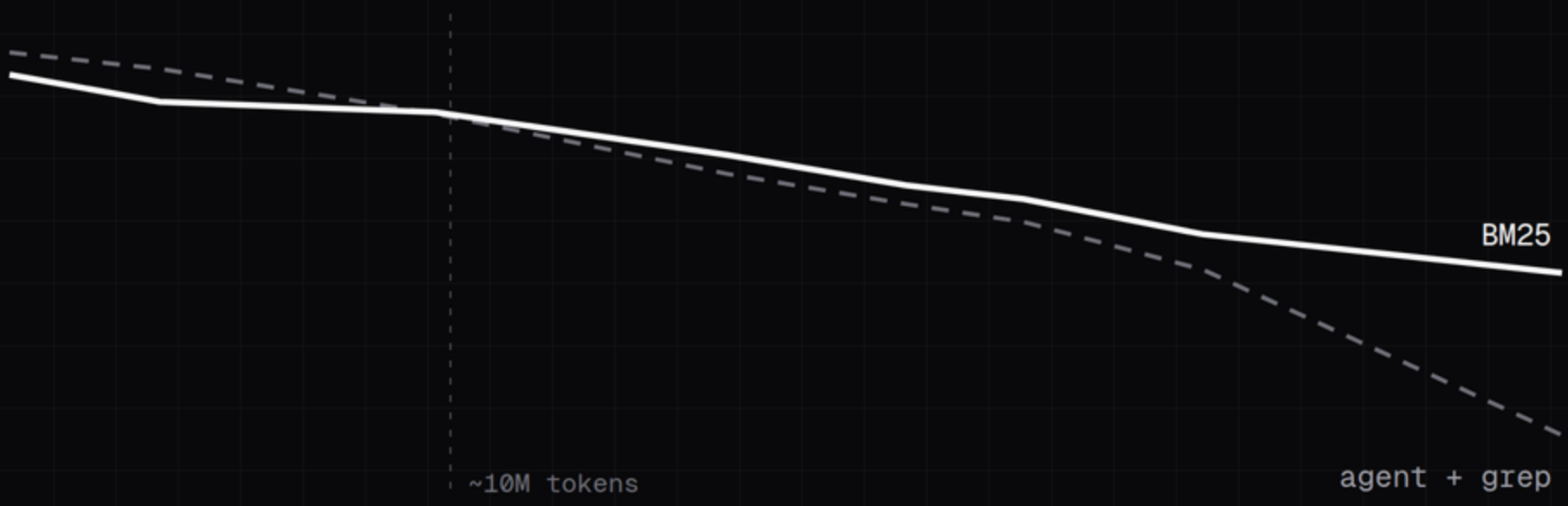
BM25 was designed in the 1990s.

In July 2026 it beat agentic search by ~20 points on large corpora.



Search & retrieval for agentic systems, mapped. What each method does, what it costs, where it breaks.

23 SLIDES

Retrieval methods do different jobs. Sort them by job before you compare them.

RANKERS score & order candidates	BM25 03-06 Dense 07 Learned sparse 08 Late interaction 09
FILTERS cut the search space	Code indexes 10 Bloom & friends 11 Dedup 12
NAVIGATORS reason their way to the answer	Agentic loops 13 Tree retrieval 14
STRUCTURES precompute relationships	Graphs 15 Agent memory 16
GLUE improves any of the above	Hybrid, rerank, context 17
SKIP IT when retrieval is the wrong move	Full context 18
REFERENCE for coming back to	Master table 19-20 Decision flow 21 Default stack 22 Sources 23

ASK
For the person paying. What it buys you, and when it's worth it.

BUILD
For the person shipping. Mechanism, cost, failure mode, evidence.

The keyword ranker inside Lucene, Elasticsearch, OpenSearch etc.

ASK

Keyword search done properly. No AI model, no GPU, and cheap at any size.

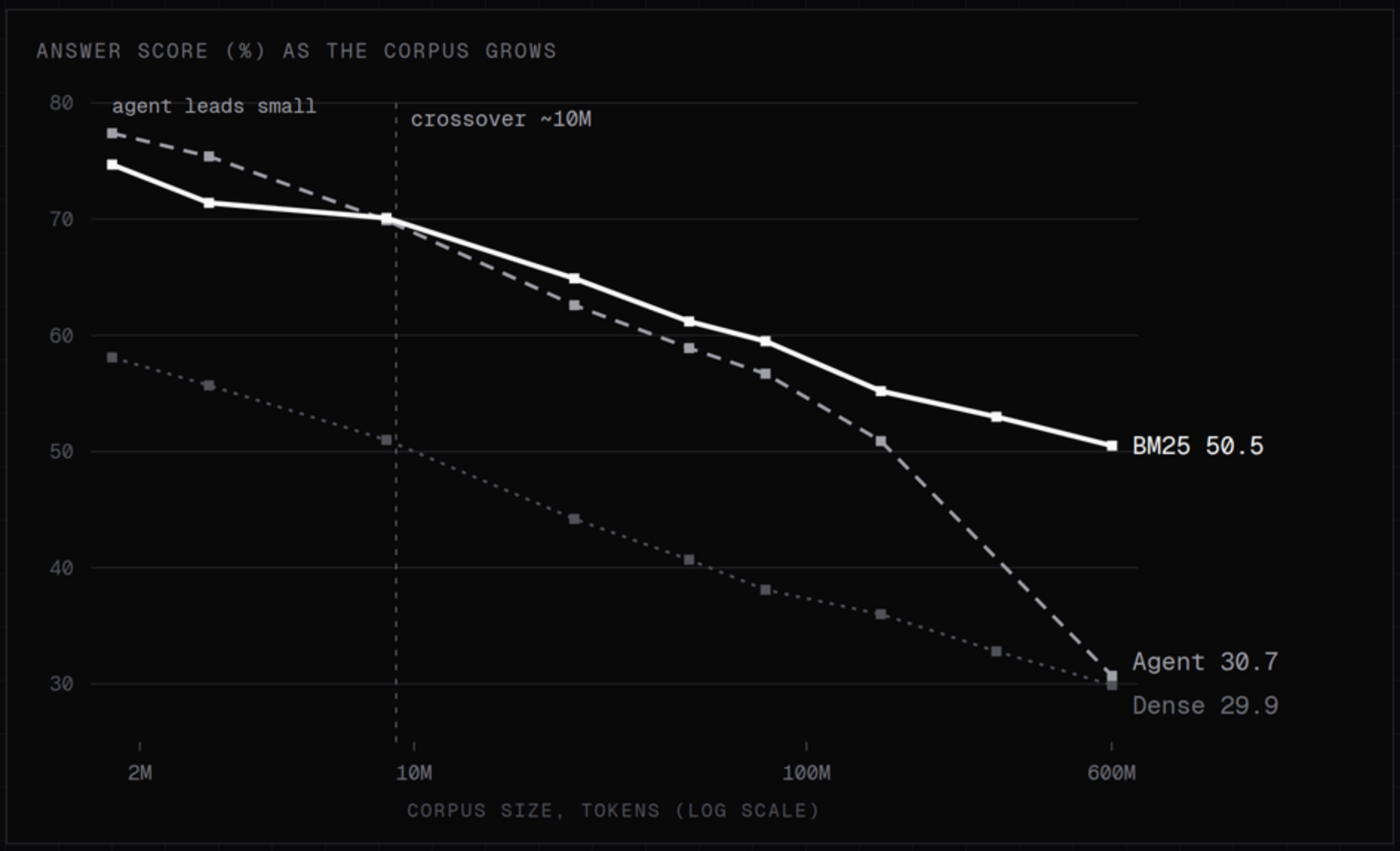
SCORE(D, Q)

$$\Sigma\ \mathrm{IDF}(q)\ \cdot\ \frac{f(q,D)\cdot(k1+1)}{f(q,D)\ +\ k1\cdot(1\ -\ b\ +\ b\cdot|D|/\mathrm{avgdl})}$$

IDF	Rare terms count more. "PLAID" says more than "the".
k1	Saturation. The 10th mention of a term adds less than the 2nd.
b	Length normalisation. A long document doesn't win just by being long.
inverted index	What it runs on: term → list of documents.

INDEX	QUERY	STORAGE
CPU only, zero LLM calls	milliseconds	a fraction of the corpus

Around 10M tokens of corpus, the ranking flips.



ASK

The agent that looks clever in a demo gets more expensive and less accurate as your data grows. At the smallest tier it already costs ~39x the tokens, while it's still winning.

BUILD

Same 500 questions, same reader model, same judge. Only the corpus grew: 28 nested tiers, 1.7M → 601M tokens. Query tokens per question at the smallest tier: BM25 **5.8K**, agent **226K**.

BM25 Wins at Scale · arXiv 2607.26497 · Jul 2026 preprint

Agents write BM25-shaped queries.

A PERSON TYPES	AN AGENT TYPES
`why does login keep failing`	`AuthError token_expired refresh middleware`

Exact terms, identifiers, error strings etc are BM25's home ground.

ASK

If your documents are long (contracts, reports, codebases, ticket histories etc), keyword ranking is a serious contender, and a cheap one.

BUILD

- On a long-conversation benchmark (LongMemEval), grep beat vector search for every harness & model pair tested: widest gap **86.2% vs 62.9%**, narrowest **76.7% vs 75.0%**. Swapping the harness moved accuracy about as much as swapping the retriever.
- Where dense still wins: short documents with the answer near the top. BrowseComp-Plus has exactly that shape and ranks dense above BM25.
- SAGE explains the flip: dense models encode the start of a document well & the rest poorly. Long documents with the evidence spread through them favour BM25.

Is Grep All You Need? · arXiv 2605.15184 / SAGE · arXiv 2602.05975

Before calling BM25 weak, check how it was tuned.

ASK

When someone tells you keyword search didn't work, ask how it was configured.

BUILD

Default k1/b suit short passages. Document-level corpora need document-level length normalisation, and with it BM25 turns competitive on deep-research benchmarks (arXiv 2602.21456).

BM25F

Weight fields: title > body

BM25+

Removes the penalty on very long documents

BMX

Entropy-weighted. Beat BM25 on 11 of 15 BEIR datasets, no training

BM25S

Same scoring, much faster implementation

FINE PRINT ON SLIDE 04

The corpus is enterprise text whose questions carry exact anchors, the traps are semantically close but factually wrong, and the dense side used a small (0.6B) embedder. All of that tilts towards lexical.

BREAKS WHEN

The query and the document use different words for the same thing ("cancel" vs "terminate subscription"). Vocabulary mismatch is the problem the next three slides exist to solve.

Search by meaning. The default most teams start with.

ASK	Finds text that means the same thing in different words. Needs an embedding model & a vector database.
BUILD	One vector per chunk, nearest-neighbour search (HNSW, IVF etc).

INDEX	QUERY	STORAGE
embedding pass over the corpus	one embedding call + ANN lookup, milliseconds	dims × chunks × 4 bytes before compression, plus the ANN graph

WINS	BREAKS
Paraphrase, conversational queries, short documents with front-loaded answers, cross-lingual search.	Exact identifiers (SKUs, error codes, names), long documents (one vector blurs the detail), large corpora (29.9 at full scale on slide 04).

> BEIR · arXiv 2104.08663 · dense models trained on one domain often fell below BM25 on others

Keyword search that knows synonyms.

ASK

Fixes BM25's synonym problem on the search engine you may already run. No separate vector database.

BUILD

A model writes weighted expansion terms into each document (a "car" document also gets "vehicle"), stored in the same inverted index BM25 uses. Inference-free variants expand documents only; the query side is a tokeniser lookup.

INDEX	QUERY	STORAGE
model pass over the corpus (GPU helps)	~1.1x BM25 latency, inference-free model	larger postings than BM25

WINS	BREAKS
OpenSearch neural sparse, Elasticsearch ELSER, Vespa, Qdrant sparse vectors etc already support it.	Query-side SPLADE puts a model on your hot path; expanded postings slow queries at scale.

> SPLADEv2 ~9.6% above BM25 on BEIR · arXiv 2307.10488
> Inference-free model +3.3 NDCG@10 over the prior best · arXiv 2411.04403

The most accurate neural option, with the biggest memory bill.

ASK

Worth it when precision matters more than infrastructure cost, or when your PDFs are full of tables & charts.

BUILD

One vector per token. Score = for each query token, its best match in the document, summed (MaxSim).

ColBERTv2	Compresses the vectors.
PLAID	7x faster on GPU, 45x on CPU than vanilla ColBERTv2.
MUVERA	Folds multi-vector into single fixed vectors for a standard ANN index. ~10% better recall, ~90% lower latency than PLAID on BEIR, 32x smaller with PQ.
ColPali	Runs it on page images, no OCR.

INDEX	QUERY	STORAGE
encode every token (GPU)	tens of ms GPU, hundreds CPU (PLAID, 140M passages)	the big one, even compressed

BREAKS

Memory budget, and fewer engines support it natively. Often deployed as a reranker over a cheaper first stage.

For code, exact search usually wins. The real question is when to index.

ASK

Coding agents grep constantly. On a small repo that's instant; on a huge monorepo every search turns into a wait.

ripgrep	No index, scans every file. Always fresh, very hard to beat on small & medium repos. Cursor cites 15-second waits in large monorepos.
trigram index Zoekt	regex → trigrams → intersect posting lists → run the real regex on the few candidates. Sub-50ms on large codebases, ~3-3.5x the corpus on disk.
suffix array livegrep	Arbitrary regex, memory-heavy.
sparse n-grams GitHub, Cursor, ClickHouse	Variable-length grams chosen by rarity. Cursor builds its index locally from git commits.

RULE OF THUMB

`rg` until latency stalls the agent, then index.

A Bloom filter answers one question: definitely not here, or maybe here.

ASK

It makes retrieval cheaper by skipping work. It never decides relevance, so if someone pitches one "to improve relevance", ask what ranks the results.

BIT ARRAY · K HASH FUNCTIONS



`all k bits set`
maybe here (false positives possible)

`any bit clear`
definitely not here (never wrong)

A few bits per key. Relatives: cuckoo (supports delete), xor / binary fuse (static, smaller & faster), ribbon (RocksDB) etc.

WHERE IT EARNS A PLACE

SKIP

Shards, segments & files that can't hold the key (LSM databases, Parquet)

PRUNE

GitHub & Cursor keep two tiny masks per posting (~2 bytes) so a trigram index answers like a quadgram index

AGENT LOOPS

A cheap "already fetched this URL / doc" check

DEDUP

At scale, see the next slide

Duplicates quietly eat your top-k.

ASK

Five near-identical chunks can fill five of your top results with one answer. Cleaning the corpus before indexing is cheap, and rarely measured.

MinHash	Estimates word overlap (Jaccard) between shingled documents.
LSH	Buckets likely pairs, so you don't compare everything with everything.
SimHash	Fingerprints near-duplicates at web scale.
LSHBloom	Swaps the LSH index for Bloom filters.

Catches reordered boilerplate, changed footers, template pages etc.

LSHBLOOM VS MINHASHLSH

0.6%

of the disk, on 39M documents

~3x

faster

BREAKS

It measures shared words only. Two documents saying the same thing in different words both survive.

LSHBloom · arXiv 2411.04257

Letting the model search the way a person would.

ASK

Good on small, messy data and on code the agent is editing. Cost grows with every step, and accuracy drops as the corpus grows.

BUILD

- The LLM issues grep / keyword searches, reads, refines, repeats (ReAct-style). No index, so all the cost lands at query time.
- **39x** BM25's query tokens at the smallest tier, **60x** a few tiers later. At full scale, **31%** of questions ran out of the 80-call budget.

THE FIX THE SAME STUDY TESTED: GIVE THE AGENT A BM25 TOOL IN PLACE OF RAW GREP

AGENT + RAW GREP	AGENT + BM25 TOOL
36.9	**69.4**
895K tokens / question	101K tokens / question (~1/9)

150-question full-scale subset. It also beat plain BM25 (54.8) on that subset.

Ranked discovery first, then let the agent reason over the shortlist.

For long structured documents, read the table of contents first.

ASK

10-Ks, contracts, manuals etc. A model navigates the document like an analyst would. Answers trace back to a section, and each question costs one or more LLM calls.

```
10-K  (root)
├── Item 1  Business
├── Item 7  MD&A            <- LLM picks this node
│   ├── Results of operations  p.41-48
│   └── Liquidity              p.49-53  <- then this
└── Item 8  Financial statements
```

PageIndex	Builds a tree (title, summary, page range per node); at query time an LLM picks the nodes. No embeddings, no chunking.
RAPTOR	Clusters & summarises chunks recursively, then retrieves at the right level of detail.

INDEX	QUERY	SCOPE
LLM pass per document	LLM navigation per question	inside a document. Pick documents first with BM25 or vectors

> PageIndex reports 98.7% on FinanceBench. Vendor-reported (system "Mafin 2.5"), so reproduce it on your own documents before trusting it.

Graphs answer questions about the whole collection. The build is the bill.

ASK

"What keeps coming up across all our incident reports" is a graph question, and chunk search can't see it. Most other questions don't need a graph.

BUILD

GraphRAG: an LLM extracts entities & relations, clusters them into communities, summarises each.

WHERE THE BUILDS STOPPED, 2026 SCALING STUDY (FULL CORPUS 512K DOCS)

METHOD	BUILT UP TO	PROJECTED FULL BUILD
MS-GraphRAG	8,750 docs	~50 instance-days
LightRAG	2,254 docs	~4 instance-years
HippoRAG 2	131,876 docs	scored ~15 points below BM25 there

LazyGraphRAG	Defers summarisation to query time. Indexing cost equal to vector RAG and 0.1% of full GraphRAG, 700x+ lower query cost than GraphRAG global search (Microsoft's own benchmark).
HippoRAG 2	Entity graph + Personalized PageRank, multi-hop in one step.

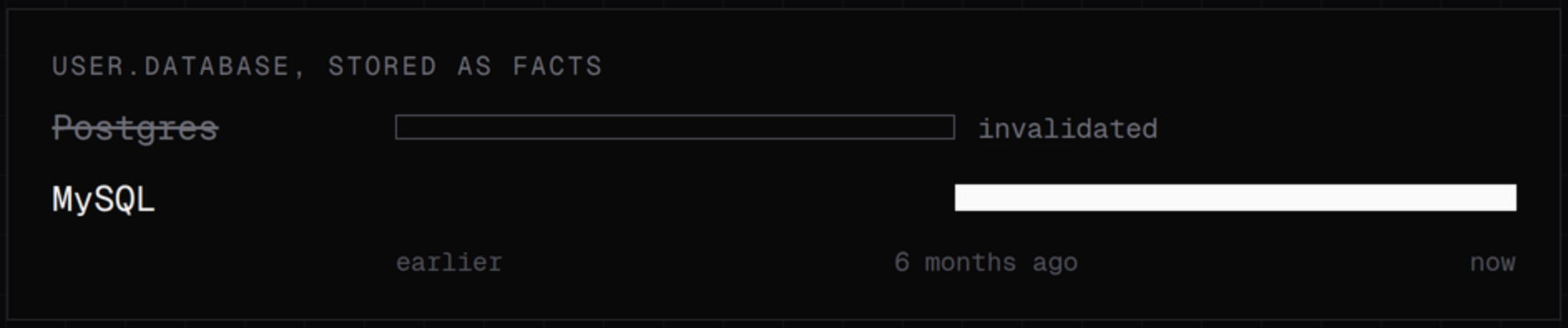
BREAKS

Extraction errors become wrong edges, and neighbourhoods pull in facts that are related but wrong.

Agent memory is a different retrieval problem. Facts expire.

ASK

A user who moved from Postgres to MySQL six months ago shouldn't get Postgres advice. Plain vector recall returns both facts, with no sense of which one is current.



Zep / Graphiti	Temporal knowledge graph. Each fact carries valid-from / valid-to, and old facts get invalidated rather than deleted.
Mem0	Extracts facts, retrieves by semantic + keyword + entity in parallel.
Letta (MemGPT)	Tiered memory the agent edits itself.
grep	Over raw history, beat vector retrieval on LongMemEval in matched harness tests (slide 05).

> Benchmarks: LoCoMo, LongMemEval, BEAM. Vendor scores conflict (Mem0 & Zep dispute each other's LoCoMo setups), so read every score next to its token cost & eval setup.

The upgrades that sit on top of any retriever.

ASK

Hybrid search plus a reranker is the safest general-purpose default. Each piece should still earn its place in your own evals.

`Hybrid + RRF`	Run BM25 & dense, fuse by rank: `Σ 1/(k + rank), k ~ 60`. No score normalisation needed (BM25 and cosine scores don't share a scale).
`Rerank`	A cross-encoder rescores the top 50-200, adding ~0.5-2s. Test on your domain; web-trained rerankers have lowered scores on technical corpora (arXiv 2604.15484).
`Contextual retrieval`	An LLM writes a line or two of context onto each chunk before it's indexed in both BM25 & embeddings. ~$1.02 per million document tokens, once, with prompt caching.
`Query rewriting / HyDE`	Rewrite the question, or embed a hypothetical answer in place of the question.

CONTEXTUAL RETRIEVAL (ANTHROPIC): TOP-20 RETRIEVAL FAILURES



Under ~200k tokens, you may not need retrieval at all.

KNOWLEDGE BASE

~200k

tokens

ROUGHLY

~500

pages

ASK

If the whole knowledge base is ~500 pages or less, put all of it in the prompt. Nothing gets missed by a bad search, and there's no pipeline to maintain.

BUILD

This is Anthropic's own guidance for knowledge bases under ~200k tokens. Prompt caching cuts latency by more than 2x & cost by up to 90% on repeated context.

REVISIT WHEN

The corpus outgrows the window, or changes faster than the cache pays off.

Master table, part 1: rankers & filters.

METHOD	INDEX COST	QUERY COST	STORAGE	BEST FOR	BREAKS ON
BM25	CPU, no LLM	ms	small	IDs, jargon, long docs, scale	synonyms
Dense	embedding pass	embed call + ms	vectors + ANN graph	paraphrase, short docs	exact tokens, long docs
Learned sparse	model pass	~1.1x BM25 (inference-free)	larger postings	BM25 + synonyms	query-side model latency
Late interaction	encode every token	tens-hundreds of ms	largest	precision, visual PDFs	memory
Trigram / n-gram	index build	<50ms	~3-3.5x corpus	code, logs, regex	meaning
Bloom & friends	tiny	O(1)	bits per key	skipping, pruning, seen checks	can't rank
MinHash / LSH	batch job	batch job	small	dedup before indexing	paraphrased duplicates

Costs are relative, from the sources on slide 23. Measure on your own corpus.

Master table, part 2: navigators, structures & glue.

METHOD	INDEX COST	QUERY COST	STORAGE	BEST FOR	BREAKS ON
Agentic loop	none	39-60x BM25 tokens	none	small corpora, live code	scale
Tree (PageIndex, RAPTOR)	LLM per document	LLM per question	tree	long structured docs	cross-document search
GraphRAG	very high LLM	moderate-high	graph + summaries	whole-collection themes	build cost at scale
LazyGraphRAG	~vector RAG	LLM at query time	light	the same, cheaper	query-time LLM work
Temporal memory graph	LLM per update	graph query	graph	facts that change	build cost, contested benchmarks
Hybrid + RRF	sum of parts	+ fusion	sum of parts	general default	weak components
Reranker	none	+0.5-2s	model	precision on top-k	off-domain training
Full context	none	tokens per call (cached)	none	under ~200k tokens	scale

Costs are relative, from the sources on slide 23. Measure on your own corpus.

Pick by the shape of your data, then prove it with evals.

CHECK IN ORDER. FIRST YES WINS.

1	Under ~200k tokens?	yes	Full context + prompt caching
2	Code?	yes	ripgrep, then a trigram / sparse n-gram index once latency stalls
3	Agent memory across sessions?	yes	Temporal graph (Zep / Graphiti) or multi-signal (Mem0)
4	Questions about the whole collection?	yes	LazyGraphRAG or HippoRAG 2
5	Long structured documents that need traced answers?	yes	BM25 picks the document, tree retrieval reads it
6	PDFs full of tables & charts?	yes	ColPali

7 · EVERYTHING ELSE

tuned BM25	+ dense or SPLADE via RRF if evals show vocabulary misses	+ rerank if evals show it helps	+ ColBERT only if precision is still short & memory allows

ALWAYS **Dedup at ingestion.**

The default stack the evidence supports.

1. **dedup** MinHash
2. **contextualise chunks**
3. **BM25** + dense / SPLADE via RRF, if evals ask for it
4. **rerank** top ~50
5. **agent reasons** over the shortlist

ASK

Every box after BM25 costs money or latency. Add it when your evals say so.

BUILD

Build ~50-100 real queries with known answers before choosing anything. Corpus size, document shape & query style pick the winner; benchmarks only tell you where to look.

Measure before you add a layer.

Sources.

ARXIV · ARXIV.ORG/ABS/ + ID

BM25 Wins at Scale	2607.26497	2026 preprint
Is Grep All You Need?	2605.15184	2026 preprint
SAGE	2602.05975	2026 preprint
BM25 tuning for deep research	2602.21456	2026 preprint
BEIR	2104.08663	
BMX	2408.06643	
SPRINT	2307.10488	
Inference-free learned sparse	2411.04403	
PLAID	2205.09707	
MUVERA	2405.19504	
ColPali	2407.01449	
LSHBloom	2411.04257	
Rerankers on technical text	2604.15484	2026 preprint
HippoRAG 2	2502.14802	
RAPTOR	2401.18059	

OTHER

Anthropic	Introducing Contextual Retrieval (2024)
Microsoft Research	LazyGraphRAG (2024)
Cursor	Fast regex search (2026)
VectifyAI	PageIndex (vendor-reported figures)