QUBER — RETRIEVAL PIPELINE REFERENCE

From Document to Answer

Extraction, cleanup, chunking, embedding, and hybrid search

This reference describes how a financial PDF becomes a set of searchable records and how a question retrieves the right one. It covers the two extraction engines and their fusion, the corrections applied before anything is stored, the design of the searchable units, the embedding rules, and the hybrid ranking that serves the answering agent. It assumes the reader knows what a RAG system is and nothing else about this codebase.

Engineering Reference · Generated July 2026

Source of truth: `src/quber/core/` · `experiments/ade_rag_playground/`

Contents

Audience and Scope

Written for an engineer joining the project. Every symbol named here exists in the codebase, and every measurement was taken on the live system against the six-document evaluation corpus: two Visa releases, a T-Mobile factbook, a Verizon information package, an Arbor Realty 10-Q, and a 160-page 3M 10-K.

1 The Pipeline at a Glance

The pipeline turns one PDF into artifacts — the unified document, the corrected tables, the match report, and the review flags — then into database records. Two engines read the PDF independently: a document engine that produces the full text in reading order, and a table engine that produces corrected tables. A fusion step reconciles the two into a unified document, corrects mislabeled headings, and writes the artifacts. Ingestion converts the artifacts into searchable chunks and citable groundings in Postgres. At question time, two rankings, vector and keyword, are fused to select the records the answering agent reads.

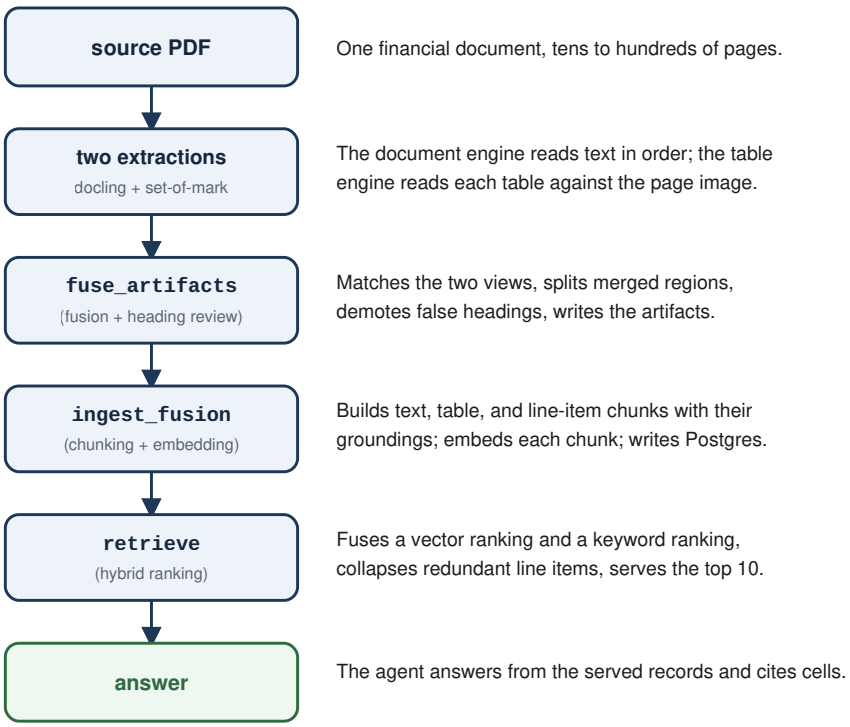

Figure 1 — The pipeline. Every stage below is one section of this document.

> *A retrieval unit must be able to answer a question.*
>
> — the design rule behind every chunking decision in section 4

2 Document Extraction

Two engines read the same PDF because neither is sufficient alone. The document engine reads everything in reading order but misreads table structure. The table engine reads table structure against the page image but knows nothing outside its regions. Each engine supplies what the other lacks.

2.1 The document engine: docling

Docling parses the PDF into typed items: text with a label (section header, paragraph, footnote, page header), tables, and pictures. Every item has provenance: its page number and its printed position on the page. The configuration is the `tuned-financial` preset (`src/quber/core/parsers/docling_parser.py`):

- TableFormer in ACCURATE mode
- the DoclingParse backend
- OCR
- picture classification
- CUDA
- a page batch size of 32

The output is the document's spine: the ordered sequence of everything printed.

2.2 The table engine: set-of-mark

The table engine treats the page image as the authority on structure and the extracted text as the authority on values. It runs in three steps.

- **Locate.** A grid locator names every table on the page and its region. Camelot, a deterministic grid extractor, then reads each region separately, so it can neither split one table into many nor merge many into one.
- **Correct.** A correction agent receives the cropped table image and Camelot's markdown, in which every cell is tagged with a printed address (`[B3] 1,637`) so the agent can name cells, and returns the corrected table plus its metadata: title, subtitle, units caption, footnotes, footnote markers, the header depth (section 3.2), and a record of every cell it merged or moved (`LLMTableCorrection`, temperature 0).
- **Guard.** A grounding guard rejects any correction containing a number absent from both Camelot's grid and the page text layer. A rejected correction is discarded whole,

and the deterministic grid render is served instead. The engine never emits an invented figure.

The result is one `ExtractedTable` per visual table, holding the corrected markdown and a `corrected_grid`: one cell record per output cell with its text, its printed position when it was traced to one, a provenance status, and an inspector's note where one was taken. Cells that could not be traced are still served, and their status says so.

2.3 Fusion

Fusion (`fuse_artifacts`, `src/quber/core/fusion/fusion.py`) reconciles the two views and writes the artifacts of record:

- **Match.** Each set-of-mark table is matched to its docling counterpart by region overlap. The match list is itself an artifact (`<base>.fusion.json`); ingestion later uses it to attach section headings to tables.
- **Split.** When docling found more tables than the table engine in one region, the merged region is split and each piece is re-extracted and re-corrected against its own crop.
- **Graft.** The corrected table bodies replace docling's table readings inside a clone of the docling spine, producing the unified document (`<base>.unified.json`). Tables docling found that the table engine missed are kept from docling's own reading, so their content is never silently absent.
- **Review headings.** The heading review (section 3.3) demotes page decoration mislabeled as section headings.

Every run also writes `<base>.flags.json`: the review queue. Each entry has full document identity. The queue lists:

- cells whose provenance status warrants inspection
- dropped printed text
- any demoted headings

3 Cleanup

Cleanup is every correction applied before anything becomes searchable. The common principle: an agent that is already looking at the evidence reports a structural fact, the fact is checkable, and the safe direction is always the default. Nothing is fixed by silent inference downstream.

3.1 Structure correction

The correction agent fixes what Camelot misread:

- spurious empty columns
- headers split across rows
- headers misaligned from their values
- rows Camelot truncated

The agent may rearrange cells; it may never invent a value. Two checks enforce this. The grounding guard (section 2.2) rejects any correction containing a number found in neither Camelot's grid nor the page text layer. The coordinate-frame check rejects a correction that copied Camelot's printed address tags into the output, where they would corrupt values downstream.

3.2 Header depth

The correction agent reports each table's header depth as `header_rows`: how many leading rows of its corrected table are column headers, counted off the table image. Figure 2 shows why the count is needed: financial tables stack their headers, so the boundary between header and data is not always one row down — and chunking (section 4.3) must know where it is. No numeric pattern finds the boundary reliably. A bare-year header row looks like data; a section-label row looks like a header.

	March 31, 2026		December 31, 2025	
	UPB	Carrying Value	UPB	Carrying Value
Multifamily	$ 479,219	$ 473,919	$ 566,906	$ 553,016
Commercial	1,700	1,700	1,700	1,700

`header_rows = 2` — the header block

data rows — one line record each (§4.3)

Figure 2 — A stacked header, from the Arbor 10-Q. The agent reports `header_rows = 2`; a one-row assumption would tell a reader "$ 473,919" is a "March 31, 2026" value and nothing more.

Three properties make the report trustworthy. The count describes the exact structure the same agent just emitted, in the same response, so it can never describe a structure its author did not produce. Zero is a legal answer: a split piece from section 2.3 whose crop starts at data rows has no header of its own, and each split piece reports the depth of its own crop. And there is no fallback inference — an artifact extracted before this field existed fails ingestion with instructions to re-extract.

> **WHY THE AGENT AND NOT A RULE**
>
> On a fresh extraction of the Arbor 10-Q, the agent and a numeric-pattern rule disagreed on 23 of 64 tables. Adjudicated against the printed pages, the agent was right in every sampled case, in both error directions: the rule counted "Assets:" section labels as header rows, and it mistook the bare-year header "Description | 2026 | 2025" for data.

3.3 The heading review

Docling sometimes labels page decoration as a section heading. Page decoration is text the page layout repeats — running headers, running footers, page numbers — rather than text the author wrote at that place in the document. The Arbor 10-Q prints its document banner at the top of nearly every notes page. Docling called it a heading on 40 pages, and chunking — which prefixes every chunk with its governing heading (section 4.1) — put it onto 112 of the document's 445 text chunks. The same defect put T-Mobile's running headers on its chunks. False section context at that scale pushes the correct records out of the ranking (section 6.3) and misstates the document's structure.

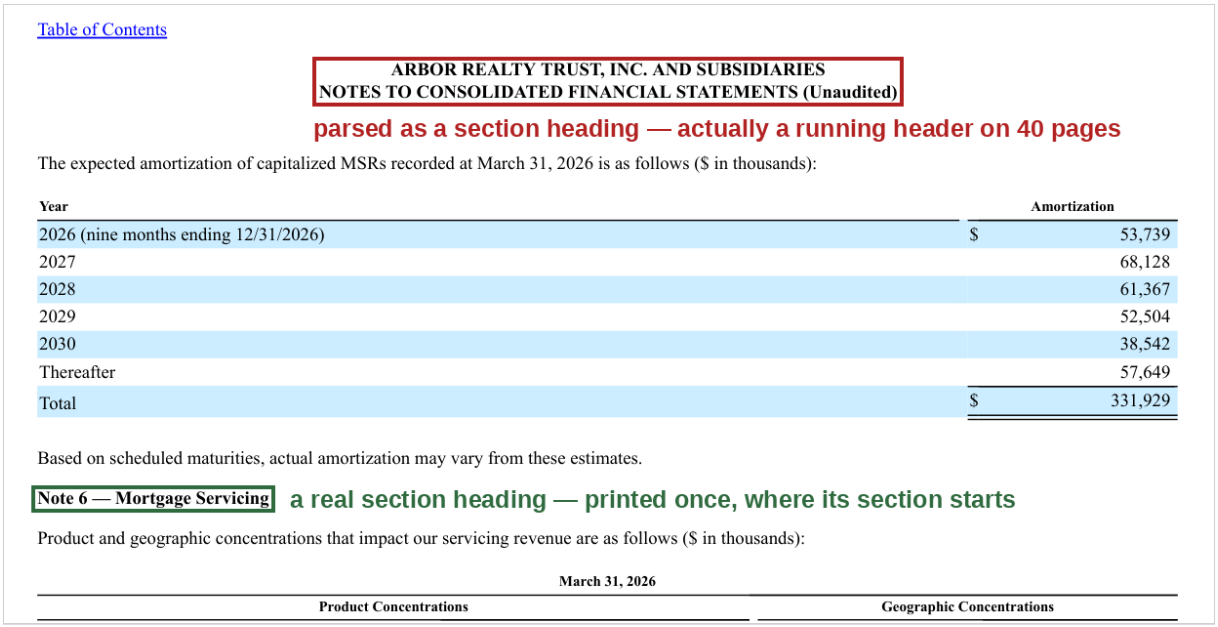

Table of Contents

ARBOR REALTY TRUST, INC. AND SUBSIDIARIES
NOTES TO CONSOLIDATED FINANCIAL STATEMENTS (Unaudited)

parsed as a section heading — actually a running header on 40 pages

The expected amortization of capitalized MSRs recorded at March 31, 2026 is as follows ($ in thousands):

Year	Amortization
2026 (nine months ending 12/31/2026)	$ 53,739
2027	68,128
2028	61,367
2029	52,504
2030	38,542
Thereafter	57,649
Total	$ 331,929

Based on scheduled maturities, actual amortization may vary from these estimates.

Note 6 — Mortgage Servicing a real section heading — printed once, where its section starts

Product and geographic concentrations that impact our servicing revenue are as follows ($ in thousands):

	March 31, 2026	
Product Concentrations		Geographic Concentrations

Figure 3 — Arbor 10-Q, page 20. The parser labeled both marked lines section headings. The banner (red) is a running header repeated on 40 pages; "Note 6" (green) is a real heading, printed once where its section starts.

The review (`src/quber/core/fusion/heading_review.py`) demotes such headings under two independent conditions, both required:

- **Geometry nominates.** The same text occurs as a heading on three or more pages (`NOMINATION_PAGES = 3`). This is computed from the parse, not asserted by a model. A heading printed once can never be demoted, whatever any verdict says.
- **The agent confirms.** One call per document judges only the nominated texts, each presented with its occurrence count, page list, and printed position. It returns one of four verdicts per heading: keep, running header, column label, or other non-heading. Keep is the default when unsure.

Position is the decisive evidence. A running header sits at a fixed position at the very top of its pages. An authored sub-heading that legitimately repeats sits lower, at varying positions, wherever its section starts: 3M prints "Year 2018 results:" in every business-segment discussion. The first version of the review lacked position evidence and wrongly demoted exactly those 3M sub-headings. Adding position fixed the verdicts, and the mistake was caught immediately because every demotion is a flag, never a silent edit.

A demotion relabels the heading items to `page_header` on the unified document and writes a flag record: the text, its pages, the verdict and reason, and how many text items it governed. No chunk loses any of its own words, only the false prefix. Re-running fusion reverses everything.

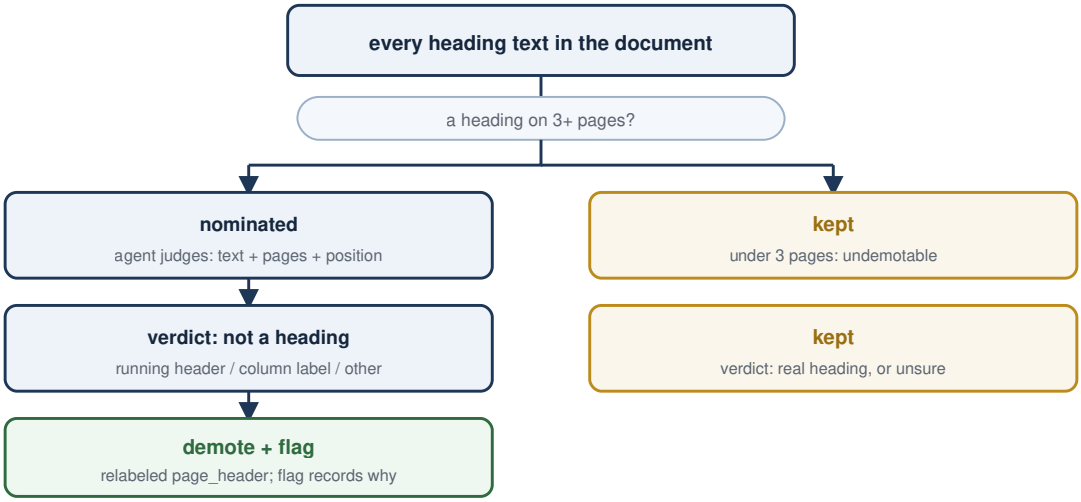

Figure 4 — Heading demotion. Both conditions must hold; every exit that keeps a heading is the default path.

4 Chunking

Chunking decides what a question can find. Ingestion (`ingest_fusion.py`) cuts the fused document into chunks: the pieces of text that get embedded, keyword-indexed, ranked, and read by the answering agent. Three kinds of chunk are stored:

- a paragraph, prefixed with its section heading (4.1)
- a whole table (4.2)
- a single printed table line (4.3)

Cell-level citation is a separate record kind, the grounding (4.4). Page decoration is excluded entirely.

One rule shaped all three kinds: a chunk must hold enough of the page to answer a question by itself. A section heading fails that test — "Liquidity and Capital Resources" names a topic and answers nothing — so a heading is never stored as its own chunk. Its text is prefixed onto every chunk beneath it, where it helps the right content match the questions that name the section.

4.1 Text chunks

One chunk per paragraph, prefixed with the section heading in effect where the paragraph sits: `"Fiscal First Quarter 2026 - Financial Highlights — Payments volume for the three months…"`. The prefix is skipped when the paragraph already contains the heading's words.

The prefix exists because questions name sections. A reader asks what the Financial Highlights section says about payments volume; the paragraph that answers speaks of payments volume but never names its own section. The prefix puts the words the question will use on the chunk that can answer it. The alternative — storing headings as chunks of their own — inverts the match: the heading fits the question perfectly, answers nothing, and outranks the content beneath it.

4.2 Table chunks

One chunk per extracted table, stored under the table's own id (`#/tables/12` is the document's thirteenth table, counting from zero): the governing section heading, the table's title, subtitle, and units caption, then the full grid as HTML with an id on every cell (`t12-2-1` = table 12, row 2, column 1). The agent cites those ids, and the application resolves them to page overlays. The section heading is attached through the fusion report's

match: the heading in effect where the table's docling counterpart sits in reading order. Tables need the prefix most: a printed table title is often just a date line, so the words a question would use appear only in the section heading above the table. Visa's non-GAAP reconciliation is the standing example — its printed title is "Three Months Ended December 31, 2025", and every word connecting it to a non-GAAP question comes from its section heading.

4.3 Line-item records

One embedding cannot represent forty printed lines. The measured failure: a question about one line of a large table ranked the table 14th–28th, behind prose, because the line's signal was diluted across the whole table's embedding. So every printed data line also becomes its own record, built to be answerable alone:

- the section heading, title, subtitle, and units — the same prefix the table chunk has;
- the full header block: every leading header row, sliced by the agent-reported `header_rows` of section 3.2, so a two-tier header keeps both tiers;
- the row-axis label governing the line ("Business Segments", "Revenues:") — a row whose only text is in its first column is a label, and it is prefixed onto the lines beneath it;
- the line itself, with its real cell ids, so a citation from a line record resolves exactly like one from the table chunk.

Header rows and label rows produce no records of their own: a record whose entire content is header words is retrieval noise that looks like data. Each line record holds its own page box, the union of its cells' boxes.

A table is therefore stored twice over: once whole, once per line. Both granularities are kept because a question's shape is unknown until it is asked — "what were preferred stock dividends" wants one line, "what does the non-performing loans table show" wants the whole table. In execution terms the redundancy is literal: the same printed cells sit in the index twice, in the table chunk and in its line records, each with its own embedding, and all of them compete in the same ranking. The records roughly triple a document's searchable units — the Arbor 10-Q ingests as 445 text chunks, 66 table chunks, and 727 line-item records — and a table-shaped question can match a dozen sibling records at once. Keeping a table and its own lines from filling the results together is a ranking problem, deferred to query time (section 6.2). Ingestion's part is one field: every line record of table 12 (their ids run `t12-line-2`, `t12-line-3`, …) stores the table chunk's id, `#/tables/12`, in `parent_chunk_id`, and the ranking groups a table with its lines by that field.

4.4 Groundings

A grounding is one citable location: a page, a box, and — for a table cell — the cell's provenance status and inspector note. Search and citation operate at different granularities by design: a table is one chunk with one embedding, but every one of its cells has its own grounding. When the agent cites `t12-2-1`, the application draws that cell's box on the page and shows its status. Every chunk is also its own grounding, so prose citations resolve the same way.

Table	One row per	What it holds
documents	ingested PDF	slug, filename, page count
chunks	searchable unit	its id, content, 1024-dim embedding, generated keyword index (`content_tsv`), page, box, `parent_chunk_id`
groundings	citable location	page, box, cell position, provenance status, inspector note, cell text

5 Embedding

Every chunk is embedded with `BAAI/bge-large-en-v1.5` (1024 dimensions), batched 32 at a time on the GPU, and stored in a pgvector column. Questions are embedded with the same weights at query time, so ingest and query share one vector space.

The model embeds what a reader reads, not what the database stores. Stored table and line-item content is HTML, because the cell ids must survive for citation. On a short line record, though, the markup outweighs the words. Ingestion therefore embeds a tag-stripped rendition (`_embed_view`): tags dropped, cells separated by a delimiter. This one change moved failing line records into the served context during evaluation. The stored HTML is untouched.

> **One space, one model.** Chunk embeddings and question embeddings must come from the same weights. A future serverless embedding endpoint loads the same model for ingest batches; question embedding stays local to the API process.

6 Hybrid Search

One ranking is not enough. Vector similarity matches paraphrased questions but cannot tell near-synonyms apart. Keyword matching is exact but fails when the question's wording differs from the document's. The retriever (`retrieve` in `retrieval.py`) runs both over the same chunks and fuses them.

6.1 The two rankings

- **Vector.** pgvector cosine distance between the question embedding and every chunk embedding; the top 200 form the vector list.
- **Keyword.** The question's lexemes are OR-joined. Requiring every word fails whole questions, because a single filler word missing from a chunk rules it out. Chunks matching any lexeme are ranked by `ts_rank_cd` with log-length normalization, and the top 200 form the keyword list. The normalization is required: without it, a subsidiary list repeating the company name hundreds of times outranked a short record that matched every query word once.

6.2 Fusion and the collapse rule

The two lists are fused by reciprocal rank: a chunk's score is `1/(20+rank)` summed over the lists it appears in (`RRF_K = 20`, grid-tested against a real failing question). A question whose keywords match nothing degrades to the pure vector ranking. The retriever takes a candidate list three times deeper than the requested ten, because the collapse rule may free slots.

The collapse rule removes redundancy that line-item records create. When a table and its own lines are retrieved together, or three or more lines of one table appear (`COLLAPSE_AT = 3`), the question is about the table, so the whole table is served once, its line records are dropped, and the freed slots refill from the candidates below. The grouping key is the `parent_chunk_id` every line record holds.

6.3 Evidence and open work

The standing evaluation is eleven questions across the corpus, each with a known printed answer. All eleven pass on the current pipeline, including the three that motivated line-item records — single values inside large tables of large documents:

Question	Answer served
3M Health Care segment operating income, FY 2018	$1,799M
3M Industrial segment operating income, Q4 2018	$627M
T-Mobile revised 2025 postpaid guidance	6.1–6.4 million
Arbor bridge loans, average carrying value Q1 2026	$11,281,488K
Arbor bridge loans, weighted average yield Q1 2026	7.35%
Visa non-GAAP operating expenses	$3,391M

Six of the eleven standing questions; the remainder cover dividends, servicing balances, free cash flow, and section-naming phrasings.

One suite question passes only because of the heading review. Before it, the question's entire served context was prose prefixed with the Arbor banner, and the correct record ranked outside the top 100 under every phrasing tried. Demoting that one banner, and nothing else, fixed both failing phrasings.

OPEN · SAME-VOCABULARY COMPETITION

When two tables share a question's words, the wrong table's line records can occupy many candidate slots at once: a fair-value table also lists "Bridge loans" beside "Carrying Value". The failure is measured; these three fixes are proposals, none yet built:

- cap candidates per parent table before fusion
- weight the embedded view so a long title cannot outweigh header rows
- an agent pass that selects from a diversified candidate pool