

ENTERPRISE RAG INGESTION

Structure-Preserving Chunking and Hierarchical Summarization

for Enterprise Retrieval-Augmented Generation

AUTHOR

Samiran Dolui

PUBLISHED

September 2026

SERIES

Corporate GPT White Paper Series

ABSTRACT

Enterprise corpora are dominated by content that is not prose: spreadsheets, slide decks, scanned documents, transcripts, source code and figures. We describe how Corporate GPT ingests them: fifteen format families are normalised into one structured document representation, chunked by tokens along document structure, and summarised per document by token-bounded map-reduce. We identify six ways a structure-aware chunker still loses information (flattened nested lists, unbounded tables, broken code fences, redundant spreadsheet serialisation, lost transcript timing and text trapped in scanned pages) and give

the root cause and repair for each. On a multi-format regression corpus processed offline, the repairs split a 118,448-token table into 62 header-bearing parts, keep all 45 chunks of four source files well-formed where the stock chunker broke every one, cut the tokens indexed for spreadsheets by 59%, timestamp every transcript chunk, and index the text of a scanned page that the stock chunker leaves out. We state plainly where the implementation still falls short of the design.

62

header-bearing
parts from a
118,448-token
table

45/45

source-file
chunks well-
formed

59%

fewer tokens
indexed for
spreadsheets

15

format families,
one
representation

01

Introduction

Most retrieval-augmented generation (RAG) systems [8] treat chunking as preprocessing: a step that turns documents into strings of roughly the right length before the interesting work begins. We argue the opposite. The chunker decides what the index can know. A table row separated from its header cannot be answered; a clause flattened out of its list loses the condition it depended on; a chart that reaches the index as “a bar chart” cannot report its values. No embedding model, reranker or prompt recovers what the chunker discarded.

The chunker decides **what the index can know**.

The problem is sharpest in enterprise corpora, which mix spreadsheets and slide decks, scanned PDFs and legacy Office files, transcripts, source code and diagrams. Each format fails in its own way when forced through a fixed-window splitter. We organise the paper around three ways a pipeline loses information before retrieval begins:

1 • Section 4

Structural amnesia

The serialisation of a nested list, a table, a code block or a transcript drops the structure that carries its meaning.

2 • Section 5

Modality collapse

Figures, charts, drawings and scanned pages carry meaning that never becomes text unless a model produces it.

3 • Section 6

Retrieval myopia

Chunk search ranks passages, so it cannot answer questions about documents: which files cover a topic, or what a report is about.

Contributions. This is an engineering account of Corporate GPT's ingestion pipeline. We contribute an architecture that normalises fifteen format families into one intermediate representation before chunking (Sections 2 and 3); six named failure modes of an off-the-shelf structure-aware chunker, with root causes and repairs that reduce to two general techniques (Sections 4 and 5); per-document summarisation by token-bounded map-reduce into a separate, access-controlled index (Section 6); and an offline structural evaluation of every repair on our regression corpus (Section 7).

Position. We build on layout-aware conversion with Docling [2], whose hybrid chunker respects element boundaries while bounding chunk size. Systematic comparisons show that the chunking strategy alone measurably changes retrieval quality [3, 14], and methods such as late chunking [7] and contextual retrieval [1] restore document context to chunks, but they operate on text that has already been linearised. We address the step before: what happens to a table, a code block or a slide as it becomes text. Our summary tree resembles RAPTOR [13] and recursive book summarisation [15], but it is scoped to one document, built by token-bounded packing rather than clustering, and exposes only its root; unlike query-time expansion such as HyDE [6], the extra text is generated once, at ingestion. Throughout, a *document* is one ingested file, a *chunk* a unit of indexed text, a *node* its stored record (text plus metadata), and a *collection* a named set of points in the vector store.

02

Ingestion architecture

Files arrive by upload or by directory synchronisation (Figure 1). Uploads are validated for size, file type, structure and password protection, and OOXML files against archive-expansion limits that refuse decompression bombs before anything is inflated. On

synchronisation, files whose path, size and modification times are unchanged are skipped. A scheduler dispatches work every 15 seconds, with back-pressure on queue depth.

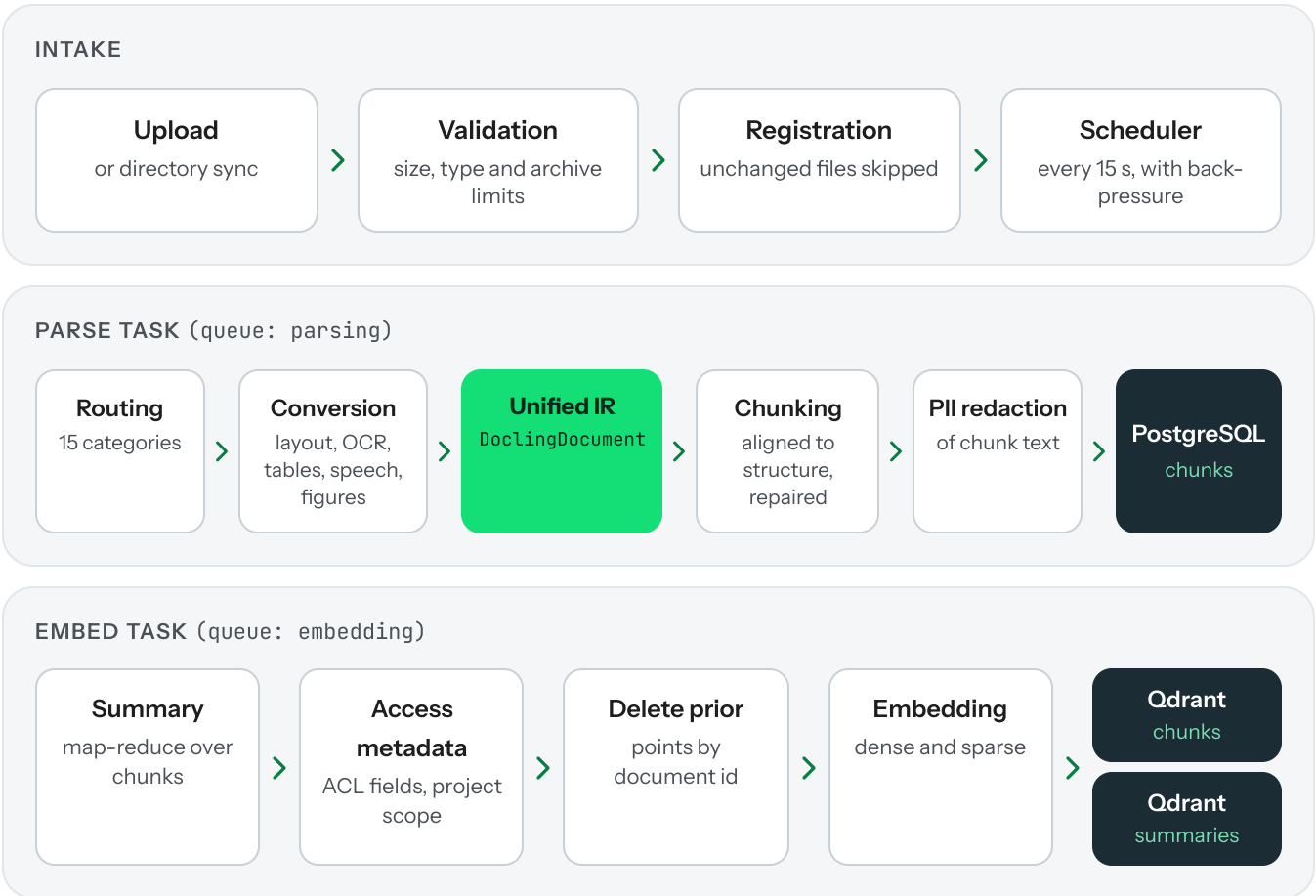


Figure 1. Ingestion pipeline. Every format is normalised into one structured intermediate representation before chunking, so the chunker has a single input contract. Parsing and embedding are separate tasks on separate queues, with the chunks persisted between them.

One intermediate representation. The decision that makes fifteen format families tractable is to normalise all of them into Docling's document model, `DoclingDocument`, before chunking: a tree of typed items (headings, paragraphs, list items within list groups, tables with cell structure, pictures with captions and annotations, code blocks) with page and bounding-box provenance where the format has pages, and a content layer that separates the body from running headers and footers. Converters differ enormously; what they emit is the same, so the chunker has one input contract instead of fifteen. The representation is also where enrichment attaches: a figure description is stored on its picture item, a transcript timestamp in its item's text.

Two asynchronous tasks. Ingestion runs on Celery workers as two tasks on separate queues. The parse task converts, chunks and redacts the file and persists its chunks in PostgreSQL; the embed task summarises the document, attaches access-control and project metadata, and writes dense and sparse vectors and the summary to Qdrant [10]. The split lets the model-bound parse stage and the embedding stage scale independently, and lets an embedding failure be retried from the persisted chunks without re-parsing. Every indexing run first deletes the document's existing points, keyed by a stable document identifier, so retries and re-indexing are idempotent; access-control changes are applied as payload updates to existing points, without re-embedding.

Personal data. With PII protection enabled (the default), chunk text is scanned after chunking with Presidio and spaCy, in English, and a detection is acted on only if its confidence reaches 0.60. That threshold sets the real coverage: card numbers, IBANs, cryptocurrency wallets and e-mail addresses are redacted reliably, and personal names when spaCy detects them; US Social Security and driving-licence numbers only when nearby context, such as the word “SSN”, raises their score; US bank-account and passport numbers score below the threshold and are not redacted; and Indian PAN and Aadhaar numbers, although on the redaction list, have no recogniser enabled. E-mail domains can be allow-listed (none are by default); dates, locations, organisations, URLs and the exact display names of the document's owners are kept. Spans become `[ENTITY_TYPE]` placeholders, and the audit trail stores a keyed hash of each span, never its value. Redaction covers chunk text only: the chunk's metadata (heading path, file name and path, owner fields and access identifiers) is neither scanned nor excluded from the embedded text (Section 8).

REDACTION COVERAGE AT CONFIDENCE 0.60			Presidio and spaCy, English
RELIABLY Card numbers, IBANs, cryptocurrency wallets, e-mail addresses; personal names when spaCy detects them	ONLY WITH CONTEXT US Social Security and driving-licence numbers	NOT REDACTED US bank-account and passport numbers; Indian PAN and Aadhaar (no recogniser enabled)	

03

Format-aware parsing

Routing is by file extension into fifteen categories (Table 1).

PDFs run through Docling's standard pipeline: a layout-analysis model segments each page, TableFormer [9] recovers table structure, and OCR runs *per layout region* (only where there is no text layer or where a region overlaps a bitmap) so digital text is read directly while scans and embedded screenshots are OCR'd. A figure classifier routes charts to a local chart-to-table model, code and formulas are recognised, and pictures covering at least 5% of a page are described by the configured vision-language model (VLM). Legacy `.doc`, `.xls` and `.ppt` files are first converted to OOXML by a LibreOffice daemon and then follow their modern equivalents. Audio is transcribed in-process by Whisper [11]. Video adds key frames described by the VLM, sampled by default every 10 seconds from the start (scene-change sampling is an alternative mode) up to 20 frames, so descriptions cover at most the first 190 seconds; durations are capped so that processing fits the worker's time limit (about six minutes of video at the default one-hour limit). Standalone images go through a VLM page pipeline that transcribes text-bearing images to markdown and describes purely visual ones; SVG files are rasterised first.

Extensions	Category	Conversion	Notable handling
<code>.pdf</code>	<code>pdf</code>	Layout model, table-structure model, region-level OCR	Figure descriptions; chart-to-table; code and formula recognition
<code>.docx</code> <code>.doc</code>	<code>word</code>	Word backend; <code>.doc</code> converted to OOXML first	Embedded pictures described by a VLM
<code>.xlsx</code> <code>.xls</code>	<code>excel</code>	Spreadsheet backend; <code>.xls</code> converted first	Table parts canonical; body triplets dropped; sheet-level stitching at query time
<code>.pptx</code> <code>.ppt</code>	<code>powerpoint</code>	Presentation backend; <code>.ppt</code> converted first	Chart series extracted; slide-level stitching at query time
<code>.csv</code>	<code>csv</code>	CSV backend	Table parts canonical; body triplets dropped
<code>.md</code>	<code>markdown</code>	Markdown backend after preprocessing	Link targets, image alt text and embedded HTML headings inlined
<code>.html</code> <code>.htm</code>	<code>html</code>	HTML backend after boilerplate removal	Flattened-text fallback on failure
<code>.tex</code>	<code>latex</code>	LaTeX backend	Drawing environments described from source

Extensions	Category	Conversion	Notable handling
<code>.vtt</code>	<code>vtt</code>	WebVTT backend	Start times inlined before chunking
<code>.py</code> <code>.ts</code> <code>.tsx</code> <code>.js</code> <code>.java</code> <code>.go</code> <code>.rb</code> <code>.php</code> <code>.sql</code> <code>.vue</code>	<code>text</code>	Built directly: one code item per file	Language-tagged, re-fenced chunks
<code>.txt</code> <code>.json</code> <code>.xml</code> <code>.yaml</code> <code>.yml</code> <code>.toml</code> <code>.ini</code> <code>.cfg</code> <code>.log</code> <code>.rst</code> <code>.css</code> <code>.sh</code> <code>.bat</code> <code>.ps1</code>	<code>text</code>	Built directly: blank-line blocks	
<code>.png</code> <code>.jpg</code> <code>.jpeg</code> <code>.gif</code> <code>.bmp</code> <code>.tiff</code> <code>.webp</code>	<code>image</code>	Vision-language page pipeline (image to markdown)	Separate detailed description node
<code>.svg</code>	<code>svg</code>	Rasterised, then the image pipeline	
<code>.mp3</code> <code>.wav</code>	<code>audio</code>	Speech recognition (Whisper)	Duration-capped
<code>.mp4</code> <code>.mov</code> <code>.avi</code> <code>.mkv</code> <code>.webm</code> <code>.m4v</code>	<code>video</code>	Speech recognition plus sampled key frames	Key frames described by a VLM
other	<code>default</code>	Format detection; only PDFs get the PDF settings, other detected formats use Docling's defaults	Unrecognised files fail without retry; <code>.tif</code> lands here, not in <code>image</code>

Table 1. Supported formats, routing category, conversion path and notable handling.

Failure handling. Failures are handled differently by stage and by format. For Office documents and video frames, a failed or timed-out figure description is recorded on the ingestion attempt and the document is indexed without it; for PDFs, a failed description comes back empty and unrecorded with an OpenAI-compatible model endpoint, while on Bedrock it aborts the parse, which is then retried; for a standalone image, a failed description fails the parse. Summarisation drops failed batches (Section 6), and an HTML page that defeats the HTML backend is flattened and converted again. Docling conversion errors and

PDF-library errors are treated as permanent and are not retried, even when their cause is transient, such as a GPU running out of memory; other failures are retried up to three times. A partial conversion is indexed with its failed pages missing, and a video whose frame sampling fails is indexed from its transcript alone; neither is currently recorded on the attempt (Section 8).

04

Structure-preserving chunking

Docling's hybrid chunker walks the document body and forms chunks from structural elements rather than character offsets: an element larger than the budget is split, and small neighbours that share a heading path are merged. The budget is `CHUNK_SIZE`, 2048 tokens by default, counted with the tokenizer named by `TOKENIZER_MODEL`. Each stored body chunk is prefixed with its heading path (for example, `Onboarding > Access requests`); table parts carry a `Table N (part i/n)` title instead. This handles most prose well. On structured content we found six failure modes. For each we give the root cause and the repair; Figure 2 shows four of them on real inputs.

4.1 Six structural failure modes

01 Flattened nested lists.

When a chunk holds several items, the chunker serialises each on its own and joins them with newlines. The list-group context that records an item's depth is not part of the item, so a three-level list is indexed flat and the dependency between its steps disappears. **Repair:** after chunking, re-walk the document tree for the chunk's items and re-emit each list item indented by its depth relative to the shallowest list in the chunk, with its own marker; ordered items that record no explicit marker, as in Markdown, are numbered by their position in the list, as Docling's own serialiser does (Figure 2a). The repair keeps every item's text: if the chunk also holds a table, code or a formula, if its list items contain formatted text or links, or if it holds only part of an item the chunker split across chunks, the chunker's own text is kept. When it does rebuild, inline markup in the chunk's other items, such as a link's target, is not carried over.

02 Unbounded tables.

Inside body chunks the chunker renders a table as row-wise triplets (`Dulce, Last Name = Abril`), bounded like any other text but stripped of the table's shape. To keep that shape, Corporate GPT also indexes every table as markdown, originally one node per table, whatever its size: a 1000-row spreadsheet became a single node of about 94 KB. **Repair:** split the markdown row-atomically into parts that fit the budget, re-prefix the header row and its separator (with the table's caption, if it has one) onto every part, and title each part `Table N (part i/n)`, with `table_part = i/n` in its metadata. A row that exceeds the budget on its own is kept whole rather than cut mid-cell, and a table whose header alone exceeds the budget is left unsplit. On a real 1066-row account statement, a table of 118,448 tokens (58 times the budget) becomes 62 parts of at most 2040 tokens, each carrying the header (Section 7).

03 Broken code fences.

A source file is one code item, serialised as one fenced block and then split by the budget, so only the first chunk opens the fence and only the last closes it; middle chunks reach the model as prose. **Repair:** position-aware re-fencing. The original opening fence is stripped from the first chunk and the closing fence from the last, and every chunk is wrapped in a fresh fence tagged with the language from the file extension and one backtick longer than any backtick run inside it, so embedded examples cannot close it early. Position, not content, decides what to strip, because content-based stripping would also remove fences that belong to the code.

04 Redundant spreadsheet serialisation.

A spreadsheet or CSV file is nothing but tables, so its triplet body text restates, less legibly, exactly the data the markdown parts carry with their headers: each row keyed by its first cell's value, the first column's header never stated. **Repair:** for these formats the chunker's body output is dropped and the table parts are canonical. In document formats the triplets remain beside the prose that discusses the table, so such tables are indexed twice.

05 Lost transcript timing.

The WebVTT parser keeps each cue's start time on the item's provenance, but chunk text carries none, so a transcript can say what was decided but not when. *Repair:* before chunking, prefix each item with its start time ([00:04] , or [1:02:03] past the first hour). The chunker is unchanged; timing travels through the pipeline as text.

(a) Nested list, Markdown source

STOCK CHUNKER

```
Complete the steps in order.
1. Install the agent
- Download the package
- Verify the checksum
- Run the installer
2. Configure access
- Request a service account
3. Validate the deployment
Escalate failures to the platform team.
```

PRODUCTION

```
Deployment checklist

Complete the steps in order.
1. Install the agent
  - Download the package
    - Verify the checksum
  - Run the installer
2. Configure access
  - Request a service account
3. Validate the deployment
Escalate failures to the platform team.
```

(b) Large table, spreadsheet source

FIXED WINDOW, 1,024/200 TOKENS

```
| 36270 |
| ORD-10299 | South | Bridge B9 | 167 | 25050 |
| ORD-10300 | West | Sensor N7 | 279 | 12555 |
| ORD-10301 | East | Bridge B9 | 57 | 8550 |
| ORD-10302 | East | Router R1 | 390 | 70200 |
| :
```

PRODUCTION: PART 2 OF 10

```
Table 1 (part 2/10)

| Order | Region | Product | Qty | EUR |
|-----|-----|-----|-----|-----|
| ORD-10065 | West | Gateway G2 | 70 | 21700 |
| ORD-10066 | East | Sensor N7 | 55 | 2475 |
| ORD-10067 | West | Sensor N7 | 270 | 12150 |
| :
```

(c) Source file, a middle chunk

STOCK CHUNKER

```
def scale_88(values, factor=88):
    """Return values scaled by 88 and clipped at 880."""
    return [min(v * factor, 880) for v in values]
    :
    return [min(v * factor, 1270) for v in values]
```

PRODUCTION

```
python
def scale_88(values, factor=88):
    """Return values scaled by 88 and clipped at 880."""
    return [min(v * factor, 880) for v in values]
    :
    return [min(v * factor, 1270) for v in values]
```

(d) WebVTT transcript

STOCK CHUNKER

```
Welcome to the release planning session.
The Plavnik release window opens in March.
Regional units are summarised in the appendix.
```

PRODUCTION

```
[00:00] Welcome to the release planning session.
[00:04] The Plavnik release window opens in March.
[00:09] Regional units are summarised in the appendix.
```

Figure 2. Baseline output (left) and Corporate GPT's output (right) for four of the failure modes, produced by running Docling's stock chunker (or, in (b), a fixed 1,024-token window) and the production code on the same inputs. Inputs (a) to (c) are synthetic; (d) is from our regression corpus. Outputs are shortened vertically (:) and otherwise unedited.

06 Text trapped inside pictures.

On a scanned page, layout analysis finds one page-sized picture, and the OCR text is attached as its children; the chunker treats pictures as leaves, so the page is indexed only through a description of its image, when a vision model provides one. **Repair:** after the first pass, each page is tested: it qualifies when more of its OCR text items sit unchunked under pictures than were chunked *and* the pictures cover at least 60% of the page. If any page qualifies, the document is re-chunked with a chunker that descends into pictures, and only the qualifying pages' chunks are spliced back, in document order. The splice falls back to the fully re-chunked set if any item would be lost or duplicated, and is discarded unless it adds text. No OCR is re-run. `PICTURE_TEXT_RECOVERY` selects `scanned` (the default, as described), `all` (always descend into pictures) or `off`.

4.2 Two generalisable techniques

The six repairs are instances of two techniques. *Pre-chunk enrichment* pushes signal into the text before the chunker sees it (transcript timestamps, figure descriptions attached before chunking, normalised links in Markdown and HTML) and the unchanged pipeline carries it for free. *Post-chunk repair* keeps the chunker's boundary decision, which is usually right, and fixes the serialisation, which is where the loss occurs: lists and code fences directly, tables in the separately exported view, trapped text by re-running the boundary decision only where the first pass provably lost text. Because boundaries are kept, the repairs compose with any chunker that exposes the items it grouped.

BEFORE THE CHUNKER

Pre-chunk enrichment

Transcript timestamps

Figure descriptions

Normalised links

AFTER THE CHUNKER

Post-chunk repair

Lists

Code fences

Tables

Trapped text

4.3 Metadata and retrieval

Every chunk leaves the chunker with metadata in three groups (Figure 3): provenance: `document_id`, `file_name` and `page_label` (page, slide or sheet, for formats that have them); structure: `section` (body, table, figure, chart or image description), heading path, `object_index`, `table_part` and language; and access: five access-control fields and the project scope, attached at indexing. Citations take their page or slide from it, and access control is enforced inside the vector store, before ranking, because every point carries the access fields.

CONTENT (EMBEDDED AND SHOWN TO THE MODEL)

Table 1 (part 2/10)

Order	Region	Product	Qty	EUR
ORD-10065	West	Gateway G2	70	21700
ORD-10066	East	Sensor N7	55	2475
⋮				

Title marks a fragment: part 2 of 10.
Header row and separator repeated, so this part is answerable alone.

METADATA SET BY THE CHUNKER

```
document_id:
file_record_7c1e9a52-3b4d-4e8f-9a61-0d2c5b7e8f13
file_name: ledger.xlsx
section: table
page_label: 1
object_index: 1
table_part: 2/10
```

Provenance and structure: citations (sheet or page), stitching order (object index, part), idempotent deletes (document id).

METADATA ATTACHED AT INDEXING (ILLUSTRATIVE VALUES)

```
file_type: xlsx
acl_user_ids: [u-7f3c...]
acl_user_group_ids: [g-19a2...]
acl_is_public: false
acl_is_admin_public: false
acl_is_curator_public: false
project_id: 0
pii_anonymized: false
```

Access: the same predicate filters chunk and summary search inside the vector store, before ranking. Updated in place on ACL changes.

Figure 3. An indexed chunk record: the second of ten parts of the synthetic ledger of Figure 2b, as produced by the production code. Keys attached at indexing time are shown with illustrative values.

Metadata also lets retrieval deliver coherent units. Each query runs a dense and a sparse, BM25-style [12] search, fused by reciprocal rank fusion [4]. Before the final top-*k* cut, chunks of workbooks and slide decks that share a document and page label are stitched into one result, in document order (body, tables, figures; then object index and table part), and

scored by their best member, so five fragments of one slide occupy one result slot instead of five. A stitched group is capped at half of the source budget below, keeping its best-scoring members. PDF and word-processing pages are not stitched: they are artefacts of layout, not units of meaning. The principle is *index fine, deliver coarse*. Chunk size also sets how many sources reach the model: sources are admitted until they fill 90% of a 16,000-token budget, about seven chunks at the default size; splitting tables to the chunk budget and capping stitched groups keep one large table from crowding out every other source.

**Index fine, deliver
coarse.**

90% of a 16,000-token budget, about seven chunks at the default
size

— 05

Multimodal content

Some content has no text until a model creates it; Figure 4 summarises the extraction paths.

Figures. Pictures in PDFs are described inside Docling's pipeline; pictures in Office documents and video key frames are described in a separate pass. Both use the VLM bound to the system-level parsing slot, so the organisation, not the individual user, chooses the model that reads its documents. Pictures too small to carry content are skipped (under 5% of a PDF page, under 64 pixels elsewhere), and calls are bounded by concurrency and per-call timeouts. For Office documents and video there is also a per-document deadline of at most 660 seconds, after which the remaining pictures go undescribed; PDFs have no per-document deadline. A description appears both inline in its body chunk, at the picture's position in reading order, and as a standalone figure node with its caption and page (and, for PDF and PowerPoint figures, its bounding box) which lets a query such as “the architecture diagram” land on the figure itself.

Charts as data. A description (“a bar chart comparing quarterly revenue”) cannot answer “what was Q3 revenue?”. In PDFs, charts classified as bar, pie or line are converted to tables by a local chart-to-table model (Granite Vision) and enter the body chunk as markdown. In presentations, the series values and category labels cached in each native chart's XML are read directly and indexed as a chart node with slide number and chart type; Docling's presentation reader also places native chart data in the slide's body chunk.

Drawings. LaTeX figures written as TikZ, PSTricks or `picture` environments reach no image model. We extract each environment from the source, ask the text model to describe what the code draws without inventing content, and index the description together with the drawing source.

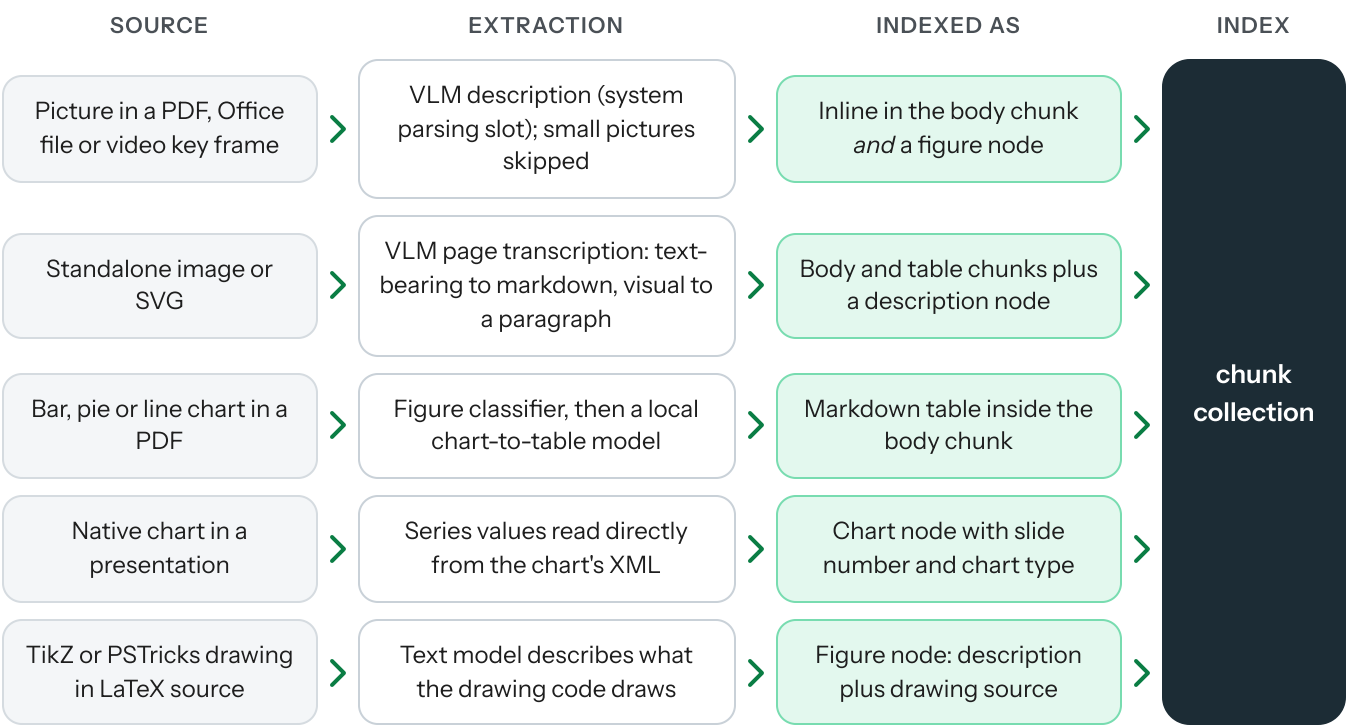


Figure 4. Content with no native text representation is made retrievable by model-generated descriptions or by structured data extraction. Every path ends in ordinary indexed nodes with structural metadata.

06

Hierarchical summarization

Chunk retrieval ranks passages, but some questions are about documents. *Discovery* (“which documents cover data retention?”) wants a ranking of documents, and a chunk ranking rewards the file with one strongly matching paragraph over the file that is about the topic throughout. *Gist* (“what is this report about?”) has an answer no single chunk contains. Both need a representation of the whole document, and Corporate GPT builds one per document at ingestion time by map-reduce over its own chunks (Figure 5):

- 1

Pack. The texts of all the document's nodes, in stored order and PII-redacted, are packed greedily into batches within `MAX_TOKEN_SUMMARY_CALL` (20,000 tokens by default; deployments may raise it). Segments are never split.

2

Map. Each batch is summarised by one call, using the system-level parsing model, so the result does not depend on who uploaded the file. Calls run concurrently on a shared pool of worker threads (`LLM_THREAD_POOL_SIZE` , 8 by default). The prompt asks for eight to ten bullet points, about 600 tokens in all, drawn only from the supplied content; reasoning-model preambles are stripped.

3

Reduce. If more than one summary survives, the summaries are re-packed, in the same order, and summarised again, until one remains.

At the prompt's target of about 600 tokens, one reduce batch absorbs thirty or more summaries even at the default budget, so documents of up to about 600,000 tokens are summarised in two levels, and a document that fits in one batch costs a single call. The target is not enforced (no output limit is set) so an individual summary can run longer.

Partial failure. Each call is attempted up to three times, with exponential backoff (1 s, then 2 s). A batch that still fails is dropped and the summary is built from the rest, so one bad call narrows coverage instead of failing the document. The guarantee has a boundary: if every batch fails, or the final reduce call does, the document has no summary, and because summarisation precedes indexing, its chunks are not indexed either until a scheduler retry (up to three) succeeds.

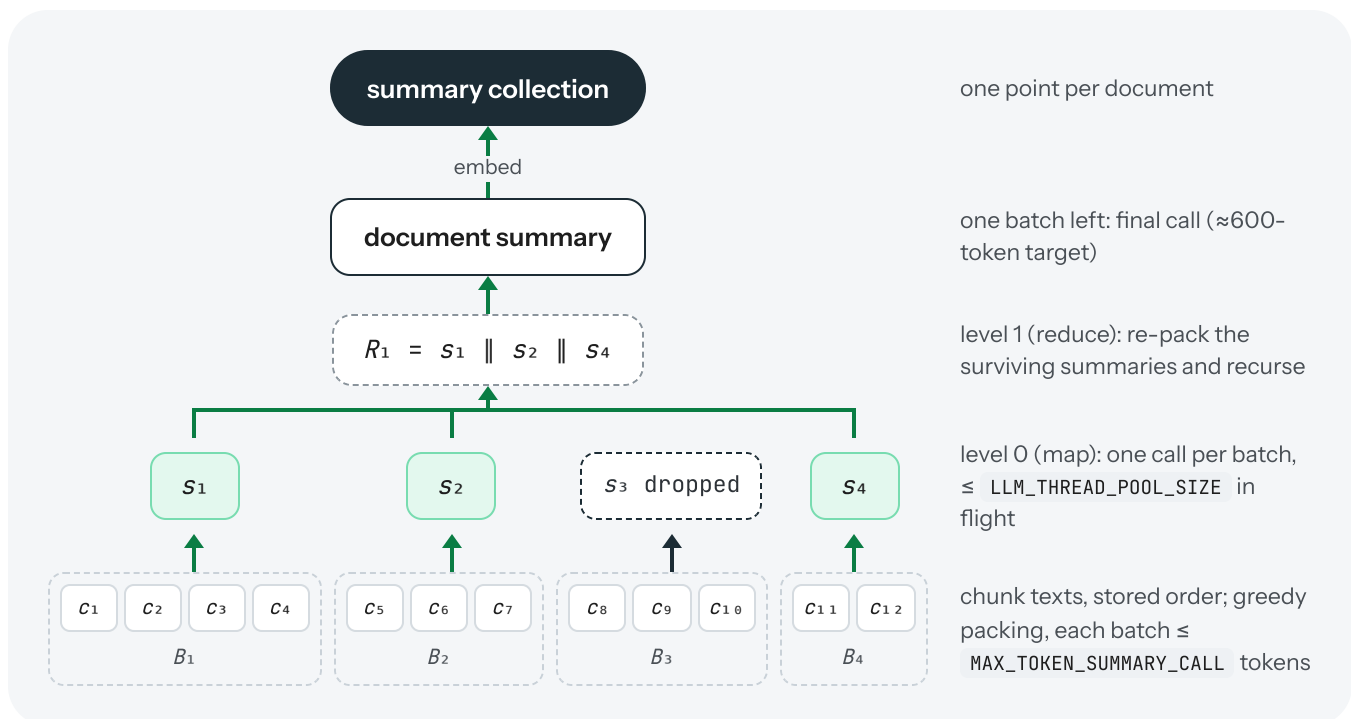


Figure 5. Hierarchical summarization of one document. Chunk texts are packed in order into token-bounded batches and summarised concurrently; the surviving summaries are re-packed and reduced until one remains. A batch that fails after its retries is dropped rather than failing the document.

The summary index. Summaries live in their own collection, one point per document, embedded dense-only with the chunks' embedding model and carrying the same access-control fields as the chunks, so summary search is filtered by the same predicate inside the vector store. The index is exposed to agents as a *file search* tool that returns the best-matching documents the caller may see. Today that tool is available to agents built in Corporate GPT's agent builder and to two built-in assistants; the default question-answering agent searches chunks directly, and its search cannot yet be narrowed to the documents file search returns. Two-stage retrieval (route to documents, then search inside them) is what the index enables, not yet the default behaviour.

— 07

Evaluation

We measure structural integrity: does each repair do what it claims on real inputs, and how often does the failure it addresses occur? Its effect on answer quality needs a mixed-format question set, for which we give the protocol rather than estimates.

Setup. The corpus is the regression set that guards Corporate GPT's ingestion against dependency upgrades: 13 single-fact canary files (digital and scanned PDF, HTML, Markdown, CSV, Word, Excel, PowerPoint, LaTeX, WebVTT, text, JSON and XML; the image and SVG canaries need a vision-model endpoint, and we did not run the audio and video canaries, which exercise speech recognition and frame description rather than chunking) and five real documents (a scanned map, a 15-page academic paper, a multi-sheet account-statement workbook, LaTeX prose and a subtitle file), plus four of Corporate GPT's own source files as of commit `8b6b6a6e` (`indexmanager.py` , `document_parser.py` , `rag_tools.py` and `api.real.ts`) and the synthetic inputs of Figure 2. We ran the ingestion code at commit `a9dab0a6` (the `quality` branch plus the repairs described here) with Docling 2.120.1, a 2048-token budget and the `Qwen3-Embedding-8B` tokenizer, offline on an Apple M3 laptop CPU with network access blocked. We disabled the stages that call a model endpoint (figure and drawing descriptions) and two local enrichment models (chart extraction, and code and formula recognition); they add text to chunks but do not change how structure is chunked. The canary tier can be regenerated from the repository; the real-tier documents are not distributable. The baselines are Docling's stock hybrid chunker with the same tokenizer and budget, and a fixed 1,024-token window with a 200-token overlap over the markdown export.

13

single-fact canary files

5

real documents

4

Corporate GPT source files

Structural integrity. Table 2 summarises the results. The failures are not hypothetical even in a modest corpus: one real workbook holds a table 58 times the chunk budget, and fixed windows separate table rows from their header in 91% of the windows that contain them. Stock serialisation produced no well-formed chunk out of 45 across four multi-chunk source files. Without trapped-text recovery none of the real scanned map's text reaches the index (with a vision model configured, only a description of the page image would), while the three other PDFs are unchanged by it. All thirteen canary facts are present in the production chunks, and chunks respect the budget except 3 of the academic paper's 53, which exceed it by 1 to 12 tokens because the stored heading prefix is slightly longer than the one the chunker counts.

Failure mode	Exercised by	Baseline	Production
Nested lists	Synthetic checklist, 4 nested items (the regression corpus has no nested lists)	Stock: 0 of 4 items indented	4 of 4 indented, numbering kept
Unbounded tables	34 tables in 10 files; 1 exceeds the budget	One node of 118,448 tokens; fixed window: 155 of 170 table-bearing windows lack the header	62 parts of at most 2040 tokens, all 62 with the header row
Broken code fences	4 source files, 45 chunks	Stock: 0 of 45 well-formed	45 of 45 fenced and language-tagged
Spreadsheet triplets	3 files (CSV and two workbooks)	Stock: 177,541 triplet tokens beside 122,722 table tokens	Triplets dropped: 59% fewer tokens indexed
Transcript timing	2 WebVTT files	Stock: 0 of 2 chunks timestamped	2 of 2
Trapped text	4 PDFs (2 scanned)	Recovery off: no chunk carries the scanned map's text	1 page recovered (367 characters); the other PDFs unchanged

Table 2. Structural integrity by failure mode: what the corpus contains, the baseline's output, and production's output.

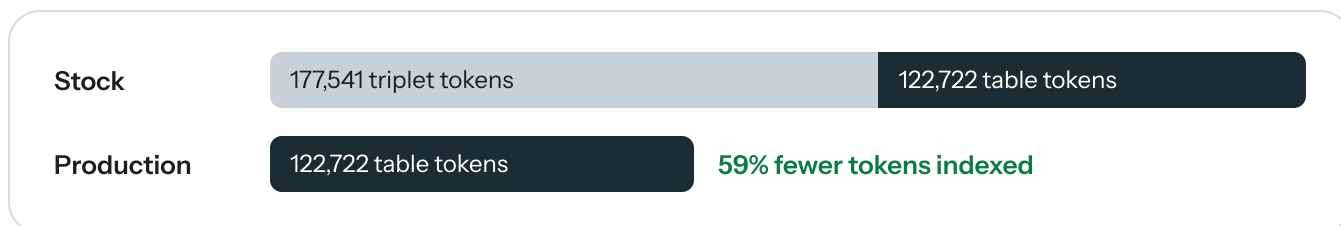


Figure 6. Tokens indexed for the three spreadsheet files, from Table 2. Production drops the triplet body text and keeps the table parts canonical.

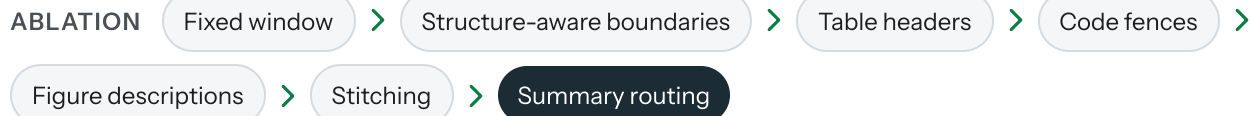
Cost. Parsing dominates CPU time, and within it layout analysis, table structure and OCR on PDF pages: about 2.5 s per page for the academic paper and 18 s for one dense scanned page (Table 3). Every other format parses in well under a second, and chunking never exceeds half a second. Summarisation needs one call for any document under the batch budget and, at the 20,000-token default, eight for a workbook of more than 120,000 tokens; at the 60,000-token budget one production deployment uses, the same workbook needs four and every other document one. Summary calls assume summaries at their 600-token target. Model latency depends on the deployed endpoint and is excluded.

Document	Pages	Parse (s)	Chunk (s)	Nodes	Figure calls	Summary calls
Digital PDF (canary)	1	3.4	<0.1	3	1	1
Scanned PDF (canary)	1	4.8	<0.1	2	0	1
Scanned map (real)	1	18.0	0.1	1	1	1
Academic paper (real)	15	37.2	0.5	53	3	4
Account statement (real)		0.1	0.2	72	0	8
Other canary files (11)		≤0.03	<0.1	1–6	0	1

Table 3. Processing time (laptop CPU, excluding model calls) and the model calls each document would incur; summary calls are at the 20,000-token default budget.

Answer quality. The ablation adds one mechanism at a time (fixed window, structure-aware boundaries, table headers, code fences, figure descriptions, stitching, summary routing) and scores each with RAGAs [5] on context precision and recall, faithfulness and answer relevancy. It needs a mixed-format question set weighted towards the formats the repairs target; our current RAGAs set of 40 questions over ten PDFs cannot exercise them,

and results will follow in a revision. Structural integrity is necessary but not sufficient. What these measurements establish is that without the repairs, specific information is unrecoverable downstream.



08

Limitations and future work

Summaries are lossy.

Quality is bounded by the summarising model, each reduction discards detail, one prompt shape serves every document type, no output limit is enforced, and a dropped batch leaves no mark on the stored summary.

Embedded metadata.

Each chunk's metadata (bookkeeping such as file paths, owner fields and access identifiers as well as the file name and heading path) is serialised into its embedded text, unredacted and outside the chunk budget, diluting both the dense and the sparse signal. Restricting it to semantically useful keys requires re-embedding existing content.

Personal-data coverage.

Detection is English-only and thresholded (Section 2): bank-account and passport numbers and non-US national identifiers are not redacted.

Budgets.

`CHUNK_SIZE` is global rather than per format, and the counting tokenizer need not match the deployed embedding model.

Failure visibility.

Partial conversions and failed PDF figure descriptions leave no record on the attempt; Docling and PDF-library errors are not retried even when transient; and the per-document description deadline can leave many pictures in very large decks undescribed.

Remaining repair gaps.

Multi-row table headers repeat only their first row; code blocks inside other documents are not re-fenced; only WebVTT transcripts carry timestamps; where a chunk mixes a list with a table, code or a formula (even an inline code span) or with part of a split item, or its list items contain formatted text or links, the list keeps the chunker's flat serialisation; and a rebuilt chunk drops inline markup, such as link targets, from its other items.

Duplication and cost.

Figure descriptions and document-format tables are indexed twice, and a native slide chart up to three times; descriptions are not cached, so a changed document is described again from scratch.

Beyond closing these gaps, future work includes per-format budgets, section-level summaries between the chunk and the document, and the answer-quality ablation on a mixed-format set.

— 09

Conclusion

Retrieval quality in enterprise RAG is decided before the embedding model runs: by whether the chunker preserves the structure that carries meaning, by whether content with no native text becomes text or data at all, and by whether documents, not only passages, can be found. Corporate GPT addresses these with one intermediate representation for fifteen format families, six repairs on a structure-aware chunker that reduce to two reusable techniques, multimodal extraction that prefers data to descriptions, and a per-document summary index. Our structural evaluation shows each repair doing what it claims on real inputs; measuring what each is worth in answer quality is the work that comes next.

Retrieval quality in enterprise RAG is decided **before the embedding model runs.**

— REFERENCES

References

1. Anthropic. Introducing contextual retrieval. <https://www.anthropic.com/news/contextual-retrieval>, 2024.
2. Christoph Auer, Maksym Lysak, Ahmed Nassar, Michele Dolfi, Nikolaos Livathinos, Panos Vagenas, et al. Docling technical report. Technical report, IBM Research, 2024. arXiv:2408.09869.
3. Tong Chen, Hongwei Wang, Sihao Chen, Wenhao Yu, Kaixin Ma, Xinran Zhao, Hongming Zhang, and Dong Yu. Dense X retrieval: What retrieval granularity should we use? In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2024.
4. Gordon V. Cormack, Charles L. A. Clarke, and Stefan Büttcher. Reciprocal rank fusion outperforms Condorcet and individual rank learning methods. In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2009.
5. Shahul Es, Jithin James, Luis Espinosa-Anke, and Steven Schockaert. RAGAs: Automated evaluation of retrieval augmented generation. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, 2024.
6. Luyu Gao, Xueguang Ma, Jimmy Lin, and Jamie Callan. Precise zero-shot dense retrieval without relevance labels. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL)*, 2023.
7. Michael Günther, Isabelle Mohr, Daniel James Williams, Bo Wang, and Han Xiao. Late chunking: Contextual chunk embeddings using long-context embedding models. arXiv:2409.04701, 2024.
8. Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, et al. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.
9. Ahmed Nassar, Nikolaos Livathinos, Maksym Lysak, and Peter Staar. TableFormer: Table structure understanding with transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.
10. Qdrant. Qdrant: Vector similarity search engine and vector database. <https://qdrant.tech>, 2024.
11. Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. Robust speech recognition via large-scale weak supervision. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, 2023.

12. Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389, 2009.
13. Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D. Manning. RAPTOR: Recursive abstractive processing for tree-organized retrieval. In *International Conference on Learning Representations (ICLR)*, 2024.
14. Brandon Smith and Anton Troynikov. Evaluating chunking strategies for retrieval. Chroma Technical Report, <https://www.trychroma.com/research/evaluating-chunking>, 2024.
15. Jeff Wu, Long Ouyang, Daniel M. Ziegler, Nisan Stiennon, Ryan Lowe, Jan Leike, and Paul Christiano. Recursively summarizing books with human feedback. arXiv:2109.10862, 2021.