QUBER — SET-OF-MARK EXTRACTION SERIES

Cell-Level Provenance and Evaluation

Every table cell traced back to its printed source, a verified status on each, and a flag when a person must look

Each cell of an extracted table ends in one of two states. Either it is traced back to its coordinates on the printed page, or it is assigned a status naming why it is not. This document specifies the data model, the workflow that assigns the coordinates, the workflow that assigns and verifies the statuses, the review artifacts, and the design decisions with their evidence. It assumes the Set-of-Mark pipeline as described in `SET_OF_MARK_DESIGN.pdf`. Vision locates each table, Camelot fills it in-region, and an LLM correction cleans structure without changing values.

Engineering Reference · Generated July 2026

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

Contents

Audience and Scope

For engineers working on the extraction pipeline and for reviewers deciding whether to trust its output. Covers the cell-provenance subsystem end to end. Table location and Camelot's own parsing are covered in `SET_OF_MARK_DESIGN.pdf`.

1 Overview

Each cell of the final corrected table ends in one of two states. A reconciled cell's text is traced back to a box on the printed page. Every other cell is assigned a status naming the observed condition that left it unlocated.

Every condition that needs a human is written to a review artifact. An auditor therefore has two guarantees. Every value traces to its printed source, and every failure to trace is recorded, never silent.

On the fourteen-document evaluation corpus, 16,173 of 16,452 non-empty cells (98.30%) are reconciled to a printed source, with zero document errors. The remainder is fully accounted for. Every unreconciled cell is classified, and an inspection agent verifies each classification against the table image. Of these, 134 cells were flagged for human review.

> *The model never produces a coordinate and never counts positions; it repeats printed labels, and a lookup resolves them.*
>
> — module docstring, `merge_grounding.py`

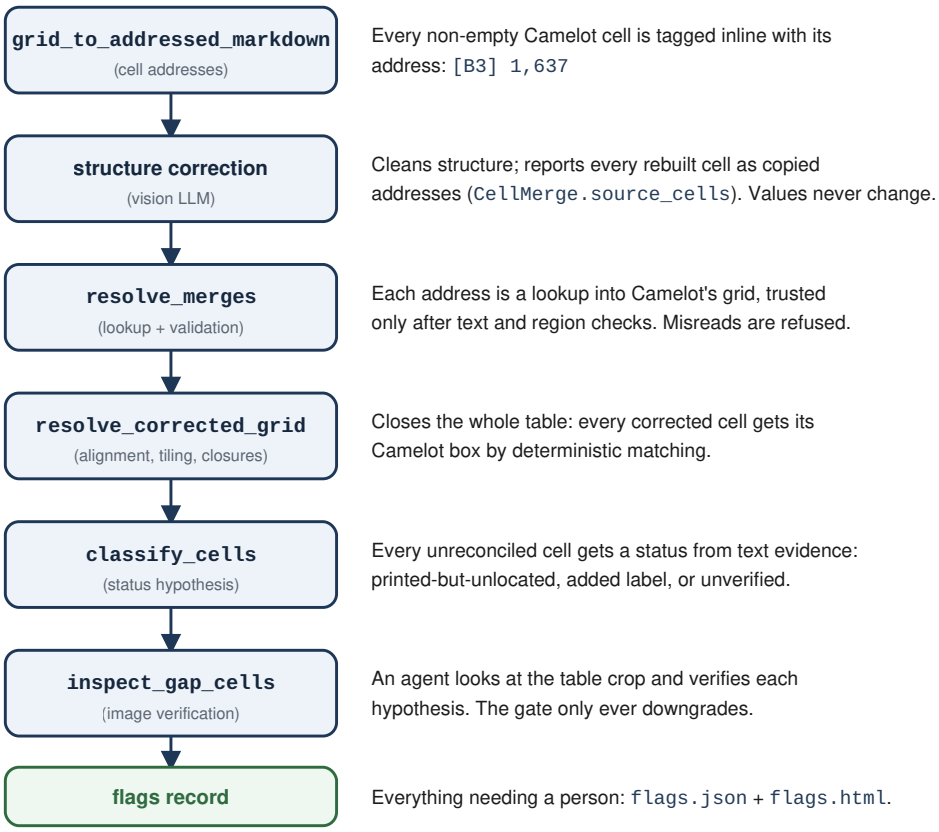

Figure 1 — The provenance pipeline. Grounding is deterministic. The two LLM agents, correction and inspection, only report and verify.

2 The Data Models

Five Pydantic models describe everything this subsystem produces:

- `ExtractedTable` — one table, raw and corrected.
- `GroundedCell` — one cell and its trace to the page.
- `CellMerge` — the agent's report of one rebuilt cell.
- `MergedCellBox` — that report after validation.
- `CellFlag` — one item for human review.

All are defined in `core/extractors/base.py` except `CellMerge`, which lives with the correction agent's schema in `agents/llm_client.py`. Each model is described below: why it exists, then what it holds.

2.1 ExtractedTable

A consumer must be able to audit an extraction without re-running it. `ExtractedTable` therefore stores the table twice: the grid as Camelot read it, and the grid after correction. The remaining fields identify the table and record what the extraction dropped.

Field	Meaning
`table_id`	Deterministic address `<doc>-p<page>-t<n>` from the document stem, page, and reading-order ordinal (`table_address`). A sub-table split from a fused region appends `-s<k>`. The same document yields the same IDs on every run.
`content_fingerprint`	First 8 hex characters of sha256 over Camelot's raw value grid (`grid_fingerprint`): identifies the table by its values, where `table_id` identifies it by page and position.
`cell_grid`	Camelot's raw grid with per-cell geometry, before correction. Values survive correction verbatim, so this stays joinable to the output.
`corrected_grid`	The corrected markdown's grid, one `GroundedCell` per markdown cell (header row 0), each with its box, status, and inspector's note.
`merged_cells`	Every corrected cell built from more than one source cell, resolved to one box per reported address plus their union.

Field	Meaning
`dropped_text`	Printed header-area text that reached no output cell. Each entry becomes a table-level review flag.
`som_region` / `content_region`	The vision locator's table boundary, and the same box with its bottom revised to the last data row, where the table truly ends.

2.2 GroundedCell

Every claim this subsystem makes is about a single cell, so each cell needs its own record. A `GroundedCell` records one table cell: the text it holds, where that text sits on the page, and what the evaluation concluded about it. The same model is used for the cells of both the raw Camelot grid and the corrected output grid.

Field	Meaning
`text`	The cell value, immutable through the pipeline.
`box`	The cell's box, normalized 0..1 with the page top-left as origin, the same frame as every other box in the table record. `None` when the cell has no traced source.
`status`	A status code naming why the cell has or lacks a box. Set on corrected-grid cells only. Validated against the status registry, so an unknown code cannot enter the record.
`note`	The inspector's one-line account of what the table image shows at this position. `None` when the cell needed no inspection.

2.3 CellMerge

When the correction rebuilds a cell, the new text no longer matches any single Camelot cell, so its geometry would be lost without a report of where the pieces came from. A `CellMerge` is that report: the result text, its row and column in the corrected output, the verbatim source texts, and `source_cells`, the printed addresses the agent read off the grid.

2.4 MergedCellBox

The agent's report is a claim, not a measurement, and is used only after every address in it is checked. A `MergedCellBox` is the validated result: one box per verified address plus their union, and a `grounded_by` state of `cell_address`, `partial`, or `none`.

2.5 CellFlag

A review item must name its own document. An earlier version rebuilt review items from log lines and assigned them to the wrong documents. A `CellFlag` is one item of the run's review queue: the source document, page, `table_id`, and title, plus the cell's position, text, status, and the inspector's note. A table-level flag for dropped text has no cell to point at, so its row and column are `None`.

3 The Grounding Workflow

3.1 Printed cell addresses

A vision model cannot produce accurate pixel coordinates. It can repeat labels it sees. The Camelot grid is therefore rendered for the agent with a spreadsheet address printed inside every non-empty cell: `grid_to_addressed_markdown` tags each cell inline, `[B3] 1,637`, with lettered columns and numbered rows.

When the agent rebuilds a cell — flattening a stacked header, rejoining a split currency symbol — it reports which addresses the pieces came from. Grounding is then a lookup into Camelot's grid, not a guess.

```
CellMerge(result="Three Months Ended December 31, 2022",
          row=0, col=1,
          sources=["Three Months Ended", "December 31, 2022"],
          source_cells=["B1", "B2"])
```

A flattened two-row header as the agent reports it: the printed addresses of both fragments, copied from the grid.

The agent must report every output cell not copied verbatim from a single Camelot cell, single-source citations included. A band label is a header printed once over several columns, replicated into each column it spans. A group label is one the correction copies into other cells. These and marker-extended labels all arrive as resolvable addresses instead of needing to be reverse-engineered later.

3.2 Resolving merge reports

The agent's merge report is a claim, not a measurement. `resolve_merges` therefore checks every claimed address before using it. Each must pass four gates before its box enters the record:

Text anchoring the text at the claimed address must appear in the merge's result

Header region a header cell may only cite sources above the grid's first value row

Position guard the corrected cell at the claimed row/col must hold the result

Full grounding only a merge with every address resolved assigns a box

Each gate exists because of a defect found on real documents. Text anchoring refuses misread addresses. The header-region check exists because header text repeats in data regions. An equity statement prints "Treasury" in both. The text check alone once anchored a header to a data cell. The last two gates came out of the coordinate readback audit (Section 8). An agent that miscounted its own output position placed a correct box on the wrong cell. A partially resolved merge produced a union box that covered only a currency symbol.

A refused address leaves the merge `partial` or `none`. Refusals never enter the record.

3.3 Closing the table

Merges cover the cells the agent rebuilt. `resolve_corrected_grid` closes everything else. To close a cell is to assign it its Camelot box, the operation the `_close_*` passes below are named for. The passes run in order, each strictly narrower than the last:

Pass	What it closes
Exact alignment	Whole-cell matches in reading order — rows first, then cells within each aligned row. A value the correction moved keeps its source cell's box. Rows with identical text map in order and never cross.
Span tiling `_tile_windows`	Camelot glues several printed columns into one grid cell and wraps labels across grid rows, so corrected cells are often *fragments* of grid text. Cells the alignment pass did not close match as ordered contiguous spans of their aligned rows' text, taking the union of the grid boxes the span touches. Skips the header region, where reading a row left to right would falsely join text from separate columns.

Pass	What it closes
Header stacks `_close_header_stacks`	A flattened label whose fragments stack within one grid column of the header region. Matched by exact concatenation of the column run. Spanning labels serve every column they span.
Unpaired rows `_close_unpaired_rows`	Two stacked printed sections the agent rebuilt side by side: corrected cells match as spans of grid rows the alignment never consumed, all-or-nothing per row.
Marker labels `_close_marker_labels`	A row label the correction extended with a footnote marker that exists only in the image ("costs(1)"): matched with the marker stripped, only for markers catalogued in `footnote_refs`.

A token that exists nowhere in the grid never acquires a box. This is the system's hard guarantee: an invented placeholder can never be traced back to the page.

3.4 Capture repair

Some gaps are Camelot's, not the agent's. Raised-baseline typography can split a totals row into two bands whose boundary falls exactly on a number's vertical center, and Camelot drops the number.

`repair_capture` closes this with a bounded loop. A deterministic detector finds region numerics absent from the grid (`dropped_numeric_keys`). An advisor agent recommends one retry knob: row tolerance, column tolerance, or flavor. A deterministic acceptor takes the retry only if it recovers every missing numeric and loses nothing.

4 The Evaluation Workflow

4.1 Deterministic classification

Every cell that could not be traced must still be accounted for, never left silent. After grounding, `classify_cells` therefore assigns every non-empty corrected cell a status. A cell with a box is `reconciled`. For the rest, the deciding test is whether the cell's text exists in the table's source. The source is the Camelot grid plus the region's text layer, joined on a sentinel character. The sentinel keeps a token from matching across the boundary between two cells. The test splits three ways:

- Text that is present classifies as printed-but-unlocated, split by row kind.
- Text that is absent but on the authorized-label set classifies as an added label, again split by row kind. `AUTHORIZED_LABELS` holds *Total*, *Item*, and *Description*, the same list the correction prompt authorizes.
- Anything else is `unverified`.

Novel conditions land in the catch-all by construction, so an uncataloged condition is surfaced for inspection and never absorbed into a known class.

4.2 Image inspection

A classified status is a hypothesis from text evidence. Whether a band really is printed, or an added *Total* really sits on a blank totals row, is visible only on the image.

The status inspector (`agents/status_inspector.py`, run by `inspection.py`) receives the table crop, the corrected markdown, and each flagged cell with its proposed condition spelled out in words. For every cell it must describe what the image shows *before* giving a verdict. The schema places `evidence` before `verdict` deliberately. The verdict is three-way: `holds`, `contradicted`, or `cannot_tell`. It is never forced to pick the nearest option.

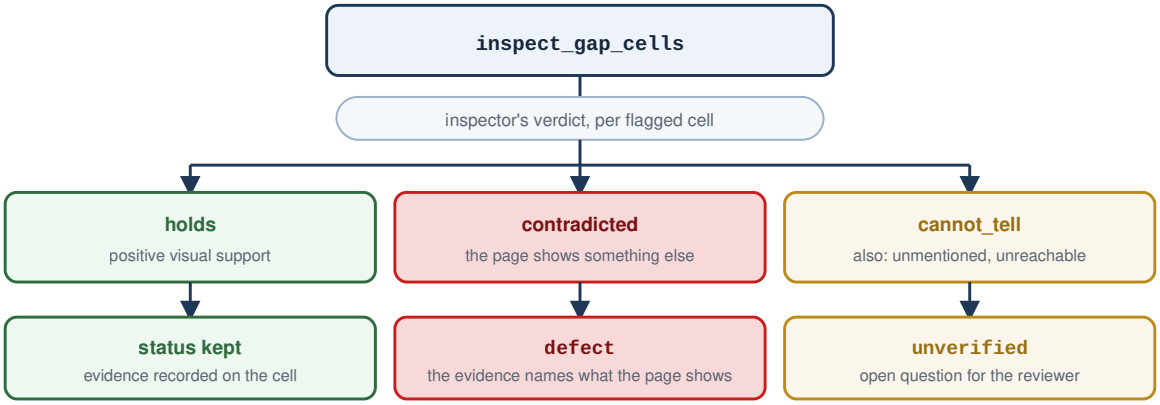

Figure 2 — The deterministic gate over the inspector's three-way verdict. The inspector reports what the image shows. The gate assigns the status.

The conditions the inspector verifies obey rules that each came out of an observed failure (Section 8). They state only claims the image can settle. The two printed-but-unlocated codes share one position-agnostic condition. An added label's condition is judged by *absence*, and it is contradicted only by a *different* printed word. The exact same text printed at an unsourced position resolves to `cannot_tell` — a review item, not a defect.

4.3 Dropped header text

One condition is not a cell at all. The correction drops a group label printed over several sub-headers (“Coverage Data”). No output cell then exists to hold a status. The correction model preserves such labels inconsistently from run to run even when instructed not to, so a prompt cannot close this.

`find_dropped_header_text` closes this deterministically: a cell in the grid's header region counts as dropped when none of its words appears in any corrected cell or in the table's absorbed metadata (title, subtitle, units, footnotes). The check is word-level because Camelot glues side-by-side stacks into single cells whose whole text exists nowhere even when every word was preserved. Each fragment is recorded on `ExtractedTable.dropped_text` and becomes a table-level review flag.

In practice the check also catches page text that Camelot swept into the grid and the correction rightly excluded: a heading, a boilerplate sentence. The flag surfaces the drop so the omission is never silent.

5 The Status Vocabulary

Every code names the observed condition, in the vocabulary of the status-case catalog (`docs/CELL_STATUS_CASES.pdf`), so a reader can tell what happened on the page without decoding project shorthand.

The codes fall in three tiers:

- **Pass** — expected behavior, nothing to review.
- **Review** — surfaced to the user when the table is consumed, but not a defect.
- **Defect** — the page positively contradicts the extraction.

Status	Case	Tier	Observed condition
`reconciled`	—	Pass	The cell's text is traced back to a printed source and takes that source's page coordinates.
`header_printed_unlocated`	1	Pass	Header or band text is printed on the page but could not be traced back to its position. Text correct, geometry missing.
`label_printed_unlocated`	2, 3	Pass	A row label is printed, usually wrapped across printed lines, but could not be traced back to its position.
`single_character`	7	Pass	One-character text (a bare `$`), excluded from location matching by design: a bare symbol would anchor inside any unrelated number.
`total_label_added`	5	Review	The totals row is printed with no label. The extraction wrote *Total*. The author could have intended something else.

Status	Case	Tier	Observed condition
`header_label_added`	6	Review	The table is printed with no header over this column. The extraction wrote a generic column name.
`unverified`	—	Review	The condition could not be confirmed. The catch-all: uncataloged failure modes land here by default.
`defect`	—	Defect	The inspector positively observed the page showing something other than what the extraction recorded.
`header_text_dropped`	4	Review	Table-level, not a cell: printed header-area text appears in no output cell.

Pass tier requires confirmation. Cases 1–3 keep their pass tier only because the inspector confirmed the condition on the image. A proposal the image cannot settle is `unverified`. One the image contradicts is `defect`.

Empty cells have no status — they have no printed mark and are not gaps.

The machine-readable registry is `CELL_STATUS_REFERENCE`: one entry per status code with a display label, a plain-language description, an `inspect` flag marking the codes a consumer surfaces for review, and the case numbers the code covers. `GroundedCell` validates its status against this registry, so a front end keying on `code` can never meet an unknown value. The prose glossary is the package README.

6 Review Artifacts and Logging

6.1 Flags artifacts

Everything a person must act on has to arrive as its own record, not be scraped from the log. Every run therefore writes its review queue to a JSON file. `cell_flags` collects every corrected cell whose status is registered for inspection, plus each table's dropped text, into `<base>.flags.json` beside the tables output. Each flag records full document identity, so it can be matched to its table without reverse attribution.

Under `--review`, `render_flags_html` writes `<base>.flags.html`, one section per table with any flagged decision. Each section shows the table as printed, the extraction output with flagged cells highlighted, and the reasoning recorded for each flag. The printed view is a page crop with the boundary drawn. Highlights are red for defects, gold for unverified, and green for added labels. The stylesheet includes print rules, so printing the HTML reproduces the paginated review document. When no table qualifies, no file is written.

6.2 Log levels

The flags record lists the action items. The log narrates the run. A log line that asks for no action is not a warning. Concretely, internal deliberations trace at DEBUG in plain words: unresolved merge sources, cells not traced back to the page, inspection downgrades. WARNING is reserved for a confirmed defect (named with the inspector's evidence) and for operational failures where inspection could not run. Everything a person must eventually see arrives through the flags artifacts, not by scraping the log.

6.3 Cloud operation

The full workflow runs against object storage so that document extraction, table extraction, and fusion can run as three separate jobs joined only by a prefix. The three commands are `quber document`, `quber table`, and `quber fuse`. Each reads the source PDF from an `s3://` URI through a persistent, ETag-validated cache (`resolve_document`) and writes all artifacts to an `s3://` prefix through `output_sink`. The sink uploads on clean exit and uploads nothing on failure. Standalone fusion reads a prior run's four artifacts from a bucket prefix through the same cache (`resolve_artifacts`).

7 Fusion

Fusion grafts each table's corrected body into the docling document spine, the unified document's ordered sequence of elements. For provenance it transfers exactly one thing per cell: geometry. `_graft_cell_geometry` copies each corrected cell's box onto the grafted `TableCell.bbox`, converted from the normalized top-left frame into docling's bottom-left page points. The graft does no re-matching. A cell the grounding stage could not locate arrives without a box and stays without one. The geometry transfers directly because `bbox` is a native `TableCell` field.

Statuses and notes deliberately stay out of the unified document. Docling's `TableCell` is a closed model with no per-cell extension point. Embedding a parallel status grid into table-level metadata would duplicate a record that already exists, and a duplicated record can drift.

Instead each grafted table binds `quber__table_id` and `quber__content_fingerprint` into its metadata. Any consumer holding the unified document joins on the ID to the same run's `tables.json` for the full cell record and `flags.json` for the review queue. One source of truth per concern.

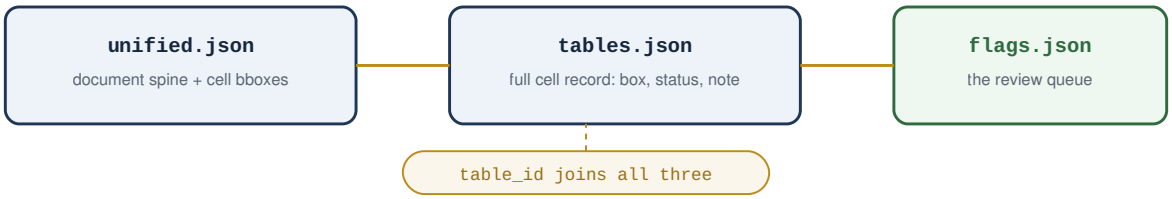

Figure 3 — `table_id` joins the three artifacts. It is deterministic across runs, so references written against one run resolve against the next.

8 Design Decisions

Each decision below is stated with the evidence that forced it.

8.1 Addresses, not coordinates

The vision model cannot produce accurate pixel coordinates, and asking for them invites invention. It can repeat printed labels. Printing an address into every grid cell turns geometry reporting into label copying, and grounding into a validated lookup.

8.2 Determinism first, agent reports only where reading order breaks

A two-table experiment, run before any full-corpus run, showed the dominant gap was not agent communication but glued Camelot grids. The worst table's five printed columns arrived as a 55×2 grid. Body rows survive correction in reading order, so the text itself proves the mapping and deterministic alignment suffices. Headers are the one place the agent rebuilds out of reading order. Merge reports are needed for grounding only there.

8.3 Report everything, resolve one way

Early versions required reports only for multi-source merges and grew a closure heuristic for each unreported pattern. The current design replaced that: the agent reports *every* non-verbatim cell, and one resolution path handles them all. Closures remain only for patterns with no reportable source.

8.4 Audit location truth, not just presence

Every coverage metric counts a wrongly placed box as covered. The coordinate readback audit reads the PDF text at each box's coordinates and compares it to the cell's value. It is the only check of location truth. It found two real defect classes: a correct box placed on the wrong cell via the agent's miscounted position, and a shrunken partial-merge union boxing only a `$`. Both are now refused by gates in `resolve_merges`. It also restated coverage: the 99.1% read before the audit was inflated, and 98.7% survived it.

8.5 Faithful provenance beats cosmetic repair

The agent may label a column from words printed in the title line. The box then points at the title cell. That is where the text genuinely came from, so it is recorded as-is.

8.6 Box granularity is the glued Camelot cell

When Camelot glues several printed values into one cell, that cell is the genuine printed source of each value. Boxes resolve to the glued cell, not to invented per-value subdivisions.

8.7 A classified status is a hypothesis until an agent looks

The classifier's text evidence cannot see the page. Assigning a catalog case by pattern — “an unlocated header cell is case 1” — would absorb conditions without inspection. Every flagged cell is therefore verified against the table image. The gate over the verdict only ever downgrades: nothing an agent says can move a cell to a better status.

8.8 Inspection conditions are image-checkable claims

Five observed failures each forced a revision of the condition texts. An internal-mechanism clause (“could not be located in the text grid”) made the model dispute the unverifiable part and contradict text it could see. Conditions now state only what the image can settle. A position-specific condition wording let a row-kind misclassification produce a false defect. Both printed-but-unlocated codes now share one position-agnostic condition. The model read “does the condition hold” as “is the word printed” and refuted 87 of 96 added-label proposals whose own evidence described the qualifying situation. An added label's condition is now judged by absence, which is evidence *for* it. Data sitting under an inserted header row was read as contradiction. An added label is now contradicted only by a *different* printed word at its position. The exact same text printed at an unsourced position was escalated to defect. It now resolves to `cannot_tell`, because the extraction is right and only the record's sourcing is open.

8.9 Status codes name the observed condition

The first vocabulary (“supplied”, “measured”) described what the software did, divorced from what a reader sees. Codes now state the condition in the case catalog's terms: `reconciled`, `total_label_added`, `header_text_dropped`. The two added-label cases got separate codes with their own inspection conditions.

8.10 Action items live in the flags, not the log

Review items used to be reconstructed from page-number log lines, which mis-assigned documents. Flags are now written as their own artifact, and each flag records its source document, page, and `table_id`. The log was re-tiered around them: WARNING only for what a person must act on.

8.11 Join, do not embed

With a deterministic `table_id` shared across all artifacts, embedding the cell statuses into the unified document would duplicate the record into a model with no field for it. Geometry grafts because docling has a field for it. Everything else is reached by joining on the ID.

OPEN · KNOWN GAPS

The split re-extraction path (sub-tables cut from a fused region) runs neither capture repair nor status inspection. Its cells classify but are not image-verified. Added-label behavior varies run to run. On re-extraction only 8 of 32 flagged tables reproduced their flags. That is why added labels are review tier, not defects.

9 Results

Results are from the fourteen-document evaluation corpus (earnings releases and filings), re-run end to end after each mechanism landed. Coverage counts non-empty corrected cells reconciled to a printed source.

Mechanism added	Coverage	What closed
Address grounding + exact alignment	91.6%	Baseline: merges by address, whole-cell matches.
Span tiling	97.1%	Fragments of glued grid cells.
Placeholder-dash removal	97.2%	Invented em-dashes blanked; printed nil dashes kept.
Capture repair	97.5%	Totals-row numbers Camelot dropped.
Header closures	98.3%	Stacked headers, cross-section rebuilds.
Widened reporting + marker strip + audit gates	98.7%	Band and spanning labels; marker-extended labels; two wrong-box classes refused.

The final captured run is the one whose artifacts the review documents are built from. It reconciled 16,173 of 16,452 cells (98.30%), with zero document errors and 134 review flags. The flags break down as 96 added conventional labels, 21 defects, and 17 unverified cells. Most of the defects are column headers the correction invented where the page prints a units caption instead.

Coverage dipped from 98.7% because the prompt now authorizes conventional labels. Each such label has no printed source to trace back to, by design, and arrives flagged. It is not lost geometry.

This run predates the vocabulary rename, so its artifacts use the older code names.

Interpreting the coverage figure. 98.30% is the share of cells reconciled to a printed source. The complement is not error. It decomposes into confirmed printed text that could not be traced back to its position (pass tier), authorized added labels (review tier), and the 38 defect-or-unverified cells above. Each is accounted for, none silent.