**CHUNKING DOCUMENTS**

**GEMINI**

**1. Handling Long Legal Descriptions and LLM Token Limits**

The core issue with long legal descriptions is that they can easily exceed the token limits of Large Language Models (LLMs), forcing you to truncate the text and lose critical information.1 A naive approach of simply splitting the text into fixed-size chunks is problematic because it disregards the meaning and structure of the legal language, often breaking a single clause or call in a metes-and-bounds description across two separate, incoherent chunks.2

The best solution is **Semantic Chunking**, a method of breaking down large texts into smaller, semantically meaningful segments.3 The goal is to preserve the complete context of each part of the description. Here are the most effective ways to implement this:

* **Heuristic-Based Segmentation:** This is a rule-based approach that leverages the predictable structure of legal descriptions. You can define rules to split the text based on common patterns, which is particularly effective for metes and bounds descriptions.4 For example, you can create rules to split the text at the end of each "call" (e.g., after phrases like "...to a point;" or "...thence North 89 degrees West..."). This method can be implemented using regular expressions or simple string-matching logic.5
* **NLP-Powered Sentence Splitting:** For a more sophisticated approach, you can use Natural Language Processing (NLP) libraries like spaCy.3 These tools are trained to recognize sentence boundaries with much higher accuracy than simple punctuation splitting. They can handle complex sentence structures found in legal documents, ensuring that each chunk represents a complete thought or clause.8
* **Embedding-Based Chunking:** This is the most advanced method. It involves using a neural network to convert each sentence or small group of sentences into a numerical vector (an "embedding"). You can then calculate the semantic similarity between adjacent vectors. A sharp drop in similarity indicates a change in topic, which is an ideal place to split the text into a new chunk.3 This ensures that each chunk is highly cohesive in its meaning.

Once you have these smaller, semantically complete chunks, you can process them individually. For tasks like summarization, you can use a **MapReduce** approach: summarize each chunk independently and then create a final summary from the collection of individual summaries.10 This allows the LLM to process the entire legal description without ever exceeding its token limit in a single step.

**2. Processing Pages Containing Multiple Deeds**

When a single scanned page from a deed book contains multiple distinct deeds, treating the entire page as one document will confuse the data extraction process. The solution here is **Document Layout Analysis (DLA)**, also known as document segmentation.11

DLA is the process of analyzing the spatial arrangement of a document to understand its structure.13 It identifies the location of different elements like text blocks, titles, headers, footers, and images.12 This allows you to programmatically identify the boundaries of each individual deed on a single page.

Here is how to approach this:

* **Identify Logical Boundaries:** The first step in DLA is to identify the logical boundaries that separate one deed from the next. This could be a larger-than-usual gap of white space, a horizontal line, or the presence of a new document title or recording stamp. For multi-page deeds within a larger bundle, some models can even be trained to classify pages as "initial," "middle," or "final" to reconstruct the individual documents.16
* **Use Machine Learning for Segmentation:** While simple rule-based heuristics can work for highly consistent layouts, a more robust solution involves using machine learning models. Modern computer vision models like **YOLOv8** can be trained on a dataset of deed book pages to automatically detect and draw bounding boxes around each individual deed.17 The model learns to recognize the visual patterns that signify the start and end of a deed, even with varying layouts.22
* **Process Each Segment as a Separate Document:** Once the DLA process has identified the coordinates of each deed on the page, you can treat each of these segments as a separate, independent document. The OCR and data extraction pipeline would then be run on each of these "sub-documents" individually. This ensures that the information from one deed (e.g., the grantor and grantee) is not incorrectly associated with an adjacent deed on the same page.

By first applying Document Layout Analysis to segment multi-record pages and then using Semantic Chunking to break down long text sections within each resulting document, you can create a robust pipeline that feeds your AI system with clean, well-structured, and contextually complete data, effectively resolving both of the issues you've encountered.

**CHATGPT**

You’ll fix both “too-long legal descriptions” and “multiple deeds on one page” by making your pipeline **layout-aware** and **boundary-driven**, not purely text-length-driven. Concretely: (1) segment pages into structural blocks using OCR coordinates, (2) detect deed/section boundaries with lightweight rules + ML features, (3) produce **small, overlapping chunks** with durable IDs and offsets, and (4) run extraction per chunk with a reducer that reconciles fields across chunks.

Below is a prescriptive design you can implement now in your Google Document AI + PostgreSQL stack.

**1) Layout-aware segmentation (foundation for both problems)**

1. **Request bounding boxes** from the OCR (paragraph/line/word level). Persist:
   * page\_no, block\_id, line\_id, bbox=(x1,y1,x2,y2), text.
   * Also persist **reading order** (y then x, with column detection).
2. **Block grouping:** cluster adjacent lines into blocks by vertical gaps and left-edge similarity (same column and indent ≈ same block).
3. **Visual separators:** detect horizontal rules, large vertical whitespace, and recorder stamps; these often delineate separate instruments on the same page.

Result: a page → blocks in **reading order** with geometry. All downstream chunking uses these blocks, not raw strings.

**2) Boundary detection rules (fast, accurate in practice)**

Use a **two-tier boundary detector** over blocks:

**A. Rule set (cheap, robust)**

Mark a block as a **start-marker** if it matches any of:

* Deed/opening phrases (case-insensitive):  
  ^(warranty|quitclaim|special warranty|deed of trust|indenture|easement|assignment|release|lien)\b  
  ^this (warranty|quitclaim|indenture)\b  
  ^prepared by\b|\breturn to\b (often header prefaces)
* Party header patterns: between .\* and .\*, grantor[: ], grantee[: ]
* Recorder headers: filed for record, instrument no\., book \d+ page \d+
* Jurat/ack start (often indicates **end** of an instrument): ^state of\b.\*\bcounty of\b, acknowledged before me, notary public

Mark a block as a **legal-description start** if it matches:

* ^legal description\b, ^description\b, ^beginning at\b|^commencing at\b, ^lot\b, ^tract\b, ^parcel\b, ^section\b, or an all-caps heading TRACT \d+, PARCEL \w+, AND ALSO:.

Mark **legal-description continuation** lines by:

* Leading “THENCE”, “N/S/E/W \d+°”, bearing/curve regexes, semicolon-terminated courses, or long numeric runs with ° ' ".

Mark **instrument end** by:

* Notary/ack block, **release clause** terminus (“discharged/satisfied in full”), or the next start-marker.

**B. Lightweight classifier (optional, improves recall)**

If rules alone aren’t perfect, train a tiny classifier over block features:

* Features: regex hits above, avg line height, indent variance, all-caps ratio, keyword counts.
* Labels: {instrument\_start, body, legal\_desc\_start, legal\_desc\_body, notary/ack, separator}.
* Output used only to **confirm** or fill gaps between rules.

**3) Chunking strategy that never blows token limits**

Once you’ve segmented instruments and their internal parts, chunk **within** those structures.

**A. Instrument-level chunking (multi-deed per page)**

* **One page can yield many instruments.** Emit (book, page, segment\_index)—e.g., 211-419-1, 211-419-2.
* Each instrument’s chunk stream = header → parties → grants → legal description → exhibits → notary.
* Keep chunks **small**: target **800–1,200 tokens** per chunk for LLMs, with **10–15% overlap** between chunks to preserve context at boundaries (store chunk\_seq, overlap\_prev\_chars).

**B. Legal description chunking (long, multi-part)**

* Split on **section headings**: TRACT \d+, PARCEL [A-Z], LOT \d+, AND ALSO:, “TOGETHER WITH”.
* Within each tract, split on **course cadence**:
  + Start at BEGINNING/COMMENCING, then create chunks of **15–25 courses** (lines beginning with “THENCE” or a bearing). Keep an overlapping **look-back of 1–2 courses**.
* If metes-and-bounds without explicit course lines, chunk on **punctuation cadence**: split at semicolons; if too long, split every ~1,200–1,600 characters near a punctuation boundary.
* Always **normalize bearings** (replace OCR confusables like l→1, O→0, rn→m) **before** chunking, but retain raw text for audit.

**C. Token-aware reflow**

Before sending to the LLM:

* Strip boilerplate (“Prepared by”, stamp text) unless needed for field extraction.
* Remove repetitive full-page headers/footers.
* Collapse multiple spaces, but **do not** collapse semicolons/line breaks in legal descriptions (course boundaries matter).
* If token estimate still high, **route**: send legal description to a specialized extractor prompt; send parties/dates to a smaller prompt. (Router = fast keyword check; no need for ML here.)

**4) Durable identifiers and offsets (stitching & audit)**

Every chunk carries:

* instrument\_id = (book, page, segment\_index)
* page\_span = list of page numbers covered
* block\_span = first/last block ids
* char\_offset\_span within the instrument concatenated text
* chunk\_seq, prev\_overlap\_len

Store outputs with these keys so your reducer can **stitch** chunk-level extractions back to the instrument and resolve duplicates/conflicts deterministically.

**5) Reducer: merge per-chunk extractions into one instrument record**

After per-chunk extraction:

* **Field precedence:** prefer values with higher model confidence; if tie, prefer from **earliest chunk** where field appears (headers > body).
* **Name/date reconciliation:** apply fuzzy merge (trigram/Jaro-Winkler) across chunks; compute a **consensus** value and record alternates with scores.
* **Legal description merge:** concatenate course lists in order; verify continuity (last course end ≈ next course start if your parser computes geometry; otherwise ensure no course lost at chunk boundary by checking overlap).
* **Document type sanity check:** header keywords override misreads in body.
* Write one Document row + parsed\_fields JSON with field-level confidences and chunk provenance.

**6) Special handling: multiple deeds on one page**

Here’s a concrete routine that works well in old deed books:

1. **Page pre-pass:** find **all start-markers** on the page; sort by top-y. If none, fall back to largest vertical whitespace + all-caps headings.
2. **Segment creation:** for each start-marker i, the **instrument region** is blocks from i up to (but not including) i+1. If a notary/ack appears earlier, cut there.
3. **Header tie-break:** if two start-markers are very close (< 60 px vertically) and one is Prepared by and the other is a deed type, the deed type wins as start; treat Prepared by as header block for that segment.
4. **Stamp collision:** if a recorder stamp lies mid-page, don’t treat it as a separator unless followed by a deed header—stamps can appear inside an instrument.
5. **Cross-page spill:** if an instrument continues to next page (no notary on current and next page begins with body text), continue the same segment\_index across pages; mark page\_span accordingly.

This yields deterministic (book, page, segment\_index) identifiers and stable segmentation across rescans.

**7) Practical token budgets & prompts**

* **Target 800–1,200 tokens** per chunk for GPT-class models; **max 1,600** only for legal descriptions where course integrity matters, still with overlap.
* Use **map-reduce prompting**:
  + **Map prompt:** “Extract only parties, dates, consideration, related-doc refs from this chunk. If absent, return nulls. Do not infer.”
  + **Map-LD prompt (legal description):** “Extract normalized course list; preserve sequence and bearings; do not summarize.”
  + **Reduce prompt:** “Given these partial fields + confidences + chunk order, produce a single consolidated instrument record; resolve conflicts by confidence then header precedence.”
* **Hard stop guards:** If a chunk contains multiple start-markers (mis-segmented), short-circuit and re-segment rather than forcing extraction.

**8) Minimal code sketch (pseudo you can drop into your pipeline)**

**Boundary regexes (Python):**

DEED\_START = re.compile(r'^(warranty|quitclaim|special warranty|deed of trust|easement|assignment|release|lien|indenture)\b', re.I)

PARTY\_OPEN = re.compile(r'\bbetween\b.+\band\b', re.I)

RECORDER = re.compile(r'\b(book\s\*\d+\s\*page\s\*\d+|instrument\s\*no\.)', re.I)

ACK\_START = re.compile(r'^(state of|county of).\*\backnowledged\b|notary public', re.I)

LD\_START = re.compile(r'^(legal description|description|tract\s+\d+|parcel\s+[A-Z]|lot\s+\d+|and also:|beginning at|commencing at)\b', re.I)

COURSE = re.compile(r'^(thence|\b[ns][ew]?\s\*\d{1,3}°\s\*\d{1,2}\'\s\*\d{1,2}"\b)', re.I)

**Instrument segmentation from blocks:**

segments = []

cur = []

for blk in blocks\_in\_reading\_order:

if DEED\_START.search(blk.text) or PARTY\_OPEN.search(blk.text) or RECORDER.search(blk.text):

if cur:

segments.append(cur); cur = []

cur.append(blk)

if ACK\_START.search(blk.text):

segments.append(cur); cur = []

if cur: segments.append(cur)

# assign segment\_index by order; compute page\_span, block\_span, y-bounds

**Legal description chunking:**

for seg in segments:

parts = split\_on\_regex(seg, LD\_START) # header/body/ld parts

for ld in parts.legal\_descriptions:

courses = split\_on\_regex(ld, COURSE, keep\_delim=True)

chunks = chunk\_by\_count(courses, max\_courses=20, overlap=2)

emit\_chunks(chunks, type='LD')

# chunk other parts by char length ~1200 with sentence/punctuation boundaries

**9) QA and safety rails**

* **Visualization:** generate a low-res PNG overlay with block/segment boxes and IDs; a reviewer can spot bad splits in seconds.
* **Round-trip test:** re-concatenate chunk texts per segment; assert they equal the original segment text (modulo whitespace). If not, your chunker is dropping/duplicating text.
* **Checksum & offsets:** store SHA-256 of original segment text; for each chunk, store start/end **character offsets** into the segment string—this makes stitching exact.

**10) Database support**

Add to your Postgres schema:

* ocr\_block(page\_no, block\_id, bbox, text, instrument\_id, segment\_index, order\_in\_segment)
* instrument(doc\_id, book, page, segment\_index, page\_span int[], block\_span int4range, sha256, ...)
* chunk(instrument\_id, chunk\_seq, kind, char\_start, char\_end, overlap\_prev, text, token\_estimate)  
  This lets you:
* Rebuild an instrument from chunks deterministically.
* Trace any extracted field back to the **exact characters** and page.

**11) Opinionated defaults (use these unless you have a strong reason not to)**

* **Always** segment by layout first; never chunk a raw page blob.
* **Never** send >1,600 tokens to an LLM for extraction; prefer more chunks + a reducer.
* **Always** overlap legal description chunks by at least **one course**; elsewhere, 10–15% character overlap.
* **Treat “Prepared by/Return to” as headers**, not instrument starts, unless no other start marker exists.
* **Use geometry gaps > 1.5× median line height** as a soft separator when rules disagree.

If you want, I can turn this into a drop-in Python module that takes Document AI JSON, writes ocr\_block rows, and emits (instrument, chunk) tables with the regexes above plus overlap logic—and a quick PNG overlay for QA.