QUBER · DOCUMENT EXTRACTION STUDY

ADE versus Docling

A comparative analysis of document parsing for equity-market filings

This study evaluates landing.ai's Agentic Document Extraction (ADE) as a parallel path to docling for the markdown and structural view that `quber document` produces. It is grounded in real runs against two SEC filings — Arbor Realty Trust's Q1 2026 10-Q and Visa's Q1 2026 earnings exhibit — and reports four measured dimensions: table representation, LLM retrieval reliability, numeric value fidelity, and document-structure faithfulness, each with its credit cost. The reader is assumed familiar with the quber extraction pipeline and the docling DoclingDocument model.

Engineering Reference · Generated 29 June 2026
Evidence: `output/ade/`, `output/*.docling.*` · Parse + Section scope

Contents

1 Context and Scope

2 The Two Pipelines

- 2.1 Docling: a typed element tree
- 2.2 ADE: grounded chunks plus a Section query

3 Table Representation: Pipe versus HTML

- 3.1 Worked example: Arbor Changes in Equity

4 The Retrieval Experiment

5 Value Fidelity

6 Document Structure: Section versus Docling

- 6.1 Worked example: the Visa earnings exhibit
- 6.2 Element typing

7 Cost Accounting

8 Findings and Recommendation

Audience and Scope

For engineers weighing whether to adopt ADE alongside docling. The scope is the two generally-available ADE tools, **Parse** and **Section**. ADE Extract, Classify, and Split are out of scope: Classify and Split are vendor-marked Preview, and Extract's field-level grounding is undocumented. Nothing here changes the existing Camelot / Set-of-Mark table pipeline; ADE is assessed only as an alternative document view.

1 Context and Scope

The `quber document` command resolves a source PDF, runs docling under the canonical tuned-financial configuration, and writes the result as JSON plus a markdown export. In the CLI this command maps to the `parse` function; the markdown is produced by a single call to `document.save_as_markdown(...)`. That one markdown artifact is the unit under examination: the question is whether ADE can produce an equal or better document view for the same source, and at what cost.

Because the markdown file is a terminal artifact — nothing downstream in quber reads it back — swapping its producer is self-contained. The table-correction path builds its own markdown in memory from the DoclingDocument and is untouched by this question. So ADE is studied as a *parallel* producer of the document view, not as a replacement for the table pipeline.

> *Surface failures explicitly; never silently judge what is acceptable to drop. On a financial filing, an unflagged extraction gap is a liability.*
>
> — standing engineering principle for this evaluation

1.1 The evidence base

Every number in this document comes from a run executed during the study, not from vendor marketing. Two filings anchor the analysis.

Arbor10Q_Q126 67-page 10-Q, 63 corrected tables

VISA_991_Q126 11-page earnings exhibit

Model ADE DPT-2; Haiku 4.5 for retrieval

2 The Two Pipelines

Both systems turn a PDF into a structured document, but they produce different shapes. Docling yields a navigable element tree; ADE yields a flat list of grounded chunks plus a full-document markdown, with hierarchy available only through a second API call.

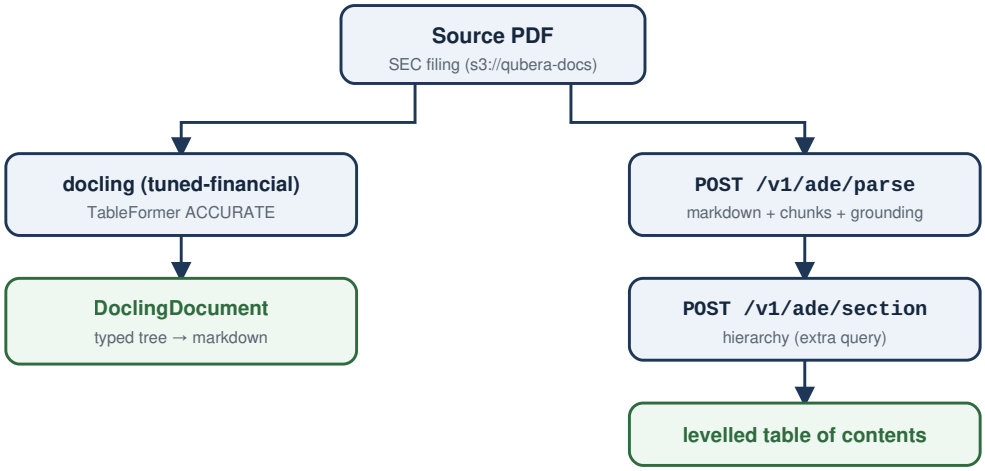

Figure 1 — The two document-view paths from one source PDF. Docling delivers hierarchy inside a single parse; ADE delivers it through a second Section call.

2.1 Docling: a typed element tree

Docling emits a tree of typed elements with a rich label vocabulary — `section_header`, `page_header`, `page_footer`, `list_item`, `footnote`, `picture`, form checkboxes — each carrying page and bounding-box provenance, with lists reconstructed as groups. It is a navigable object: you can ask for every footnote, or strip running headers, by label.

2.2 ADE: grounded chunks plus a Section query

ADE Parse returns one JSON with five top-level keys: `markdown`, `chunks`, `splits`, `grounding`, and `metadata`. The markdown is not plain GitHub-flavoured markdown — tables are HTML `<table>`, every chunk is prefixed with an `<a id=…>` anchor, and figures use a `<::…::>` delimiter. The chunk type vocabulary is coarse and comes from the DPT-2 ontology.

> **Provenance note.** ADE chunk types observed on the Arbor filing were `text`, `marginalia`, `table`, and `attestation`. There is no heading chunk type — headings exist only as markdown # levels, not as typed structural elements.

3 Table Representation: Pipe versus HTML

Docling's markdown serialises tables as pipe grids; ADE serialises them as HTML. The difference matters only where a table's structure exceeds what a pipe grid can express: a pipe table cannot carry a column span, a row span, or a multi-row header. For most financial statements the column headers are a single flat row, so the two forms are equivalent. The divergence appears at full-width section banners inside a table.

3.1 Worked example: Arbor Changes in Equity

The Consolidated Statements of Changes in Equity stacks two periods in one table, each introduced by a full-width banner row. Docling's pipe form ran the two periods together and shredded the banner across cells; ADE split them into two tables and captured the banner as a `colspan` cell.

Figure 2 — The 2025 period banner, three views (Arbor10Q, p.6)

1 · DOCLING PIPE (shredded)	2 · ADE HTML (bounded)	3 · WHAT IT MEANS

`| | | Three | Months Ended | March 31, | 2025 | | | |`
`| Balance - January 1, 2025 | 42,585,589 | ...`
	`<table id="2-1E">`
	`<tr><td colspan="10">Three Months Ended March 31, 2025</td></tr>`
	`<tr><td>Balance - Jan 1, 2025</td><td>42,585,589</td> ...`
		The banner is a section marker spanning all 10 columns. Pipe cannot express a span, so it disintegrates into four junk cells. HTML keeps the period boundary intact.

WHERE HTML ACTUALLY WINS

Not on column headers, which financial statements keep flat, but on full-width banners and table bounding. ADE gave each period its own table; docling merged them with no boundary, so a reader has no structural cue that the 2025 rows belong to a different period than the 2026 rows above them.

4 The Retrieval Experiment

A richer table representation is only worth its cost if a downstream model reads it more reliably. We tested that directly: does an LLM retrieve a value more reliably from ADE's nested HTML than from quber's Camelot / Set-of-Mark corrected pipe table? Docling's raw markdown was excluded as a baseline because quber discards docling's table bodies and replaces them with the corrected ones — the corrected pipe is the real incumbent.

4.1 Method

The probe was the Arbor Changes-in-Equity table, the hardest case: row labels such as *Net income* and *Balance – January 1* appear in both period blocks, so a correct answer must resolve period, then row, then column. Ten such lookups were posed against three representations of the same table, each run five times at temperature 0.7, scored by exact numeric match against quber's validated values.

quber_pipe corrected pipe, as shipped

quber_html same values, rendered as nested HTML

ade_html ADE's HTML, as returned

4.2 Result

Representation	Score	Accuracy
quber_pipe — corrected pipe	50 / 50	100%
quber_html — same values as HTML	50 / 50	100%
ade_html — ADE HTML	50 / 50	100%

All three forms scored a clean sweep, and the model genuinely disambiguated periods — it returned the 2025 figure where asked, not the 2026 one. The format made no difference to retrieval reliability.

WHY THIS MATTERS

For a clean, correctly-extracted equity table read in isolation, HTML structure buys no retrieval advantage over the corrected pipe. Since clean equity statements are the representative case for this document universe, the result is not a toy: it is the expected case. The simplest version of the "HTML helps the LLM" hypothesis is falsified for these documents.

OPEN · LIMIT OF THE NULL

This is a null from a non-discriminating probe — the task was easy enough that all forms hit the ceiling. It rules out a format advantage on clean tables; it does not prove format never matters on degraded or embedded tables. Those remain untested.

5 Value Fidelity

Before trusting either side, the numbers must agree. We diffed quber's corrected values against ADE's, cell by cell, across the five consolidated-statement tables of the Arbor slice, normalising away pure formatting ($, em-dash, parentheses) and comparing only the digits and signs.

Statement (Arbor10Q)	Rows compared	Numeric mismatches
Balance Sheet	30	0
Statements of Income	32	0
Changes in Equity	12	0
Cash Flows	48	0
Cash Flows (continued)	13	0
Total	**135**	**0**

Across 135 value-bearing rows the two extractions agreed on every digit and every sign, including parenthesised negatives. The only differences were cosmetic: ADE places `$` on more cells, and the two render zeros as `—`, `-`, or an empty cell interchangeably. There were no parsing errors on either side for these tables.

On the earlier "value differences." Those were docling *raw* markdown versus ADE — docling dropping a leading $. Docling raw is the side quber throws away, so that comparison used the wrong baseline. Against the side quber keeps, the values are identical.

QUBER · STRUCTURE COMPARISON

6 Document Structure: Section versus Docling

The sharpest difference between the two systems is hierarchy. Examined against the source PDF, ADE's Section query reconstructs the document's nesting; docling flattens it.

6.1 Worked example: the Visa earnings exhibit

Page one of VISA_991_Q126 shows a clear visual hierarchy: a large banner headline, “Visa Reports Fiscal First Quarter 2026 Results,” with smaller blue sub-headings — Income Statement Summary, Key Business Drivers — beneath it, and a CEO quote set off in the right column. ADE Section reproduced that nesting; the lineage below is its actual output for the document head.

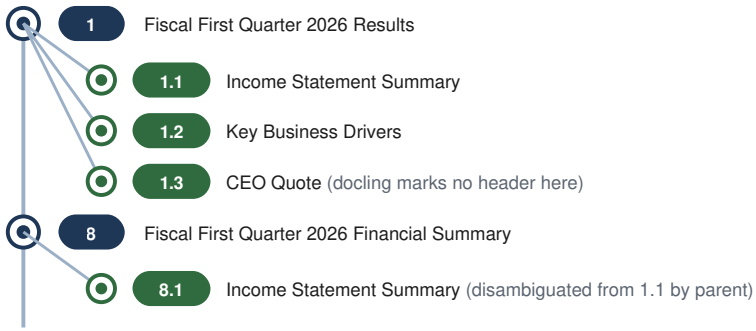

Figure 3 — ADE Section output for the Visa head (navy = level 1, green = level 2). The same title “Income Statement Summary” appears under two parents and is told apart by section number; docling lists both as flat, indistinguishable headers.

Docling found the same headings but assigned every one the same level. Its sixteen `section_header` elements are all `level: 1` — a flat list that contradicts the visible size hierarchy and cannot disambiguate the two “Income Statement Summary” blocks.

Structural property	ADE Parse + Section	Docling
Section hierarchy	2 levels, nested, matches the PDF	flat — all 16 headers level 1
Heading → content link	chunk `start_reference` per section	reading-order tree position
Duplicate-title disambiguation	by section number (1.1 vs 8.1)	none
CEO quote as a unit	captured (1.3)	not marked as a header

6.2 Element typing

Docling's advantage is the inverse: a finer element vocabulary. On the same Visa document it separately typed what ADE's base Parse folds into plain text or marginalia.

Typed elements (Visa, docling)	Count
`section_header`	16
`list_item` (in groups)	32
`picture` (logos / icons)	21
`footnote`	4
`page_footer` / `page_header`	10 / 1

ADE's coarse set — text, marginalia, table, attestation — cannot answer “give me every footnote” or “list the bullet items” from chunk types alone. For a fine-grained element inventory, docling is richer, and it is already produced at no extra cost.

7 Cost Accounting

Every ADE response reports `credit_usage` in its metadata, so cost is measured, not estimated. Two rates hold across runs.

Operation	Scope	Credits	Rate
Parse — Arbor10Q	67 pages	201.0	3.0 / page
Parse — Visa	11 pages	33.0	3.0 / page
Section — Visa	whole doc	12.46	content-priced
Parse + Section — Visa	11 pages	**45.46**	≈ 1.38× Parse

Parse is a flat three credits per page, predictable across both filings. Section is not priced per page; on the Visa exhibit it added roughly 38 percent on top of Parse. Docling, by contrast, produces its hierarchy as part of the parse you already run, at no marginal cost.

Completeness signal. The 67-page Arbor parse returned `failed_pages: []` — no silent page gaps, which is the property that matters most for a financial filing.

8 Findings and Recommendation

The two systems are complementary, and which is “better” depends on what the document view is for.

Dimension	Winner	Evidence
Section hierarchy / levels	**ADE**	2 nested levels vs docling's flat 16
Fine-grained element types	**Docling**	footnote / picture / list typed
Table value accuracy	Tie	135 rows, 0 mismatches
LLM retrieval reliability	Tie	100% / 100% / 100%
Cost of the structure	**Docling**	free vs +38% Section surcharge

RECOMMENDATION

For an outline or hierarchy view — “what are the sections and how do they nest” — ADE Parse + Section is the more faithful model, paid for with the ~38% Section surcharge. For a typed element inventory, docling is richer and already free in the pipeline. Their table outputs are interchangeable on value, and neither earns an LLM-retrieval edge on clean equity tables. Adopt ADE where document hierarchy is the deliverable; keep docling where typed structure and zero marginal cost are.

OPEN · KNOWN GAPS

Section's `start_reference` anchors a section to its *heading* chunk, not the table beneath it — tying a section to its table needs a small follow-on match. ADE Extract's field-level grounding is undocumented, and Classify and Split are vendor Preview. The retrieval null is established only for clean, isolated tables.

ARTIFACTS

Where the evidence lives

Generated during this study · 29 June 2026.

ADE outputs: `s3://qubera-extracts/<ticker>/ade/` and `output/ade/`.

Per-document: `<stem>.ade.json` (full Parse response), `<stem>.ade.md` (markdown), `<stem>.section.json` / `.section.md` (hierarchy). Docling sides at `output/<stem>.docling.{json,md}` and `<stem>.tables.json`.