OCRia-nano-1: distilling a 4B document VLM into an 8.1M-parameter CPU pipeline

Logan Markewich

Abstract

Accurate CPU-only parsing is rare. Recent vision-language models (0.6B-4B parameters) parse documents end to end, but need expensive GPUs to reach usable throughput. Other works use smaller models, or collections of models in a pipeline, that are small enough to run on CPUs while trading accuracy for throughput. We present OCRia-nano-1, an 8.1M-parameter model pipeline that pairs a detector–recognizer OCR stack with a joint table and layout model. Used inside the LiteParse engine, it raises ParseBench-2 from 48.7 to 59.2, ahead of the CPU parsers Indra (55.1) and Marker (52.5), at a comparable 27.8 pages per second on 16 vCPUs (2.0× the Tesseract default). Its models are distilled from OCRia-1, keeping only training lines that a second reader (the PDF text layer or PP-OCRv6) agrees with. With the PDF text layer alone at 46.3 on ParseBench-2, the models lift LiteParse to two-thirds of the teacher's score (59.2 vs. 87.0) with 1/500th of its parameters. The weights are open on Hugging Face under a modified OpenRAIL license and work in LiteParse today as an opt-in OCR backend.

Technical report · November 23, 2026

Code Models

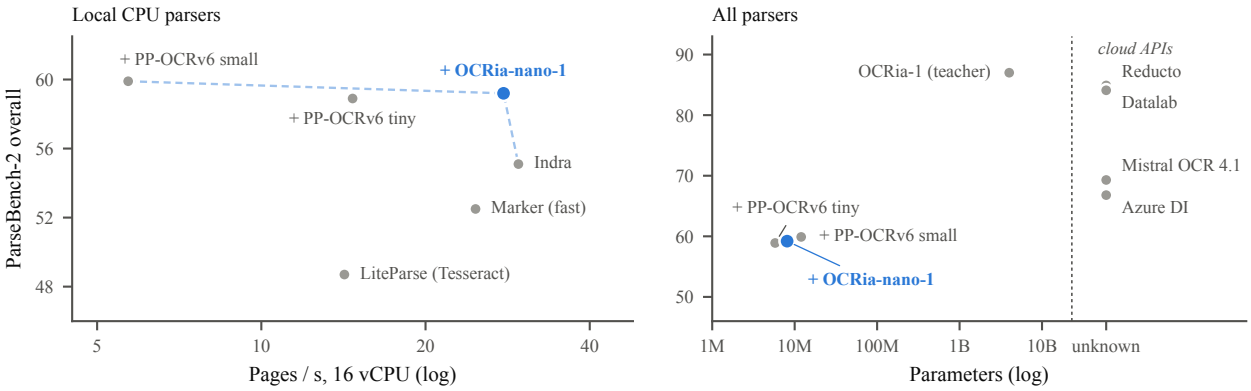

Figure 1: ParseBench-2 overall. Left: local CPU parsers against throughput (pages per second on a 16-vCPU machine, every configuration in the same container); dashed: the speed–accuracy frontier. Right: against parameters where published, with cloud APIs (sizes unpublished) in their own column; OCRia-1 is the 4B teacher.

1 Introduction

Converting a document to a structured representation like Markdown typically covers a broad spectrum of complexity, from simple paragraphs to complex forms and nested tables. This spectrum leads to a diverse landscape of approaches to document parsing, from single-shot VLM systems that process purely from screenshots and leverage heavy GPU resources [2, 19, 24, 34], to heuristic-based systems tuned for running as fast as possible on CPU [1, 28]. This leads to a common decision point for users and consumers of these systems: whether to run accurate models at the cost of compute, or decide that the data is straightforward or latency-sensitive enough for CPU-based systems.

Document parsers have largely taken the compute path: recent document parsers and OCR systems rely on vision-language models (VLMs), typically in the 1–4B parameter range. OCRia-1 (4B parameters) [26], LightOnOCR-3 (0.8B, 1B, and 4B parameter versions) [32, 33] and NVIDIA Nemotron 3.5 Parse (0.6B parameters) [20] are all recent models that parse extremely well, at a significant compute cost. Benchmarks like ParseBench-2 [6] and olmOCR-bench [24, 25] help categorize the strengths and weaknesses of these models. At the same time, smaller CPU-based systems have seen less development. The

most notable is Paddle's PP-OCRv6 (1.5M to 34.6M parameters) [37], although as a classical OCR system, it does not produce markdown without additional heuristics. Indra [4] is a 12M parameter pipeline with a similar architecture, whose additional models for page orientation, reading order and math, combined with heuristics, enable markdown output.

In this paper, we introduce OCRia-nano-1, our 8.1M parameter CPU OCR pipeline for document parsing. OCRia-nano-1 demonstrates that we can distill strong teachers like OCRia-1 into smaller CPU-based models, with LiteParse and OCRia-nano-1 together reaching two-thirds of the teacher's ParseBench-2 score at 1/500th of its parameters. Combined with our existing heuristic pipeline, LiteParse [16], it can replace LiteParse's built-in Tesseract OCR as an opt-in backend, improving both throughput and accuracy. On ParseBench-2, OCRia-nano-1 raises LiteParse's score from 48.7 to 59.2 overall, OmniParseBench rises from 42.3 to 74.8, all while pages per second increases 2.0× to 27.8 pages/second on a 16-vCPU machine. It leads the closest CPU pipelines, Indra and Marker, on ParseBench-1/2, OmniParseBench and BHL while tying them on olmOCR-bench. Swapping in OCR models 2–9× larger (PP-OCRv6 small and medium) buys at most one point on ParseBench-2, while PP-OCRv6 small has roughly 4.8× lower throughput (Fig. 1).

Concretely our work contributes:

1. **An OCR stack**: combines a detector, script router, and per-script recognizers, for a total of 3.9M parameters.
2. **LayoutNet**: a joint transformer model that combines OCR and text information to predict table and layout structure, adding +22.1 points on ParseBench-2's table dimension over LiteParse's heuristic tables.
3. **Automated distillation**: All models are trained on OCRia-1 labels and public data, keeping only the training lines that a second reader (the PDF text layer or PP-OCRv6) agrees with.
4. **Broad evaluation**: OCRia-nano-1 is benchmarked across ParseBench-2, ParseBench-1, OmniParseBench, olmOCR-bench, and BHL, with a measured noise floor.
5. **Open-weight release**: we release all OCRia-nano-1 models as open-weights on HuggingFace, available for use with the LiteParse parser.

2 System overview

LiteParse was originally built as a rich PDF processing tool that mines text and geometry information from documents and uses it to create layout projections and markdown [16]. We built OCRia-nano-1 on that same information, as a pipeline of models that work together to improve the output where plainly written heuristics could not. We graft OCRia-nano-1 into LiteParse's existing pipeline in two main places, the OCR and layout understanding, as shown in Fig. 2.

Model	Role	Params
`det`	Text-line detector	0.08M
`router`	Script router	0.04M
`latin_rec`	Latin recognizer	0.80M
`world_rec`	World-script recognizer	0.97M
`rtl_rec`	RTL recognizer	0.71M
`cjk_rec`	CJK recognizer	1.27M
`layout_net`	Joint table + layout model	4.27M
Total		8.1M

Table 1: The OCRia-nano-1 pipeline of models. Parameter counts from the released model weights.

The OCR stack has three parts: a line detector, a set of script recognizers, and a router on top of the recognizers. The line detector predicts one rotated quad per text line. Then, the router takes line crops to predict "is-upside-down" and selects a proper script recognizer. The router decides a default script per-page while allowing for overrides. The detector supplies each line's box and rotation, and the recognizer reads its text. LiteParse's existing algorithm merges OCR text with native PDF text while ensuring OCR is only applied on pages where it is needed.

LiteParse's grid projection places the text from the PDF and OCR on an ASCII grid that mimics the page's original spacing and layout. As a side-effect of this process, a wealth of statistics are gathered about the document. We select eight signals from the grid projection (Section 5.1), plus the text geometry, and finally the page image, as input to a joint table and layout model. This model jointly predicts "in-table", "is-table-header", pairwise relations, and a layout class for each text unit (Section 5.1).

3 Data

OCRia-nano-1 is an pipeline of models, and requires unique training data for each component. The detector learns from line boxes, transcribed line crops train the recognizers, and page-level layout annotations train our LayoutNet. Public training data for document OCR and table structure is scarce and often licensed for non-commercial use only [12, 17]. We use permissive public sets where they exist, and build everything else from two sources of labels, checked by an agreement filter:

1. **the page itself**: the PDF text layer, and synthetic

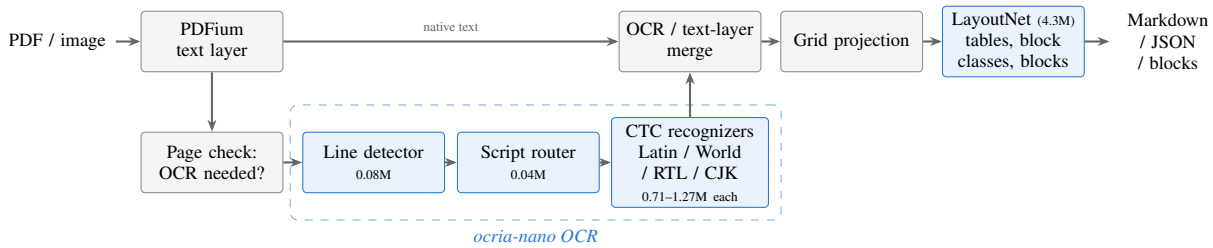

Figure 2: LiteParse with OCRia-nano-1: model stages in blue, existing LiteParse stages in grey.

Type	Source	Label	Size	Used by
OCR				
PDF text layer	web-crawled PDFs	PDFium text, PP-OCRv6 must agree	35.2k pages	det, all rec., router
Scans	scanned pages from the same crawl	PP-OCRv6 text, a second reader must agree	10.1k pages	det, Latin
Synthetic text	Wikipedia sentences, 47 languages	exact	30.0k pages	det, all rec., router
Grounded teacher	OCRia-1 SFT corpus, 6 sets	teacher text + word boxes	11.1k pages	det, Latin, RTL, world
OCR layers	grounded scans with an invisible OCR layer	layer text, teacher must agree	4.8k crops	world
Handwriting	Wikimedia Commons	PP-OCRv6 box + OCRia-1 text	2.2k pages	Latin
Typewriting	CIA CREST (Internet Archive mirror)	Tesseract box + OCRia-1 text	400 pages	Latin
Table cells, IDs	SFT-corpus tables; synthetic	PDF text; exact	658k crops	Latin
Public	HierText, GNHK, CORD, CORU, CATMuS	human	381k crops	det, Latin, RTL
LayoutNet (tables + layout)				
Corpus tables	OCRia-1 SFT corpus, 9 sets	teacher tables grounded to units	18.3k pages	LayoutNet
PMC	PMC Open Access (CC BY, BY-SA, CC0)	OCRia-1 tables (direct call), text match	7.0k pages	LayoutNet
Layout	OCRia-1 SFT corpus, 18 sets	teacher blocks, 13 classes	23.7k pages	LayoutNet

Table 2: Training data sources. Sizes count usable pages or crops after filtering. The detector trains on the text-layer, scan, synthetic, grounded, HierText and GNHK lanes. The corpus-table and PMC pages are each built twice, once with PDF-text units and once with OCR units.

pages whose text is known exactly;

2. **the teacher**: OCRia-1 [26], a 4B document VLM, that we use in three forms:
   - *direct*: we call OCRia-1 on our own pages for transcription;
   - *in LlamaParse*: OCRia-1 running inside LlamaParse, which adds layout blocks and word boxes around its output;
   - *SFT data*: the annotations OCRia-1 itself was trained on.

   We call all three *the teacher*; Table 2 lists which form labels each lane.
3. **an agreement filter**: an independent reader (PP-OCRv6 [7, 21], or Tesseract) reads each line again, and a label is kept only when the two agree.

PDF data and OCRia-1 give high quality transcriptions, with OCR agreement helping to supply text geometry when it is not available. Table 2 lists every data type, source, annotation process, and the models trained on it.

Training data alignment. We hand-label none of our training data, unless it came from an existing human-labelled corpus. Instead, we rely on an alignment pipeline to safely verify training data. Each line is read by two independent readers, which will sometimes disagree. The teacher is our main label source for image-born documents, while PDF text segments are used when available. Text lines are kept when the readers agree to within a character error rate of 0.15. We set this threshold by inspecting crops sorted by disagreement, where we found this to be an acceptable balance between readers. We also drop chart values OCRia-1 transcribes as tables, since they can't be grounded to the input. These filters drop 3% of the transcribed Latin text, while dropping 63% of the right-to-left and 53% of the world script lines, due to weak off-the-shelf OCR on these texts. Synthetic texts make up the bulk of our non-Latin text to compensate for the impact of filtered data.

From teacher tables to unit labels. LayoutNet is a joint model, where for every text unit it predicts a layout class and whether that unit is in a table. For every pair of units a relational head predicts whether they share a layout block, a table, a row, a column or a cell (Section 5). The teacher transcribes each table as HTML, but HTML

only provides a row and column location, not a box location. The text units, which come from either the PDF data or our OCR, have the boxes but no table structure. To connect these two, we assign each text unit to a teacher cell using the overlap of word boxes and matching text. Layout classes are labelled the same way, where overlap determines the class of a text unit, assigning one of our 13 layout classes (text, heading, title, page header and footer, list, table, caption, footnote, formula, code, figure text and key-value).

4 OCR

The OCR half of OCRia-nano-1 is a pipeline where a detector finds text lines, each line is cropped and straightened, and a CTC recognizer [9, 27] reads the crop. PP-OCR [7, 21] popularised this pipeline, and routing lines to script-specific recognizers is common in multilingual OCR (Section 7). Our six OCR models in Table 1 total 3.86M parameters.

LiteParse runs page checks (Section 2) to decide which pages need OCR, triggering for scans, image-only regions, and invisible or garbled text layers. On mixed pages, text the PDF already provides is masked out of the page render, so the OCR models read only what the text layer is missing.

Section 6.4 compares OCRia-nano-1 with other OCR models swapped into the same pipeline, and **??** breaks its accuracy down by script.

4.1 Line detector

The detector is a small segmentation network from the DBNet family [14]. It predicts a single map of shrunken line cores, and connected components of that map grow back into rotated quads. The shipped detector has 82k parameters and is built almost entirely from depthwise-separable convolutions.

Most labelled pages are born-digital, so training pages pass through an online scan simulator that teaches the detector scan conditions from them, applying perspective warps, paper texture, uneven lighting, fax binarisation, show-through, blur and compression at random. Non-text distractors (photos, halftones, barcodes) are also pasted into empty regions as negatives, which cuts false detections on pictures and barcodes.

4.2 Script router and recognizers

A 39k-parameter router reads a text line crop and jointly predicts the script family (Latin, CJK, right-to-left, or "world") and whether the line is upside down. The page takes a majority script, and individual lines may override it when the router is confident (Fig. 3). The upside-down votes rotate the whole page before recognition.

Each script family has its own recognizer, and all four are the same convolution-only CTC architecture. Greedy decoding happens inside the exported graph, so the runtime only maps indices to characters. The models range from 0.71M parameters (RTL) to 1.27M (CJK, which carries the largest character set). A page of another script sent to the Latin recognizer reads as confident-looking nonsense, so when a page's Latin reading is unusually unsure, a sample of its lines is re-read by every recognizer, and the page moves to another recognizer if that one is clearly more confident. A line-level version of the same check catches mixed pages, such as English labels on an Amharic map.

5 Tables and layout: LayoutNet

The second half of the OCRia-nano-1 pipeline, LayoutNet, is a joint table and layout detection model. LayoutNet produces several predictions in a single pass for each text unit, namely is it in a table (and which row, column and cell), which block does it belong to, and what class is that block. Text-unit transformers using pairwise heads are not new, and graph and pairwise-relation methods have a long history in table structure recognition (Section 7). What makes LayoutNet unique is the tight integration with LiteParse, using outputs and measurements from LiteParse's grid projection algorithm, along with typical text information, geometry, and page images.

5.1 Units and inputs

The inputs to LayoutNet are page screenshots and *units*. Units are runs of text split heuristically where a table cell or layout block could have a boundary. Unit texts themselves come from either embedded PDF data or OCR results.

Each unit also supplies its geometry and position, its text shape (digits, case, punctuation, length), and its style when available from the PDF. During training, we mask styles so that units from OCR stay in-distribution. LiteParse itself adds an additional eight features from the grid projection algorithm (Section 2). They are a flag for whether the projection placed the unit, whether the text flows as prose, how many text items share its left, right and centre alignment, and which of those three anchors it snapped to, with the text unit's projection block and XY-cut leaf passed alongside. Detected ruling lines also provide further geometric information to the text unit.

A small CNN image branch gives the model visual features. A strided CNN turns a greyscale page screenshot into patch tokens, and each unit samples the feature map



Figure 3: Detected lines coloured by the recognizer the script router picks.

under its box. The top encoder layers cross-attend to the image tokens, carrying visual features through the model.

5.2 Model and decoding

The encoder is a 6-layer pre-norm transformer at width 256. *Node* heads predict whether a unit is in a table or header. *Pair* heads predict pairwise relations, whether units share a table, row, column, cell, or layout block. A *class* head assigns one of our 13 layout classes (text, heading, title, page header and footer, list, table, caption, footnote, formula, code, figure text and key-value). In total with the image CNN branch, LayoutNet amounts to 4.3M parameters.

We decode tables by clustering the same-table prediction graph. Rows, columns, and cells come from the clusters inside each table. Some post-processing works to remove false positives like tables of contents, chart axes, and other table-like structures. These elements are not tables, and rendering them as tables makes the markdown harder to read. Layout blocks are connected components of the same-block graph, with each taking the class with the largest summed probability (Fig. 4).

LayoutNet's tables claim their lines first. LiteParse's heuristic table detectors then run on the remaining lines, which recovers tables the model misses, but they also read aligned prose, stat callouts and timelines as grids. A heuristic table is therefore kept only when LayoutNet's class head agrees that its lines are tabular (mean table probability at least 0.3); otherwise its lines fall back to paragraphs. This adds 1.2 points on ParseBench-2 [+0.9, +1.4], mostly layout (+6.0) with text, formatting and forms each +0.4, for -0.1 on tables, and 0.4 on OmniParseBench, with olmOCR-bench unchanged.

6 Evaluation

This section answers three questions: how accurate OCRia-nano-1 is against various local and hosted parsers (Section 6.2), how fast it runs on CPU (Section 6.3), and which of its parts contribute accuracy (Section 6.4).

6.1 Benchmarks and protocol

Our headlining benchmark is ParseBench-2 [6], measuring accuracy on text, formatting, tables, charts, layout grounding, and forms across real business documents. ParseBench-2 uses severity-aware scoring, where a page's score is capped by its most severe error, meaning a single changed number or broken table relation can cost most of the page. We also report OmniParseBench [5], ParseBench-1 [36], olmOCR-bench [24, 25], and the historical print BHL IMPACT dataset [8, 11] for OCR character error rates (CER).

We compare OCRia-nano-1 to similar classical OCR systems, Tesseract 5 [29], PP-OCRv6 [37], Indra [4], and Marker [22]. These systems are all detector recognizer pipelines of various sizes. Tesseract 5 is an LSTM line recognizer without a published parameter count (its English model is about 16 MB). PP-OCRv6 comes in three sizes, tiny (1.5M parameters), small (7.7M) and medium (34.6M). Indra is the most similar pipeline to OCRia-nano-1, consisting of a similar set of script recognizers behind a router, along with satellite models for page orientation, reading order and math, amounting to $\sim$12M pa-

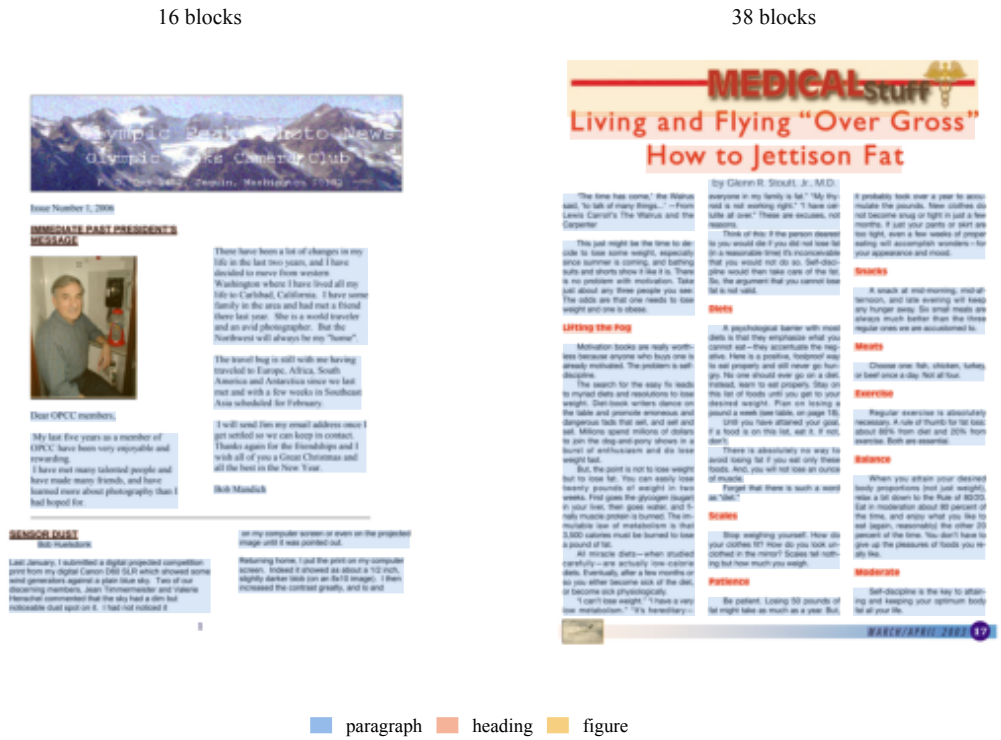

Figure 4: LayoutNet layout blocks on two multi-column pages (text layer only).

rameters in total. Marker is a pipeline around Indra similar to LiteParse, adding slightly more heuristic processing for documents.

6.2 Main results

Table 4 shows that OCRia-nano-1 closes the gap with commercial parsers and beats the other local CPU parsers on ParseBench-2. It adds 10.6 points to the default LiteParse configuration with Tesseract for OCR. For systems that can run locally on CPU, OCRia-nano-1 scores the biggest lead in tables. Indra is ahead of OCRia-nano-1 only on forms (70.2 against 69.8), and the two are level on text (68.5 against 68.4). OCRia-nano-1's lead comes from tables (59.5 against 47.6), layout grounding (72.4 against 65.2) and formatting (82.2 against 76.1). OCRia-nano-1 does not ship with chart understanding and trails every other VLM and specialized parser that supports chart-to-table conversion.

On OmniParseBench's official harness (Table 3), LiteParse with OCRia-nano-1 scores 74.8 (text 59.3, tables 74.3, layout 90.9), against 42.3 for the default configuration. The tables family accounts for most of the gain since the default configuration has no table model and scores 14.5 there, while LayoutNet brings it to 74.3.

On BHL's historical print (Fig. 5), OCRia-nano-1 reaches a character error rate of 0.048, while Tesseract 5 scores 0.064 on the leaderboard; LiteParse's default Tesseract configuration scores 0.275 in our run. The VLMs on the leaderboard reach 0.024–0.031, and PP-OCRv6-medium (34.6M parameters, 9× our 3.9M OCR stack) reaches 0.034. Our run of Indra 1.1.2 and Marker 2.2.0 score 0.062 and 0.067 respectively. On ParseBench-1 and olmOCR-bench (Table 3), OCRia-nano-1 adds 12.2 and 15.2 points to the default Tesseract configuration. olmOCR-bench moves more than ParseBench-1 because it expects page furniture (running headers, footers and page numbers) to be removed. Its headers-and-footers category rises from 50.8% with the default configuration to 88.2% with OCRia-nano-1.

6.3 Inference efficiency

OCRia-nano-1 has 8.1M parameters, roughly 500× fewer than OCRia-1, and it runs on CPU only. The OCR models run through ONNX Runtime, in-process. LayoutNet runs through ONNX Runtime as well, except on macOS, where a plain-Rust implementation of the same weights is faster. The installed package is 59.9 MB on Linux and 75.3 MB on macOS, of which 30 MB are the models and the rest is ONNX Runtime.

Table 5 measures throughput on a 16-vCPU cloud container across five hosts, for LiteParse's default Tesseract settings (with and without LayoutNet), Indra, Marker,

System	ParseBench-2	OmniParseBench	ParseBench-1	olmOCR-bench	BHL CER ↓	pages/s
Local, CPU (run by us)						
LiteParse (Tesseract)	48.7	42.3	40.5	45.7	0.275	14.2
Indra 1.1.2	55.1	70.9	42.6	60.8	0.062	**29.6**
Marker (fast)	52.5	69.8	40.4	60.8	0.067	24.7
LiteParse + OCRia-nano-1	**59.2**	**74.8**	**52.7**	**60.9**	**0.048**	27.8
GPU / cloud (published numbers)						
OCRia-1 (teacher)	87.0	–	82.8	84.6	–	–
Infinity-Parser2-Pro	78.5	87.0	71.2	87.6	–	–
LightOnOCR-3 4B	78.4	82.9	75.3	86.3	0.031	–
Chandra 2	78.9	90.1	70.6	85.8	0.029	–
Mistral OCR 4.1	69.3	89.5	–	81.9	–	–

Table 3: Top: systems that run locally on CPU, all run by us with LiteParse's release build where applicable (pages/s on 16 vCPU, every system in the same container). Bottom: published numbers on the same benchmark versions (ParseBench-2 v2.07, re-scored from published outputs with the official scorer at 2.2.0; OmniParseBench v1.7 board; ParseBench-1 official leaderboard; olmOCR-bench overall from the OCRia-1 report; BHL board). Best local value in bold. Indra's OmniParseBench board entry is 71.4; we report our run of the official harness using 1.1.2. Marker is also our run using the 2.2.0 release.

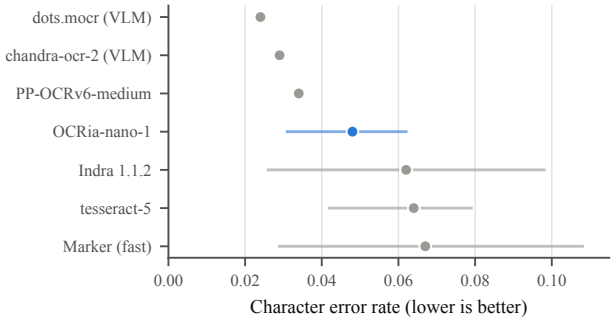

Figure 5: BHL IMPACT GT, 2,165 historical pages, reading-lane CER (scorer 3.0). Bars = 95% CI where reported.

LiteParse with PP-OCRv6 tiny and small in place of our OCR models, and LiteParse with OCRia-nano-1 (with and without LayoutNet). The PP-OCR configurations run the same pipeline as Table 6, including LayoutNet, with one RapidOCR server per worker on the worker's core. The workload is 300 random pages from ParseBench, 60 from each of its five categories (tables, charts, layout, text content and text formatting). Almost all are PDFs, a mix of born-digital pages and scans.

OCRia-nano-1 parses 27.8 pages per second. That is 2.0× the default configuration, 1.9× PP-OCRv6 tiny and 4.8× PP-OCRv6 small, and it is within 0.7 ParseBench-2 points of the PP-OCRv6 small model (Table 5). On one core, an average page takes 426 ms with OCRia-nano-1, 751 ms with Tesseract, and 773 ms with PP-OCRv6 tiny. At P90, pages finish within 0.9 s, compared to 2.4 s with Tesseract and 1.6 s with PP-OCRv6 tiny. Indra runs at 29.6 pages per second and an average of 440 ms, landing within machine variation when compared with OCRia-nano-1 while performing worse on accuracy benchmarks except for a near-tie on olmOCR-bench (Table 3). Marker's bundled Indra predictor defaults to 4 ONNX threads, which oversubscribes during the throughput benchmark. Adjusting this to one thread per worker to match the other configurations parses 24.7 pages per second (0.9× OCRia-nano-1).

Table 5 separates costs by running the same OCR with and without LayoutNet. LayoutNet adds about 100 ms per page on one core (426 against 325 ms mean latency). Without it, OCRia-nano-1's OCR parses 37.2 pages per second, more than two and a half times the Tesseract default.

These latency and throughput numbers were achieved using several optimizations across the pipeline and models. On mixed pages we mask the text the PDF already provides before OCR, PNG image inputs enter the PDF without re-encoding, and OCRia-nano-1 renders pages at a default 250 dpi, which in our tests matched 300 dpi on benchmarks with about a third fewer pixels to process.

6.4 Ablations

In this section, we perform ablations on several components to determine which parts of OCRia-nano-1 are capacity limited vs. limited by labels. We grow or shrink each model with the general recipe and keep data the same. We find that OCRia-nano-1 is capacity-bound in its Latin recognizer, while the detector and LayoutNet are not. We also find that each component in OCRia-nano-1 adds step-changes in accuracy, as described in Figure 6.

System	Overall	Text	Formatting	Tables	Charts	Layout	Forms
LlamaParse Agentic (OCRia-1)	87.0	87.0	92.7	89.9	77.4	89.2	85.6
Reducto	84.9	81.8	82.6	84.9	89.1	86.0	85.2
Datalab (accurate)	84.1	91.1	92.5	90.4	67.8	82.4	80.6
Chandra 2	78.9	77.2	88.8	85.6	62.5	80.6	78.7
MinerU2.5-Pro	76.9	77.1	79.0	80.6	64.0	86.2	74.5
PaddleOCR-VL 1.6	71.5	73.3	79.4	73.0	57.5	74.5	71.2
Mistral OCR 4.1	69.3	85.0	81.9	81.8	1.2	85.9	79.7
Azure DI Layout	66.8	79.8	77.8	77.6	1.5	83.9	80.3
LiteParse + OCRia-nano-1	**59.2**	68.5	82.2	59.5	3.1	72.4	69.8
Indra (indra-ocr)	55.1	68.4	76.1	47.6	3.1	65.2	70.2
Marker (fast)	52.5	62.5	75.5	43.6	3.1	61.0	69.2
LiteParse (Tesseract)	48.7	58.8	78.0	36.2	1.2	49.4	68.5

Table 4: ParseBench-2 v2.07 with the official `parse-bench` 2.1.3 scorer (overall = mean of the six dimensions). Competitor and teacher rows are re-scored from their published outputs (ParseBench-2-results, layout with block HTML reduced to text); local rows are our runs. Top block: GPU-hosted or cloud parsers. Bottom block: runs locally on CPU.

	PB-2	Pages/s	ms per page, one core	
Configuration	overall	16 vCPU	mean	p90
LiteParse (Tesseract)	48.7	14.2 ± 1.6	751	2,420
+ LayoutNet	56.9	12.8 ± 1.4	859	2,626
Indra 1.1.2	55.1	29.6 ± 5.3	440	871
Marker (fast)	52.5	24.7 ± 3.5	578	1,111
LiteParse + PP-OCRv6 tiny + LayoutNet	58.9	14.7 ± 2.7	773	1,568
LiteParse + PP-OCRv6 small + LayoutNet	59.9	5.7 ± 0.4	1,963	4,988
LiteParse + OCRia-nano-1 OCR only	51.2	37.2 ± 4.8	325	751
LiteParse + OCRia-nano-1 (release)	59.2	27.8 ± 4.4	426	862

Table 5: Accuracy and speed per configuration on the same 300 ParseBench pages: ParseBench-2 overall, pages per second with 16 single-core workers in a 16-vCPU container (mean ± sd over five cloud hosts, every configuration in the same container on each host) and per-page latency on one core (warm, mean over the hosts). PP-OCR runs through LiteParse's HTTP OCR interface, which adds a small per-page request cost. LayoutNet runs on every page since it produces the layout blocks as well as the tables.

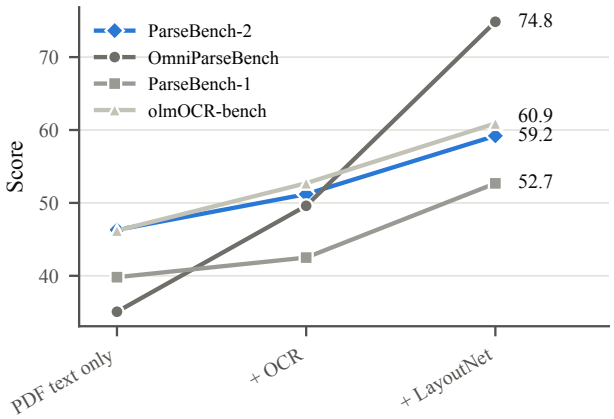

Figure 6: Score as components are added, all on the release build.

Each stage has impact. Figure 6 adds the components one after the other. On ParseBench-2 the PDF text layer alone scores 46.3. LayoutNet without OCR raises that to 53.7, OCR without LayoutNet to 51.2, and both together combine to 59.2. Replacing our OCR models with Tesseract costs 2.4 points, and dropping OCR entirely costs 5.5 (Table 6). OCR models 2–9× larger (PP-OCRv6 small and medium) recover at most 0.9 points.

Each LayoutNet input and head adds accuracy. Table 6 swaps LayoutNet variants into the release pipeline:

- **Layout heads**: replacing LiteParse's heuristic-based blocks with LayoutNet's raises layout 16.7 points from 53.0 to 69.7, while tables drop by 1.8 points.
- **Image branch**: integrating image features adds 1.8 table and 3.0 layout points (+1.1 overall).

The image-branch gain is measured from the wider joint model (d256), which on its own is within noise of the narrower one (58.2 against 58.3).

The Latin recognizer is capacity-bound. The Latin recognizer improves steadily with size (Table 6 and Fig. 7), at the cost of latency. Trained from scratch on the same data, moving from 0.76M to 9.6M parameters lowers HierText CER from 0.162 to 0.116 and adds 2.4 ParseBench-2 text points (0.7 overall). The shipped slim model reaches 0.137 at nearly the same size (0.80M, covering 791 characters). Since the accuracy gains are small overall, we ship the slim model and spend the savings on throughput.

OCR models (LayoutNet on)	Params	Overall	Δ overall [95% CI]	Text		
none (text layer only)	–	53.7	−5.5 [−6.2, −4.9]	57.1		
Tesseract 5, 300 dpi	–	56.9	−2.4 [−2.8, −1.9]	59.2		
PP-OCRv5 mobile	3.1M	58.7	−0.5 [−0.8, −0.2]	66.7		
PP-OCRv6 tiny	1.5M	58.9	−0.4 [−0.7, −0.1]	66.6		
OCRia-nano-1	3.9M	**59.2**	(release)	68.5		
PP-OCRv6 small	7.7M	59.9	+0.7 [+0.4, +1.0]	71.9		
PP-OCRv6 medium	34.6M	60.2	+0.9 [+0.6, +1.2]	72.3		
OCRia-nano-1, no LayoutNet	3.9M	51.2	−8.1 [−8.6, −7.5]	68.1		
LayoutNet variants	Params	Overall	Δ overall [95% CI]	Tables	Layout	
Tables only (no layout heads, d192)	1.9M	55.3	−3.9 [−4.3, −3.4]	58.1	53.0	
Joint (tables + layout, d192)	1.9M	58.3	−1.0 [−1.3, −0.7]	56.4	69.7	
Joint, wider (d256, layout loss ×0.5)	3.4M	58.2	−1.1 [−1.4, −0.7]	57.7	69.4	
+ image branch (shipped)	4.3M	**59.2**	(release)	59.5	72.4	
+ capacity (d384, 8 layers)	11.6M	59.2	±0.0 [−0.3, +0.2]	59.2	71.9	
Latin recognizer	Params	Overall	Δ overall [95% CI]	Text	HierText CER	ms/line
slim, from scratch	0.76M	58.9	−0.4 [−0.6, −0.2]	67.0	0.162	2.9
large, from scratch	4.26M	59.3	+0.1 [−0.1, +0.2]	68.4	0.127	10.5
xl, from scratch	9.56M	59.5	+0.3 [+0.1, +0.4]	69.4	0.116	21.6
shipped, fp32	0.80M	59.3	±0.0 [0.0, +0.1]	68.7	0.132	3.4
shipped (int8)	0.80M	**59.2**	(release)	68.5	0.137	1.5

Table 6: Ablations on ParseBench-2, each a swap inside the release pipeline; Δ = paired difference to the release with its 95% bootstrap CI over documents. *OCR models*: other OCR models in place of ours (PP-OCR through RapidOCR); the last row drops LayoutNet. *LayoutNet variants*: the tables-only model has no layout heads, so layout falls back to LiteParse's heuristic blocks. *Latin recognizer*: three sizes trained from scratch on the same data and schedule; the shipped model has the slim architecture.

The detector is not capacity-bound. Swapped into the release package, detectors of 0.08M (shipped), 0.38M and 1.48M parameters all score 59.2–59.3 on ParseBench-2, with text between 68.3 and 68.5. The shipped detector is 18× smaller than the largest at no measurable cost.

LayoutNet is also not capacity-bound. Growing it to 11.6M parameters changes the overall score by -0.0 $[-0.3, +0.2]$: 59.2 against the shipped 4.3M model's 59.2 (Table 6). LayoutNet improves with better inputs (the image branch) and better labels (grounded boxes), but not with size (Table 6). Future work would see us improve our labelling pipeline to increase performance at this size, or even look at smaller models.

7 Related work

End-to-end VLM parsers. Vision-language models (VLMs) have become a popular approach to end-to-end document parsing, especially for markdown output. Nougat [2] (a Swin Transformer image encoder with an mBART text decoder), GOT-OCR 2.0 [34] (a vision encoder and Qwen-0.5B decoder) and olmOCR [24] (a Qwen2-VL-7B fine-tune) were early examples. Recent works have similar designs while they differ mainly in the specifics around vision encoders, sizes, and training recipes. OCRia-1 [26], the teacher for OCRia-nano-1, is a 4B fine-tune of Qwen3.5-4B. Other peers include Chandra 2 [3], LightOnOCR-3 [33], Infinity-Parser2-Pro [10], dots.ocr [13] and Nemotron 3.5 Parse [20], all being various flavors of text decoders with vision encoders. VLMs are at the frontier of accuracy, with OCRia-1 scoring 87.0 on ParseBench-2, well above any CPU system (Table 3). However, VLM parsers all require a GPU for useful throughput and latency. For example, OCRia-1 parses between 2 and 10 pages per second on a single B200, depending on request concurrency.

Small CPU OCR pipelines. Small detector–recognizer systems sit at the opposite end of the document parsing spectrum. Their output is typically text lines with boxes rather than markdown, and they run on CPUs at high throughput and low latency. Tesseract [29] is among the most popular legacy systems for OCR. Since version 4 it has used an LSTM line recognizer, and it is the default OCR engine in LiteParse. One of the most widely used is the PP-OCR series from Paddle [7, 21]. PP-OCRv6 [37] pairs DBNet detectors with CTC recognizers, in various sizes (1.5M to 34.6M parameters), and produces only text

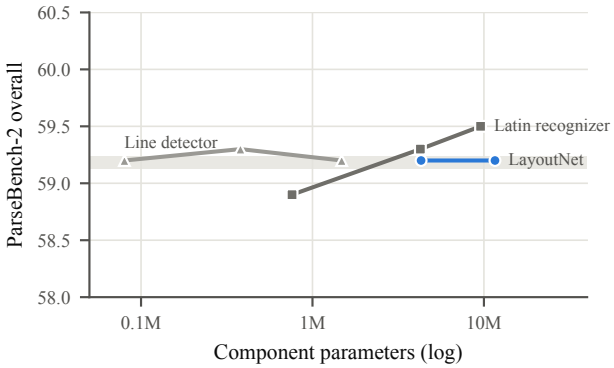

Figure 7: ParseBench-2 overall against component size: the line detector, the Latin recognizer, and LayoutNet, each swept inside the release pipeline. Band: the release with LayoutNet retrained from three seeds; differences inside it are noise. Only the Latin recognizer rises clearly with size.

and boxes. We use PP-OCRv6 as an agreement check when building labels (Section 3). Swapped into LiteParse in place of our OCR models, its larger recognizers provide a small increase in accuracy at a large cost in throughput compared to OCRia-nano-1 (Table 5).

Indra [4] is the closest comparison to OCRia-nano-1. It has the same overall pipeline shape, containing a text detector, per-script CTC recognizers behind a script router, and smaller models for page orientation, reading order and math. In our runs Indra roughly matches OCRia-nano-1's throughput but trails it on accuracy (Table 3). Marker [22] wraps Indra in an end-to-end converter with additional heuristics, similar to LiteParse with OCRia-nano-1. Unlike both, OCRia-nano-1 is built around LiteParse's spatial projection of the PDF text layer, which it uses as input features. It also predicts table structure and layout blocks jointly with a single learned model, LayoutNet.

Text-layer converters. Text-layer converters such as PyMuPDF4LLM [1] and pdfplumber [28] apply heuristics to the text embedded in born-digital PDFs. They are the fastest option for those documents, but they have weak or no OCR for scanned pages. LiteParse [16] also belongs to this category depending on the settings used. It extracts text with PDFium and projects it onto a grid that preserves the page's spacing. Figure 6 measures what each learned stage of OCRia-nano-1 adds on top of the text layer alone.

Layout analysis and table structure. Once text extraction works, the next problem is labelling layout and table structure, which let downstream systems organize and process the text. DocLayNet [23] is a core dataset for training layout models. These range from the LayoutLM family [35], which models text and layout together, to purely visual box detectors such as the Heron models [15]. Table structure recognition has its own datasets, such as PubTables-1M [30] and FinTabNet [38], and its own models, including TableFormer [18] and the lightweight POTATR [31]. Visual models like Heron and TableFormer detect regions from page pixels. LayoutNet instead labels the text pieces that LiteParse already has, from either the PDF text layer or OCR. In one pass it assigns them to table cells and to 13 block classes, and a small image branch adds visual context. Replacing LiteParse's heuristic blocks with LayoutNet substantially improves layout accuracy (Section 6.4).

Benchmarks. End-to-end parsing benchmarks are the main way to compare document parsers today. We evaluate on ParseBench-1 [36], ParseBench-2 [6], OmniParseBench [5], olmOCR-bench [24, 25] and the BHL IMPACT historical print set [8, 11] (Section 6.1). Their published leaderboards mostly cover VLMs and cloud APIs, and small CPU systems are often missing from them.

8 Conclusion and future work

We presented OCRia-nano-1, an 8.1M-parameter model pipeline for CPU document parsing. It is a pipeline of models made up of a text-line detector, a script router, per-script recognizers (3.9M parameters in total), and LayoutNet (a 4.3M-parameter joint table and layout model). Training data is distilled from OCRia-1, using multi-reader agreement to filter and collect training data. With LiteParse as the orchestrator, OCRia-nano-1 can replace the Tesseract default as an opt-in OCR backend, and raises ParseBench-2 from 48.7 to 59.2, OmniParseBench from 42.3 to 74.8, ParseBench-1 from 40.5 to 52.7, olmOCR-bench from 45.7 to 60.9, and lowers BHL's character error rate from 0.275 to 0.048. It does this on CPU at 27.8 pages per second on 16 vCPUs or 426 ms per page on a single core (Tables 3 and 5).

Among other parsers that are meant to run on CPU, OCRia-nano-1 leads Indra and Marker on ParseBench-2 (by 4.1 and 6.7 points), ParseBench-1, OmniParseBench and BHL, and ties them on olmOCR-bench. Most of the lead is attributed to tables and layout, where LayoutNet helps LiteParse reach 59.5 and 72.4 on ParseBench-2. It runs at comparable speed to other CPU pipelines, slightly below Indra's throughput and slightly above its per-page latency. We showed that each component in the pipeline provides accuracy through ablations, where on ParseBench-2 the text layer alone scores 46.3 through existing heuristics, while the entire pipeline stacks on up

to 13 more points (Fig. 6). The final pipeline reaches about two-thirds of the teacher's ParseBench-2 score (59.2 against 87.0) with roughly 1/500th of its parameters.

Limitations and future work.

- **Recognition is the main remaining gap.** The Latin recognizer is the one component that improves with size (Table 6), and text recognition is where OCRia-nano-1 could gain against competing models. Ablations show that a 9.6M model adds 2.4 text points but costs 7× more CPU per line. Training longer on more data moved the shipped slim model further than the larger architectures did, so we expect data (higher quality and more quantity) to drive future gains. Handwriting and scripts with little training data remain the weakest aspects as observed in benchmarks.
- **Charts.** OCRia-nano-1 has no chart model to process charts into tables, which leaves the Charts dimension near zero (3.1) on ParseBench-2, which is the largest single gap to the VLM parsers.
- **Layout grounding and scanned math.** Layout grounding (72.4 on ParseBench-2) still trails the VLM parsers (80–89), and equations on scanned pages are not recovered at all (olmOCR-bench old-scans math). Both point to LayoutNet's region labels and a math recognizer for image inputs as possible next steps.

References

[1] Artifex Software. PyMuPDF4LLM. https://github.com/pymupdf/pymupdf4llm, 2024. AGPL-3.0.

[2] Lukas Blecher, Guillem Cucurull, Thomas Scialom, and Robert Stojnic. Nougat: Neural optical understanding for academic documents. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2308.13418.

[3] Datalab. Chandra OCR 2. https://huggingface.co/datalab-to/chandra-ocr-2, 2026. Model card; code at https://github.com/datalab-to/chandra.

[4] Datalab. Indra OCR. https://github.com/datalab-to/indra, 2026. Code Apache-2.0; weights modified OpenRAIL-M (https://huggingface.co/datalab-to/indra-ocr).

[5] Datalab. Omni parse bench: A document-parsing benchmark. https://github.com/datalab-to/omni_parse_bench, 2026. Version 1.7.0.

[6] Pierre-Loïc Doulcet, Boyang Zhang, Sebastián G. Acosta, Sacha Bron, Daniel B. Ospina, Antonio Jimeno Yepes, Nutchapol Dendumrongsup, and Simon Suo. ParseBench-2: Severity-aware evaluation of document parsing, 2026. URL https://arxiv.org/abs/2611.17400.

[7] Yuning Du, Chenxia Li, Ruoyu Guo, Xiaoting Yin, Weiwei Liu, Jun Zhou, Yifan Bai, Zilin Yu, Yehua Yang, Qingqing Dang, and Haoshuang Wang. PP-OCR: A practical ultra lightweight OCR system. *arXiv preprint arXiv:2009.09941*, 2020.

[8] FineBooks. BHL OCR leaderboard. https://huggingface.co/spaces/finebooks/bhl-ocr-leaderboard, 2026. Hugging Face Space.

[9] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 369–376, 2006.

[10] Zuming Huang, Jun Huang, Kexuan Ren, Baode Wang, Weizhen Li, Jianming Feng, Yu Wang, Yichen Yao, Shijun Lin, Yige Tang, Cheng Peng, Weidi Xu, Wei Chu, Yinghui Xu, and Yuan Qi. Infinity-Parser2 technical report, 2026. URL https://arxiv.org/abs/2607.07836.

[11] IMPACT Centre of Competence and Biodiversity Heritage Library. IMPACT-BHL ground truth. https://github.com/impactcentre/groundtruth-bhl, 2012. PRImA PAGE XML, CC-BY 3.0. Hugging Face redistribution: https://huggingface.co/datasets/finebooks/bhl-impact-gt.

[12] Guillaume Jaume, Hazim Kemal Ekenel, and Jean-Philippe Thiran. FUNSD: A dataset for form understanding in noisy scanned documents. In *International Conference on Document Analysis and Recognition Workshops (ICDAR-OST)*, 2019.

[13] Yumeng Li, Guang Yang, Hao Liu, Bowen Wang, and Colin Zhang. dots.ocr: Multilingual document layout parsing in a single vision-language

model, 2025. URL https://arxiv.org/abs/2512.02498.

[14] Minghui Liao, Zhaoyi Wan, Cong Yao, Kai Chen, and Xiang Bai. Real-time scene text detection with differentiable binarization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 11474–11481, 2020.

[15] Nikolaos Livathinos, Christoph Auer, Ahmed Nassar, Rafael Teixeira de Lima, Maksym Lysak, Brown Ebouky, Cesar Berrospi, Michele Dolfi, Panagiotis Vagenas, Matteo Omenetti, Kasper Dinkla, Yusik Kim, Valery Weber, Lucas Morin, Ingmar Meijer, Viktor Kuropiatnyk, Tim Strohmeyer, A. Said Gurbuz, and Peter W. J. Staar. Advanced layout analysis models for Docling, 2025. URL https://arxiv.org/abs/2509.11720. Heron model: https://huggingface.co/docling-project/docling-layout-heron.

[16] LlamaIndex. LiteParse: fast, local document parsing with spatial text extraction. https://github.com/run-llama/liteparse, 2026.

[17] Urs-Viktor Marti and Horst Bunke. The IAM-database: An English sentence database for offline handwriting recognition. *International Journal on Document Analysis and Recognition*, 5(1):39–46, 2002. doi: 10.1007/s100320200071.

[18] Ahmed Nassar, Nikolaos Livathinos, Maksym Lysak, and Peter Staar. TableFormer: Table structure understanding with transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022. URL https://arxiv.org/abs/2203.01017.

[19] Junbo Niu, Zheng Liu, Zhuangcheng Gu, Bin Wang, Linke Ouyang, Zhiyuan Zhao, Tao Chu, Tianyao He, Fan Wu, Qintong Zhang, Zhenjiang Jin, Guang Liang, Rui Zhang, Wenzheng Zhang, Yuan Qu, Zhifei Ren, Yuefeng Sun, Yuanhong Zheng, Dongsheng Ma, Zirui Tang, Boyu Niu, Ziyang Miao, Hejun Dong, Siyi Qian, Junyuan Zhang, Jingzhou Chen, Fangdong Wang, Xiaomeng Zhao, Liqun Wei, Wei Li, Shasha Wang, Ruiliang Xu, Yuanyuan Cao, Lu Chen, Qianqian Wu, Huaiyu Gu, Lindong Lu, Keming Wang, Dechen Lin, Guanlin Shen, Xuanhe Zhou, Linfeng Zhang, Yuhang Zang, Xiaoyi Dong, Jiaqi Wang, Bo Zhang, Lei Bai, Pei Chu, Weijia Li, Jiang Wu, Lijun Wu, Zhenxiang Li, Guangyu Wang, Zhongying Tu, Chao Xu, Kai Chen, Yu Qiao, Bowen Zhou, Dahua Lin, Wentao Zhang, and Conghui He. MinerU2.5: A decoupled vision-language model for efficient high-resolution document parsing, 2025. URL https://arxiv.org/abs/2509.22186.

[20] NVIDIA. Nemotron 3.5 parse. https://huggingface.co/nvidia/Nemotron-3.5-Parse, 2026. Model card.

[21] PaddlePaddle Authors. PaddleOCR. https://github.com/PaddlePaddle/PaddleOCR, 2026. PP-OCRv6 models; Apache-2.0.

[22] Vik Paruchuri and Datalab. Marker: Convert PDF to markdown + JSON quickly with high accuracy. https://github.com/datalab-to/marker, 2026. Version 2.2.0 (marker-pdf).

[23] Birgit Pfitzmann, Christoph Auer, Michele Dolfi, Ahmed S. Nassar, and Peter W. J. Staar. DocLayNet: A large human-annotated dataset for document-layout analysis. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 2022. URL https://arxiv.org/abs/2206.01062.

[24] Jake Poznanski, Aman Rangapur, Jon Borchardt, Jason Dunkelberger, Regan Huff, Daniel Lin, Christopher Wilhelm, Kyle Lo, and Luca Soldaini. olmOCR: Unlocking trillions of tokens in PDFs with vision language models, 2025. URL https://arxiv.org/abs/2502.18443.

[25] Jake Poznanski, Luca Soldaini, and Kyle Lo. olmOCR 2: Unit test rewards for document OCR, 2025. URL https://arxiv.org/abs/2510.19817.

[26] Aman Rangapur, Antonio Jimeno Yepes, Daniel Bustamante, Nutchapol Dendumrongsup, Sebastián G. Acosta, Pierre-Loïc Doulcet, and Simon Suo. OCRia-1 technical report, 2026. URL https://arxiv.org/abs/2611.00092.

[27] Baoguang Shi, Xiang Bai, and Cong Yao. An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(11):2298–2304, 2017.

[28] Jeremy Singer-Vine. pdfplumber. https://github.com/jsvine/pdfplumber, 2015.

[29] Ray Smith. An overview of the Tesseract OCR engine. In *Proceedings of the Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, pages 629–633. IEEE Computer Society, 2007. doi: 10.1109/ICDAR.2007.4376991.

[30] Brandon Smock, Rohith Pesala, and Robin Abraham. PubTables-1M: Towards comprehensive table extraction from unstructured documents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022. URL https://arxiv.org/abs/2110.00061.

[31] Brandon Smock, Libin Liang, Maxim Sokolov, Amrit Ramesh, Valerie Faucon-Morin, Tayyibah Khanam, and Maury Courtland. POTATR: A lightweight image-to-graph model for page-level table extraction. In *International Workshop on Document Analysis Systems (DAS)*, 2026. URL https://arxiv.org/abs/2606.09788.

[32] Said Taghadouini, Adrien Cavaillès, and Baptiste Aubertin. LightOnOCR: A 1b end-to-end multilingual vision-language model for state-of-the-art OCR, 2026. URL https://arxiv.org/abs/2601.14251.

[33] Said Taghadouini, Adrien Cavaillès, Baptiste Aubertin, and LightOn AI. LightOnOCR-3: High-performance OCR and layout extraction in one model. https://huggingface.co/blog/lightonai/lightonocr-3, 2026. Hugging Face blog, 8 October 2026.

[34] Haoran Wei, Chenglong Liu, Jinyue Chen, Jia Wang, Lingyu Kong, Yanming Xu, Zheng Ge, Liang Zhao, Jianjian Sun, Yuang Peng, Chunrui Han, and Xiangyu Zhang. General OCR theory: Towards OCR-2.0 via a unified end-to-end model, 2024. URL https://arxiv.org/abs/2409.01704.

[35] Yiheng Xu, Minghao Li, Lei Cui, Shaohan Huang, Furu Wei, and Ming Zhou. LayoutLM: Pre-training of text and layout for document image understanding. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 2020. URL https://arxiv.org/abs/1912.13318.

[36] Boyang Zhang, Sebastián G. Acosta, Preston Carlson, Sacha Bron, Pierre-Loïc Doulcet, Daniel B. Ospina, and Simon Suo. ParseBench: A document parsing benchmark for AI agents, 2026. URL https://arxiv.org/abs/2604.08538.

[37] Yubo Zhang, Xueqing Wang, Manhui Lin, Yue Zhang, Penglongyi Deng, Ting Sun, Tingquan Gao, Zelun Zhang, Jiaxuan Liu, Changda Zhou, Hongen Liu, Suyin Liang, Cheng Cui, Yi Liu, Dianhai Yu, and Yanjun Ma. PP-OCRv6: From 1.5M to 34.5M parameters, surpassing billion-scale VLMs on OCR tasks, 2026. URL https://arxiv.org/abs/2606.13108.

[38] Xinyi Zheng, Douglas Burdick, Lucian Popa, Xu Zhong, and Nancy Xin Ru Wang. Global table extractor (GTE): A framework for joint table identification and cell structure recognition using visual context. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, 2021. URL https://arxiv.org/abs/2005.00589.

A Data details

Licensing. We train on every source except those under non-commercial or no-derivatives licences (e.g. IAM [17], FUNSD [12]). Pages from the OCRia-1 corpus are dropped when their licence field matches either restriction or when the corpus marks them excluded.

Benchmark decontamination. Before any dataset is built, documents are checked against a blocklist of every benchmark we report: file hashes and 8-word text shingles over ParseBench-1, ParseBench-2, OmniParseBench, olmOCR-bench and ODL are used for filtering.

Label agreement. A line from the PDF text layer or from the teacher is kept only when a second reader agrees with it to within a character error rate of 0.15 (Section 3). We found this to be an acceptable deviation while still capturing enough data to train a capable model.

Synthetic text. Synthetic pages render Wikipedia sentences in 47 languages (CC BY-SA) with open-licensed fonts, including handwriting fonts, under random layouts, sizes and ink. Each line's label is its exact source text, and its box is the rendered ink. Scripts with little real data (Ethiopic, Armenian, Lao, most Indic scripts) rely mostly on these pages to train the associated recognizers.

B Training details

Recognizers. All recognizers are CTC models over 32-pixel-high line crops: a convolutional stem followed by token mixers and a linear head over the script's character set. The Latin recognizer (0.80M parameters, 791 classes) is quantized to int8 (static QDQ with ONNX Runtime). The world (0.98M) and right-to-left recognizers share the slim body. The CJK recognizer uses a narrower head and a stride of 8 pixels per CTC step, since CJK glyphs are roughly square. Training uses batch 128 with a cosine learning-rate schedule. Every crop is augmented randomly using crop jitter, perspective and elastic warps, ruled paper and bleed-through, low resolution,

Configuration	ParseBench-2	OmniParseBench	ParseBench-1	olmOCR-bench
PDF text only	46.3	35.1	39.81	46.2
+ LayoutNet (no OCR)	53.7	58.6	49.27	52.9
+ OCR (no LayoutNet)	51.2	49.6	42.50	52.7
Release (OCR + LayoutNet)	**59.2**	**74.8**	**52.68**	**60.9**

Table 7: Each pipeline stage on every benchmark (the numbers behind Fig. 6).

System	CER ↓	95% CI	Recall
dots.mocr (VLM)	0.024	–	–
chandra-ocr-2 (VLM)	0.029	–	–
PP-OCRv6-medium	0.034	–	0.98
OCRia-nano-1	0.048	[0.031, 0.062]	0.93
Indra 1.1.2	0.062	[0.026, 0.098]	0.94
tesseract-5	0.064	[0.042, 0.079]	0.92
Marker (fast)	0.067	[0.029, 0.108]	0.94

Table 8: BHL IMPACT ground truth, 2,165 historical pages: reading-lane character error rate with 95% confidence intervals where we ran the system ourselves.

blur, JPEG, noise, faded or heavy ink, binarisation and slight skew.

Detector and router. The line detector (0.08M parameters) predicts text-line probability maps on pages resized to a long side of 1600 pixels. Boxes are cut at column gutters and printed column rules. The router (39k parameters) classifies each line crop into one of four script groups and predicts whether the line is upside down.

LayoutNet. A 6-layer pre-norm transformer at width 256 over text units, with a strided CNN image branch whose patch tokens the top layers cross-attend to (4.3M parameters in total). Node, pair and class heads are trained jointly on teacher labels grounded to the units (Section 3).

C Evaluation details

Benchmark versions and settings. ParseBench-2 is the public release v2.07 (`llamaindex/ParseBench-2`) scored with the official `parse-bench` 2.2.0. Reference systems are re-scored from their published outputs. OmniParseBench is scored with the official v1.7 harness. We set `-keep-headers-footers` for LiteParse on ParseBench-1, ParseBench-2 and OmniParseBench, whose ground truth keeps running heads, and `-no-links` everywhere. olmOCR-bench runs with LiteParse's defaults, which strip running heads, plus `-no-links`. BHL uses the leaderboard's scorer (v3.0) on all 2,165 pages.

Host	CPU	Tess.	Indra	v6 tiny	v6 small	OCRia-nano-1
Azure 1	AMD EPYC, 3.1 GHz	13.5	26.6	12.6	5.4	25.3
Azure 2	AMD EPYC, 3.0 GHz	12.6	26.0	13.0	5.4	24.7
Azure 3	AMD EPYC, 3.1 GHz	13.4	27.7	13.8	5.7	25.6
AWS	Intel Xeon, 3.8 GHz	16.6	38.8	19.4	6.2	35.3
GCP	AMD EPYC, 2.4 GHz	15.0	29.0	14.6	6.0	28.3
Mean		14.2	29.6	14.7	5.7	27.8
sd		1.6	5.3	2.7	0.4	4.4

Table 9: Pages per second on 300 random ParseBench pages, 16 single-core workers in a 16-vCPU container. All configurations run in the same container on each host. Tess. = LiteParse default; v6 tiny / small = LiteParse + LayoutNet + PP-OCRv6 through RapidOCR.

Reproducing the ParseBench-2 rows. With `parse-bench` installed and the ocria-nano models pulled (`lit models pull ocria-nano`), `parse-bench run liteparse_ocria_nano -dataset parsebench-2` reproduces the release row; `indra` and `marker_fast` run Indra 1.1.2 and Marker 2.2.0 locally (their CLIs on `PATH`), and `liteparse_tesseract` the default configuration.

Competitor runs. Indra 1.1.2 and Marker 2.2.0 (fast mode) are evaluated. Indra runs with math enabled, as LiteParse's math is always on, and with the same per-benchmark header and footer setting as LiteParse. Marker runs with its defaults, except that its bundled Indra predictor is limited to one ONNX Runtime thread per worker, like every other configuration in the speed runs.

Speed protocol. Throughput is measured on the same 300 random ParseBench pages as Table 5 with 16 single-core workers in a 16-vCPU container. Latency is one worker on the same pages in four interleaved batches. Every configuration runs in the same container on each of five hosts (Table 9) to keep numbers comparable.

D Runtime

The models run in-process in LiteParse's Rust engine on ONNX Runtime, loaded at run time. The engine uses only the CPU execution provider, with one intra-op thread pool shared by every model session. The models download with `lit models pull`, which pins each file's hash. Installed with ONNX Runtime, the package takes 59.9 MB on Linux and 75.3 MB on macOS.