QUBER · DOCUMENT & TABLE EXTRACTION STUDY

ADE versus the Quber Stack

Landing.ai measured against docling and the Set-of-Mark / Camelot table workflow

This study compares landing.ai's Agentic Document Extraction (ADE) against the two systems quber runs today: docling for the document view, and the Set-of-Mark / Camelot workflow for table extraction. 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. Part I weighs the document view (ADE Parse + Section against docling); Part II weighs the table workflow on coverage, accuracy, provenance, and structure; Part III records the ADE features we are not using and the overall recommendation. The reader is assumed familiar with the quber pipeline.

Engineering Reference · Generated 29 June 2026

Evidence: `output/ade/`, `output/*.docling.*`, `output/*.tables.json`

Contents

Audience and Scope

For engineers weighing whether ADE earns a place alongside quber's stack. Part I scopes the two generally-available ADE document tools, **Parse** and **Section**. Part II maps the `quber table` work flow — Set-of-Mark location, Camelot extraction, grounded correction, and fusion into the docling spine — against ADE's table chunks. Nothing here proposes changing the production table pipeline; ADE is assessed as a candidate, not a replacement.

PART I · THE 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 — the command maps to the `parse` function, and the markdown comes from one call to `document.save_as_markdown(...)`. Separately, `quber table` (the `extract` command) runs the Set-of-Mark workflow to produce vetted tables. ADE is evaluated against both: its Parse + Section output against the document view, and its table chunks against the Set-of-Mark tables.

> *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 here comes from a run executed during the study, not vendor marketing.

Arbor10Q_Q126 67-page 10-Q, 63 corrected tables

VISA_991_Q126 11-page earnings exhibit

Models ADE DPT-2; Haiku 4.5 for correction & retrieval

2 The Document-View Pipelines

For the document view, 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 call.

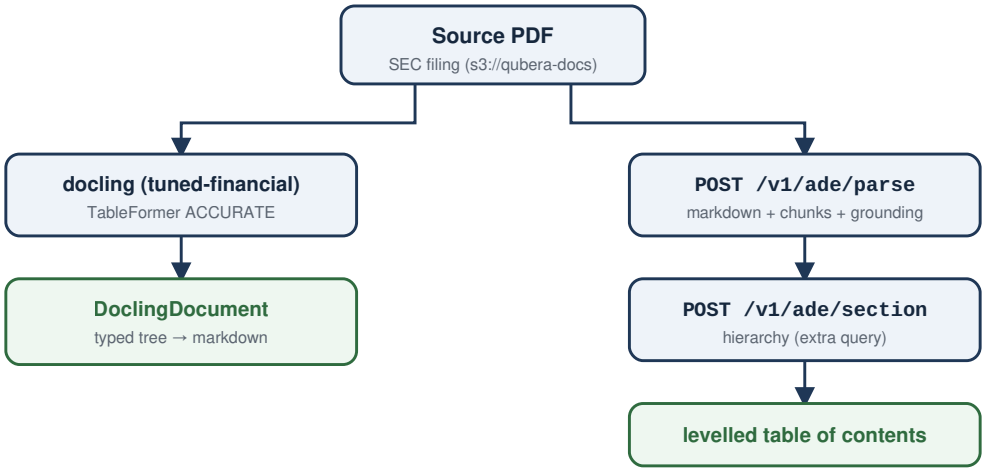

Figure 1 — The two document-view paths from one source PDF. Docling delivers hierarchy inside a single parse; ADE delivers it through a second Section call.

2.1 Docling: a typed element tree

Docling emits a tree of typed elements — `section_header`, `page_header`, `page_footer`, `list_item`, `footnote`, `picture`, form checkboxes — each with page and bounding-box provenance, lists reconstructed as groups. It is a navigable object: every footnote, or all running headers, is reachable 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`, `metadata`. The markdown is not plain GitHub-flavoured markdown — tables are HTML `<table>`, each chunk is prefixed with an `<a id=…>` anchor, and figures use a `<::…::>` delimiter. The chunk type vocabulary is coarse, from the DPT-2 ontology.

> **Provenance note.** ADE chunk types observed on Arbor were `text`, `marginalia`, `table`, `attestation`. There is no heading chunk type — headings exist only as markdown # levels.

3 Document Structure: Section versus Docling

The sharpest document-view difference is hierarchy. Against the source PDF, ADE's Section query reconstructs the document's nesting; docling flattens it.

3.1 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 in the right column. ADE Section reproduced that nesting; the lineage below is its actual output for the document head.

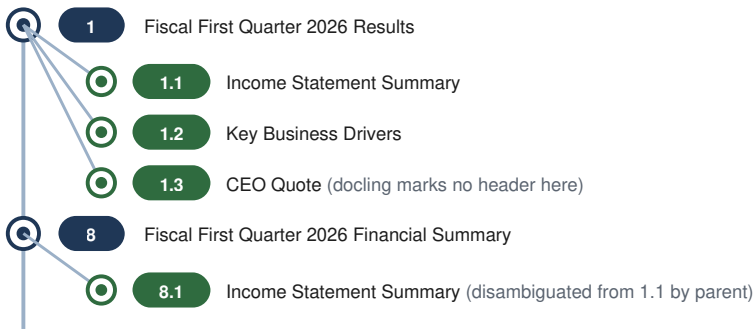

Figure 2 — ADE Section output for the Visa head (navy = level 1, green = level 2). The same title appears under two parents and is told apart by section number; docling lists both as flat, indistinguishable headers.

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 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`	reading-order tree position
Duplicate-title disambiguation	by section number (1.1 vs 8.1)	none

3.2 Element typing

Docling's advantage is the inverse: a finer element vocabulary. On the same Visa document it separately typed **16** section headers, **32** list items in groups, **21** pictures, **4** footnotes, and page headers/footers. ADE's coarse set folds footnotes and bullets into plain text, so it cannot answer “give me every footnote” from chunk types alone. For a typed element inventory docling is richer, and it is already produced at no extra cost.

4 The Quber Table Workflow

The `quber table` command runs the Set-of-Mark engine, where the page image is the arbiter. A vision grid-locator names every table and its region; Camelot extracts each one constrained to its box, so it can neither shatter one table into many nor merge many into one; a grounded structure-correction model then cleans headers, spans, and merged cells *without altering a value*. The result is one `ExtractedTable` per visual table, which fusion grafts into the docling spine.

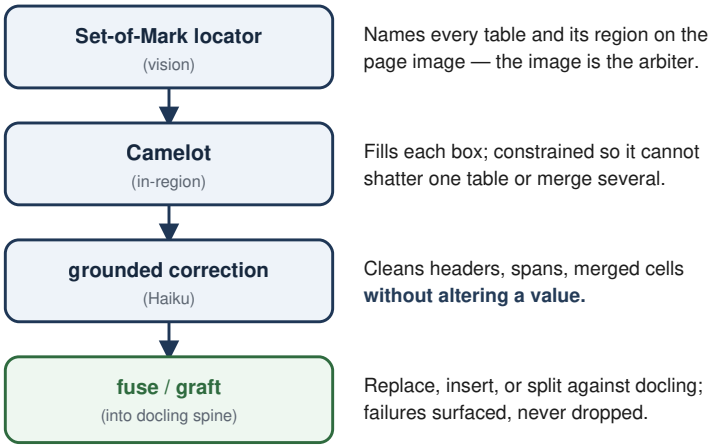

Figure 3 — The `quber table` Set-of-Mark workflow. Each stage is constrained by the one before it, and the final fusion reconciles against docling.

An `ExtractedTable` is more than a body: it carries the vision-read title, subtitle, scale/currency `units`, footnote text and footnote reference markers, a `content_region` (the corrected content extent), a `kind` (`text_table`, `chart`, or `image_table`), and an `extraction_record` auditing how it was produced.

5 Coverage

On raw table detection the three systems agree to within one table.

Document	Quber (Set-of-Mark)	Docling	ADE (table chunks)
Arbor10Q (67 pp)	63	63	64
Visa (11 pp)	—	11	10

Detection parity is not the whole story. The quber workflow adds a reconciliation layer that ADE has no equivalent for: it matches the vision-located tables against docling and assigns every region an explicit verdict — `replace`, `docling_miss` (a table docling missed, inserted), `docling_undercount` (docling merged or dropped, split apart), `som_merged` (one region docling saw as several, split), `image_table` (a table rendered as an image), `chart` (a picture mistaken for a table), and `som_miss` (a table docling found but the vision pass did not).

> **THE COVERAGE PROPERTY THAT MATTERS**
>
> The quber workflow never silently drops a table. A located-but-unread table, a docling table with no vision match, an empty body — each is emitted as an explicit error, not omitted. ADE reports completeness only at page granularity (`failed_pages`, empty on the 67-page Arbor parse); it has no per-table "I saw a table here but could not read it" signal.

6 Accuracy and Provenance

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 and comparing only digits and signs.

Statement (Arbor10Q)	Rows compared	Numeric mis-matches
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 sign, parenthesised negatives included. The differences were cosmetic: `$` placement and the rendering of zeros as `—`, `-`, or an empty cell. No parsing errors on either side.

> **ACCURACY IS A TIE; PROVENANCE IS NOT**
>
> The values match, but the two arrive there differently. In the quber workflow each cell's value comes from Camelot and the correction model is forbidden from altering it; any text-layer fill is recorded with its origin. Every figure is traceable. ADE's table is a single model read — accurate here, but opaque: no per-cell provenance, no guarantee a figure was not silently normalised. For a financial-liability posture, that traceability is quber's distinguishing property, independent of the matching accuracy.

On the earlier "value differences." Those compared docling *raw* markdown to ADE — the side quber discards. Against the corrected values quber keeps, the numbers are identical.

7 Representation and Attribution

The two table outputs differ on two axes that pull in opposite directions: the body's structural fidelity, where ADE leads, and the financial attribution around the body, where quber leads.

7.1 Body structure: pipe versus HTML

Quber serialises a corrected table as a pipe grid; ADE serialises it as HTML. A pipe grid cannot carry a column span, a row span, or a multi-row header, so a full-width banner inside a table degrades. The Changes-in-Equity statement stacks two periods, each under a banner row.

Figure 4 — 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> ...`
		A banner spanning all 10 columns. Pipe cannot express a span, so it breaks into junk cells. HTML keeps the period boundary intact.

Quber's corrected pipe keeps the banner as a ragged one-cell row — cleaner than docling's shredding, but still a flat grid where the corrector collapses every cell to a single span. ADE's HTML preserves the span natively. For body structure, ADE's representation is richer.

7.2 Financial attribution

The inverse holds for everything *around* the body. An `ExtractedTable` types the scale/currency cap tion, the footnote text, the footnote reference markers read off the image, and the corrected content extent — the attribution a financial table depends on. ADE's table chunk is the HTML body plus grounding; it does not separately type any of these.

units scale / currency caption

footnote_refs superscript markers, read by vision

content_region corrected content extent

kind text_table / chart / image_table

8 Retrieval Reliability

A richer body is only worth its cost if a model reads it more reliably. We tested directly: does an LLM retrieve a value more reliably from ADE's nested HTML than from quber's corrected pipe? The probe was the Arbor Changes-in-Equity table — row labels like *Net income* recur in both period blocks, so a correct answer must resolve period, row, then column. Ten such lookups, three representations of the same table, five runs each at temperature 0.7, scored by exact match against quber's validated values.

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%

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, this is the expected case, not a toy. 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.

PART III · ASSESSMENT

9 ADE Features Not in Use

ADE ships more than Parse and Section. The rest we are deliberately leaving on the shelf for now, for the reasons below.

Feature	Status	Why little opportunity now
Extract	GA	Schema-based field pull, but field-level grounding is undocumented — a value with no provenance back to the page is a liability gap for filings.
Classify	Preview	Per-page categorisation; vendor-marked not-production, and our inputs are single filings of known type.
Split	Preview	Separates multi-document packets; our inputs are single filings, so nothing to split.
custom_prompts	GA	Controls figure descriptions only; our documents are table-heavy with few figures.
DPT-2 mini	Preview	Lightweight, English digital-native only; not for production financial documents.
Chunk images / Visualize	GA	Review and debug aids; `quber table --review` already emits a before/after HTML and an annotated PDF.

The one latent opportunity is deeper use of Section: its `start_reference` can tie a section to the table beneath it, giving each table its governing section path. That is unbuilt, and needs a small follow-on match because the reference anchors to a section's heading chunk, not its table.

10 Cost

The two systems have different cost models. ADE is a per-page metered SaaS; every response reports `credit_usage`, so its cost is measured exactly.

ADE 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; Section added roughly 38 percent on top of Parse for the Visa exhibit. The quber stack carries no per-page SaaS charge: docling and the Set-of-Mark locator run on CPU, and the only metered cost is the Haiku correction pass. Docling's hierarchy and the vetted tables both come out of compute you already run.

> **Completeness signal.** The 67-page Arbor parse returned `failed_pages: []` — no silent page gaps, the property that matters most on a filing.

11 Findings and Recommendation

The systems are complementary across both axes. The scorecard:

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 detection (coverage)	Tie	63 / 63 / 64 on Arbor
Failure surfacing	**Quber**	per-table verdicts vs page-level only
Table value accuracy	Tie	135 rows, 0 mismatches
Value provenance	**Quber**	per-cell traceable vs opaque read
Body structure (spans)	**ADE**	HTML keeps spans; pipe flattens
Financial attribution	**Quber**	units / footnote_refs / content_region
LLM retrieval reliability	Tie	100% / 100% / 100%
Cost of structure	**Quber**	no per-page SaaS credit

RECOMMENDATION

ADE earns its place in exactly one role today: the **document hierarchy view**, where Parse + Section is the more faithful model than docling's flat headers, for the ~38% Section surcharge. On tables, the quber Set-of-Mark workflow should stand: it matches ADE on detection and value accuracy, and beats it on the two properties a financial filing demands — per-table failure surfacing and per-cell value provenance — while adding the units, footnotes, and content-extent attribution ADE does not type. ADE's one table edge, native HTML spans, did not translate into better LLM retrieval on clean equity tables.

OPEN · KNOWN GAPS

Section's `start_reference` anchors to a heading chunk, not the table beneath it. ADE Extract's field grounding is undocumented; Classify and Split are vendor Preview. The retrieval null holds only for clean, isolated tables — degraded and embedded cases are untested.

ARTIFACTS

Where the evidence lives

Generated during this study · 29 June 2026.

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

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