WHITE PAPER · RETRIEVAL-AUGMENTED GENERATION

RAG: Coming together at the Seams

John Kelly · 5 November 2026

ABSTRACT

Retrieval-augmented generation (RAG) answers questions by searching a collection of documents and passing the best-matching passages to a language model. Before a document can be searched, it is cut into pieces called chunks, and every cut is a place where an idea can be split in two. This paper describes that boundary problem, reviews how current chunking strategies try to manage it, and identifies the gap they leave: a fact that is split across two large chunks may not be found at all, because neither half resembles the question closely enough. It then introduces **Seams**, a chunking strategy that indexes a small, separate search vector over every chunk boundary and links it to both chunks it straddles. Seams is designed for long evidence in long chunks: conceptual material, where meaning develops over several paragraphs and chunks larger than 512 tokens are preferred. Seams keeps every chunk clean and disjoint, duplicates no text, works within a fixed context budget, adds to an existing index without rebuilding it, and has only two settings of its own. Published results at smaller chunk sizes found little benefit from keeping split evidence whole, so the paper confines its claim to this setting. This paper is a proposal: it sets out how Seams works, the conditions under which it applies, what it costs, and how its benefit can be measured, which is left to future work.

1. The problem space: where chunks come apart

1.1 Why documents are cut into chunks

A retrieval-augmented generation system keeps its knowledge outside the language model, in an ordinary collection of documents, and looks up the relevant passages each time a question arrives. The founding RAG paper by Lewis and colleagues paired a document retriever with a text generator and used Wikipedia as its knowledge source, splitting every article "into disjoint 100-word chunks, to make a total of 21M documents"[1, p. 4]. The retriever it relied on, Dense Passage Retrieval (DPR), made the same choice and split each article "into multiple, disjoint text blocks of

100 words"[2, p. 4]. Later systems kept the pattern: In-Context RALM, which simply places retrieved passages in front of an unmodified language model, used "non-overlapping passages of 100 words"[3, p. 5].

Today the most common method is still to cut documents "on a fixed number of tokens (e.g., 100, 256, 512)," where a token is a word or a piece of a word[5, p. 8]. The trade-off is well described in the RAG survey by Gao and colleagues: "Larger chunks can capture more context, but they also generate more noise," while smaller chunks "may not fully convey the necessary context"[5, p. 8].

1.2 The boundary problem

Every fixed cut is made without looking at the text, so some cuts land in the middle of an idea. The same survey states the problem directly: chunking "leads to truncation within sentences"[5, p. 8]. A 2026 study names it boundary fragmentation, "where critical evidence is split across segments," and reports that it lowers retrieval recall[11]. Anthropic's guidance on its Contextual Retrieval method likewise warns that "the choice of chunk size, chunk boundary, and chunk overlap can affect retrieval performance"[7, p. 9].

How often does this happen? A simple model helps. If a piece of evidence is L tokens long and sits at a random position in text that is cut every C tokens, the chance that a cut falls inside it is roughly $L \div C$. Figure 1 plots this model. It is an illustration, not a measurement, but it shows the key point: the risk is small for short facts and large for long evidence such as procedures, arguments and tables.

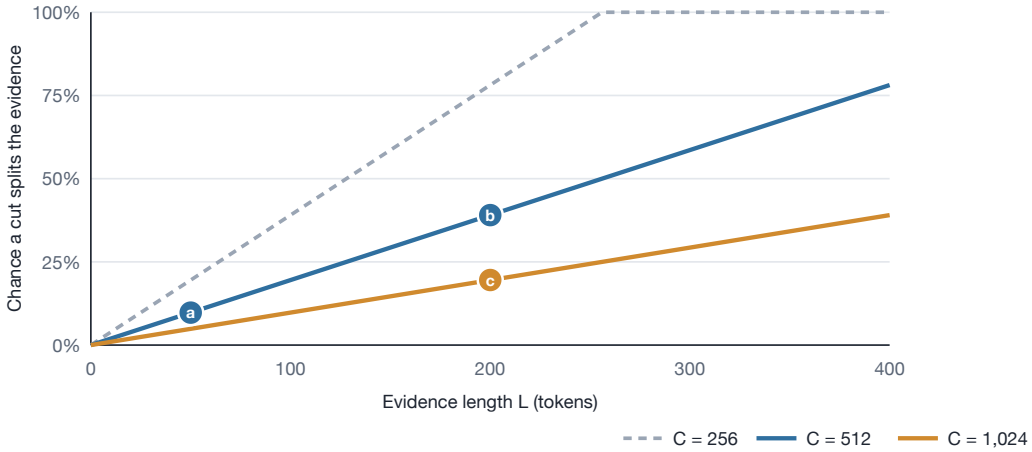


Figure 1. An illustrative model of boundary risk: the chance that a fixed cut splits a piece of evidence is about L ÷ C, where L is the evidence length and C is the chunk size. Long, conceptual evidence is at much greater risk than short facts. This is a geometric model, not measured data, and it shows how often a cut falls inside evidence, not how often a split causes evidence to be missed (see Section 2.3).

1.3 How current strategies optimize chunking

Practitioners and researchers have developed several ways to manage chunk size and chunk boundaries. Table 1 summarizes the main families and what the evidence says about each.

Strategy	How it works	What the evidence shows
Fixed-size chunks	Cut every N tokens, without looking at the text.	In preliminary trials reported in a footnote, DPR found that "fixed-length passages performs better in both retrieval and final QA accuracy" than natural paragraphs[2, p. 2]. A 2024 Vectara study found fixed-size chunking beat semantic chunking on real documents, for example an F1 score of 90.59 against 87.37 on HotpotQA, with five chunks retrieved, against the breakpoint semantic chunker[8].
Overlap	Repeat the end of each chunk at the start of the next, so a split idea survives whole in at least one chunk.	Mixed. Chroma's evaluation found overlap raised recall for small chunks (77.1 to 82.4 at 250 tokens), suggesting that "for smaller context, overlapping chunks are necessary for high recall"[9]. At 400 tokens it slightly lowered recall and reduced the share of useful text in results[9]. DPR tried overlapping passages and did "not find [them] advantageous compared to the non-overlapping version"[2, p. 4].
Semantic chunking	Cut where the meaning of neighbouring sentences changes most.	Its clearest wins came on artificially "stitched" documents with high topic diversity (63.93 against 43.79 on NQ). The authors concluded that "fixed-size chunking remains a more efficient and reliable choice for practical RAG applications"[8].
Small-to-big and hierarchies	Search small units and return the larger text around them. Small2Big uses "sentences (small)" as the retrieval unit and supplies the "preceding and following sentences" as context[5, p. 8]. LlamaIndex's auto-merging retriever builds levels of 2,048, 512 and 128 tokens[13].	Widely used. The auto-merging retriever "recursively 'merges' subsets of leaf nodes that reference a parent node beyond a given threshold"[13]. Merging works upward within a parent, so evidence split between two parents is not reunited.
Neighbour expansion	After a chunk is found, also fetch the chunks beside it.	SCAR expands neighbours selectively and reaches "92.8% recall on boundary-fragmented queries with only 7.84 chunks," against 10.16 for static windowing[11].

Strategy	How it works	What the evidence shows
Late chunking	Embed the whole text first, then split, so each chunk's vector reflects its neighbours.	Described as preserving "cross-chunk context by applying long-context embeddings before splitting"[6, p. 2]. Limited to text that fits the embedding model's window.
Contextual Retrieval	Prepend a short, model-written description of each chunk's place in its document before indexing.	Reduced the top-20 retrieval failure rate by 49% (5.7% to 2.9%), and by 67% with reranking (5.7% to 1.9%)[7, pp. 8–10]. A 2026 benchmark confirmed consistent gains[6, p. 4].

1.4 The gaps these strategies leave

Gap 1: a split fact can be lost before anything is fetched. Neighbour expansion and small-to-big retrieval both repair context *after* a chunk has been found. They cannot help when a fact is split so evenly that neither half resembles the question closely enough to be retrieved. Overlap does address this case, because it keeps the fact whole inside one chunk, but in a large chunk that fact is surrounded by hundreds of unrelated tokens. Chroma's authors speculate that relevant information can become "diluted within chunks"[9]. The argument that dilution makes a split fact hard to find is this paper's reasoning, built on that suggestion.

Gap 2: overlap pays for recall with duplicated text. Overlap places the same text in two chunks, so neighbouring chunks are often retrieved together and the duplicate fills slots that could hold different evidence. Figure 2 shows Chroma's measurements. Overlap helped recall for small 250-token chunks, but at 400 tokens it reduced the share of useful text in the results for both splitters tested[9]. Chroma measures that share as precision Ω, defined "for the case that all chunks containing excerpt tokens are successfully retrieved," which "gives an upper bound on token efficiency given perfect recall"[9]. Chroma also notes that "reducing chunk overlap improves IoU scores, as this metric penalizes redundant information"[9]. IoU (intersection over union) compares the relevant text with the retrieved text and counts every retrieved token, so duplicated text lowers it[9].

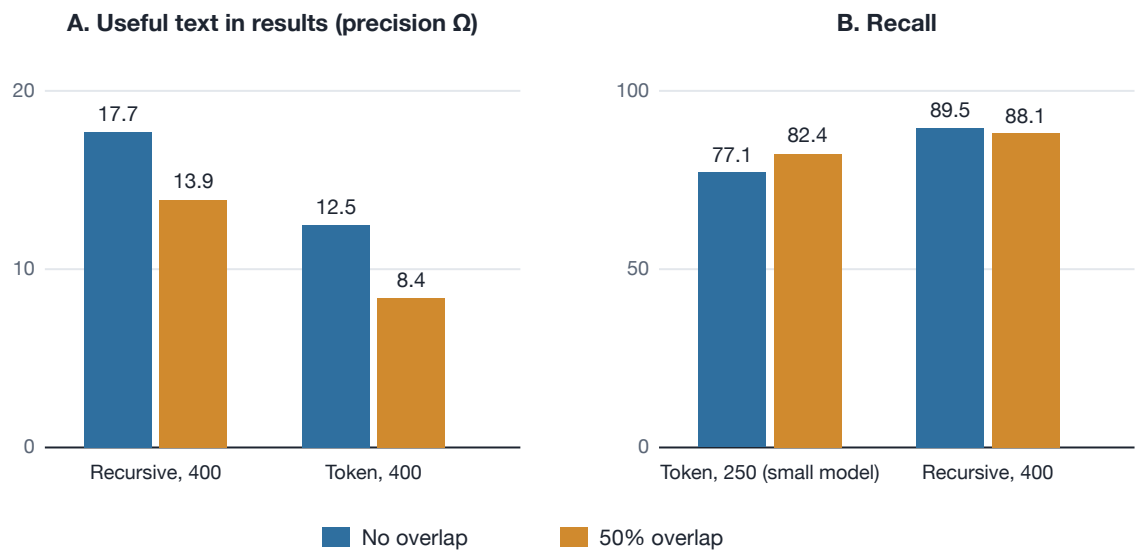


Figure 2. Chroma's chunking evaluation, retrieving five chunks[9]. Panel A: with 400-token chunks, overlap lowered the share of useful text in the results (precision Ω) for both splitters. Panel B: overlap raised recall for small 250-token chunks with a small embedding model, but slightly lowered it for 400-token recursive chunks. All runs use text-embedding-3-large except the 250-token pair, which uses all-MiniLM-L6-v2.

Gap 3: the larger the chunk, the more each repair costs. To guarantee that evidence of a given length is never split, overlap must be at least that long. For conceptual material with long evidence, that means a large fraction of every chunk is duplicated. Neighbour expansion instead sends whole extra chunks to the model, and with large chunks each one adds a great deal of text. That matters because "overly long contexts can also lead LLM to the 'Lost in the middle' problem," in which a language model attends mostly to the beginning and end of its input[5, p. 10]. The RAGAS authors make the same point about retrieved context that is too long[4, p. 3].

Gap 4: meaning-based cutting does not reliably fix boundaries. Semantic chunking tries to put cuts in the right places, but on real documents it did not beat fixed-size chunking, and its advantage was clearest when topics changed abruptly[8]. Its results are also hard to separate from chunk size, because the same study notes that "precision rises significantly as chunk size decreases"[8].

Gap 5: boundaries are rarely measured directly. Much of the evidence above concerns chunk size or chunking method, not boundaries themselves. A 2026 benchmark of ten retrieval strategies indexed each document "as a single unit without chunking," and lists chunking-strategy ablations as future work[6, pp. 3, 8]. SCAR is a notable exception, because it built a set of 160 boundary-fragmented questions specifically to measure this failure[11].

2. Introducing Seams, and when to use it

> **DEFINITION**
>
> **Seams** is a chunking strategy that keeps a document's ordinary, non-overlapping chunks and adds one small search vector over every boundary between adjacent chunks. Each of these **seam vectors** embeds the text that straddles the cut and is linked to both chunks it joins. A seam is used only to *find* chunks: when a seam matches a question better than either chunk beside it, both of those chunks are passed to the language model, and the seam's own text is never shown to the model or cited.

The name reflects the idea. A chunked document is like a garment cut into panels, and the seams are where the panels meet. Ordinary retrieval looks only at the panels. Seams adds a way to search the joins, so that evidence lying across a join can be found and then read in full.

Seams builds on documented patterns rather than replacing them. LangChain's `MultiVectorRetriever` already "supports multiple embeddings per parent document," so that a match on any small vector returns the larger document it belongs to[12]. Small2Big already searches small units and returns their surroundings[5, p. 8]. Seams applies the same principle at exactly one place, the boundary, and gives each small vector two parents instead of one. The design is new, so the conditions below are reasoned from the evidence in Section 1.

The closest broader alternative is to index sliding windows over the whole document. The RAG survey lists a "sliding window approach" among the indexing refinements of advanced RAG[5, pp. 3, 8]. A multi-vector version would embed, for example, 256-token windows at a stride of 128 tokens and link each window to the chunk or chunks it overlaps. That design covers chunk interiors as well as boundaries, and Seams is a special case of it: one window per boundary, placed so that it straddles the cut. The difference is cost. Windows at a stride of s tokens add about $C \div s$ vectors to every chunk of C tokens, which is 8 extra vectors per 1,024-token chunk at a 128-token stride, or about nine times the vectors of the chunks alone. Seams adds one vector per boundary, which is about twice the chunk-only index. The argument for indexing only the boundaries is that evidence lying inside a chunk is already whole, so it can be found by the chunk's own vector. The extra interior windows mainly address a different problem, the dilution of short facts inside long chunks. Seams provides the boundary coverage with less than a quarter of those vectors, and Section 3.5 includes sliding windows among the comparisons for measuring its benefit.

2.1 The central case: conceptual content and chunks larger than 512 tokens

Some documents carry their meaning in short, self-contained facts, such as a product catalogue or a list of error codes. Others build meaning gradually, across several paragraphs: research papers, policy documents, legal agreements, design documents and technical procedures. In conceptual material like this, the answer to a question is often an argument or a sequence of steps rather than a single sentence.

The evidence suggests that such material is better served by larger chunks. In NVIDIA's chunking benchmark, datasets of factual-lookup questions did best with chunks of 256 to 512 tokens, while analytical datasets "generally benefited from larger chunk sizes (1,024 tokens) or page-level chunking"[10]. Chroma's authors observe that "chunks which are too small fail to capture necessary context within a single unit"[9], and the RAG survey notes that coarser retrieval units "can provide more relevant information for the problem," at the cost of extra noise[5, p. 7].

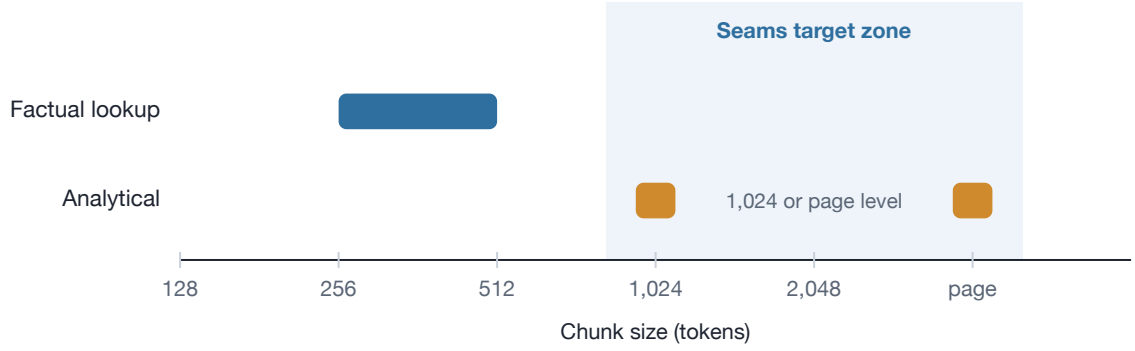


Figure 3. Best-performing chunk sizes by question type in NVIDIA's benchmark[10]. Factual-lookup datasets favoured 256–512 tokens; analytical datasets favoured about 1,024 tokens or whole pages. The bars show dataset results; NVIDIA's general guidance also allows page-level chunking for factoid questions. Seams is aimed at the shaded zone, where chunks are large and conceptual.

Large chunks make the boundary problem harder, not easier, for three reasons. First, the evidence in conceptual material is long, so by the model in Figure 1 a substantial share of it still straddles a cut: a 200-token argument has about a one-in-five chance of being split even with 1,024-token chunks. Second, each repair is expensive at this size, because overlap must duplicate long stretches of text and neighbour expansion must send whole extra chunks of 1,024 tokens or more. Third, the dilution described in Gap 1 is at its worst, because a split fact inside an overlapping 1,024-token chunk shares that chunk with roughly a thousand other tokens. Seams is designed for exactly this situation: it searches the boundary with a small vector, where a straddling fact is concentrated, and expands to the two large chunks only when that search succeeds.

2.2 Conditions under which Seams is the right choice

Seams is the recommended strategy when conditions 1, 3 and 7 hold and most of the others do. Each condition follows from the gaps in Section 1.

1. **Chunks are large**, above 512 tokens, because the content is conceptual and needs room for its meaning to develop.
2. **Cuts are arbitrary**, as with fixed-size chunking or text with little usable structure, so boundaries often fall inside ideas.
3. **Evidence is long**, roughly 150 tokens or more, such as procedures, clauses, multi-sentence arguments or tables, so the share of split evidence is high (Section 2.3 explains why this matters).
4. **Each passage should be stored and returned once.** Overlapping chunks can also record exact source positions, but they store shared text twice, so results must be deduplicated and the same passage can be cited from two chunks. Non-overlapping chunks keep every passage in exactly one chunk.
5. **Re-indexing is unwelcome.** Seams can be added beside an existing chunk index without re-chunking or re-embedding a single chunk.
6. **Late chunking is not available**, for example because no long-context embedding model is in use or documents exceed its window[6, p. 2].
7. **Index memory can grow.** Seams roughly doubles the number of vectors, as Section 3.4 explains.
8. **The collection is much larger than the final budget.** The final budget is the number of chunks sent to the model, for example the top 20. If every chunk fits in the answer anyway, no evidence can be lost at a boundary. Anthropic notes that a knowledge base "smaller than 200,000 tokens" can simply be included in the prompt in full[7, p. 2].

2.3 The counter-evidence, and the narrower claim

Two of the sources this paper relies on cut against its premise, and they should be faced directly. DPR, working with 100-word passages and factoid questions, tried overlapping passages and did not find them advantageous[2, pp. 2, 4]. Chroma, at 400 tokens, measured recall of 89.5 without overlap and 88.1 with 50 percent overlap for its recursive splitter[9]. If keeping evidence whole inside one chunk does not raise recall at these sizes, then boundary splits may cause less harm than Figure 1 suggests. Figure 1 shows how often a cut falls inside evidence, not how often that cut causes the evidence to be missed.

There is a reasonable explanation that keeps Seams in play, but it narrows the claim. Both results come from settings where evidence is short relative to the chunk. A factoid answer inside a 100-word passage is short, and when a short fact is split, one half usually still matches the question well enough to be retrieved. Neither study tested long evidence in long chunks, which is the setting this paper targets. Seams is therefore claimed only for **long evidence in long chunks**: evidence of roughly 150 tokens or more (a working threshold, not a measured one), in chunks of about 1,024 tokens, sought by questions that match only part of that evidence. The seam itself must be wide enough to hold such evidence, so Section 3.1 sets the seam half-width to at least 150 tokens. Outside that setting, the counter-evidence suggests plain disjoint chunks are already adequate. Within it, Section 3.5 describes how the benefit of Seams can be measured.

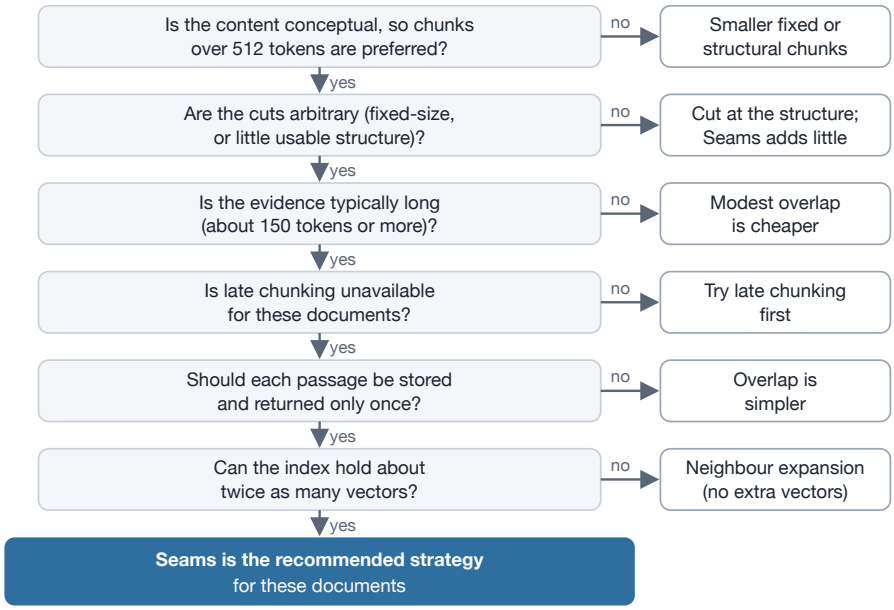


Figure 4. A decision chart that condenses the conditions in Section 2.2, one question per box. A yes to every question is the clearest case for Seams; Section 2.2 states the minimum, which is conditions 1, 3 and 7 plus most of the others. The chart leaves out condition 5, which is a benefit rather than a test, and condition 8, which holds for almost any collection large enough to need retrieval. Section 3.5 describes how the benefit of Seams can be measured.

2.4 When Seams is the wrong choice

Seams adds little when chunks follow real section breaks, because few cuts then fall inside ideas. It is also a poor fit for small chunks of about 256 tokens or fewer, because modest overlap gives the same protection at lower cost and Chroma found overlap genuinely helps recall at that size[9]. For short factual content, most evidence is never split (Figure 1), so doubling the index buys little. Finally, where late chunking or adaptive neighbour expansion is available and performing well, those methods may already cover the boundary problem without extra vectors; SCAR, for example, reports high recall on boundary-split questions[11].

3. How Seams works

3.1 Indexing

Seams starts from the baseline used by the founding systems: plain, disjoint, fixed-length chunks. This is the form used by the original RAG and DPR systems[1, p. 4; 2, p. 4]. Indexing then proceeds in four steps.

1. **Chunk the document** into non-overlapping chunks of the chosen size, for example 1,024 tokens, and give each one a stable identifier and its exact position in the source.
2. **Cut a seam window at every boundary.** Take the last k tokens of each chunk and the first k tokens of the next, snapping to sentence edges where they exist. The seam must be wide enough to hold the evidence Seams targets, so this paper recommends a half-width k of at least 150 tokens, matching the evidence length in Section 2.3. A value of $k = 150$ gives a seam of about 300 tokens, which is still less than a third of a 1,024-token chunk. This setting is this paper's recommended starting point. Snapping to a sentence edge can shorten one side below k, which weakens the guarantee described in Section 3.4, so an implementation that needs the full guarantee should snap outward, keeping at least k tokens on each side.
3. **Embed each seam** with the same embedding model as the chunks, and store it with two links: `left_chunk_id` and `right_chunk_id`.
4. **Keep seams separate from chunks**, either in their own collection or with a type field, so that seam text can never be returned as evidence.

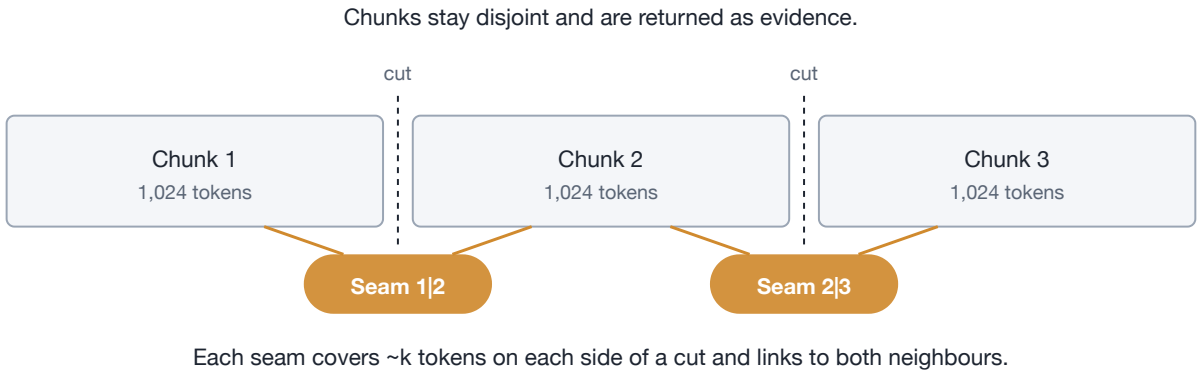


Figure 5. The Seams index. Ordinary chunks (grey) are stored and searched as usual. Each seam vector (amber) embeds a short window that straddles one cut and carries links to the chunk on each side.

3.2 Querying

At query time, Seams adds one decision to an ordinary retrieval pipeline. Figure 6 shows the flow, and a worked example follows it.

1. **Search chunks and seams together**, ideally with hybrid search that combines meaning-based vectors with keyword matching. Anthropic found that "Embeddings+BM25 is better than embeddings on their own," where BM25 is a standard keyword-ranking function[7, p. 11]. Seam text repeats text from chunk edges, so if keyword search runs over one collection holding both, words near boundaries are counted twice. Keyword statistics should be computed for each collection separately, and the resulting lists fused.
2. **Combine the rankings, not the raw scores.** A short seam and a long chunk score differently against the same question, so their scores are not directly comparable. Reciprocal rank fusion (RRF) merges ranked lists using positions only and "requires no score normalization"[6, p. 3]. In one benchmark, a convex combination, which is a weighted average of normalized scores, reached higher recall than RRF (0.726 against 0.695 in the top five)[6, p. 4]. RRF remains the better fit for Seams, because a weighted average needs seam and chunk scores to be normalized against each other, and those scores are distorted by the difference in length that RRF ignores.
3. **Apply the fire test.** A seam is short, and short passages can rank above long ones for reasons that have nothing to do with boundaries, so a seam may outrank its neighbours even when the evidence lies wholly inside one chunk. The fire test therefore requires a clear win. With rank 1 as the best match, a seam fires only when rank(seam) + m ≤ rank(left chunk) and rank(seam) + m ≤ rank(right chunk). A linked chunk that does not appear in the ranked list counts as ranked last, so it never prevents a firing. A seam that does not fire is dropped from the ranking, and its linked chunks keep their own places. The margin m is a setting chosen for each deployment, and it can be calibrated without hand-labelled data: sample sentences that lie wholly inside one chunk, near its edge, so that they also appear in that chunk's seam; have a language model write a question that each sentence answers; run those questions; and record how many places the seam ranks above the chunk that truly holds the answer. Setting m just above the 90th percentile of that head start means that about nine in ten unsplit facts do not fire. The same sampling, repeated with sentences that cross a cut, shows how far seams outrank their chunks when a fact really is split; those sentences lie inside the seam window by construction, so the second sample costs nothing extra. The working range for m lies between the two distributions. If they overlap heavily, the collection lies outside the setting Seams is designed for, because its long evidence is rarely split evenly. Because positions in the fused list depend on how many seams land among the chunks, m is calibrated for each deployment and does not carry over between collections or retrieval depths. This calibration is this paper's proposal. A score-ratio threshold, like the relative threshold SCAR uses[11], is an alternative form of the same setting, but it compares raw scores that the difference in length distorts.
4. **Rerank, if the pipeline uses a reranker.** A reranker reads the question together with each candidate and scores it afresh; Anthropic, for example, reranked its top 150 candidates and kept the top 20[7, p. 10]. The fire test runs on the first-stage fused ranking, before any

reranker. Seams are never passed to the reranker, because their text is never used as evidence, and the chunks a fired seam brings in are protected from its cutoff, because each holds only part of a split fact and, judged on its own, could be wrongly discarded. The reranker receives the first N ordinary chunks of the fused ranking after seams are removed (Anthropic used $N = 150$). Chunks brought in by a fired seam are left out of reranking and pinned at their seam's position, even if one of them was also retrieved directly. The reranker's order fills the positions around them.

5. **Fill the final budget in rank order.** The final budget, such as the top 20, is filled strictly in rank order. An ordinary chunk is added where it ranks. A fired seam adds both of its linked chunks where it ranks, skipping any chunk already included, so two adjacent firings that share a chunk cost three slots rather than four. A fired seam needs one slot for each linked chunk not already included, so it needs two slots, one, or none. If it needs more slots than remain, it is skipped and those slots go to the next results, so both halves of a split fact always arrive together and the budget stays fixed. Seams therefore take space only where they outrank the alternatives, and no separate cap on firings is needed. Adding both chunks is safe because chunks are disjoint and each seam is cut from the last k tokens of one chunk and the first k tokens of the next, so every token the seam matched lies inside one of the two chunks, and sending both guarantees that the matched evidence reaches the prompt. A stricter rule that added a chunk only if it also ranked among the first search results would not be safe, because each half of an evenly split fact is diluted inside its large chunk and can rank too low to be found, which is exactly the case Seams exists for. The cost is that when the evidence sits wholly at the edge of one chunk, the other chunk is wasted; because it occupies a slot in the fixed budget, it displaces a lower-ranked result rather than adding extra text. If no seam fires, ordinary chunks fill the budget as usual.
6. **Build the prompt from original chunk text only.** Seam text is a pointer for finding chunks and is never cited.

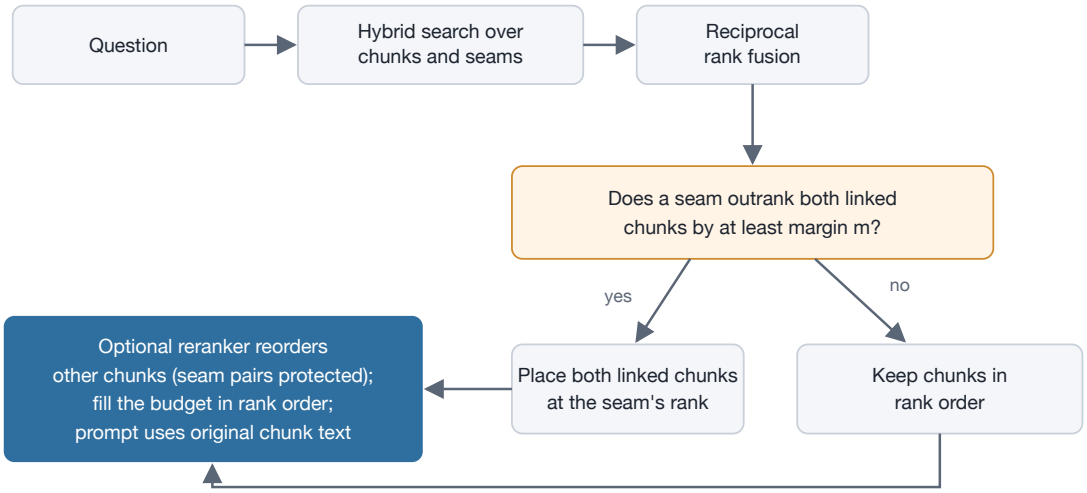


Figure 6. The query-time flow. The only addition to a standard hybrid retrieval pipeline is the fire test: a seam must outrank both linked chunks by a margin m, and when it does, both chunks take the seam's place in the ranking. Any reranker runs afterwards and cannot remove a seam's pair, and the final budget is filled in rank order.

WORKED EXAMPLE

This example traces one question through the steps above. Its ranks are illustrative, not measured.

The document. A design document for an order-processing service is cut into 1,024-token chunks, and each seam takes k = 150 tokens from each side of its cut. One 200-token passage argues a decision in three short paragraphs. The first two set out the trade-off: the service must keep taking orders during a regional outage, so it accepts briefly stale reads. The third states the decision and its exception: eventual consistency, except for billing records, which must always be strongly consistent. The cut between chunks 7 and 8 falls after the trade-off: its 110 tokens end chunk 7, and the 90 tokens of the decision and exception begin chunk 8. Seam 7|8 holds the whole argument, with only 40 tokens of other text before it and 60 after, so two-thirds of the seam is the argument. In each chunk, by contrast, the argument is only about a tenth of the text.

The question. "Why does the order service accept stale reads?" matches the trade-off in chunk 7 most closely, but a correct answer also needs the exception in chunk 8: billing records are never read stale. The final budget is five chunks, the margin is m = 3, and there is no reranker.

The fused ranking mixes seams and chunks:

Position	Entry	What it holds
1	Seam 7|8	The whole argument

Position	Entry	What it holds
2	Chunk 12	An earlier section on read replicas
3	Seam 12|13	The cut after chunk 12; chunk 13 is not in the list
4	Chunk 7	The trade-off: availability over fresh reads
5–7	Chunks 3, 20 and 15	Background material
11	Chunk 8	The decision, and the exception for billing

Without Seams. An index of chunks alone returns chunks 12, 7, 3, 20 and 15. Chunk 8 falls outside the five, because its 90 tokens of argument are diluted in 1,024. The model receives the trade-off but not the exception, and can confidently answer that the service accepts stale reads everywhere, billing included.

The fire test. Seam 7|8 is at position 1, and 1 + 3 = 4, which is no worse than chunk 7's position of 4 or chunk 8's position of 11, so it fires. Seam 12|13 is at position 3, and 3 + 3 = 6, which is worse than chunk 12's position of 2, so it does not fire. It is dropped from the ranking and chunk 12 keeps its own place. Chunk 13, which is absent and so counts as ranked last, would not have blocked the firing; chunk 12 does.

Filling the budget. The list is then read in order:

Position	Entry	Action	Slots used
1	Seam 7|8 (fired)	Needs two slots, for chunks 7 and 8, and five remain, so both are added	2 of 5
2	Chunk 12	Added at its rank	3 of 5
3	Seam 12|13	Did not fire; already dropped	3 of 5
4	Chunk 7	Already included; skipped	3 of 5
5	Chunk 3	Added	4 of 5
6	Chunk 20	Added; the budget is full	5 of 5

The result. The model receives chunks 7, 8, 12, 3 and 20, in their original text, and together they hold the whole argument, exception included. The prompt is the same size as before, five chunks of 1,024 tokens. The cost is chunk 15, the fifth result of the plain ranking, which chunk 8 displaces. If the argument had sat wholly inside chunk 7, chunk 7 would usually rank close to the seam and the margin would stop it from firing. When such a false fire does happen, the price is the same: one displaced chunk.

3.3 How Seams addresses each gap

Gap from Section 1	How Seams responds
1. A split fact can be lost before anything is fetched	The seam vector holds the straddling text whole, surrounded by only about *k* tokens on each side instead of a full chunk, so the fact is less diluted and can be matched directly. This is the Small2Big principle[5, p. 8], applied at the boundary.
2. Overlap duplicates text	Chunks remain disjoint, so no passage is returned twice or cited twice. Seams copy text from chunk edges only for searching, and that text is never returned.
3. Repairs are costly for large chunks	The two neighbouring chunks are sent only when the seam outranks both of them by a margin, rather than for every match. They take the seam's place in the same final budget, filled in rank order, so the context never grows beyond it.
4. Meaning-based cutting is unreliable	Seams does not depend on finding the right cut. It accepts arbitrary cuts and covers them, and it can also sit on semantic cuts if those are used.
5. Boundaries are rarely measured	Seam hits are directly observable, so a deployment can count how often evidence actually crosses a boundary in its own corpus (see Section 3.5).

3.4 What Seams costs

Seams is not free, and its main cost is easy to underestimate. A vector index's memory depends on the number and size of its vectors, not on how long the text behind them is. HNSW (Hierarchical Navigable Small World), the default vector index in Weaviate, is "an in-memory index, where each node in the graph as well as each edge between nodes are stored in memory"[15]. Weaviate's resource guide states that "the HNSW vector index is the primary driver of memory usage" and gives the rule of thumb "Memory usage = 2 * (the memory footprint of all vectors)"[14]. A 300-token seam therefore costs as much index memory as a 1,024-token chunk, and adding one seam per boundary roughly doubles the vector count. The doubling of memory applies to indexes that hold their vectors in memory, such as HNSW. Compression lowers the cost of each vector, since Weaviate describes product quantization as "a technique that reduces the size of vectors," and Weaviate also offers a disk-based index for cases where "memory is your main constraint"[14]. Either option reduces the absolute cost, although the number of vectors still doubles.

Overlap is cheaper in vector count for the same guarantee. A seam that takes k tokens from each side is certain to contain any split evidence up to about k tokens long, which is the same guarantee as k tokens of overlap. It can hold evidence up to $2k$ tokens long when that evidence is centred on the cut, and the guarantee holds only if snapping leaves at least k tokens on each side (Section 3.1). With k = 150 and 1,024-token chunks, the equivalent 150-token overlap needs about 1.17 times as many vectors, while Seams needs about 2 times as many (Figure 7). Raising k does not

change the number of Seams vectors, because there is still one seam per boundary; it changes only how much text each seam embeds. Evidence longer than *k* is held whole only when it sits close to the centre of the cut, so corpora with longer evidence call for a larger *k*. The case for Seams therefore rests on cleaner chunks, passages stored and returned once, more focused boundary matching and selective expansion, not on index size.

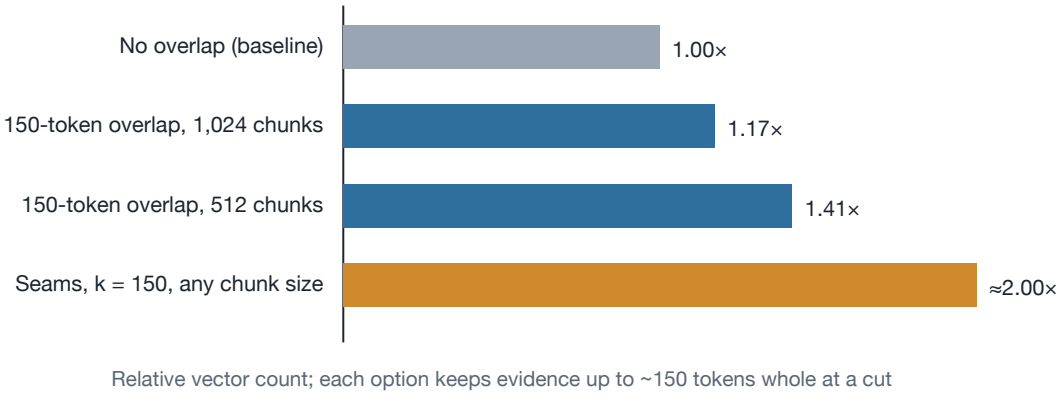


Figure 7. Relative vector count for options with the same boundary guarantee, calculated from the definitions: overlap of o tokens on chunks of C tokens needs about C ÷ (C – o) as many chunks, and Seams adds one vector per boundary. Index memory grows with vector count[14].

Sliding windows, the broader alternative discussed in Section 2, cost far more. Windows at a stride of *s* tokens add about *C* ÷ *s* vectors per chunk, so 256-token windows at a 128-token stride need about 9 times the chunk-only vector count with 1,024-token chunks and about 5 times with 512-token chunks, against about 2 times for Seams.

Seams also has a context cost when it fires, because one piece of evidence then occupies two large chunk slots instead of one. Because added chunks count against the same final budget (Section 3.2), the total context does not grow, but a firing can push another relevant chunk out of that budget. When the evidence sits wholly at one edge, one of the two chunks is wasted, and a false fire wastes both slots. The extra irrelevant text also matters because "overly long contexts can also lead LLM to the 'Lost in the middle' problem"[5, p. 10]. The calibrated margin limits firings to questions whose evidence appears to cross a boundary, and a chunk reached both directly and through a seam is included only once.

3.5 Measuring the benefit of Seams

This paper is a proposal, and measuring the benefit of Seams is future work. The benefit can be measured by comparing Seams with the alternatives on the same questions: no overlap, modest overlap (about 12.5 to 15 percent), sliding windows (for example 256-token windows at a 128-token stride, linked to their chunks), sliding windows at a stride of about one chunk length, which

gives the same number of vectors as Seams and shows whether placing each window on a cut is what helps, adaptive neighbour expansion, and late chunking where available, with Seams measured at more than one seam width (for example k = 100, 150 and 200). The question set should include questions whose answers deliberately cross a chunk boundary, alongside ordinary questions, following the approach SCAR used[11].

The main retrieval measure is the failure rate used by Anthropic: "1 minus recall@20," the percentage of relevant passages not retrieved within the top 20 results[7, p. 8]. The model in Figure 1 gives that measure a target. With 1,024-token chunks, about one 200-token piece of evidence in five straddles a cut. If, for illustration, half of those straddling cases are missed today because neither chunk matches well enough, Seams can recover up to about ten percentage points of recall on questions with long evidence. That ceiling is a reasoned estimate, not a measurement, and how close Seams comes to it is the main thing to measure. Context cost is measured directly, as tokens sent per answer, alongside the failure rate, because each firing places two chunks for one piece of evidence. The precision of the fire test is measured by its **false-fire rate**: the share of firings in which the evidence lies wholly inside one chunk. The boundary questions make this measurable, because the location of each answer is known, and the calibration of m in Section 3.2 is designed to keep the rate low. A second measure is the share of firings in which one of the two added chunks holds none of the relevant passage; where that waste is significant, a side can be dropped when its half of the seam window scores close to zero against the question. RAGAS context relevance, computed as the number of sentences a model extracts as needed divided by the total sentences in the context[4, p. 4], is a useful supporting measure because it penalizes an unneeded second chunk. It is best read alongside the other measures rather than alone, because it was the least reliable RAGAS metric, agreeing with human judges in 0.70 of cases against 0.95 for faithfulness, and its authors note that it struggles "especially for longer contexts"[4, p. 5].

3.6 Summary

Seams answers one specific weakness of chunked retrieval: evidence that lies across a cut between two large chunks can be missed, because neither chunk resembles the question closely enough. It keeps chunks clean and disjoint, adds a small, focused search vector over each cut, and brings both neighbouring chunks into the answer, at the seam's place in the ranking, only when that vector outranks both of them by a calibrated margin. Because published results at smaller sizes found little benefit, the claim is confined to long evidence in long chunks: conceptual documents with chunks above 512 tokens, where each passage should be stored once and late chunking is unavailable. Seams costs about twice the vectors of an ordinary index, less than a quarter of what sliding windows over the whole document would need, and it fits on top of any retrieval pipeline that exposes its first-stage ranking, with only two settings of its own: the seam half-width k and the margin m. Section 3.5 sets out how its benefit can be measured.

References

1. P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. Yih, T. Rocktäschel, S. Riedel and D. Kiela. "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks." *Advances in Neural Information Processing Systems 33 (NeurIPS 2020)*. arXiv:2005.11401.
2. V. Karpukhin, B. Oğuz, S. Min, P. Lewis, L. Wu, S. Edunov, D. Chen and W. Yih. "Dense Passage Retrieval for Open-Domain Question Answering." *Proceedings of EMNLP 2020*, pp. 6769–6781. ACL Anthology.
3. O. Ram, Y. Levine, I. Dalmedigos, D. Muhlgay, A. Shashua, K. Leyton-Brown and Y. Shoham. "In-Context Retrieval-Augmented Language Models." *Transactions of the Association for Computational Linguistics*, vol. 11, pp. 1316–1331, 2023. ACL Anthology.
4. S. Es, J. James, L. Espinosa-Anke and S. Schockaert. "RAGAs: Automated Evaluation of Retrieval Augmented Generation." *Proceedings of EACL 2024: System Demonstrations*, pp. 150–158. ACL Anthology.
5. Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, M. Wang and H. Wang. "Retrieval-Augmented Generation for Large Language Models: A Survey." arXiv:2312.10997, version 5, 2024.
6. M. Akarsu, R. K. Karaman and C. Mierbach. "From BM25 to Corrective RAG: Benchmarking Retrieval Strategies for Text-and-Table Documents." arXiv:2604.01733, 2026.
7. Anthropic. "Introducing Contextual Retrieval." Published 19 September 2024; accessed 5 November 2026. anthropic.com/research/contextual-retrieval. Archived at web.archive.org. Page numbers refer to the dated PDF snapshot.
8. R. Qu, R. Tu and F. Bao. "Is Semantic Chunking Worth the Computational Cost?" Vectara and University of Wisconsin–Madison. arXiv:2410.13070, 16 October 2024.
9. B. Smith and A. Troynikov. "Evaluating Chunking Strategies for Retrieval." Chroma Technical Report, 3 July 2024. trychroma.com/research/evaluating-chunking.
10. S. Han. "Finding the Best Chunking Strategy for Accurate AI Responses." NVIDIA Technical Blog, 18 June 2025. developer.nvidia.com.
11. N. Langlois. "SCAR: Semantic Continuity-Aware Retrieval for Efficient Context Expansion in RAG." arXiv:2606.16661, 2026.
12. LangChain. "MultiVectorRetriever" (API reference). reference.langchain.com. Accessed 5 November 2026.
13. LlamaIndex. "Auto Merging Retriever." developers.llamaindex.ai. Accessed 5 November 2026.
14. Weaviate. "Resource Planning." docs.weaviate.io. Accessed 5 November 2026.
15. Weaviate. "Vector Indexing." docs.weaviate.io. Accessed 6 November 2026.

Page numbers for references 1–7 refer to the copies held in this library. Figures 1, 4 and 7 are derived from definitions and stated reasoning rather than measured data; Figures 2 and 3 reproduce published results.