QUBER · DOCUMENT & TABLE EXTRACTION STUDY

ADE versus the Quber Stack

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

Quber runs two systems today. Docling parses a whole filing. The Set-of-Mark / Camelot workflow extracts tables. This study measures landing.ai's Agentic Document Extraction (ADE) against both. Part I compares how docling and ADE parse the document and recover its section hierarchy. Part II compares the table workflow on coverage, accuracy, and structure. Part III lists the ADE features we are not using and gives the recommendation. Every figure comes from a real run against two SEC filings.

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` workflow against ADE's table output. Nothing here proposes changing the production table pipeline. ADE is assessed as a candidate, not a replacement.

1 Context and Scope

Quber processes a filing through two separate commands. ADE is measured against both, so this section states what each command does.

1.1 The two quber commands

`quber document` runs docling under the tuned-financial configuration. It reads the whole filing and writes two things: a JSON tree of typed elements, and a markdown export. Together these hold the headings, prose, and tables as docling sees them, each with its page position. ADE `Parse` and `Section` are compared against this output.

`quber table` runs the Set-of-Mark workflow. It locates each table on the page, extracts it with Camelot, and corrects its structure, producing one table per visual table on the page. ADE's table chunks are compared against this output.

> *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

1.2 Test documents

The tests use two SEC filings: the 67-page Arbor Realty Trust 10-Q (`Arbor10Q_Q126`, 63 tables) and the 11-page Visa earnings exhibit (`VISA_991_Q126`, 10 tables). Both are dense and table-heavy, the conditions the application is built to handle. ADE runs on its DPT-2 model. Haiku does the table correction and the retrieval scoring. Every figure below comes from a real run against these two filings.

2 Parsing the Document

Docling and ADE both turn a PDF into a structured document, but the shapes differ. Docling produces a tree of typed elements from a single parse. ADE produces a flat list of chunks plus a full-document markdown, and its parse carries no section hierarchy. Recovering the hierarchy takes a second call, to `Section`. The ADE path therefore requires two API calls.

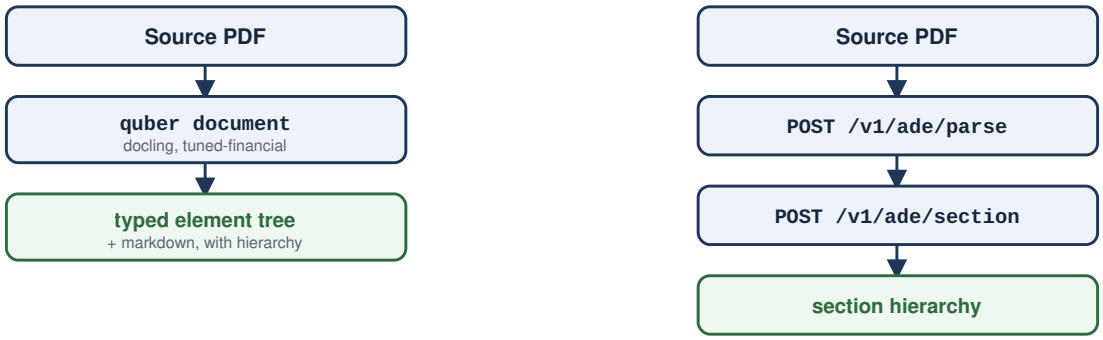

Figure 1 — The two parse pipelines. The same PDF is submitted to each; they do not share a parse. Docling returns the hierarchy in one parse. ADE needs a second call, to `Section`, to produce it.

2.1 Docling: a typed element tree

Docling returns a tree of typed elements — `section_header`, `page_header`, `page_footer`, `list_item`, `footnote`, `picture`, even form checkboxes — each with its page and bounding box. Because the elements are tagged by type, you can pull every footnote, or every running header, directly.

2.2 ADE: chunks plus a Section call

ADE `Parse` returns one JSON with five top-level keys: `markdown`, `chunks`, `splits`, `grounding`, `metadata`. Each chunk is a piece of the document — a paragraph, a table, a heading — tagged with its type and its location on the page, called its grounding. Its markdown also differs from docling's in a way that matters. Where docling emits ordinary markdown, ADE's is structured: tables come back as HTML `<table>`, every chunk is prefixed with an `<a id=…>` anchor, and figures use a `<::…::>` delimiter. That structure is what makes the table comparison in Section 7 possible, and it is the main thing that sets ADE's markdown apart from docling's.

Element-typing note. The chunk types seen on Arbor were `text`, `marginalia`, `table`, and `attestation`, a small fixed set. `Parse` has no heading type, and so no section hierarchy: headings appear only as markdown # levels. The hierarchy is recovered by the separate `Section` call, which Section 3 covers.

3 Document Structure: ADE Section versus Docling

The sharpest difference in the parsed document is hierarchy. Run against the source PDF, ADE's `Section` call reconstructs the document's nesting. Docling flattens it.

3.1 Example: the Visa earnings exhibit

On page one of `VISA_991_Q126` the headline “Visa Reports Fiscal First Quarter 2026 Results” sits above two sub-headings and a CEO quote. ADE `Section` recovers that nest ing, shown in Figure 2.

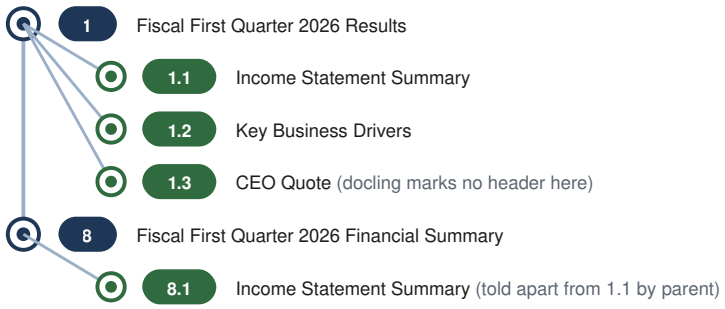

Figure 2 — ADE `Section` output for the Visa head (navy = level 1, green = level 2). The same title appears under two parents and is told apart by its section number. Docling lists both as flat, identical headers.

Docling found the same headings but assigned each the same level. Its sixteen `section_header` elements are all `level: 1`, a flat list that contradicts the visible hierarchy and cannot tell the two “Income Statement Summary” blocks apart.

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 opposite one: a finer set of element types. On the same Visa document it separately tagged section headers, list items in groups, pictures, footnotes, and page headers and footers. ADE's smaller set folds footnotes and bullets into plain text, so it cannot answer “give me every footnote” from the chunk types. For a typed element inventory, docling is richer.

4 The Quber Table Workflow

The `quber table` command runs the Set-of-Mark engine in three stages, each constrained by the one before it.

First, a vision model locates every table on the rendered page image and returns its bounding box. The page image decides where each table is.

Second, Camelot extracts each table, held to that box, so it can neither split one table into several nor merge several into one.

Third, a structure-correction model fixes what Camelot got structurally wrong: misread headers, dropped spans, mis-merged cells. The correction is necessary and constrained. The model adjusts only formatting and cell arrangement, never a value.

A final step, fusion, merges each corrected table back into the docling document, the same reconciliation quber uses elsewhere. It replaces the matching docling table, inserts one docling missed, or splits a region docling merged. A table that matches nothing docling found is reported as an error, never dropped.

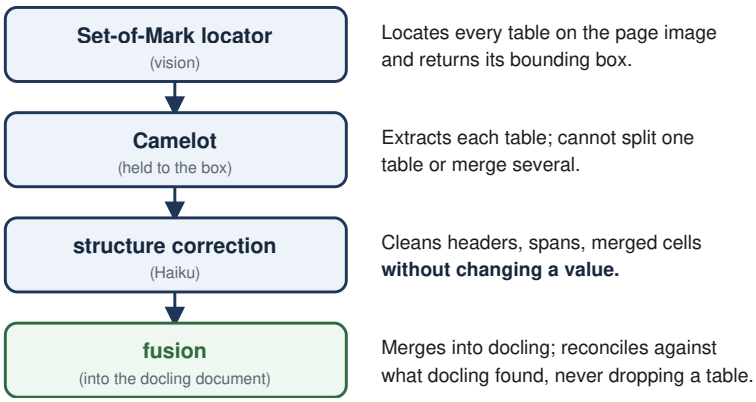

Figure 3 — The `quber table` Set-of-Mark workflow. Each stage is constrained by the one before it, and the final fusion reconciles against docling.

Each table comes out as an `ExtractedTable` object. It holds the corrected body, a vision-read title and subtitle, the scale and currency `units`, the footnote text and the footnote reference markers, a `content_region` giving the corrected extent of the table's content, a `kind` that is one of `text_table`, `chart`, or `image_table`, and an `extraction_record` that logs how the table was produced.

Units and footnotes get their own handling. The scale-and-currency caption is read once and forward-filled across split sibling tables, so a value never loses its unit. Footnote markers are read off the image, since the text layer often drops superscripts, and they key the lookup that attaches each footnote's text to the table.

5 Coverage

Coverage has two parts. Section 5.1 asks whether the systems find the same tables. Section 5.2 covers what quber does with the ones it finds.

5.1 Table identification

Two of the three columns are ours: Set-of-Mark is the `quber table` engine, docling is the `quber document` parse. The third is the count of chunks ADE tagged as `table`. The three agree to within one table on both filings.

Document	`quber table` (Set-of-Mark)	`quber document` (docling)	ADE (tables found)
Arbor10Q (67 pp)	63	63	64
Visa (11 pp)	10	11	10

The single-table disagreement in each row is the document's Table of Contents. It is technically a table, so one system counts it where another does not. It is not a financial table the tools need to agree on, so the off-by-one is expected.

5.2 Quber error detection

After extracting, the `quber table` workflow matches each table against what docling found and records how the two line up, a step ADE has no equivalent for. Every region gets one of these statuses.

Status	Meaning
`replace`	quber's table replaces the docling table in the same place
`docling_miss`	a table docling missed; quber inserts it
`docling_undercount`	docling merged or dropped tables; quber splits them apart
`som_merged`	one region docling saw as several; quber splits it
`image_table`	a table printed as an image
`chart`	a picture docling classified, not a real table
`som_miss`	a table docling found but Set-of-Mark did not

The coverage property that matters

The workflow never silently drops a table. A located-but-unread table, a docling table with no Set-of-Mark match, an empty body — each is reported as an error, not omitted. The ADE `Parse` response we inspected reports completeness only per page, through `failed_pages`, which was empty on the 67-page Arbor run. It carries no per-table read-failure field.

6 Accuracy and Provenance

We diffed quber's values against ADE's, cell by cell, across the consolidated-statement tables of both filings, normalising away formatting and comparing only digits and signs.

Filing	Statements	Rows compared	Mis-matches
Arbor10Q	Balance Sheet, Income, Changes in Equity, Cash Flows (2)	135	0
Visa	Balance Sheet, Operations, Cash Flows	92	0
Total	**two filings, eight statements**	**227**	**0**

Across 227 value-bearing rows in two filings, 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. This covers the consolidated statements. The other tables in each filing, the notes and schedules, were not diffed.

> **Accuracy is a tie; provenance is not**
>
> The values match, but how they get there differs. In the `quber table` workflow each value comes from Camelot, and the correction model is not allowed to change it, so every figure traces back to a source cell. ADE returns the table as a chunk whose cells carry no per-cell origin field. The chunk records its type, id, and page grounding, but not where each number came from. For a financial filing, where every figure must be traceable, that is quber's advantage, separate from the matching accuracy.

7 Representation and Attribution

Quber and ADE each do one thing better than the other:

- ADE's HTML preserves complex tabular structure, the spans and stacked headers (§7.1).
- Quber captures the units and footnotes around the table that ADE leaves untyped (§7.2).

7.1 Body structure: pipe versus HTML

Quber writes a corrected table as markdown, a pipe grid. ADE writes it as HTML. HTML can express what a pipe grid cannot: a header stacked across several rows, where one header spans the columns beneath it. The Arbor income statement has exactly this — "Three Months Ended March 31," sits over its 2026 and 2025 columns.

Figure 4 — A two-tier column header, rendered two ways (Arbor10Q income statement, p.5)

Quber pipe (SoM-corrected) · period folded into each column

	Three Months Ended March 31, 2026	Three Months Ended March 31, 2025
Interest income	$ 235,047	$ 240,693

ADE HTML · period spans both year columns

	Three Months Ended March 31,	
	2026	2025
Interest income	$ 235,047	$ 240,693

Neither side misreads the header. Both read it correctly. The difference is structural. ADE keeps the two tiers, so the period spans both year columns. Quber's pipe grid has no way to express a spanning header, so the corrector folds the period into each column, giving a clean but flat single row. The values are intact on both sides. Only the header's grouping is lost in pipe.

7.2 Financial attribution

Quber leads on everything around the table body. An `ExtractedTable` tags the finan cial context a table depends on: the scale, the footnotes, the true extent of the data. ADE's table chunk is just the HTML body plus its page location, and tags none of it.

`ExtractedTable` field	What it captures	In ADE's chunk?
`units`	scale / currency caption, e.g. “$ in thousands”	no
`footnote_refs`	superscript footnote markers, read off the image	no
`content_region`	the corrected extent of the table's content	no
`kind`	text_table, chart, or image_table	no
body + grounding	the table body and its location on the page	yes

8 Retrieval Reliability

The problem. A table is only useful if a model can read values back out of it. ADE's HTML body carries more structure than quber's pipe table. The question is whether that extra structure lets an LLM retrieve a value more reliably. If it does, that is a reason to prefer the HTML body.

The test. We posed value-lookup questions against three representations of the same table — quber's pipe, quber's values rendered as HTML, and ADE's HTML — scoring exact matches against quber's validated values. Two probes ran. A breadth probe covered all seven consolidated-statement tables in both filings with straightforward row-and-column lookups. A depth probe ran on the Arbor Changes-in-Equity table, where labels like *Net income* recur in both period blocks, so a correct answer must resolve period, then row, then column.

The result. Every representation read every value correctly.

Probe	Scope	quber_pipe	quber_html	ade_html
Breadth (cell lookups)	7 tables, 2 filings	126/126	126/126	126/126
Depth (repeated-label)	Changes in Equity	50/50	50/50	50/50

> **What this shows**
>
> Across seven tables in two filings, HTML structure bought no retrieval advantage over the corrected pipe, on the straightforward lookups and on the harder repeated-label case alike. The “HTML helps the LLM” idea does not hold for these tables.

PART III · ASSESSMENT

9 Other ADE Features

ADE ships more than `Parse` and `Section`. Several of the rest surfaced during this study and may have merit. The table gives what each offers and the catch.

Feature	Status	What it offers, and the catch
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.
`cus-tom_prompts`	GA	Shapes figure descriptions — potentially useful for RAG embeddings. Limited to figures, which our table-heavy filings have few of.
DPT-2 mini	Preview	Lightweight — a cost-saving option on simpler documents. Still Preview, and English digital-native only.
Chunk images / Visualize	GA	Review and debug aids; `quber table --review` already emits a before/after HTML and an annotated PDF.

The most promising 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.

Quber is not free either. It has its own LLM and GPU costs, shaped differently. The Haiku correction runs per page and per table, on the order of $0.05 per page. The GPU cost for docling is variable and depends on how it is hosted: AWS versus RunPod, and a cold, ephemeral start versus a warm one. So the comparison is not paid-versus-free but two different cost structures: ADE's fixed per-page credit against quber's per-page LLM cost plus a deployment-dependent GPU cost.

11 Findings and Recommendation

ADE and quber are complementary: ADE is the stronger document parser, quber the stronger table engine. The recommendation follows directly.

Use ADE for the section hierarchy. `Parse` plus `Section` reconstructs the document's nesting better than docling's flat headers, and that is worth the roughly 38% `Section` sur charge. **Keep the quber Set-of-Mark workflow for tables.** It matches ADE on detection and on value accuracy, and beats it on the two properties a financial filing demands, per-table failure surfacing and per-cell value provenance, while tagging the units, footnotes, and content extent ADE leaves untyped. ADE's one table advantage, native HTML spans, did not improve LLM retrieval on the tables we tested.

The scorecard behind that recommendation:

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 and 10 / 11 / 10
Failure surfacing	**Quber**	per-table statuses vs page-level only
Table value accuracy	Tie	227 rows, 0 mismatches
Value provenance	**Quber**	per-cell traceable vs no origin field
Body structure	**ADE**	HTML keeps spans; pipe flattens
Financial attribution	**Quber**	units / footnote_refs / content_region
LLM retrieval reliability	Tie	100% across 7 tables, 2 filings
Cost model	Tie	different shapes; see §10

11.1 Artifacts

The files behind every figure. ADE outputs live at `s3://qubera-extracts/<ticker>/ade/`, mirrored locally under `output/ade/`. Docling and table outputs live under `output/`. Each filename carries the document stem, `Arbor10Q_Q126` or `VISA_991_Q126`:

Artifact	Arbor10Q_Q126	VISA_991_Q126
ADE Parse	.ade.json / .ade.md	.ade.json / .ade.md
ADE Section	not run	.section.json / .section.md
Docling parse	.docling.json / .md	.docling.json / .md
Quber tables	.tables.json	.tables.json