions, so the exact rates are scenario-dependent and should be read as the size of the problem, not fixed constants. The models are small and mid-size open models, served locally at 4-bit quantization with reasoning turned off; larger, closed, or reasoning models may behave differently. One 2026 model (Qwen3.5) was excluded because its reasoning could not be disabled in our serving stack. The two 2026 Gemma-4 models are multimodal and were used in text-only mode, so the generation comparison mixes a change in model generation with a change in model type. StaleBench is built so readers can run these same measurements on their own data and systems, which is the right way to draw conclusions about a specific system.

Future work. Natural next steps are real-world corpora in place of synthetic facts, a wider model set that includes frontier models, and richer answer types such as numbers and dates. The benchmark already supports custom answer checkers and custom systems, so these extensions do not need changes to the core.

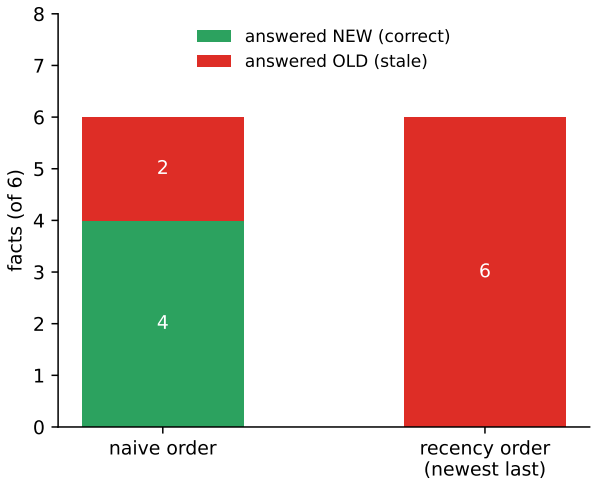

Figure 4: Primacy bias in Qwen2.5-1.5B on six changed facts. With the newest document placed last, the model answers with the old value on all six, which explains why the recency fix collapses its recovery.

8 Conclusion

A RAG system can be fresh in its index and stale in its answer. StaleBench measures answer freshness through catch-up latency and recovery rate, across refresh policies, with exact-match scoring and confidence intervals. Across ten models from three families, about half of answers stay stale even under immediate refresh, and the cause is the position of the documents, not the retriever. A recency-order fix removes the staleness entirely for the most capable models but backfires on others, so it must be checked per system. StaleBench gives teams a clear way to see this gap in their own systems and to act on it.

Availability

StaleBench is released as open-source software. The code is licensed under Apache-2.0 and the experiment data under CC-BY-4.0. See `https://github.com/KaranSinghDev/stalebench`.

References

[1] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[2] Edwin B. Wilson. Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association*, 22(158):209–212, 1927.

[3] Peter Bailis, Shivaram Venkataraman, Michael J. Franklin, Joseph M. Hellerstein, and Ion Stoica. Probabilistically bounded staleness for practical partial quorums. In *Proceedings of the VLDB Endowment*, 2012.

[4] Jie Ouyang, Tingyue Pan, Mingyue Cheng, Ruiran Yan, Yucong Luo, Jiaying Lin, and Qi Liu. HoH: A dynamic benchmark for evaluating the impact of outdated information on retrieval-augmented generation. *arXiv preprint arXiv:2503.04800*, 2025.

[5] Fedor Chernogorskii, Sergei Averkiev, Liliya Kudraleeva, Zaven Martirosian, Maria Tikhonova, Valentin Malykh, and Alena Fenogenova. DRAGOn: Designing RAG on periodically updated corpus. *arXiv preprint arXiv:2507.05713*, 2025.

[6] Shayne Longpre, Kartik Perisetla, Anthony Chen, Nikhil Ramesh, Chris DuBois, and Sameer Singh. Entity-based knowledge conflicts in question answering. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2021.

[7] Evgenii Kortukov, Alexander Rubinstein, Elisa Nguyen, and Seong Joon Oh. Studying large language model behaviors under context-memory conflicts with real documents. *arXiv preprint arXiv:2404.16032*, 2024.

[8] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics (TACL)*, 2024. arXiv:2307.03172.

[9] An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, et al. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2025.

[10] Aaron Grattafiori et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[11] Gemma Team. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.

[12] Shitao Xiao, Zheng Liu, Peitian Zhang, and Niklas Muennighoff. C-Pack: Packed resources for general chinese embeddings. *arXiv preprint arXiv:2309.07597*, 2024.