Surya and ocr

BCUL Historical Newspaper Ingestion Pipeline (Surya Layout + Surya-Aware Quadrant Apple Vision OCR)

1. Overview & Architecture

The ingestion pipeline (`data_ingestion/ingest_pipeline.py`) processes historical Swiss newspaper PDFs from the BCUL Scriptorium archive (`67,493` issues across 26 titles, 1732–2000) into structured layout segments and OCR text stored in PostgreSQL (`article_segments` and `article_ocr`).

Core Design Principles

1. **Decoupled Layout & OCR**: - **Layout Segmentation (`surya-ocr` `LayoutPredictor` on MPS/GPU)** detects structural blocks (`Text`, `SectionHeader`, `Title`, `PageHeader`, `PageFooter`, `Picture`, `Figure`, `Caption`, `Table`, `ListItem`, `Formula`, `Footnote`) at the semantic block/column level. - **Text Recognition (`ocrmac` / Apple Vision `VNRecognizeTextRequest`, `recognition_level="accurate"` on Apple Neural Engine / CPU)** extracts individual text lines with bounding boxes and confidence scores.
2. **Parallel Execution**: - On every page, Surya layout segmentation (MPS GPU) and full-page Apple Vision OCR (ANE/CPU) run concurrently via a 2-thread `ThreadPoolExecutor`.
3. **Adaptive Surya-Aware Quadrant Splitting (Fallback for Dense Broadsheets)**: - Full-page Apple Vision OCR internally downscales input images to ~2048px on the longest edge and caps line proposals at ~350–400 lines per call. On dense 19th/20th-century broadsheet pages (`40–120` text segments, `500–800+` lines of 5–6 pt type), a single full-

page call can drop entire columns of text. - Whenever a dense page (≥ 10 `Text` segments) has $< 60%$ of its `Text` segments populated by the initial full-page OCR pass, the pipeline automatically upgrades that page to **Surya-Aware Quadrant Splitting**, recovering **99–100%** of segments without cutting any Surya segment or word across quadrant boundaries.

---

2. Step-by-Step Page Processing Pipeline

```
flowchart TD
    A["PDF Issue (pypdfium2)"] --> B["Render Page Bitmap (scale = 2.0–2.5, 144–180 DPI)"]
    B --> C["Surya LayoutPredictor (MPS GPU)"]
    B --> D["Full-Page Apple Vision OCR (ANE/CPU)"]
    C --> E["Spatial Line-to-Segment Matching (Pass 1 + Pass 2)"]
    D --> E
    E --> F{"Dense Page Check:<br/>Text segs >= 10 AND<br/>Filled Text segs < 60%?"}
    F -- "No (99.4% of pages)" --> H["Multi-Column Reading-Order Line Sort"]
    F -- "Yes (Dense Broadsheet)" --> G["Surya-Aware 2x2 Quadrant OCR<br/>(4 Non-Splitting Cluster Crops)"]
    G --> H
    H --> I["Normalize Coordinates to 300 DPI Space"]
    I --> J["Bulk Upsert into PostgreSQL<br/>(article_segments + article_ocr)"]
```

Step 1: PDF Rasterization

- Each page of the PDF is rendered to an RGB PIL image using `pypdfium2` (`pdf_page.render(scale=render_scale)`), and C-level PDF/bitmap handles are explicitly closed after each page to prevent memory leaks over multi-day runs.
- **Scale factor**: - Standard ingestion uses `render_scale = 2.0` (`144 DPI`). - Dense broadsheet repair (`repair_dropped_ocr_pages.py`) renders at `render_scale = 2.5` (`180 DPI`).

Step 2: Concurrent Layout Detection & Full-Page OCR

- **Surya Layout Detection**: - Runs `LayoutPredictor([page_img])`, returning a list of layout boxes $[x_1, y_1, x_2, y_2]$ in page pixel coordinates, each with a `label` and `confidence`.
- **Full-Page Apple Vision OCR**: - Runs `ocrmac.OCR(page_img, recognition_level="accurate").recognize()`, returning `(text, confidence, [nx, ny, nw, nh])` in normalized bottom-left coordinates, which are converted to top-left pixel coordinates $[lx_1, ly_1, lx_2, ly_2]$ via `ocrmac.convert_coordinates_pil`.

Step 3: Two-Pass Line-to-Segment Matching (`match_ocr_lines_to_segments`)

Each OCR line is matched to a Surya segment using its center point $(c_x, c_y) = \left(\frac{lx_1 + lx_2}{2}, \frac{ly_1 + ly_2}{2}\right)$:

1. **Pass 1 — Tightest Enclosing Segment**: - Finds all Surya segments whose bounding box (expanded by a `6px` tolerance margin) contains (c_x, c_y): $$(bx_1 - 6) \le c_x \le (bx_2 + 6) \quad \text{and} \quad (by_1 - 6) \le c_y \le (by_2 + 6)$$ - If multiple nested/overlapping Surya boxes contain the line center, the line is assigned to the segment with the **smallest area** $\text{area} = (bx_2 - bx_1)(by_2 - by_1)$ so specific inner boxes (`SectionHeader`, `Title`, `Caption`) win over coarse outer containers.
2. **Pass 2 — Vertical Near-Miss Snapping**: - Any OCR line not matched in Pass 1 (e.g., first/last lines of a paragraph slightly outside Surya's tight crop) is checked against segments whose horizontal span contains c_x within `20px`: $$(bx_1 - 20) \le c_x \le (bx_2 + 20)$$ - Assigned to the vertically closest segment if vertical distance $\Delta y < 35\text{ px}$.

---

3. Surya-Aware Quadrant Splitting (`ocr_page_with_surya_quadrants`)

Why Naive Grid Splitting Fails vs. Why Surya-Aware Quadrants Succeed

Approach	OCR Calls / Page	Dense Broadsheet Recall	Boundary / Slicing Artifacts

Full-Page (`scale=2.0–3.0`)	`1`	`0%–60%` on dense pages	None, but Apple Vision hits its ~2048px internal downscale & ~350-line cap, dropping whole columns
Naive 2×2 Grid (`W/2, H/2`)	`4`	`~98%`	Cuts mid-page Surya segments and words in half across the $x=W/2$ and $y=H/2$ cut lines
Per-Segment Cropping (`1 crop / box`)	`50–120`	`~99%`	`10x` slower (`50–120` CoreML launches/page); clips ascenders/descenders if Surya box is 2px tight
Surya-Aware 2×2 Quadrants	`4`	**`100%` (`10/10` pages)**	**Zero split segments, zero split words, zero duplicate lines**

Algorithm

Given a page of size (W, H) and N Surya layout segments $S = {s_1, \dots, s_N}$:

1. **Partition Surya Segments into 4 Quadrant Clusters (`TL=0`, `TR=1`, `BL=2`, `BR=3`)**: - For each Surya segment s_i with bounding box $[bx_1, by_1, bx_2, by_2]$, compute its center: $$c_x = \frac{bx_1 + bx_2}{2}, \qquad c_y = \frac{by_1 + by_2}{2}$$ - Assign s_i to exactly one quadrant cluster G_k ($k \in {0, 1, 2, 3}$): $$k = \mathbb{I}!\left(c_x \ge \frac{W}{2}\right) + 2 \cdot \mathbb{I}!\left(c_y \ge \frac{H}{2}\right)$$ - Every Surya segment belongs to **one and only one** cluster G_k.
2. **Compute Non-Splitting Outer Bounding Box for Each Cluster**: - Instead of cropping at $(W/2, H/2)$, compute the tight union bounding box of all Surya segments in G_k, padded by `12px`: $$gx_1 = \max!\left(0,; \min_{s \in G_k} bx_1 - 12\right), \qquad gy_1 = \max!\left(0,; \min_{s \in G_k} by_1 - 12\right)$$ $$gx_2 = \min!\left(W,; \max_{s \in G_k} bx_2 + 12\right), \qquad gy_2 = \min!\left(H,; \max_{s \in G_k} by_2 + 12\right)$$ - Because $[gx_1, gy_1, gx_2, gy_2]$ completely encloses every segment $s \in G_k$, **no segment or word is ever cut in half**, even when a segment straddles the horizontal or vertical midline of the newspaper page.
3. **Quadrant-Scoped OCR & Exclusive Matching**: - Crop `page_img.crop((gx1, gy1, gx2, gy2))` and run `ocrmac.OCR(crop_im, recognition_level="accurate").recognize()`. - Shift detected line coordinates back into full-page pixel space by adding (gx_1, gy_1). - Match the OCR lines from crop G_k **exclusively** against the Surya segments in G_k (using Pass 1 + Pass 2 above). - Because lines from crop G_k can only bind to segments in G_k, overlapping crop margins near the center of the page **never produce duplicate lines**.

---

4. Intra-Segment Multi-Column Reading Order (`_sort_lines_reading_order`)

Occasionally, Surya groups two adjacent narrow newspaper columns into a single wide `Text` segment. Simply sorting OCR lines top-to-bottom (`by y, then x`) would interleave lines across the two columns horizontally.

`_sort_lines_reading_order(lines, box_width)` prevents horizontal column interleaving recursively:

1. Computes the median line width w_{med} of all OCR lines inside the segment.
2. If the segment is wide ($\text{box_width} > 280\text{ px}$), has ≥ 6 lines, and $w_{\text{med}} < 0.55 \times \text{box_width}$: - Sorts all line centers c_x horizontally and locates the largest horizontal gap $\Delta x_{\max}$ between adjacent line centers (the column gutter). - If $\Delta x_{\max} > 25\text{ px}$ at gutter position x_{split}, recursively sorts all lines with $c_x < x_{\text{split}}$ (left column top-to-bottom) followed by all lines with $c_x \ge x_{\text{split}}$ (right column top-to-bottom).
3. Within a single column, lines are sorted by vertical row bucket `(round(y1 / 8.0), x1)`.

---

5. Coordinate Normalization & Database Schema

300 DPI Coordinate Normalization

To ensure compatibility with the downstream `BCUL-historic-data` image renderer (`TARGET_DPI = 300`), all segment bounding boxes are normalized from `render_scale` pixel coordinates to `300 DPI` pixel coordinates before insertion: $$\text{dpi_scale} = \frac{300.0 / 72.0}{\text{render_scale}}$$ $$\text{bbox}_{300\text{dpi}} = \bigl[\text{round}(bx_1 \cdot \text{dpi_scale}, 2),; \text{round}(by_1 \cdot \text{dpi_scale}, 2),; \text{round}(bx_2 \cdot \text{dpi_scale}, 2),; \text{round}(by_2 \cdot \text{dpi_scale}, 2)\bigr]$$

PostgreSQL Tables Written During Ingestion

1. **`article_segments`**: - `newspaper_issue_id` (`FK -> newspaper_issues.id`) - `page_number` (`1-indexed`) - `coordinates` (`JSON` string `[[x1, y1, x2, y2]]` in `300 DPI` pixel space) - `label` (`Text`, `SectionHeader`, `Title`, `PageHeader`, `Picture`, `Caption`, `Table`, etc.) - `method` (`'surya'`) - `title` (auto-populated from the first line for headers or single-line short segments) - `processor_host`, `processor_run_id`, `created_at`
2. **`article_ocr`**: - `article_segment_id` (`FK -> article_segments.id`) - `ocr_text` (newline-joined lines sorted in reading order) - `ocr_method` (`'apple'`) - `processor_host`, `processor_run_id`, `created_at` - Upserted with `ON CONFLICT (article_segment_id, ocr_method) DO UPDATE SET ocr_text = EXCLUDED.ocr_text`.

---

6. Retroactive Repair Utility (`repair_dropped_ocr_pages.py`)

For pages ingested prior to the Surya-Aware Quadrant upgrade where full-page OCR under-matched dense text:

- Queries PostgreSQL for pages where `text_segs >= 10` and `filled_text_segs / text_segs < max_fill_ratio` (ordered from lowest fill ratio `0%` upward).
- Reuses the **existing** `article_segments` bounding boxes in PostgreSQL (no Surya GPU re-run required).
- Renders the PDF page at `render_scale = 2.5` (`180 DPI`), runs Surya-Aware Quadrant OCR across the 4 segment clusters, and keeps `max(new_text, old_text)` by length per segment so no existing text is ever lost.
- Upserts the recovered `article_ocr` rows in place without altering `article_segments.id` foreign keys.

Revision #1
Created 2026-11-12 18:24:22 UTC by Stergios
Updated 2026-11-12 18:25:16 UTC by Stergios