QUBER · ADVANCED TABLE EXTRACTION

Splitting Over-Merged Tables

Refining Set-of-Mark regions that fused two tables into one

A reference for the `split_table` step: how a Set-of-Mark region that has already been identified as over-merged is split back into the separate, correctly-titled tables it should have been. Each sub-table is re-extracted from the source PDF, never from post-processed output. Assumes familiarity with the Set-of-Mark extraction workflow.

Engineering Reference · Generated 17 June 2026

Source of truth: `src/quber/core/extractors/set_of_mark/`

Contents

Audience and Scope

This reference assumes you already understand the Set-of-Mark extraction workflow. In that workflow a page-scale vision locator (`locate_tables`) marks each table region on a rendered page, `capture_table` replays region-constrained Camelot on the source PDF to recover values and runs structure correction, and the graph emits a `list[ExtractedTable]`.

It explains one thing only: how a table already identified as over-merged is refined back into the separate tables it should have been. This is not a redesign of Set-of-Mark. It is a narrow second pass that runs after Set-of-Mark has done its job. Everything upstream of the captured `ExtractedTable` stays exactly as it is today.

1 The Defect This Refines

The Set-of-Mark locator occasionally merges two or more vertically-stacked, independently-titled tables into one region. The fused region extracts with correct values but a broken structure. The two tables' period labels, headers, and footnotes get crossed or lost.

The canonical case is the Visa Q1 FY26 release (`VISA_991_Q126.pdf`), page 11. The "Three Months Ended December 31, 2025" and "…2024" Non-GAAP reconciliations were located as a single region and emitted as one fused markdown table.

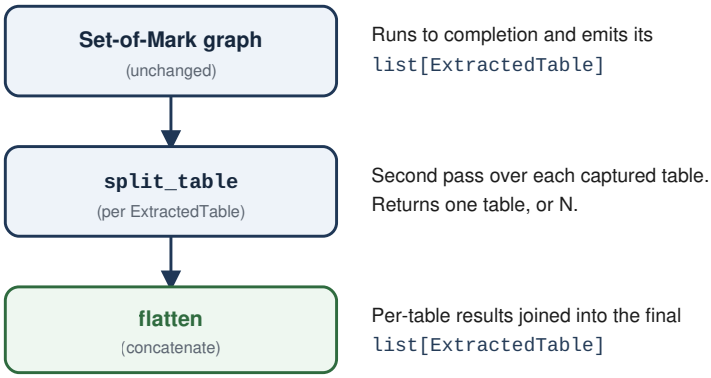
This is not a prompt bug. The page-scale grid-locator prompt already instructs the model to split stacked tables that each carry their own title and header block. At page scale it merged anyway. The fix is therefore a second, narrower pass over the already-captured table, not another tweak to the locator prompt.

The defect is rare, but on a financial filing it is material, not cosmetic. A reconciliation labelled with the wrong quarter, or shipped with its footnotes dropped, is a correctness failure with no defense.

> **THERE IS NO PRIOR MECHANISM**
>
> The "before" is simply this defect. Today an over-merged region ships as one fused `ExtractedTable` and nothing detects or corrects it. The step described here is new. There is no earlier splitting mechanism to replace.

2 Where the Step Sits: A Separate Second Pass

`split_table` is a self-contained function over a single already-captured table. It is not a graph node wedged into the middle of the Set-of-Mark pipeline. The Set-of-Mark graph runs to completion and produces its `list[ExtractedTable]`. The second pass then maps each table to one or more tables and flattens the result.



Same type in, same type out. A caller that never splits anything sees no change.

2.1 The Contract

```
split_table(table: ExtractedTable, count, boundaries) ->
list[ExtractedTable]
```

The step takes three inputs and one internal helper, and always returns a list.

Input	Meaning
`table`	The post-processed `ExtractedTable` for the region: Set-of-Mark's captured output (corrected markdown plus source, page, bbox, and som_region provenance).
`count`	The expected number of tables in the region. This is sourced upstream and handed in. docling is one possible source of that count, but the count, not docling, is the input.
`boundaries`	The per-table boundaries used as the cut seam when a split is performed. Also sourced upstream; docling provenance is one source.

A vision count probe runs inside the step. It exists only to check the supplied `count`, and its result never leaves the step.

The output is always a `list[ExtractedTable]`, in one of two shapes:

- **Split succeeded.** N re-extracted, cleaned `ExtractedTable`s, only when vision confirms N > 1 and subdivision, re-extraction, and correction all succeed. Each sub-region is re-extracted from source.
- **No-op, the default.** The input table unchanged, returned as a single-element list. This is the fallback for everything except a clean split. Vision does not confirm N > 1, the counts disagree, or a confirmed split's fix attempt fails. The region ships exactly as Set-of-Mark produced it.

2.2 Source Resolution

The step is standalone, so it does not borrow the Set-of-Mark pipeline's dependency object. It reaches the source PDF through `table.source` on the input `ExtractedTable`, not through a separate argument. The Set-of-Mark pipeline already stamps `source = str(deps.source)` on every table it emits, so the path is present.

> **Known dependency.** `ExtractedTable.source` is currently `Optional[str]` (default `None`), so the step guards for a missing source. Making the field non-nullable is tracked separately as a future change and is out of scope here.

3 The Rule Behind Every Stage: Re-Extract From Source

One rule constrains every stage below it. When isolating a merged table, we always re-extract from the source PDF. Region-constrained Camelot reads the source within a bounding box. We never re-parse, slice, or reinterpret already-extracted markdown, JSON, or rendered output to produce the split tables.

This matters because post-processed output has already lost fidelity through merged headers, dropped cells, and reflowed text. Splitting that output would compound the errors. Re-reading the source within a tighter box recovers the true cells every time.

Input	Allowed source
Values	Only ever Camelot re-parsing the source PDF, region-constrained.
Cut line (seam)	docling's per-table boxes — geometry only, never values.
Image for title / footnote inference	A render of the source PDF, used by the correction LLM for captions only, never for numbers.

> *Vision owns identity and region; Camelot owns the values inside that region; re-reading the source within a tighter box recovers the true cells — splitting post-processed output would only compound what was already lost.*
>
> — guiding principle, src/quber/core/extractors/set_of_mark/

Any future stage that needs table content obeys the same rule: target a region, re-extract from source.

4 Detection: When Two Count Signals Agree

The decision to split rests on two independent count signals.

- **The supplied count.** The expected number of tables in the region, handed in as the `count` argument. docling is one possible source of it.
- **The vision count probe.** A minimal internal call — "How many independent tables are viewable?" — returning a single integer. The model only confirms a count. It never produces coordinates. This reuses the existing `count_tables` pattern in `quber.agents.llm_client`.

The rule is a plain agreement test:

- Both signals agree on N > 1 → **split**.
- Anything else — agreement on 1, or any disagreement — no split. Camelot's extraction of the region stands.

There is no review queue and no third outcome. When the two signals disagree, Camelot is the dependable default. We keep the table exactly as Camelot already extracted it rather than guess at a split. On the Visa run, the "Contacts" block is an example. docling counted 1 and vision counted 0, so it simply stays as Camelot extracted it, with no split and no flag.

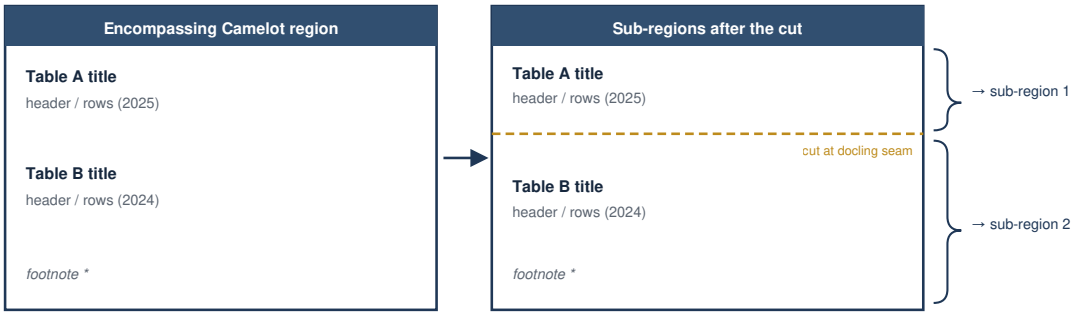
Splitting is reserved for the high-confidence case where two independent signals agree there is more than one table. Everything else defers to Camelot.

5 Subdivision: Deterministic, Nothing Dropped

When a region is marked to split, it is subdivided without losing any content.

- **Outer extent** comes from the encompassing Camelot region. Camelot is accurate. Its only miss is the subdivision itself. It captures the full bounding box correctly.
- **Cut line(s)** come from the `boundaries` input, which is docling's per-table box edges. The seam between boxes sits in the blank gap.
- **Invariant:** the union of the sub-regions equals the original region. Nothing outside a docling box but inside the encompassing region is dropped.

The invariant is non-negotiable because docling's tight boxes exclude the table footnotes. The footnotes sit below docling's box but inside the encompassing region. Using docling's boxes verbatim would silently drop the footnotes. Subdividing the encompassing region at the docling seam preserves them.



union(sub-regions) == original region — the footnote never falls outside a cut.

The cut is deterministic geometry. There are no LLM-estimated coordinates anywhere in the split execution.

6 Re-Extraction and Cleanup

For each sub-region the split produces three steps:

1. Replay region-constrained Camelot on the source PDF with `table_areas` set to the sub-region (`camelot_targeted`). Per the rule in Section 3, this is the only source of values.
2. Run the existing structure-correction step (`correct_structure` / `vet_structure`). It fixes header structure and infers the title, subtitle, and footnotes from the source-rendered image, preserving values exactly.
3. Emit one `ExtractedTable` per sub-region, each with its own `som_region` and `bbox`, its correct period label, and its preserved footnotes.

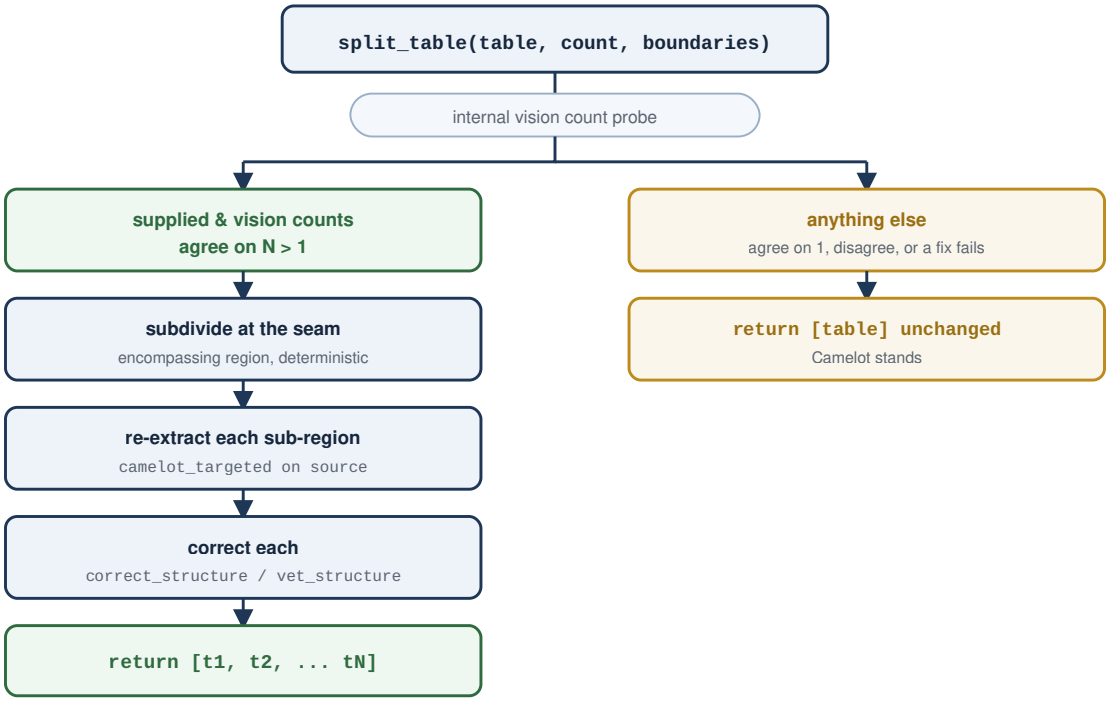
> **NO NEW EXTRACTION ENGINE**
>
> Step 1 reuses the Camelot-replay path (`camelot_targeted`). Step 2 reuses the correction path the pipeline already runs per table. The split step is orchestration over existing parts.

7 End to End: Visa Page 11, Walked Through

1. Set-of-Mark produces 10 regions for the page set. Page 11's two stacked Non-GAAP reconciliations are captured as one fused `ExtractedTable` — correct numbers, crossed 2025/2024 period labels.
2. The second pass runs `split_table` on that table. The supplied count for the region is 2 (docling saw two boxes). The internal vision probe returns 2. Both agree on $N > 1 \rightarrow$ split.
3. The encompassing Camelot region is cut at the docling seam, which lands in the blank gap between the two reconciliations. Two sub-regions result; their union is the original region, so both footnotes stay inside the cut.
4. Each sub-region is re-extracted with `camelot_targeted` and cleaned with the correction step. Out come two `ExtractedTable`s: one subtitled 2025, one subtitled 2024, each with the correct values and its own footnote.
5. The other 9 regions return a count of 1 from the probe, or disagree as the “Contacts” block does, so `split_table` returns each unchanged. Camelot stands.

8 Control Flow at a Glance

The step has exactly two outcomes.



Two outcomes only. Disagreement is not an error state. It is the ordinary "Camelot stands" default.

There is no review-routing and no separate disagreement-handling path. Disagreement is not an error state. It is the ordinary default in which Camelot stands.

9 Validation Evidence

Run against the source `VISA_991_Q126.pdf` using the Claude CLI vision backend.

Check	Result
Vision count probe, all 10 Set-of-Mark regions	Flagged exactly page 11 as 2; 8 single tables as 1; the “Contacts” block as 0. Zero false splits.
Agreement gate, all 10 regions	Split only page 11 (count 2 / vision 2); kept all others, so Camelot stands.
Subdivide without dropping	Cutting the encompassing Camelot region at the docling seam produced both tables correctly (subtitles 2025 / 2024), all values correct, with Camelot accuracy 97.4 and 98.0. Both footnotes were recovered — dropped when docling's tight boxes were used verbatim, preserved when subdividing the encompassing region.

10 Scope and Open Validation Gaps

Resolved during design review

- **Disagreement handling** — Camelot stands; no review queue (Section 4).
- **Source resolution** — via `table.source`; making it non-nullable is a separate future change (Section 2).
- **Unit attributions** — out of scope here. They belong to the general structure-correction prompt process, not the split step, and are tracked separately.

Open — validation gaps, not design choices

OPEN · UNTESTED CASES

Different-header case. All evidence to date is one document where the stacked tables had repeated headers. A different-header, same-geometry stacked pair is untested.

N > 2. The mechanism is identical for N tables, but only N = 2 has been validated.

11 Code Touchpoints

Indicative, to be confirmed during implementation.

Concern	Location
The `split_table` second pass over Set-of-Mark output (`list[ExtractedTable]`)	`src/quber/core/extractors/set_of_mark/pipeline.py`
Vision count probe	reuse `count_tables` in `src/quber/agents/llm_client.py`
docling box read / containment	docling provenance bboxes (already produced)
Camelot replay per sub-region	`camelot_targeted` in `…/camelot/correspondence/recovery.py`
Structure correction per sub-region	`correct_structure` / `vet_structure` in `src/quber/agents/llm_client.py`