QUBER RESEARCH — EXTRACTION PIPELINE

The Docling Accuracy Ceiling

Our installation against the latest release, and the most accurate configuration docling can express with unlimited GPU compute

docling is the open-source document parser our extraction pipeline is built on. We run it at 2.113.0 under the tuned-financial preset, our production configuration, and its handling of scanned pages, stamps, and charts has been a persistent disappointment. This document compares our installation with the latest release. It lists the settings docling offers today that change accuracy: OCR engines, layout models, the VLM (vision-language model) pipeline, and chart extraction. It then measures what our own corpus can measure. We ran chart extraction on a real earnings deck and scored it value by value against a verified ground truth, on our pinned version, the latest version, and the strongest layout model; and we scored every other route to the same values — the text layer, full-page OCR with two engines, native and rasterized. The ceiling of the title means the best accuracy docling can be configured to produce. The reader is assumed to know our pipeline at the level of `src/quber/core/parsers/docling_parser.py`.

Engineering Reference · Generated 2026-07-31, revised 2026-08-01 with measured results
Primary sources: `docling-project.github.io`, `github.com/docling-project`, measurements in this repository · QUE-322

Contents

Audience and Scope

Written for engineers working on quber's extraction pipeline. The reader knows what the tuned-financial preset enables and where docling runs in our pipeline. Claims about docling's current state are sourced from its official documentation, changelog, and source code as of July 31, 2026. Claims about our installation were verified by reading and importing against the pinned 2.113.0 in this repository. Chart-extraction results are our own measurements. We ran them July 31–August 1, 2026 on this repository's RTX 3090 (CUDA 13.0) against the LFT Q1 2026 earnings supplemental — Lument Finance Trust's quarterly earnings deck, a document our pipeline already processes — and scored them against the 68 chart values read and verified during the figure-scanning work (section 4). Any claim weaker than a cited source or a measurement is labeled in place, as "our synthesis" or with its uncertainty stated in the sentence itself. Source URLs are collected in section 10 and cited by number.

1 The Findings That Matter

Eight findings organize everything below. They mix our own measurements — run on this repository's hardware for this document — with claims sourced from docling's documentation and code; each measurement is identified as ours where it appears.

- **We measured docling's chart extraction on a real deck. 61 of 68 verified chart values came back with correct labels, and nothing in the output signals which 7 went wrong.** The `do_chart_extraction` flag is off by default and unset in our preset. We ran it on the LFT Q1 2026 earnings supplemental against the 68 values our figure-scanning path — the shipped stage of our own pipeline that reads each chart on a nominated page with a vision service and records its values — had read and verified on the same deck. It attributed Georgia's $96.4M to a nonexistent "CA" exposure, dropped every donut chart's printed center total, merged two donuts into one table, and fabricated 24 values that are contradicted by the printed page — all without any confidence signal. The flag is useful only behind reconciliation. It does not replace the figure-scanning path, which read 68 of 68. Section 4 has the full measurement, and the region-level failures our earlier figure-scanning work already documented.
- **The chart values were never unreadable — full-page OCR with the engine we already run reads all 68, with zero wrong numbers, even with the text layer stripped, on CPU.** RapidOCR read 68 of 68 on the native pages and 68 of 68 on 300 DPI rasterized copies, inventing no values either time, and it did so on the CPU execution provider. The production parse already holds all 68 as bare picture-child fragments. So the gap on chart values is attribution — which number belongs to which label — never character capture; that is the gap the figure-scanning path fills, and the gap an OCR grounding well with per-fragment boxes can ground. Section 4.4.
- **NVIDIA's Nemotron OCR runs on our hardware today, but it read our deck less accurately than the engine we already run.** The blocker to running it is a docling version gate, not NVIDIA support: the nemotron-ocr 2.0.2 wheel installs on Python 3.13 and docling's docs say 3.11–3.13, but docling's runtime check refuses everything except exactly Python 3.12, in the current release and on main. In a 3.12 environment on our RTX 3090 (CUDA 13.0) it ran end to end — and across the six chart pages it captured 66 of 68 values on native pages and 67 of 68 rasterized, and corrupted four printed numbers, against RapidOCR's 68 of 68 and 68 of 68 with none. It is 1.7x faster and takes 19.1 GB of VRAM against RapidOCR's CPU execution. On this evidence it is not a candidate for accuracy work. Section 3.

- **We are 4 releases and 16 days behind, and the upgrade changes less than the changelog suggests.** 2.116.0 added configurable layout-driven OCR modes and 2.117.0 made the OCR render scale configurable. Both aim at the subsystem we are unhappy with, and neither is tested on our corpus. On chart extraction, measured, 2.117.0's output is identical to our pin's. A fresh 2.117.0 install also pulls in a transformers version that crashes the chart model. Sections 2 and 3.
- **The strongest on-domain accuracy number anywhere in docling is the VLM pipeline's.** Granite-Docling-258M scores 0.97 on FinTabNet table structure, against 0.82 for its predecessor. FinTabNet is a financial-table benchmark, and TEDS is the 0-to-1 table-similarity measure it is scored by. No official comparison against the classic pipeline exists, so this is a lead to measure, not a result. Section 5.
- **Unlimited GPU raises throughput everywhere and accuracy in only three places:** a stronger layout model, the VLM pipeline, and the chart model. TableFormer, docling's table-structure model, does not use GPU batching at all, and OCR gains depend on the engine, not the card. On our deck the stronger layout model fixed the donut merge and misread a bar value. Sections 6 and 7.
- **There is no documented maximum-accuracy recipe.** The pieces exist — the heron-101 layout model, ACCURATE tables, chart extraction, the VLM enrichments — but no doc or maintainer statement assembles them. The configuration in section 7 is our synthesis and is labeled as such.
- **Docling reads SEC XBRL natively.** XBRL is the machine-readable data format SEC filings are published in alongside the PDF. A 10-K's XBRL converts to a DoclingDocument with tables and key-value facts — a second, already-structured copy of the numbers in exactly the filings we parse from PDF today. Section 8.

2 Where Our Installation Stands

Our pin — the docling version this repository's dependency file fixes — is close to current. We run 2.113.0, released July 14, 2026; the latest is 2.117.0, released July 30, 2026 — 4 releases and 16 days apart [1, 2]. The changes in that window that matter to us:

Release	Change relevant to quber
2.114.0	Native chart parsing for Word documents — PPTX chart parsing is already in our pinned 2.113.0, Excel landed in 2.111.0–2.112.0. Office formats, not PDF, so not our path.
2.115.0	Chart extraction became lazy-loaded, so an installation without torch can skip installing the chart model's dependencies — a packaging change, no behavior change for us. Also added per-file CLI progress output, which we do not use.
2.116.0	Layout-driven OCR pipeline with configurable OCR modes — the `OcrMode` enum in section 3.
2.117.0	OCR render scale configurable rather than hardcoded — the resolution at which pages are rasterized for OCR is now ours to set.

What we run today, from `docling_parser.py`: the tuned-financial preset sets `TableFormerMode.ACCURATE`, the DoclingParse backend, `do_ocr=True`, cell matching (mapping predicted table cells back to the PDF's own text), picture classification, `generate_picture_images`, CUDA, and `page_batch_size=32`. On GPU hosts the preset names an OCR engine explicitly. `TunedFinancialParser` sets `RapidOcrOptions` with the onnxruntime CUDA execution provider (`docling_parser.py:165-178`). It does so because docling propagates the CUDA device to RapidOCR's paddle and torch engines but not to the default onnxruntime engine. Only CPU hosts fall through to docling's default `auto` engine selection, described in section 3.

Two capabilities we do not use are already installed. We verified that 2.113.0 contains the chart-extraction options (section 4) and the higher-capacity layout model specs (section 7). The `OcrMode` enum is absent from 2.113.0, consistent with the changelog placing it in 2.116.0.

The upgrade is small and targeted, with one measured trap. The window covers four releases and sixteen days with no documented breaking changes, and two of

the four releases change the exact subsystem we are dissatisfied with. A fresh 2.117.0 environment resolves transformers 5.14.1, under which the chart-extraction model crashes (`create_causal_mask() got an unexpected keyword argument 'cache_position'`). Pinning transformers to 5.8.0 fixes it, and that is the version our repository already resolves. Measured in this repository on August 1, 2026.

3 OCR

Docling makes two separate OCR choices: which engine reads the text, and which regions of the page are sent to that engine. Both gained new options after our pinned 2.113.0.

3.1 Engines

Eight engine option classes exist in current source, each selected by its `kind`; the table groups the two Tesseract variants into one row [3]:

Engine (kind)	What it is
`auto`	The default. The docs describe it as EasyOCR when a GPU is present and Tesseract when none is [3]. The installed 2.113.0 code does something different. In `auto_ocr_model.py` on Linux it tries Nemotron if installed, then RapidOCR on onnxruntime, then EasyOCR, then RapidOCR on torch. Tesseract is not in the chain at all. Its language list is deliberately empty, and the docs say to name an engine explicitly when language must be controlled.
`easyocr`	The docs' stated GPU default. We do not run it. The tuned-financial preset names RapidOCR explicitly on GPU hosts (section 2).
`tesseract` / `tesserocr`	Tesseract via CLI or bindings.
`rapidocr`	The one engine the GPU docs confirm working with GPU batching (torch backend) [8].
`ocrmac`	macOS only; irrelevant to our Linux workers.
`nemotron-ocr`	NVIDIA's `nvidia/nemotron-ocr-v2`, the newest engine. By default it groups recognized words into sentences, the granularity its docs say "maps most directly to Docling OCR cells" — the per-region text fragments docling stores [5].
`kserve_v2_ocr`	Remote OCR served over KServe.

Nemotron's published result is a throughput claim with accuracy held flat, not an accuracy gain. Across 19,252 pages, average throughput rose 60%, with a best case of +112% on French finance documents. Accuracy was "maintained or improved" on ViDoRe v3 [6].

ViDoRe v3 is a document-retrieval benchmark, so it asks whether OCR'd documents are still retrieved correctly rather than reading text quality directly.

We ran Nemotron on this repository's hardware. One version gate blocks it, and that gate is docling's, not NVIDIA's. The `nemotron-ocr` 2.0.2 wheel installs and imports on Python 3.13 (cp311/cp312/cp313 wheels on PyPI, `requires-python >=3.11,<3.14`). Docling's OCR concepts page says v2.0.2 supports 3.11–3.13 [5, 20]. But docling's own runtime check refuses anything but exactly Python 3.12. That check is `validate_runtime` in `nemotron_ocr_model.py`, identical in released 2.117.0 and on main. On a Python 3.13 environment the engine raises `RuntimeError: Nemotron OCR requires Python 3.12` before loading. On a Python 3.12 environment on this repository's RTX 3090 (CUDA 13.0) it ran end to end.

Run end to end, it read our deck less accurately than the engine we already use. Measured over the six LFT chart pages against the 68 verified chart values (the section 4 ground truth), full-page Nemotron captured 66 of 68 on the native pages and 67 of 68 on 300 DPI rasterized copies, and it corrupted four printed numbers across the two conditions: ($46.8) became (44.8), $99,500 became $99.500, 88.1% became 88.7%, and ($10.5) became (110.5). Every corruption is a well-formed, plausible value. RapidOCR full-page on the same pages captured 68 of 68 in both conditions and corrupted nothing (section 4.4). Nemotron was 1.7x faster (16.3–16.9 s against 27.5–30.1 s for the six pages, model load included) and held 19.1 GB of VRAM, while the RapidOCR runs executed on the CPU provider. Its published claim — throughput up, accuracy held — matches what we measured on speed and fails against our own engine on accuracy. Adopting it would also mean a Python 3.12 worker image until docling relaxes the gate its own docs already contradict. On this evidence there is no accuracy case for it.

The model catalog also lists SuryaOCR (90+ languages, "good for complex layouts"). No option class exists in docling itself; it loads from the separate `docling-surya` plugin package, which is GPL-licensed — a license review precedes any use [7].

3.2 Where OCR runs on the page

The `OcrMode` enum from 2.116.0 decides which regions are rasterized and read. It has four members — the three operational modes below plus `DEFAULT`, the field's default value, which the source comment says is currently wired to run `PDF_AWARE_LAYOUT_REGIONS` [3]:

- `full_page` — the whole page through OCR, ignoring PDF text. The docs recommend it "when layout extraction is unreliable or the PDF contains scanned pages", at a speed cost [4].

- `layout_regions` — OCR only where the layout model found content; no PDF text information used.
- `pdf_aware_layout_regions` — like the above, but skips regions that are entirely native PDF text. The default in current releases; our 2.113.0 predates `OcrMode` entirely and runs the older hardcoded behavior.

The upgrade adds two OCR settings that did not exist at our pin — region mode, and the render resolution from 2.117.0's configurable render scale — and gives us a reason to revisit a third, engine choice, which existed but we never exercised. Our scanned-page and stamped-page failures should be re-tested against `full_page` mode at a raised render scale before we conclude anything about the best OCR docling can produce.

4 Charts and Figures

`do_chart_extraction` converts bar, pie, and line charts into structured tables, and it is off by default — including in our preset. Enabling it auto-enables picture classification, which we already run. The extracted data is stored on the picture item itself, at `PictureItem.meta.tabular_chart.chart_data`, as table cells with row and column offsets [9, 10]. We ran the flag on a real deck and scored its output value by value, below.

4.1 What our own history already showed

We have exercised docling's chart and figure handling before, and it failed in documented ways. The figure-scanning work that shipped in this repository (merged as pull request #175) exists because docling without this flag never reads plotted values. It detects and classifies a chart region, but a number printed only in a chart appears nowhere in the extracted document [21]. Runs on real decks found four region-level failures, each recorded in a commit in the merged history [21]:

- Docling merged two side-by-side donut charts into a single picture.
- On another page it split a chart pair the reading model treated as one figure.
- It filed a printed table as a picture, which lost a 14-row table entirely: the step that selects which regions get scanned walks the parse's list of tables, and a region filed as a picture is never in that list.
- A picture the run read nothing into looked identical to a picture correctly left alone, so the loss was silent until an explicit report was added.

Any claim about docling's chart capability has to be read against that record.

4.2 The flag, measured on our deck

We enabled `do_chart_extraction` on the tuned-financial preset and changed nothing else from production. We then ran the LFT Q1 2026 earnings supplemental. That deck's 12 charts and 68 chart values were read during the figure-scanning work and verified correct by review [21]. Three configurations ran on this repository's RTX 3090: our pinned 2.113.0, the latest 2.117.0, and 2.117.0 with the heron-101 layout model (section 7). Scoring is per ground-truth value: correct, wrong label, wrong value, or missing.

Result on the pinned 2.113.0	Count
Ground-truth chart values reproduced with the correct label	**61 of 68**
Values attached to a wrong label	2 — Georgia's $96.4M and 8.7% labeled "CA"
Values missing entirely	5 — every donut chart's printed center total
Fabricated values, verified against the rendered pages	**24** — 20 stacked-bar segments, 4 dollar amounts
Chart tables produced / pictures detected	11 / 34
Parse time with / without the flag	37.0 s / 19.7 s for the 23-page deck

The wrong-label case is the one that matters. In the geographic-concentration donut the model produced a state named CA holding Georgia's dollar and percentage values — the deck has no California exposure at all. The output is well-formed, plausible, and wrong. No confidence field is populated — `confidence` is null on every extraction — so nothing downstream can tell this row from a correct one.

The missing center totals are the second pattern. The model tabulates the segments and drops the one number the chart prints largest.

The model also fabricates. We rendered the source pages and checked every value docling emitted beyond the ground truth. 24 of the 25 extras are printed nowhere on their page and are contradicted by what is printed. In the risk-rating chart the model emitted 20 segment values for the unlabeled stacked bars. That row is identical for all four quarters, while the printed bars visibly change proportion. A constant composition also cannot produce weighted averages that move 3.5, 3.6, 3.2, 3.1. The values differ between the 2.113.0 and 2.117.0 runs of the same page. In the capital-structure donut it emitted a dollar column (930.7, 155.2, 59.7, 50.2) that matches none of the amounts the adjacent printed table states. That column sums to 1,195.8 against the printed 1,201.7, so the figures are reconstructions from segment geometry, each slightly wrong. The 25th extra is benign, a real "0 bps" axis category the ground truth simply omits. The chart's labeled series (the weighted-average line) was read correctly in every run.

Two structural failures repeat the region-level history of section 4.1. The two page-9 donuts (Q1 fundings, Q1 payoffs) were merged into a single two-row table with the same label on both rows and no chart identity on either — exactly the class of merge the figure-

scanning work had to repair. The portfolio-activity waterfall came back with its seven values under correct row labels, but the grid's column headers assert a quarter split the chart does not have. Every extraction also sets `title: null`, so which chart a table describes must be inferred from page number and picture order. One weak advance signal exists: the two worst tables — the merged donuts and the fabricated risk-rating grid — carry the deck's two lowest picture-classification confidences (0.722 and 0.535). That is the classifier's confidence in the picture's type, not the extraction's correctness, but it is the only number in the output that flagged trouble at all.

On this deck 2.117.0 produced output identical to 2.113.0 except for those guessed segments, so the four-release upgrade changes nothing here. The heron-101 layout model changed region detection in both directions. It split the merged page-9 donut pair into two correct single-chart tables, and it misread one bar value on page 12 (1131 became 1.131). Layout choice visibly changes chart output, and no configuration was clean.

> **WHAT THE MEASUREMENT MEANS FOR US**
>
> 61 of 68 values with correct labels is far better than nothing — without the flag, all 68 are absent from the document. It is also below what we ship: the figure-scanning path read 68 of 68 on this same deck, with review. And the error count is worse than 7 in 68, because the model added 24 fabricated values alongside them. A mislabeled state, dropped totals, and invented numbers in a financial document are the most expensive failures we can take, because all of them are well-formed and silent, with the confidence field null on every extraction. The flag is worth enabling only where its output is reconciled against another read, never as an unchecked source.

4.3 The model behind the flag

The extraction model is configurable through `ChartExtractionModelOptions`. The default kind is `GRANITE_VISION_V4`, which the installed source maps to `ibm-granite/granite-vision-4.1-4b` at a pinned revision. The non-default `GRANITE_VISION` kind maps to `ibm-granite/granite-vision-3.3-2b-chart2csv-preview` [22]. Three toggles control what the model produces [10]:

- `chart2csv` — on by default; produces the data table.
- `chart2code` — off; produces Python that recreates the chart.
- `chart2summary` — off; produces a prose summary.

> **WHICH MODEL THE PUBLISHED CHART BENCHMARK DESCRIBES**

> The published benchmark belongs to the older model, not the default. Docling's blog reports Chart2CSV 60.1 against GPT-4o's 46.7 and Qwen2-VL-72B's 50.3, plus ChartQA 0.87, for `granite-vision-3.3-2b-chart2csv-preview`, which is the `GRANITE_VISION` kind. The blog calls the numbers "directional signals" [9]. No published benchmark exists for the `GRANITE_VISION_V4` default that actually ran in our measurement. Our 61-of-68 above is, as far as we found, the only accuracy number anyone has for the default model on financial-deck charts.

Docling ships three other enrichment models alongside chart extraction. Its API calls figures pictures, and the two words mean the same thing here [11].

- **Picture classification** — our installed 2.113.0 resolves its default preset to DocumentFigureClassifier-v2.5, the same version the current release ships; verified in the installed `pipeline_options.py` and model specs.
- **Picture description** — a VLM writes a caption per figure (Granite Vision, SmolVLM, any HF model, or a remote API), with filters by figure class and minimum area.
- **Code and formula enrichment** — CodeFormulaV2 reads code blocks and equations. Financial filings contain few of either.

The install question, settled by running it. The docs disagree on which extra installs chart extraction — the example says `docling[granite_vision]`, the blog says `[vlm]`, the installation page lists neither. In this repository's existing environment the flag ran with no new install at all. The model downloads from Hugging Face on first use, and the torch and transformers already present sufficed. The version caveat in section 2's note applies: transformers 5.14.1 crashes it, and 5.8.0 works.

4.4 The values, read as plain text

The chart model is not the only way to get chart values out of a page, so we measured the alternatives on the same 68-value ground truth. Every source below ran over the six LFT chart pages; the rasterized condition strips the text layer (300 DPI render), the proxy for scanned pages and image-rendered charts.

Source	Values present	Wrong numbers
Production preset, unmodified (as picture-child text fragments)	68 of 68	0
PDF text layer	68 of 68	0
RapidOCR full-page, native pages (CPU)	**68 of 68**	**0**
RapidOCR full-page, rasterized pages (CPU)	**68 of 68**	**0**
Nemotron full-page, native pages	66 of 68	3
Nemotron full-page, rasterized pages	67 of 68	1
Chart model (`do_chart_extraction`, section 4.2)	61 of 68 correct-label	26

Wrong numbers were adjudicated against the printed page, not inferred: every numeric token each source emitted was checked against the page's text layer, and the mismatches were read off the rendered page. RapidOCR's only artifacts were two junk tokens — a rotated y-axis label read sideways and one stray "13" — neither resembling a value. All four OCR runs read the "GA, $96.4" label the chart model turned into "CA".

Three conclusions follow. First, character capture was never the gap: the production parse already lifts all 68 values as bare picture-child fragments — the context-free fragments the figure-scanning history documents — and the failure that matters is attribution, which number belongs to which label. Second, for the class where the parse truly holds nothing — charts rendered as images, scanned pages — full-page RapidOCR recovers every value with zero invention, on CPU, so a local read path exists where ADE is today the only reader. Third, the zero-invention property is what the chart model lacks: OCR output can serve as a grounding source precisely because it contains only what is printed. Work to attach these fragments, with their boxes, as the grounding well for no-text-layer regions is underway on its own branch; section 9 places it in the sequence.

5 The VLM Pipeline

The VLM pipeline replaces the classic layout-then-OCR-then-tables sequence with one vision model that reads the whole page image at once. It targets exactly the pages that fail for us today: scanned, stamped, chart-heavy, and tables that exist only as images.

The recommended local model is Granite-Docling-258M, the default in docling's list of VLM model configurations (its docs call these presets too; they are unrelated to our tuned-financial preset). It beats its predecessor SmolDocling on every metric published for both [12, 13]. In the table below, TEDS is the 0-to-1 table-similarity score from section 1: "table structure" scores the grid alone, "with content" also scores the cell text. OCRBench is scored out of 1,000. The financial-table rows matter most to us:

Metric	Granite-Docling-258M	SmolDocling-256M
FinTabNet 150dpi — TEDS, table structure	**0.97**	0.82
FinTabNet 150dpi — TEDS, with content	**0.96**	0.76
Full-page OCR F1	**0.84**	0.80
OCRBench	**500**	338
Equations F1	**0.968**	0.947

FinTabNet is a financial-table benchmark, so the 0.97 is directly on-domain. Two caveats keep this a lead rather than a conclusion:

- **No official head-to-head against the classic pipeline exists.** The comparison example in the docs reports per-page timings only. Claims that the VLM path avoids the classic pipeline's cascading errors come from third-party blogs, not the project [14].
- **Only the Granite-Docling and SmolDocling families emit DocTags** — the markup that converts losslessly into a DoclingDocument, docling's in-memory document object that the rest of our pipeline consumes. (One catalog oddity: the SmolVLM-256M MLX spec also declares DocTags.) Every larger option in the catalog (Qwen2.5-VL-3B, Pixtral-12B, Gemma-3-27B, DeepSeek-OCR-3B, GOT-OCR-2.0, Dolphin) emits markdown, which discards that structure [7, 13].

Unlimited GPU helps most when serving the model. Any of these model configurations can run against an OpenAI-compatible endpoint — vLLM, Ollama, or LM Studio — by setting `enable_remote_services=True`. Docling's own RTX guidance recommends vLLM on Linux, at roughly 4x the throughput of llama-server (an alternative local model server) and up to 6x the throughput of running the model on CPU alone [13, 15].

6 Table Extraction

We run TableFormer in ACCURATE mode with cell matching. TableFormer remains docling's default table model. Two successors appear in the options reference, neither documented enough to adopt [3, 7]:

- `TableStructureV2Options` (kind `docling_tableformer_v2`) — present in the API reference with only two fields, `do_cell_matching` and `kind`. Its only official trace is two changelog lines: 2.78.0 "Add support for TableFormer v2" and 2.79.0 "Use OCR cells with TableFormer v2". No benchmark or prose documentation exists. The description of it as "an improved TableFormer" comes from an auto-generated third-party site and should not be trusted.
- `GraniteVisionTableStructureOptions` (kind `granite_vision_table`) — a VLM-based table reader; the catalog lists Granite-Vision-4.1-4B emitting OTSL, docling's compact table-structure markup — a structured format that converts into a DoclingDocument like DocTags does, not plain markdown. No published accuracy comparison against TableFormer.

A bigger GPU does not help tables. The GPU docs state that `table_batch_size` is "currently not using GPU batching" [8]. TableFormer's accuracy and speed are what they are regardless of the card. Table gains, if any, come from the two successors above or from the VLM pipeline — all unmeasured.

7 The Most Accurate Configuration

No documented maximum-accuracy recipe exists, so this section assembles one from the documented pieces. The only configuration discussion we found, on the project's GitHub Discussions, is answered by a bot with no maintainer response, and we do not treat it as authority [16]. The documentation gives two things: one upgrade with a published measurement, and settings for running the pipeline in large batches on a GPU. Figure 1 places each accuracy setting on the stage of the classic pipeline it affects.

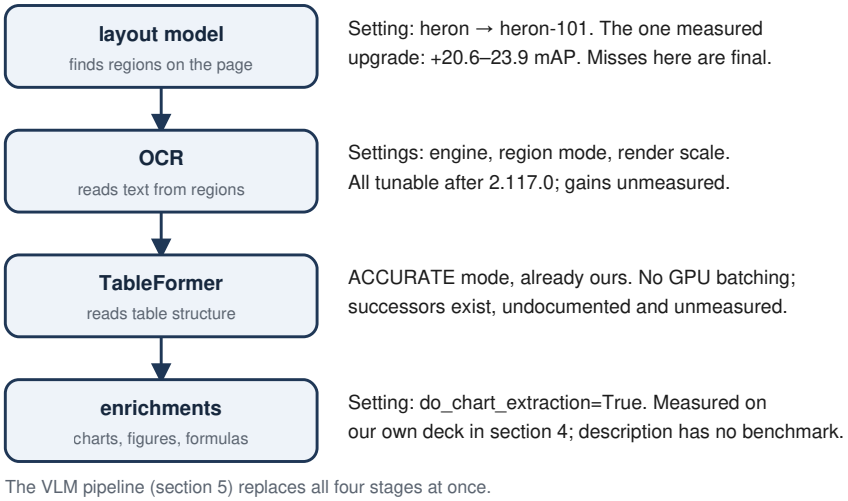

Figure 1 — The classic pipeline's four stages and the accuracy setting at each. The layout swap has a published measurement; chart extraction now has our own (section 4).

7.1 The layout model is the one measured upgrade

The default layout model is `DOCLING_LAYOUT_HERON`; `heron-101` is a higher-capacity variant of it, not the default. The project's own paper measures heron-101 at 78% mAP (mean average precision, a 0-to-100 detection score) — a 20.6–23.9-point gain over the project's previous layout models, the range spanning the paper's evaluation sets — at 28 ms per image on a single A100, trained on 150,000 documents [17]. Larger members of the same layout-model family, named `egret` (medium, large, xlarge), exist as specs in our installed version; the paper's accuracy recommendation is heron-101. Layout errors are upstream of everything: a region the layout model misses never reaches OCR, tables, or charts. Swapping the layout model therefore changes more of the output than any other single change in the classic pipeline.

7.2 What the GPU budget actually changes

Everything in docling's GPU guidance is throughput, not accuracy. It covers batch sizes: `ThreadedPdfPipelineOptions` with `ocr_batch_size=64`, `layout_batch_size=64` (defaults are 4), VLM concurrency up to 64, and VRAM bands that put a 24 GB card — the tier we rent from RunPod, the GPU cloud our workers run on — at batch 32–64 [8, 15]. Exactly three settings change accuracy: the layout model (measured, above), the VLM pipeline (on-domain benchmark, section 5), and the chart model (measured on our deck, section 4).

7.3 The candidate configuration

Every accuracy-relevant option at its strongest documented setting, assembled. No doc states this combination, and it has never been run as a whole. This is our synthesis:

```
docling >= 2.117.0, transformers pinned 5.8.0 (chart model
breaks under 5.14.1)
# layout: the one published measured upgrade (78% mAP);
# on our deck it split the merged donut pair but misread one
bar value
layout   = heron-101
# tables: unchanged; successors unmeasured
tables   = TableFormer ACCURATE
# ocr: scale was hardcoded before 2.117.0
# engine: RapidOCR today, and measured better than Nemotron on
our deck
ocr      = full_page mode, raised render scale
charts   = do_chart_extraction=True (GRANITE_VISION_V4) —
measured, section 4
# description adds captions; no accuracy benchmark
figures  = classification + description via vLLM-served VLM
batching = ThreadedPdfPipelineOptions, batch 32-64 on 24 GB
```

The VLM pipeline is deliberately absent from this block: it replaces the classic pipeline rather than configuring it. The candidate is therefore a pair — this configuration, and the VLM pipeline run beside it on the same pages — and section 9 recommends measuring both.

The documentation cannot tell us how accurate this configuration is as a whole. The chart component now has our own measurement (section 4). Every other component has either a benchmark on someone else's corpus or no benchmark at all. The configuration is cheap to

assemble, and the only way to learn how the rest performs is to run it against our own flagged pages with the current preset as control, per the protocol in section 9.1.

8 Opportunities Beyond Configuration

Two capabilities have nothing to do with parse quality and could still change our workflow.

- **XBRL gives us SEC filing numbers without parsing the PDF.** `InputFormat.XML_XBRL` converts a filing's XBRL directly to a DoclingDocument. The official example runs an SEC EDGAR 10-K and reports 292 document items, 23 of them tables, plus 319 key-value pairs representing XBRL facts — the example's own counts, and the 319 pairs are additional to the 292 items, not part of them [18]. For the 10-Ks we currently parse from PDF, the same numbers exist as machine-readable facts: a reconciliation source, or, for questions that only need reported financial-statement values, a replacement for parsing those pages at all. The extra name is `xbrl`, verified from the 2.117.0 wheel metadata; the installation docs still omit it.
- **Conversion confidence is on the official roadmap.** The roadmap lists "Conversion confidence" alongside chart understanding and key-value extraction [19]. A native confidence signal would address the docling-level silent gaps that `docs/EXTRACTION_PIPELINE.md` documents as our accepted baseline — worth watching, not waiting for.

Two items surfaced in search results but not in the roadmap itself — multi-pass extraction and an experimental table-crops layout model. Neither could be verified in any official source; treat both as rumor.

9 Assessment

Docling's chart extraction reads most values and fails silently on the rest, so it can supplement our figure-scanning path but cannot replace it — while plain full-page OCR with our own engine reads every chart value with zero invention, native or rasterized. Both are now measurements, not speculation. The open questions that remain are OCR capture on genuinely degraded scans and the VLM pipeline; Nemotron's question is answered and closed.

- **Charts: measured. Use the flag only behind reconciliation.** On the LFT deck, 61 of 68 verified values came back with correct labels. The failures are a mislabeled state, dropped center totals, a merged donut pair, and 24 fabricated values contradicted by the printed page, all silent and with no confidence field. The figure-scanning path read 68 of 68 on the same deck. The right role for the flag, if any, is a cheap second read. Reconcile its table against the figure scan, treat agreement as confirmation, and send disagreement to review. Section 4.
- **Chart values as text: measured. Build the grounding well on it.** Full-page RapidOCR reads 68 of 68 with nothing invented, text layer or no text layer, on CPU (section 4.4). That gives no-text-layer regions what they have never had: a local, fabrication-free record of what the page prints, for the figure agent's attributed reading to ground against. The work — per-fragment boxes, region scoping, attribution reconciled against the fragments — is underway on its own branch; its open measurement is capture on genuinely degraded scans, not the clean rasters measured here.
- **OCR: upgrade to 2.117.0, then tune the settings — on the scanned-page corpus, with the current preset as control.** Region mode and render scale did not exist at our pin. Engine choice existed, our preset already exercises it (RapidOCR, explicitly), and the engine comparison is settled: Nemotron captured less and corrupted four numbers where RapidOCR corrupted none, so it is closed as an accuracy candidate regardless of its 1.7x speed (section 3). `full_page` at a raised render scale is the first thing to try on our scanned-page failures.
- **Scanned pages and image-only tables: run the VLM pipeline.** Granite-Docling's 0.97 TEDS on FinTabNet is the best on-domain number docling publishes. The model is 258M parameters, trivial for our GPU budget. DocTags round-trips into the structure we already consume. The absent head-to-head is the experiment to run, not a reason to wait.

Unlimited GPU compute makes every experiment above fast to run, but it improves accuracy only through the layout model, the VLM pipeline, and the chart model. The most accurate setup docling can express today is one of the two candidates in section 7.3: the fully configured classic pipeline, or the VLM pipeline run beside it.

9.1 The measurement protocol

The chart measurement in section 4 is the template every remaining question should follow. Its five rules, stated once so the next experiments inherit them:

- **Ground truth first.** An experiment exists only where a verified reference exists. The LFT deck worked because the figure-scanning run had already read every chart value and a reviewer had confirmed them. The scanned-page and VLM experiments need the same: a small set of affected pages whose correct content is established by review before any configuration runs.
- **One variable per run, current preset as control.** The chart runs changed exactly one thing at a time — the flag, then the version, then the layout model — so every output difference had one cause. The control run is what turns "it produced output" into "it changed this."
- **Score per value, and classify the failures.** A single accuracy percentage hides what matters. "61 of 68" mattered less than the kind of failure behind it. Wrong-label and dropped-total failures are silent — nothing in the output marks them — and therefore expensive. A garbled grid like the portfolio-activity waterfall is visible on inspection and therefore cheap. Rank configurations by their silent-failure count, not their overall score.
- **Check the extras against the rendered page.** Recall against ground truth misses the other direction: values the tool emits that the page never printed. The chart measurement found 24 of these, and rendering the pages and reading them was the only way to tell they were fabricated. Every extra is either verified on the page or counted against the configuration.
- **Affected documents plus a clean control first, full corpus only as confirmation.** One deck answered the chart question in an afternoon. The full corpus run comes after the finding, to confirm scope — never as the discovery vehicle.

9.2 Recommended sequence

- Finish the OCR grounding well for no-text-layer regions — per-fragment boxes in page coordinates, region scoping, and figure-agent attribution reconciled against the

fragments. In progress on its own branch; its acceptance measurement is capture on genuinely degraded scans, extending the clean-raster 68 of 68 in section 4.4.

- Upgrade the pin to ≥ 2.117.0 with transformers held at 5.8.0 (the measured trap in section 2). The upgrade is prerequisite for the OCR-mode experiments below, and on charts it changes nothing.
- Build the scanned-page ground-truth set, then run the OCR grid: `full_page` mode at raised render scale against the current preset, per the protocol above.
- Run the VLM pipeline (Granite-Docling via vLLM) over the same ground-truth pages plus the image-table pages, scored the same way.
- If chart values are wanted in the docling output, wire `do_chart_extraction` behind reconciliation with the figure-scanning path — never unreconciled.
- Prototype XBRL ingestion on one 10-K we already process, as a reconciliation source.

10 Sources

All sources fetched July 30–31, 2026. Bracketed numbers in the text refer to this table. Local verification against the pinned 2.113.0 was done by reading and importing the installed package in this repository on July 30–31, 2026. The chart-extraction measurements in section 4, the OCR value-capture comparison in sections 3 and 4.4, and the Nemotron runs were executed in this repository on July 31–August 1, 2026; each run's configuration is stated where it is reported. The RapidOCR evaluation runs executed on the CPU provider (the CUDA execution provider requires the runtime-library preload our package applies, which the isolated evaluation environment lacked), so their timings are CPU timings.

#	Source
1	PyPI release history — `pypi.org/project/docling`
2	Changelog — `github.com/docling-project/docling/CHANGELOG.md` and the releases page
3	Pipeline options source — `docling/datamodel/pipeline_options.py` (main)
4	Full-page OCR example — `docling-project.github.io/docling/_generated/examples/full_page_ocr/`
5	Installation guide (Nemotron constraints) — `docling-project.github.io/docling/getting_started/installation/`
6	Nemotron OCR benchmark — `docling.ai/blog/20260311_00_docling_at_gtc/`
7	Model catalog — `docling-project.github.io/docling/usage/model_catalog/`
8	GPU usage — `docling-project.github.io/docling/usage/gpu/`
9	Chart understanding blog — `docling.ai/blog/20260203_00_chart_understanding_in_docling/`
10	Chart extraction example and options source — `docs/examples/chart_extraction.py`, `docling/datamodel/chart_extraction_options.py`

#	Source
11	Enrichments — `docling-project.github.io/docling/usage/enrichments/`
12	Granite-Docling model card — `huggingface.co/ibm-granite/granite-docling-258M`
13	Vision models — `docling-project.github.io/docling/usage/vision_models/`
14	VLM comparison example (timings only) — `compare_vlm_models` in the docs examples
15	RTX / vLLM guidance — `docling-project.github.io/docling/getting_started/rtx/`
16	Configuration discussion, bot-answered — `github.com/docling-project/docling/discussions/2516`
17	Layout model paper — `arxiv.org/abs/2509.11720`
18	XBRL conversion example — `docling-project.github.io/docling/_generated/examples/xbrl_conversion/`
19	Community roadmap — `github.com/docling-project/community/docs/roadmap.md`
20	Nemotron OCR versions and wheels — `docling/docs/concepts/OCR.md` (main) and `pypi.org/project/nemotron-ocr`
21	Figure-scanning ground truth and docling failure history — this repository: the merged figure-scanning pull request (#175) and the LFT deck's reviewed figure output in `data/playground`
22	Chart model repository ids — `docling/models/stages/chart_extraction/granite_vision.py` in the installed 2.113.0