

OCRia-1 Technical Report

Aman Rangapur* Antonio Jimeno Yepes*[†] Daniel Bustamante*
Nutchapol Dendumrongsup Sebastián G. Acosta Pierre-Loïc Doulcet Simon Suo

amanrangapur@gmail.com, {antonio, daniel, simon}@runllama.ai

* core contributors · [†] corresponding author

Abstract

We introduce **OCRia-1**, a 4B vision-language model (VLM) for document parsing that excels at preserving table structure, extracting chart data and retaining text formatting. OCRia-1 handles content that has traditionally been challenging in document parsing: merged cells and spanning headers, tables with thousands of cells, chart data, and meaningful text formatting such as bold, italics, subscripts and superscripts, preserving details needed for business-critical document analysis. At the time of this report, it achieves the highest scores on tables, charts and text formatting among the systems evaluated, including a 35B model and Mistral OCR 4.1. Its mean ParseBench score over the four dimensions we evaluate is 0.87, against 0.75 for the next-best system.

We emphasize data quality and reward design in training OCRia-1. Synthetic pages provide exact ground truth for long tables and rich formatting, while real pages are transcribed using a pipeline grounded in the PDF text layer. We attribute OCRia-1’s performance primarily to careful data curation that prioritizes quality and diversity over quantity. Dimension-specific rewards evaluate table reconstruction, chart-data extraction and text-formatting fidelity. We train the model on the curated data using supervised fine-tuning (SFT), then refine it through reinforcement learning with verifiable rewards (RLVR).

To our knowledge, OCRia-1 is the fastest published document-parsing VLM in its class, decoding at about 4,700 tokens per second with a single request on one B200 GPU, about nine times faster than the base model on stock vLLM. This is achieved through a block-diffusion speculator, quantization and custom kernels.

October 2026

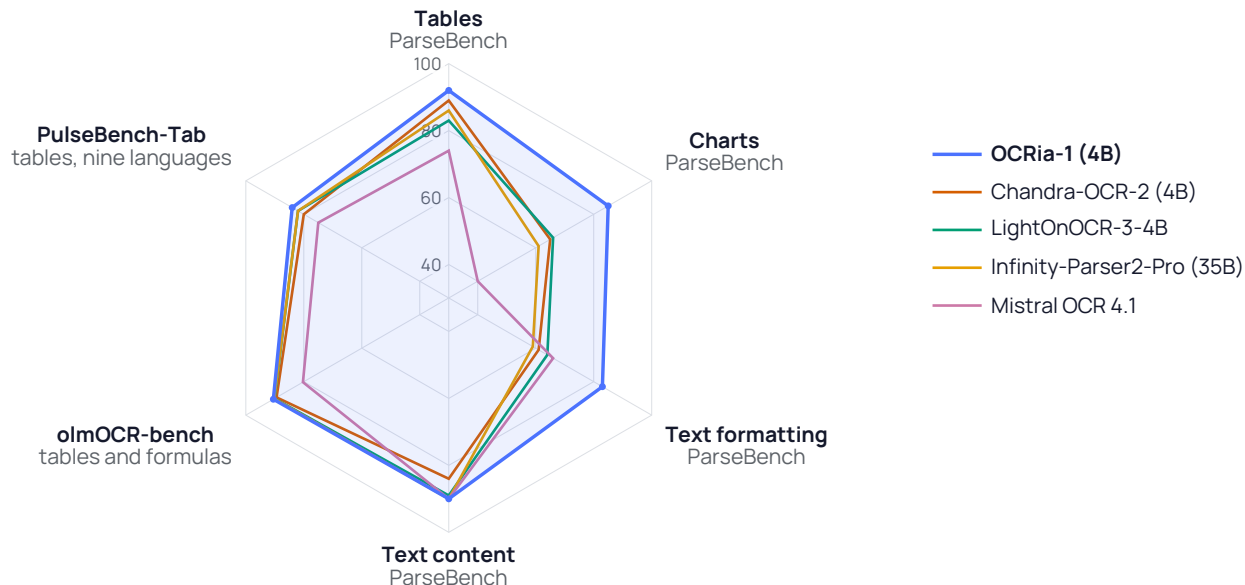


Figure 1 OCRia-1 has the highest score on every axis against leading document-parsing VLMs, on the four ParseBench dimensions and on olmOCR-bench and PulseBench-Tab from Table 3, with olmOCR-bench restricted to its table and formula categories. Scores are on a 0–100 scale, and the radial axis starts at 30.

1 Introduction

Document parsing converts a document page into text that keeps the structure of the page, such as its headings, reading order, tables and formulas. Some documents store their text but not its structure; a born-digital PDF, for example, records characters and their positions, but not which characters form a heading or a paragraph [29]. Others, such as scanned or photographed pages, are only images. In both cases, tables, charts, formatting and formulas are among the hardest elements to recover. Parsing mistakes are costly, because retrieval systems, extraction pipelines and document agents read only the parsed text and cannot correct it [48–50]. For example, a value placed under the wrong column header becomes a wrong answer, although the table still looks well formed [48]. Parsing must also be fast, because training-data generation and retrieval indexing process millions of pages, and document agents need low-latency parsing to answer questions quickly.

Compact document-parsing VLMs have become the standard way to parse documents, and a 1–4B parameter model that reads the page now replaces the heuristic pipelines in most open systems [2, 10, 24, 25, 40]. These models read plain text well. They are less accurate on complex tables and charts. Tables with merged cells, spanned headers or irregular shapes come out with cells in the wrong place, and long tables are often cut short or lose rows. Charts are usually treated as pictures, so most models describe them or skip them instead of writing out their data. These models are also slow. They write one token at a time, and a dense page of tables runs to thousands of tokens, so parsing thousands of pages takes hours of GPU time. We built OCRia-1 for these elements and for speed.

We introduce **OCRia-1**, a 4B document-parsing VLM built on Qwen3.5-4B [32, 33]. OCRia-1 transcribes the page into Markdown, writing tables as HTML with their full structure, charts as tables of their data points, and formulas as $\text{\LaTeX}$. OCRia-1 is more accurate on tables, charts and text formatting than every system we compare it with (Figure 1), and its mean ParseBench score over the four dimensions we evaluate is 0.87, against 0.75 for the next-best system (Table 3). It is also, to our knowledge, the fastest published document-parsing VLM. On one B200 GPU, the served model decodes at about 4,700 tokens per second with a single request and about 1,900 tokens per second per request with eight concurrent requests. With 64 concurrent requests, it parses about 10 pages per second, at no measurable cost on the benchmarks (Section 5.3). We plan to make a version of OCRia-1 available.

We gave priority to tables, charts and text formatting in both the data and the training. These elements carry much of the information in enterprise documents, such as financial reports, insurance filings, contracts and medical publications. Our approach is to invest in training data for these elements, in rewards that score them directly, and in inference speed. The training set has two kinds of pages (Section 2). Real pages come from public document collections, chosen where document-parsing VLMs still fail, such as dense tables, tables with cells that span several rows or columns, and charts. A pipeline of proprietary vision-language models transcribes each real page, and we keep the transcription only when independent checks agree with it (Section 2.2). Synthetic pages come from our own generators, which render varied page layouts together with their exact transcription. They supply what real collections do not contain in quantity, such as long tables, icon tables and rich inline formatting. Supervised fine-tuning (SFT) teaches the model to transcribe these pages, and reinforcement learning with verifiable rewards (RLVR) then rewards it for table cells placed under the right headers, chart values and formatting marks.

To accelerate decoding, we use speculative decoding [4, 19] with a block-diffusion draft model [5], and adapt the number of draft tokens to the batch size to maintain high throughput at larger batch sizes. We also quantized OCRia-1 to four-bit NVFP4, while retaining FP8 or BF16 in layers where lower-precision quantization reduced evaluation scores. Finally, we customized vLLM with architecture-specific CUDA kernels and fused operations to improve GPU compute and memory-bandwidth utilization.

This report provides more information about how we trained OCRia-1 and is organized as follows. Section 2 describes our diverse, high-quality mix of real and synthetic data, which we view as the primary driver of OCRia-1’s performance. Section 3 describes SFT and RLVR with rewards that score table cells and chart values. Section 4 describes speculative decoding, quantization and serving optimizations. Section 5 compares OCRia-1 with eight other systems on four transcription benchmarks and measures its inference speed on one

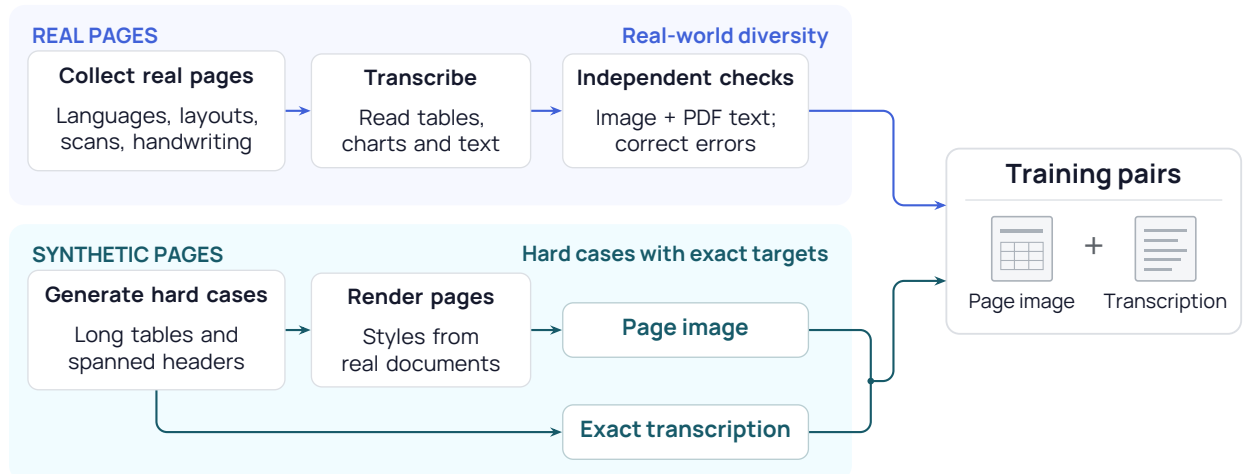


Figure 2 How the training set is built. For real pages, we generate transcriptions, check them against the page image and available PDF text, and correct errors before accepting them as ground truth. For synthetic pages, we generate the content first and render it using styles from real documents, so the correct transcription is already known.

B200 GPU. [Section 6](#) places this work among other document-parsing VLMs.

2 Data

Our data is collected strategically around the most challenging aspects of document parsing, including charts and long tables or tables with many spanning cells ([Section 6](#)). We developed a proprietary workflow to build a high-quality training corpus from both transcribed public documents and synthesised pages, to close the gaps that remain in public datasets. We used synthetic data to fill the gaps left by real data, which ensures the variety of hard cases needed for training. This allows us to include the long tail of difficult cases without the struggle of finding them in real data. Synthetic data comes with training ground truth by design, and we carefully designed the layouts of the synthetic examples to be as realistic as possible. This strategy ensures that the training set covers the aspects a model needs to get right, while keeping our customer data outside the training corpus. [Section 2.1](#) describes what we collected and why. [Section 2.2](#) describes how we generate ground truth for a real page: a teacher pipeline writes it, and independent checks decide whether we keep it. [Section 2.3](#) describes how the synthetic pages are generated.

2.1 Building the corpus

We built the corpus targeting the areas where document-parsing VLMs still fail, including dense and spanned tables, mathematics, non-Latin scripts, old scans and handwriting, and charts. We iteratively refine the SFT dataset, using model evaluations to guide targeted data collection and synthesis. We retain data from earlier rounds to preserve existing capabilities. The training set has 100,979 pages, about 0.37 B target tokens: 82,943 real pages and 18,036 synthetic pages (≈ 121 M target tokens). Real and synthetic pages share the ground-truth format shown in [Appendix A.1](#).

Real pages come from eleven source families, chosen for breadth of layout and a permissible license ([Table 1](#)). We collected about one million PDFs from these sources and kept the pages counted in [Table 1](#). To ensure diversity, we take at most three pages per document (six for table collections) and let no single source family, language or query group dominate. We use the PDF text layer to filter for pages that contain tables, charts and mathematics.

Synthetic pages let us generate challenging examples at scale with exact ground truth. Long tables, for example, are hard to find in real collections and difficult to annotate accurately. Pages emphasizing each element type are produced by a dedicated generator ([Table 2](#)). We ensure that the generators produce diverse

Source	Pages	Document class
Internet Archive	23,722	public-domain scans (1800-1975 books on science and mathematics, statistical annexes, encyclopaedias, and letters) and magazines
arXiv	16,142	scientific formula-dense pages
US / UK government statistical publications	12,966	Federal Reserve, IRS SOI, Census, FDIC, BEA, EPA, USDA, EIA.
PubMed Central	6,988	flowcharts (PRISMA, decision trees), scientific articles
Contracts and public documents with tracked changes	6,903	CUAD contracts, UN and EMA public documents
Library of Congress	5,199	handwritten manuscripts and early printed matter with volunteer transcriptions
OpenAlex-indexed articles	2,447	rare languages, tables in Arabic / Chinese / Korean / Japanese / Persian, charts
EUR-Lex	1,720	22 non-English language editions
govinfo, NTRS, ERIC, DailyMed, SEC EDGAR, municipal ACFRs, BIS, EDINET	5,879	government and financial reports, born-digital tables, NASA and education scans
World Bank Open Knowledge Repository	1,479	chart-dense reports
Publisher sites	492	magazines

Table 1 Real pages collected by source. We exclude 994 of these pages from training because they exceed the context limit.

Element type	Pages
Tables and charts	13,953
Forms and formatting	3,189
Text variants	894

Table 2 We focus synthetic generation on tables and charts to obtain complex examples with exact ground truth.

formatting that still reflects what real documents look like.

2.2 Transcribing and verifying real pages

We use our proprietary ground truth generation workflow to produce accurate ground truth for SFT at a large scale. The workflow consists of the following steps (Figure 2).

Teacher annotations. The teacher is composed of workflows to parse different complex page elements such as tables and charts, whose outputs are then combined with text predictions into the transcription of the entire page. The transcription follows the output format described in Appendix A.1.

Annotation audit. Since teacher annotations might not be perfect, once the initial transcription of a page has been produced, a mid-size VLM checks that the transcription is correct with respect to the source page image using a set of auditing rules, which target different aspects of the transcription (Appendix A.2).

The rules check for various common failure modes, such as incorrect table headers and table cells holding the wrong value. When a rule fails, the VLM is called to correct the transcription, focusing on the detected issues.

Text layer grounding. For digital PDF pages in which the text content can be extracted directly, the ground truth is compared against the PDF text to ensure the completeness of the transcription. While this text layer lacks the semantic structure of the page, the ground truth needs to at least match the text for it to be correct.

Real pages

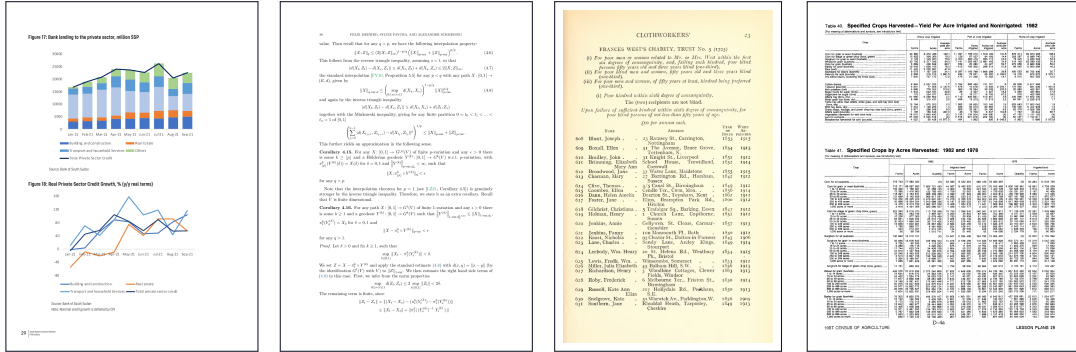


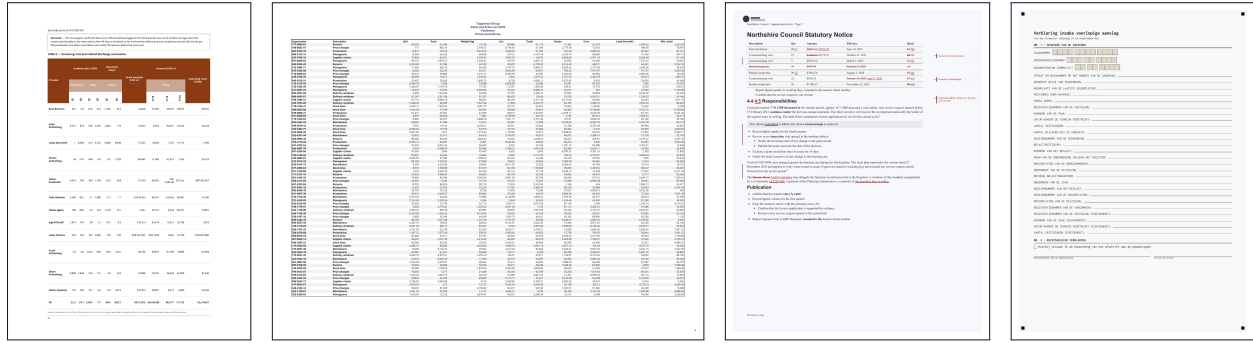
Chart-dense report page

arXiv, formulas

Internet Archive, scan

Government statistical table

Synthetic pages



Spanned table

Very long, dense table

Text with tracked changes

Form with fields

Figure 3 Example training pages. Top: real pages from the chart, arXiv, Internet Archive and government statistical collections. Bottom: synthetic pages from the spanned-table, very-long-table, tracked-changes and form generators.

Correction & re-evaluation. In correction & re-evaluation, we use a large VLM to correct the page transcription and re-evaluate the correctness of the generated ground truth.

A digital PDF page is approved as ground truth when it passes both the audit and the text layer grounding; a page that fails either check, and every scanned page, goes to correction & re-evaluation (Appendix A.2).

Only when the large VLM’s validation and correction attempts fail do we route the page for human revision.

2.3 Generating synthetic pages

Real documents best represent what the model will see in use. However, some of the complex document types that these models struggle with are scarce, and for others precise ground truth cannot be extracted reliably at scale. To close those gaps, we use synthetic data that follows the distribution of real documents while keeping a high diversity of styles.

We developed an internal workflow that generates PDF pages from pre-synthesised ground truth: the generator samples document styles and templates from real documents while rendering that ground truth. The generators cover tables and charts, including long tables of 5k–12k tokens, spanned headers, icon cells and 28 chart types; forms and inline formatting, including checkboxes and fill-in fields; and documents shown with tracked changes (Table 2). The synthetic ground truth is generated in the same format as the ground truth of the real pages (Appendix A.1).

3 Training

We train OCRia-1 in two stages for accurate document parsing. SFT teaches Qwen3.5-4B to transcribe document pages in our output format. RLVR addresses errors left by SFT by scoring page structure and content directly, such as table cells, chart values and formatting marks, while penalizing degenerate output such as repetition. A separate draft model accelerates inference (Section 4).

SFT. We fine-tune Qwen3.5-4B on 100,979 pages (Section 2). We prompt the model to transcribe each page (Appendix F), using Markdown with HTML tables to represent its content and structure.

We render each page at 1,024px on its long side and use a 20,480-token context to accommodate long tables. Appendix B.1 gives the filtering and training settings; Appendix B.3 reports results at higher image resolutions.

RLVR. After SFT, we train the model with GRPO [8, 36] with the goal of refining and enhancing the behaviors that the model has already learned to some extent. We also want it to improve its formatting and consistency. Our RLVR setup differs from previous work in two ways. First, it uses a larger set of rewards covering different page elements, along with additional penalties. Second, it weights the rewards dynamically according to their variance across rollouts, so that page elements that appear less often still contribute meaningfully to the update. Appendix B.2 gives the settings.

- **Training data.** The stage trains on 11,234 pages (Table 7). For rollout generation we keep the image at its stored resolution, mostly at or above 150 dpi (Table 8). We train on the full corpus without removing pages with low GRPO advantages, as removing such pages gave mixed results (Appendix B.5).
- **Reward design.** Our reward function combines terms across multiple dimensions of transcription quality (Section B.4.3), giving partial credit for correct aspects of imperfect outputs. Most of its thirty-four terms (Appendix B.4) are scores, which give credit for what is right. Twelve are penalties, which are zero unless the output has the error they check for, e.g., a penalty for merged or split tables (Section B.4.1).

To ensure the reward function is well defined and hard to game, we test each reward term against ground truth before training (Section B.4.2).

- **Reward weighting.** We carefully weight reward terms with variance-based reweighting to balance their influence on group-relative updates. A term’s influence depends on both its weight and its variation across rollouts, so fixed weights alone can let one dimension dominate at the expense of others (Section B.4.4). We also avoid normalizing each term by its own within-group statistics [14], which can give penalties as much influence as the quality terms they are intended to protect.

We train several checkpoints that are stronger in different dimensions and merge them into the final OCRia-1 model by checkpoint averaging [34, 45].

4 Inference

We accelerate inference through speculative decoding, mixed-precision quantization, and optimizations to the vLLM serving runtime.

Speculative decoding. We use speculative decoding [4, 19] for a lossless inference speedup: a small draft model proposes several tokens ahead and the full model verifies them all in one forward pass. We trained a six-layer DFlash 2 [5, 15] block-diffusion speculator with 0.69B parameters on document pages from our training corpus. We use a block of 32 tokens, against the 8 to 16 tokens used by DFlash and by the draft of HunyuanOCR-1.5 [20], so a verification step can produce up to 32 tokens.

We implemented a dynamic draft length, using longer drafts at low concurrency to improve per-request decoding speed and shorter drafts at higher concurrency to preserve throughput as verification costs grow. With 31 draft tokens, each verification step emits an average of 12.5 to 17.5 tokens depending on the page set, including one token from the target. These averages remain consistent across evaluated batch sizes (Table 22).

We also refined DFlash 2’s draft-token selection with dynamic programming, slightly increasing accepted tokens per step at negligible cost. [Appendix E.2](#) provides further implementation details.

Quantization. We quantize OCRia-1 to a mix of NVFP4 and FP8 [1], while retaining BF16 for the vision encoder. We calibrate on the model’s own transcriptions and select each layer’s precision, keeping only configurations whose scores remain within measurement noise of the BF16 baseline.

Serving optimizations. We customize vLLM [18] to reduce repeated work and improve GPU utilization. Our port of ReplaySSM [16] replays accepted tokens from a saved recurrent state, avoiding a state snapshot at every draft position. We write several CUDA kernels to optimize inference. We also fuse activation and quantization operations, build batch metadata once per step, and reuse the shared prompt prefix across requests. [Appendix E](#) gives the quantization, implementation and validation details.

5 Evaluation

We evaluate OCRia-1 as a document-parsing VLM along two axes: parsing quality and serving speed. We measure parsing quality across four public benchmarks with complementary coverage, from enterprise pages and broad document layouts to cropped tables in nine languages; we measure serving speed on one B200 GPU. We first describe the benchmark scope and the evaluation protocol ([Section 5.1](#)), then report parsing results ([Section 5.2](#)), and finally measure inference speed ([Section 5.3](#)).

5.1 Benchmarks and protocol

We have considered four well-known public benchmarks. The benchmarks rely on different document distributions due to their focus, and this variety ensures a thorough evaluation of our model across a diversity of document parsing problems. Unless otherwise specified, and to ensure fair evaluation, we have evaluated OCRia-1 with the transcription prompt of [Appendix F](#), on the same page renderings at each benchmark’s standard resolution, and scored it with the benchmark’s own scorer. [Appendix D](#) provides additional details about the evaluation protocol. The benchmarks are:

- ParseBench [48] contains pages from enterprise documents such as insurance, finance and government filings, and grades text content, text formatting and charts with rules written against each page, and tables with a table-structure metric ([Appendix D](#)). Its fifth dimension, visual grounding, scores whether each extracted block is located, labelled and matched to its content; OCRia-1 does not output bounding boxes at this moment, so we leave that dimension out.
- olmOCR-bench [29] runs 7,010 pass-or-fail unit tests on 1,403 pages, grouped into arXiv mathematics, old scans with mathematics, old scans, tables, multi-column pages, headers and footers, and long passages of small text.
- PulseBench-Tab [31] scores 1,820 cropped tables in nine languages by comparing the predicted cell grid with the reference grid.
- fr-bench-pdf2md [35] runs olmOCR-style tests on 721 difficult French pages.

5.2 Main results

OCRia-1 has the highest score on every benchmark in [Table 3](#), and its lead comes from the content we focused on: tables, charts and text formatting. Its margin is largest on charts and text formatting, where some other systems score close to zero, and on plain text it is as accurate as the best system on ParseBench.

On ParseBench it scores 0.85 on charts against 0.66 for the next-best system, 0.83 on text formatting against 0.68, 0.92 on tables against 0.89, and 0.90 on text content, tied with the best other systems ([Table 4](#)). It also scores highest on the two other benchmarks that contain these elements: PulseBench-Tab, which contains only tables (0.84 against 0.82), and fr-bench-pdf2md, which contains forms, graphics and long tables (72.1 against 71.4, [Table 3](#)).

Model	ParseBench*	olmOCR-bench [†]	PulseBench-Tab	fr-bench-pdf2md
Infinity-Parser2-Pro (35B)	0.74	90.2	0.82	54.9
LightOnOCR-3-4B	0.75	90.1	0.82	71.4
olmOCR-2 (7B)	0.43	83.1	0.80	56.9
Chandra-OCR-2 (4B)	0.75	89.4	0.80	65.5
dots.mocr (3B)	0.56	87.4	0.82	56.8
Surya OCR 2 (0.7B)	0.58	85.4	0.71	50.9
LightOnOCR-2-1B	0.61	88.1	0.81	53.5
Mistral OCR 4.1	0.68	80.3	0.75	54.1
OCRia-1	0.87	90.5	0.84	72.1

*Mean over text content, text formatting, tables and charts; excludes visual grounding.

[†]For olmOCR-bench, the mean of its three categories on tables and formulas (Tables, arXiv math and Old scans math); Table 16 gives the official score over all eight categories. Appendix D.1 breaks down olmOCR-bench, PulseBench-Tab and fr-bench-pdf2md by category.

Table 3 OCRia-1 leads on four well-known benchmarks, especially in tables, charts and text formatting.

Model	Mean*	Tables	Charts	Text formatting	Text content
Infinity-Parser2-Pro (35B)	0.74	0.86	0.61	0.59	0.90
LightOnOCR-3-4B	0.75	0.83	0.66	0.64	0.89
olmOCR-2 (7B)	0.43	0.73	0.14	0.02	0.84
Chandra-OCR-2 (4B)	0.75	0.89	0.65	0.61	0.84
dots.mocr (3B)	0.56	0.85	0.00	0.47	0.90
Surya OCR 2 (0.7B)	0.58	0.80	0.01	0.68	0.84
LightOnOCR-2-1B	0.61	0.78	0.15	0.64	0.88
Mistral OCR 4.1	0.68	0.74	0.40	0.66	0.90
OCRia-1	0.87	0.92	0.85	0.83	0.90

*Mean over the four dimensions shown; excludes visual grounding.

Table 4 On ParseBench, OCRia-1 leads on tables, charts and text formatting, and ties for best on text content.

olmOCR-bench has no test of charts or text formatting, so for it [Table 3](#) gives the mean of the three categories that test tables and formulas, two of the elements OCRia-1 is built to transcribe. OCRia-1 has the highest score on these categories, 0.3 points ahead of Infinity-Parser2-Pro ([Table 16](#)). In the additional comparison in [Table 17](#), OCRia-1 also has the highest score over all eight categories, again by 0.3 points. This comparison combines our runs at 1,288px with published scores for Mistral OCR 4.1 [27] and olmOCR-2 [30].

On PulseBench-Tab OCRia-1 also leads on the other three measures of [Table 18](#): its median table score is 0.95 against 0.93 for the next-best system, it gives a parseable table for 99.1% of the tables against 97.7%, and it reproduces 485 of the 1,820 tables perfectly against 431.

[Appendix G](#) shows outputs of OCRia-1 on tables, charts, text formatting and handwritten formulas. Also, [Appendix D.2](#) shows cases where OCRia-1’s performance is lower. Most of the low-performing cases are the hardest cases within the areas we targeted, e.g. old scanned documents and less frequent languages.

5.3 Inference efficiency

For inference speed, we use a separate comparison set of fast open-weight document-parsing VLMs, including compact models and systems with speculative decoding. On one B200, OCRia-1 leads the five evaluated systems in both decoding speed and page throughput at one and eight concurrent requests ([Figure 4](#)). Its per-request decoding speed is respectively $3.4\times$ and $2.0\times$ that of the next-fastest system.

In a separate concurrency sweep, OCRia-1 parses 10.21 pages per second at 64 concurrent requests, the highest concurrency tested with p90 latency below 10s ([Table 20](#)). We evaluate the serving stack described in [Section 4](#); [Appendix E.1](#) gives the measurement protocol, full results and comparison with the BF16 checkpoint on stock vLLM.

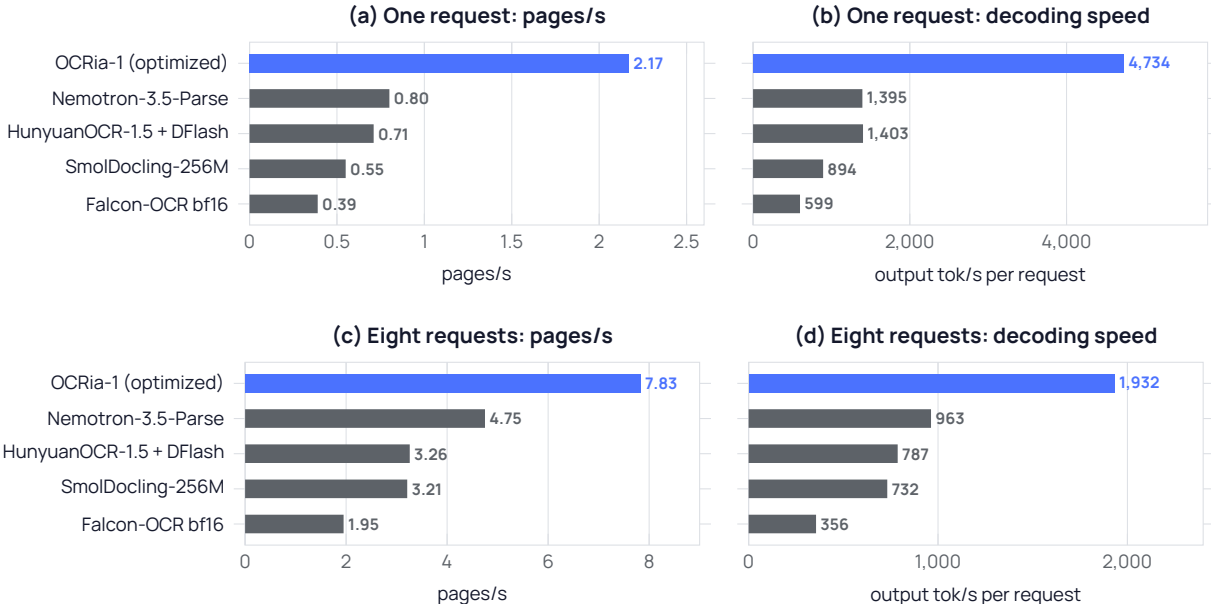


Figure 4 OCRia-1 leads in page throughput (left) and decoding speed per request (right) at one (top) and eight (bottom) concurrent requests. Each system runs on one B200. [Table 21](#) gives the exact values.

6 Related Work

OCRia-1 reads the whole page with one VLM, and it differs from earlier document-parsing VLMs mainly in its training data and its rewards. The document-parsing VLMs we compare against include olmOCR-2 [30], dots.mocr [51], LightOnOCR [25, 40], Chandra [7], Infinity-Parser2-Pro [13] and Surya [28], and other recent

ones include the DeepSeek-OCR series [42, 43] and TeleOCR [2]. Pages with correct annotations are scarce, so many training sets are labeled by a larger model or generated synthetically. These models are weakest on the page types with the fewest annotated examples, such as charts, long or heavily spanned tables, low-resource languages and scanned pages. We built our training data around these page types, and we believe this data is one of the reasons OCRia-1 outperforms the models we compare it with (Sections 2 and 5).

Some of these models continue training with reinforcement learning on a reward computed from the parsed output [13, 30, 40, 41]. Compared to previous work, our RLVR has two main differences. The first is a larger set of rewards for different page elements and more penalties (Section B.4.3). The second is a mechanism to dynamically weight the different rewards based on rollout variance (Section B.4.4), which gives a higher contribution to less frequent and varied rollout elements.

We combine a block-diffusion draft model with quantization and custom kernels, and we report the resulting inference performance (Section 5.3). There are other models using several techniques to speed up inference, such as GLM-OCR [10], which predicts several tokens per step, and diffusion-based models [9, 26], which write many tokens in parallel. Some document-parsing VLMs also use speculative decoding: HunyuanOCR-1.5 with a DFlash draft model [20], HPD-Parsing with multi-token prediction heads [44], and GravityOCR with drafts from the same network [17]; Appendix E.2 compares how many draft tokens they accept per step with ours.

7 Conclusion and Future Work

At the time of this report, OCRia-1, a 4B document-parsing VLM, has the highest scores on tables, charts and text formatting among all the systems we compare it with (Table 3), and its largest leads are on charts and text formatting in ParseBench (Table 4). At ten pages per second, one GPU parses about 860,000 pages a day (Section 5.3), so the training data for the next model, the backlog of a document store or the attachments a document agent receives can be parsed on one machine. We plan to make a version of OCRia-1 available soon.

Most of the work for OCRia-1 went into preparing the training data. Our focus on high quality data rather than quantity is one of the reasons of OCRia-1’s performance. Synthetic pages came with ground truth from their generators, and real pages were transcribed by a multi-stage pipeline whose output we checked by rule and against the text stored in the PDF. After SFT, we trained the model with RLVR and averaged checkpoints, and a draft model significantly speeds up decoding.

The same way of collecting and annotating data can be applied to prepare data for the cases where OCRia-1 currently scores lowest, such as old scans, tables in Arabic, Greek and Korean, and long tables whose rows run past the output budget (Appendix D.2). Also, we have not yet released a model capable of predicting layout, which is left as work for a future release.

References

- [1] Eduardo Alvarez, Omri Almog, Eric Chung, Simon Layton, Dusan Stosic, Ronny Krashinsky, and Kyle Aubrey. Introducing NVFP4 for efficient and accurate low-precision inference. NVIDIA Technical Blog, <https://developer.nvidia.com/blog/introducing-nvfp4-for-efficient-and-accurate-low-precision-inference/>, June 2025.
- [2] Peng Cai, Zhaofan Zou, Shifa Liu, Yikun Wang, Jiawei Tang, Kaicheng Yang, Meng Tong, MingKun Jiang, Zhongjiang He, and Hao Sun. TeleOCR: Navigating Document Parsing Across Digital and Camera-Captured Documents, 2026. URL <https://arxiv.org/abs/2608.12898>.
- [3] Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. Medusa: Simple LLM inference acceleration framework with multiple decoding heads, 2024. URL <https://arxiv.org/abs/2401.10774>.

- [4] Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling, 2023. URL <https://arxiv.org/abs/2302.01318>.
- [5] Jian Chen, Yesheng Liang, and Zhijian Liu. DFlash: Block diffusion for flash speculative decoding, 2026. URL <https://arxiv.org/abs/2602.06036>.
- [6] Xiaoyin Chen et al. Self-evolving curriculum for LLM reasoning. *arXiv preprint arXiv:2505.14970*, 2025.
- [7] Datalab. Chandra, 2026. URL <https://github.com/datalab-to/chandra>. Open model release, version 2.
- [8] DeepSeek-AI. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*; *journal version in Nature* 645:633–638, 2025.
- [9] Hejun Dong, Junbo Niu, Bin Wang, Weijun Zeng, Wentao Zhang, and Conghui He. MinerU-Diffusion: Rethinking Document OCR as Inverse Rendering via Diffusion Decoding. In *European Conference on Computer Vision (ECCV)*, 2026.
- [10] Shuaiqi Duan, Yadong Xue, Weihang Wang, Zhe Su, Huan Liu, Sheng Yang, Guobing Gan, Guo Wang, Zihan Wang, Shengdong Yan, Dexin Jin, Yuxuan Zhang, Guohong Wen, Yanfeng Wang, Yutao Zhang, Xiaohan Zhang, Wenyi Hong, Yukuo Cen, Da Yin, Bin Chen, Wenmeng Yu, Xiaotao Gu, and Jie Tang. GLM-OCR Technical Report, 2026. URL <https://arxiv.org/abs/2603.10910>.
- [11] Roger Girgis et al. Constrained group relative policy optimization. *arXiv preprint arXiv:2602.05863*, 2026.
- [12] Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Rozière, David Lopez-Paz, and Gabriel Synnaeve. Better & faster large language models via multi-token prediction, 2024. URL <https://arxiv.org/abs/2404.19737>.
- [13] 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>.
- [14] Yuki Ichihara, Yuu Jinnai, Tetsuro Morimura, Mitsuki Sakamoto, Ryota Mitsuhashi, and Eiji Uchibe. MO-GRPO: Mitigating reward hacking of group relative policy optimization on multi-objective problems. *arXiv preprint arXiv:2509.22047*, 2025.
- [15] Inco AI. DFlash 2: Keep drafting parallel. Inco AI blog, <https://inco.ai/blog/dflash2/>, August 2026.
- [16] Johnny-Liou. [RFC]: ReplaySSM: Cache SSM inputs instead of state for faster standard and speculative decode (Mamba2 + GDN). vLLM GitHub issue #47572, <https://github.com/vllm-project/vllm/issues/47572>, July 2026.
- [17] Dohyun Kim, Sungjun Han, Hyunguk Kim, Yusik Kim, Jamin Shin, Paul Hongsuck Seo, and Hongjoon Ahn. Diffusion drafts, AR verifies: Accelerating document OCR with self-speculative decoding, 2026. URL <https://arxiv.org/abs/2609.26638>.
- [18] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th ACM Symposium on Operating Systems Principles (SOSP)*, 2023. URL <https://arxiv.org/abs/2309.06180>.
- [19] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning (ICML)*, 2023. URL <https://arxiv.org/abs/2211.17192>.

- [20] Gengluo Li, Xingyu Wan, Shangpin Peng, Weinong Wang, Hao Feng, Yongkun Du, Binghong Wu, Zheng Ruan, Zhiqiong Lu, Liang Wu, Pengyuan Lyu, Huawen Shen, Zibin Lin, Shijing Hu, Jieneng Yang, Hongbing Wen, Guanghua Yu, Hong Liu, Bocho Wang, Can Ma, Han Hu, Chengquan Zhang, and Yu Zhou. HunyuanOCR-1.5: Making lightweight OCR VLMs faster and better, 2026. URL <https://arxiv.org/abs/2607.04884>.
- [21] Renyu Li, Antonio Jimeno Yepes, Yao You, Kamil Pluciński, Maximilian Operlejn, and Crag Wolfe. SCORE: A semantic evaluation framework for generative document parsing. *arXiv preprint arXiv:2509.19345*, 2025. evaluation code <https://github.com/Unstructured-IO/unstructured-eval-metrics>.
- [22] Xuefeng Li, Haoyang Zou, and Pengfei Liu. LMR: Less is more for RL scaling. *arXiv preprint arXiv:2502.11886*, 2025.
- [23] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-3: Scaling up inference acceleration of large language models via training-time test, 2025. URL <https://arxiv.org/abs/2503.01840>.
- [24] 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>.
- [25] LightOn. LightOnOCR-3. <https://huggingface.co/blog/lightonai/lightonocr-3>, October 2026.
- [26] Sean Man, Gilad Deutch, Roy Ganz, Roi Ronen, Shahar Tsiper, Shai Mazor, and Niv Nayman. DODO: Discrete OCR Diffusion Models, 2026. URL <https://arxiv.org/abs/2602.16872>.
- [27] Mistral AI. Mistral OCR 4.1. <https://docs.mistral.ai/models/ocr-4-1>, 2026. Model mistral-ocr-4-1, released July 2026.
- [28] Vikas Paruchuri and Datalab Team. Surya: A lightweight document OCR and analysis toolkit. <https://github.com/datalab-to/surya>, 2025. Surya OCR 2 model card: <https://huggingface.co/datalab-to/surya-ocr-2>.
- [29] Jake Poznanski, Jon Borchardt, Jason Dunkelberger, Regan Huff, Daniel Lin, Aman Rangapur, 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>.
- [30] Jake Poznanski, Luca Soldaini, and Kyle Lo. olmOCR 2: Unit test rewards for document OCR, 2025. URL <https://arxiv.org/abs/2510.19817>.
- [31] Pulse. PulseBench-Tab. <https://github.com/Pulse-Software-Corp/PulseBench-Tab>, 2026.
- [32] Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL <https://qwen.ai/blog?id=qwen3.5>.
- [33] Qwen Team. Qwen3.5-4B model card, 2026. URL <https://huggingface.co/Qwen/Qwen3.5-4B>.
- [34] Alexandre Ramé, Guillaume Couairon, Corentin Dancette, et al. Rewarded soups: Towards Pareto-optimal alignment by interpolating weights fine-tuned on diverse rewards. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.
- [35] Bruno Rigal, Victor Dupriez, Alexis Mignon, Ronan Le Hy, and Nicolas Mery. Benchmarking vision-language models for French PDF-to-Markdown conversion. <https://huggingface.co/datasets/pulsia/fr-bench-pdf2md>, 2026.
- [36] Zhihong Shao, Peiyi Wang, Qihao Zhu, et al. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.
- [37] Guangming Sheng et al. HybridFlow: A flexible and efficient RLHF framework. *arXiv preprint arXiv:2409.19256*, 2024. released as verl, <https://github.com/volcengine/verl>.

- [38] Taiwei Shi et al. Efficient reinforcement finetuning via adaptive curriculum learning. *arXiv preprint arXiv:2504.05520*, 2025.
- [39] Brandon Smock, Rohith Pesala, and Robin Abraham. GriTS: Grid table similarity metric for table structure recognition. In *International Conference on Document Analysis and Recognition (ICDAR)*, 2023.
- [40] 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>.
- [41] Baode Wang, Biao Wu, Weizhen Li, Meng Fang, Zuming Huang, Jun Huang, Haozhe Wang, Yanjie Liang, Ling Chen, Wei Chu, and Yuan Qi. Infinity Parser: Layout aware reinforcement learning for scanned document parsing, 2025. URL <https://arxiv.org/abs/2506.03197>.
- [42] Haoran Wei, Yaofeng Sun, and Yukun Li. DeepSeek-OCR: Contexts optical compression, 2025. URL <https://arxiv.org/abs/2510.18234>.
- [43] Haoran Wei, Yaofeng Sun, and Yukun Li. DeepSeek-OCR 2: Visual Causal Flow, 2026. URL <https://arxiv.org/abs/2601.20552>.
- [44] Shu Wei, Jingjing Wu, Lingshu Zhang, Qunyi Xie, Hao Zou, Le Xiang, Xu Fan, Yangliu Xu, Manhui Lin, Xiaolong Ma, Cheng Cui, Tengyu Du, and YY. HPD-Parsing: Hierarchical parallel document parsing, 2026. URL <https://arxiv.org/abs/2607.18839>.
- [45] Mitchell Wortsman, Gabriel Ilharco, Samir Ya Gadre, et al. Model soups: Averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *International Conference on Machine Learning (ICML)*, 2022.
- [46] Qiyang Yu, Zheng Zhang, Ruofei Zhu, et al. DAPO: An open-source LLM reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.
- [47] Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.
- [48] 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>.
- [49] Junyuan Zhang, Qintong Zhang, Bin Wang, Linke Ouyang, Zichen Wen, Ying Li, Ka-Ho Chow, Conghui He, and Wentao Zhang. OCR hinders RAG: Evaluating the cascading impact of OCR on retrieval-augmented generation. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2025.
- [50] Qintong Zhang, Bin Wang, Victor Shea-Jay Huang, Junyuan Zhang, Zhengren Wang, Hao Liang, Conghui He, and Wentao Zhang. Document parsing unveiled: Techniques, challenges, and prospects for structured information extraction, 2024. URL <https://arxiv.org/abs/2410.21169>.
- [51] Handong Zheng, Yumeng Li, Kaile Zhang, Liang Xin, Guangwei Zhao, Hao Liu, Jiayu Chen, Jie Lou, Qi Fu, Rui Yang, Shuo Jiang, Weijian Luo, Weijie Su, Weijun Zhang, Xingyu Zhu, Yabin Li, Yiwei Ma, Yu Chen, Yuqiu Ji, Zhaohui Yu, Guang Yang, Colin Zhang, Lei Zhang, Yuliang Liu, and Xiang Bai. Multimodal OCR: Parse anything from documents, 2026. URL <https://arxiv.org/abs/2603.13032>. dots.mocr.
- [52] Xu Zhong, Elahesh ShafieiBavani, and Antonio Jimeno Yepes. Image-based table recognition: Data, model, and evaluation. In *European Conference on Computer Vision (ECCV)*, 2020.

Appendix

A Data

This appendix gives the output format ([Appendix A.1](#)) and the verification rules ([Appendix A.2](#)) behind [Section 2](#).

A.1 Output format

■ OUTPUT FORMAT	
Mathematics	Inline formulas use <code>\$...\$</code> and display formulas use <code>\$\$...\$\$</code> ; a printed equation number such as (3) is written as plain text after the closing <code>\$\$</code> , not inside the formula.
Tables & Charts	Write each table as one line of HTML with <code><th></code> header cells and exact <code>rowspan</code> / <code>colspan</code> spans. Cells can contain <code></code> , <code></code> , <code><u></code> , <code>
</code> and <code><sup></code> / <code><sub></code> . Captions appear on the line before the table; charts are transcribed as tables.
Inline formatting	Use <code><sup></code> / <code><sub></code> tags rather than Unicode superscripts; <code>**bold**</code> , <code><i>*italic*</i></code> , <code><u></code> and <code>~~strike~~</code> mark bold, italic, underlined and struck-through text; <code><hr/></code> marks a printed horizontal rule.
Lists and forms	List items use <code>*</code> at the top level and <code>-</code> for nested items; checkboxes use <code>[x]</code> / <code>[]</code> , and fill-in fields use <code>[___]</code> .
Page furniture	Place the running header on the first line in <code><page_header>...</page_header></code> , the running footer on the last line in <code><page_footer>...</page_footer></code> , and the printed page number in <code><page_number>...</page_number></code> inside its header or footer.
Images	Write <code>![label: content]</code> without a filename: the label identifies the image type (logo, photo, chart, flowchart, ...), and the content is a sentence describing a photograph, the printed words of a logo or seal, or the digits of a barcode. Signatures and seals use <code>[signature: Name]</code> and <code>[seal: text]</code> ; an overlay watermark appears as one <code><watermark></code> line at the top.
Flowcharts	Place a fenced <code>mermaid</code> block after the image marker.

A.2 Verification rules

For a digital PDF page, the transcription is approved as ground truth if it passes both the audit and the text layer grounding; if either fails, the page is sent to correction and re-evaluation. A scanned page always goes to correction and re-evaluation, since there is no text layer to confirm the transcription.

[Table 5](#) lists the rules of the audit in [Section 2.2](#), by group. The audit and the reviewer answer every rule with pass, fail, uncertain or not applicable, and quote the part of the page that decided the answer.

B Training

This appendix gives the training details left out of the main text. [Appendices B.1](#) and [B.2](#) list the SFT and RLVR settings, [Appendix B.3](#) reports the experiments with the training resolution, [Appendix B.4](#) lists the components of the reward, and [Appendix B.5](#) reports the tests on pages that give no training signal.

Group	Rules	What it catches
■ TABLE RULES		25 rules, asked of every table
Cells	T01–T04	one wrapped cell split into several rows; separately printed records merged into one multi-line cell; invented spacer rows; real blank cells removed or filled
Spans	T05, T07–T09, T23	a category or subtotal label spanned across cells that hold values; wrong grid or span dimensions; repeated values merged into one cell; a real merged cell that loses what it covers; empty regions the rules treat as equivalent flagged as errors
Headers	T10–T12	header tiers flattened, dropped or made up; a header span over the wrong columns; missing or invented headers and columns
Content	T06, T13–T16	sparse or subsection values left out; a cell's text or number corrupted; superscripts, footnote marks or symbols detached from their value; a value in the wrong row or column; a long table that stops early or repeats
Boundaries	T17–T22	a borderless grid flattened into prose; one table split at a visual band; neighbouring tables woven together; a continuation that invents row relationships; printed occurrences duplicated or suppressed; captions, notes and totals attached to the wrong table
Source image	T24, T25	a source image or render that is incomplete or misleading; emphasis in the source turned into table structure
■ PAGE RULES		7 rules, asked once per page
Omissions	P01	text outside the tables that is missing: labels, captions, notes, headers, footers
Inventions	P02	invented or duplicated text, including a plausible footer number
Pictures	P03	a picture that is unaccounted for or given the wrong kind
Reading order	P04	a wrong order across columns, sidebars or right-to-left text; tables and prose out of place
Ownership	P05	a caption, note, total or detached cell held by the wrong block at a table boundary
Character fidelity	P06	a character outside the tables that differs from the print, including a printed typo that was corrected
Image completeness	P07	a clipped or unreadable source image, or a missing illustration
■ SCRIPT RULES		added for non-Latin pages
Image markers	P08, P09	a picture with no marker, two markers or the wrong class; a marker, header band or footer band that is not in the output format
Script fidelity	P10	an invented, dropped or substituted character; transliteration; digits converted to another numeral system (for example Arabic-Indic digits written as 0–9); a line repeated more often than it is printed

Table 5 Rules of the audit. The table rules apply to every table of a draft and the page rules once to the page. The last group is added for pages in non-Latin scripts.

Setting	Value
Rollouts per page	12
Sampling	temperature 0.8, top- p 0.9
Batch	64 pages, 12 rollouts each (768 sequences)
Optimiser	AdamW, learning rate 1×10^{-6}
Gradient clipping	0.5
KL penalty	in the loss, coefficient 0.04
Trust region	0.2 below, 0.28 above
Prompt and image	at most 5,632 tokens
Sampled transcription	at most 16,384 tokens
Hardware	eight H200 GPUs per node

Table 6 RLVR settings.

B.1 Supervised fine-tuning settings

The SFT context length is 20,480 tokens. Context-length filtering excludes 994 real pages from the collected corpus, leaving 100,979 pages.

We keep the training recipe fixed when comparing data mixes to isolate their effects. The SFT checkpoint for OCRia-1 was trained for two epochs with AdamW at a global batch of 64 sequences, a peak learning rate of 1×10^{-5} , a cosine schedule with linear warm-up over 3% of the steps (94 of 3,154), and zero weight decay. Sequences use bf16 without sequence packing (each GPU processes two sequences at a time). We chose the learning rate by starting from a setting that had already worked well (batch 16, learning rate 5×10^{-6}), scaling it with the square root of the batch size up to batch 64, and then keeping it fixed.

We train on four nodes with eight B200 GPUs each.

B.2 Reinforcement-learning settings

We train with verl [37] and the settings in Table 6. The trust region is asymmetric so that low-probability tokens have more room to rise [46]. The limits on the prompt and the sampled transcription add up to more than the 20,480-token context of the SFT stage, because a sampled transcription can be longer than its reference.

Table 7 gives the composition of the RLVR pages and Table 8 the resolution of their images.

B.3 Training resolution

Every training page is rendered at 1,024px on its long side, about 93 dpi and 800 image tokens for a portrait letter page, while the benchmarks serve pages at a higher resolution (Section 5). We also trained at a resolution closer to the serving one, and no transcription score changed by more than about a point.

We trained Qwen3.5-4B twice on a 30k-page subset of the SFT data, with the same recipe as the main runs, once with every page rendered at 1,024px and once at 1,536px, which gives 2.25 times the image tokens. The 1,536px run took 1.2 times as long. Table 9 gives the scores, with every benchmark served at its usual rendering. On ParseBench and PulseBench-Tab no score changed by more than 0.01, and on olmOCR-bench the table tests rose by 0.8 points and the math tests fell by 1.5.

The resolution at serving time matters far more than the resolution at training time. We served the same two models on ParseBench in two ways: at the benchmark’s standard rendering (150 dpi, about 1,650px on the long side of a letter page) and with every page downscaled to at most 1,024px. Downscaling cost both models about 0.08: the 1,536px model fell from 0.83 to 0.75 and the 1,024px model from 0.84 to 0.76. A

	Pages	Share
■ DOCUMENT TYPE		
Synthetic pages from our generators	3,788	33.7%
Curated table, chart and figure collections	2,609	23.2%
Government and regulatory filings (FEC, IRS)	1,799	16.0%
Public document-layout datasets (DocLayNet, RVL-CDIP)	952	8.5%
Forms	803	7.1%
Economic and central-bank publications (NBER, BIS, ECB, OECD)	451	4.0%
Historical manuscripts and magazines	258	2.3%
Financial filings	123	1.1%
Contracts and redlines (CUAD)	121	1.1%
Miscellaneous	330	2.9%
■ HOW THE REFERENCE TRANSCRIPTION WAS MADE		
Teacher model, from the page image only	5,613	50.0%
Generator (synthetic page)	3,669	32.7%
Teacher model, anchored to the PDF text layer	1,229	10.9%
PDF text layer only, no model	723	6.4%

Table 7 The 11,234 pages of the training split, by what kind of document each page comes from and by how its reference transcription was made. Document types are inferred from the page identifier, so the groups are reliable and the individual counts approximate. Of the references, 6.4% come from a page’s own text layer and 32.7% from the generator that drew the page; the rest were written by a model.

model trained only at 1,024 px therefore still benefits from the extra image tokens of a larger page at serving time ([Appendix C](#)), and downscaling pages to save tokens is expensive.

B.4 Reward components

[Tables 10](#) and [11](#) list every component of the reward, what it scores, and the benchmark rule it mirrors where one exists. The components are grouped by what they act on, and each is labelled with its panel, the group of components that shares one weight ([Section B.4.1](#)). The weights are a setting of each run, so the tables give the composition of the reward and not its weights.

Implied resolution	Share
~200 dpi	56.2%
~150 dpi	28.7%
~100 dpi	7.4%
Square (non-page aspect)	6.8%
Other (≤ 75 dpi)	0.8%
Other (≥ 225 dpi)	0.3%

Table 8 Page-image resolution in RLVR. The implied resolution is the long edge of the image divided by 11 inches.

Score	1,024 px	1,536 px
■ PARSEBENCH		
Mean	0.83	0.83
Table	0.87	0.88
Chart	0.79	0.79
Text content	0.85	0.85
Text formatting	0.80	0.80
■ OLMOCR-BENCH		
Overall	80.5	80.4
Table tests	88.0	88.8
Math tests	80.8	79.3
Reading order	71.8	71.6
■ PULSEBENCH-TAB		
Mean	0.82	0.83
Perfect tables	400	390

Table 9 Training at 1,024 against 1,536 px on the same 30k pages. Qwen3.5-4B, plain transcription, two epochs at batch 64 and learning rate 1×10^{-5} ; every benchmark served at its usual rendering.

Component	What it scores	Mirrors
■ WHOLE-PAGE TRANSCRIPTION		panels <code>bench</code> , <code>text</code>
<code>bench</code>	benchmark pass rate on the completion, weighted by content type; presence-based, so it rewards recall at no precision cost	olmOCR-bench
<code>cct</code>	transcription precision, $1 - \text{lev}(p, g) / \max(p , g)$ on de-tabbed plaintext; penalises extra, reordered and marked-up text	edit distance, SCORE [21]
<code>eos</code>	clean termination	
■ TABLES		panels <code>table</code> , <code>tbl_structure</code>
<code>table_align</code>	lead table term: mean of detection, cell-index accuracy and cell-content accuracy	SCORE <code>score_tables</code> , by direct call
<code>table_teds</code>	tree-edit-distance similarity, from the same call, kept because both benchmarks report it	TEDS [52]
<code>table_grits</code>	grid table similarity	GriTS [39]
<code>table_grits_trm</code>	ParseBench’s own table-dimension metric	the table dimension [48]
<code>table_struct</code>	structure-only tree-edit distance, text-blind and span-aware, which catches the <code>colspan</code> and <code>rowspan</code> cases the content metrics read as duplicates; implemented behind a switch and not enabled in the runs reported here	TEDS structure variant
<code>tbl_structure</code>	structure agreement scored in a panel of its own, the only structural table signal carrying weight in the runs reported here	
<code>table_seg_penalty</code>	$1 - \min(n_p, n_g) / \max(n_p, n_g)$ over table counts, so a merge and a split are penalised alike; a regex tag count, never on the per-cell path	
■ CHARTS		panel <code>chart</code>
<code>chart_keying</code>	fraction of reference data cells placed under the same row and column headers	<code>chart_data_point</code>
<code>chart_key_penalty</code>	penalty on a chart page that prints the reference numbers but leaves them unaddressable, counting only points whose failure reason is unassociated labels	<code>chart_data_point</code> , by direct call
<code>chart_caption</code>	caption bound to its chart	
■ INLINE FORMATTING AND FIGURE CONTENT		panels <code>formatting</code> , <code>imgalt</code> , <code>furniture</code> , <code>diagram</code>
<code>formatting</code>	inline formatting marks	the formatting tests
<code>latex</code>	fraction of the reference’s formulas reproduced, scored with the benchmark’s own formula extractor	<code>is_latex</code>
<code>imgalt</code>	image description content, graded	feature-bench alt axis
<code>furniture</code>	running heads, footers and page numbers lifted into their bands	
<code>diagram</code>	flowchart rendered as a fenced diagram block	

Table 10 Reward components that score what the page says: the transcription itself, its tables and charts, and its inline formatting. Panel names are given with each group.

Component	What it scores	Mirrors
■ REGIONS AND BOXES		panels <code>layout</code> , <code>coverage</code> , <code>binding</code> , <code>emission</code> , <code>bbox</code>
<code>layout</code>	F1 of expected against emitted region identifiers	layout element rule
<code>coverage</code>	recall of reference regions the output attributed, counting a region as attributed when its marker carries non-empty text; separates attributing a region from transcribing it well	layout element rule
<code>binding</code>	of the regions attributed, the fraction whose text is routed to the right identifier, by requiring that its bound text best-match that region's reference over all regions; targets right content bound to the wrong region	layout attribution
<code>grounding_bind</code>	mean soft-distance credit over emitted point markers	
<code>emission_penalty</code>	fraction of emitted region markers that are waste: an empty body on a text-bearing region, a repeated identifier, or one absent from the region table	
<code>bbox</code>	graded IoU F1 of emitted against reference boxes, gated on the label	box F1 at IoU 0.5
<code>bbox_leak_penalty</code>	layout boxes emitted on a page whose prompt did not ask for them, the failure mode the two-prompt scheme of the SFT stage has to avoid	
■ OUTPUT GUARDS		panels <code>penalties</code> , <code>structure</code>
<code>order</code>	reading-order fidelity, longest increasing subsequence over reference-ordered anchors	reading-order rule
<code>format_penalty</code>	on a table or chart page, requires an HTML table that parses to at least one real cell; a missing table, prose, a markdown table and an empty envelope fail equally	the detection gate
<code>length_penalty</code>	output running long against the reference past a 1.3× dead band	
<code>rep_penalty</code>	repeated n-gram fraction, against loops and runaway generation	
<code>marker_penalty</code>	marker hygiene	
<code>style_penalty</code>	one-sided penalty when styling exceeds the page's verified mark budget; the formatting tests are recall-only, so indiscriminate styling is otherwise profitable	
<code>double_emit_penalty</code>	the same content emitted twice	
<code>table_finish_penalty</code>	a page whose reference has a real table and whose prediction leaves a table unbalanced or far short of its rows; both legs are required, since the unbalanced leg alone makes omitting a table cheaper than truncating one	
<code>struct_penalty</code>	a degenerate-output guard, a flat penalty when the output runs past 20,000 characters at over 0.8 repetition or leaves runaway unclosed table tags; despite the name it is unrelated to table structure	

Table 11 Reward components that score where content is placed and that guard against degenerate output. Panel names are given with each group.

B.4.1 How the components are combined

The thirty-four components are grouped into sixteen panels, and the panels into nine targets. The nine targets are a benchmark pass rate, content fidelity, table content, table structure, chart value-label keying, layout and attribution, layout boxes, formatting marks, and output hygiene. Each component k is either a score s_k or a penalty p_k , both in $[0, 1]$, and it contributes to the reward with its weight w_k :

$$c_k(y) = \begin{cases} w_k s_k(y) & \text{if } k \text{ is a score,} \\ -w_k p_k(y) & \text{if } k \text{ is a penalty,} \end{cases} \quad (1)$$

where a penalty is zero when the output does not have the error that it checks for. The components of a panel P_j are added, and the panels are combined with weights ω_j normalised to sum to one:

$$R(y) = \sum_{j=1}^{16} \frac{\omega_j}{\sum_l \omega_l} \sum_{k \in P_j} g_k(y) c_k(y). \quad (2)$$

A length gate multiplies the seven content scores, such as `cct` and the table scores, and removes or reduces their credit when the output is much longer or much shorter than the reference or repeats itself. The `length_penalty` also lowers the reward directly when the output is too long. For short outputs we use only the gate, because a penalty for short outputs can give an empty output a higher reward than an output with half of the page.

In Equation (2) the gate is the factor g_k . Let n and n^* be the lengths in characters of the output and of the reference. For `bench`, `cct`, `table_align`, `table_teds`, `table_grits`, `table_grits_trm` and `chart_keying`, g_k is the product $g_{\text{over}} g_{\text{under}} g_{\text{rep}}$, and for every other component it is one. The first factor removes the credit of an output that is much longer than the reference, and the second reduces the credit of an output that is much shorter:

$$g_{\text{over}} = \begin{cases} 1 & \rho \leq 1.5, \\ (4 - \rho)/2.5 & 1.5 < \rho < 4, \\ 0 & \rho \geq 4, \end{cases} \quad \rho = \frac{n}{n^* + 250/1.5}, \quad (3)$$

$$g_{\text{under}} = \begin{cases} 1 & \rho \geq 0.6, \\ 0.3 + 0.7(\rho - 0.2)/0.4 & 0.2 < \rho < 0.6, \\ 0.3 & \rho \leq 0.2, \end{cases} \quad \rho = \frac{n + 250}{n^*}. \quad (4)$$

The third factor is $g_{\text{rep}} = 1 - f_{\text{rep}}$, where f_{rep} is the fraction of the raw output that is a runaway repetition. The 250 characters added in both ratios prevent a short page from being gated for a small absolute difference in length. The under-generation gate has a floor of 0.3. The gates, like the weights, are set per run.

The `length_penalty` component is a penalty for the same error as g_{over} . It is zero while the output is within its allowed length, and then it increases linearly to one:

$$p_{\text{len}} = \min\left(1, \max\left(0, \frac{n - \max(1.3n^*, n^* + 250)}{\max(n^*, 250)}\right)\right). \quad (5)$$

B.4.2 Tests for reward terms

Every reward term is tested on constructed answers before training.

The tests led to three rules for writing a reward term. A component with nothing to compare must return zero, because returning a fixed middle score instead gave an empty answer a quarter of the summed score. A check on order must require content, because an empty sequence is trivially in order and would otherwise score well on reading order. A check on counts or labels must require correct placement, because otherwise an answer with every element in the wrong place still scores well.

B.4.3 How the reward compares

The document-parsing VLMs that add a reinforcement-learning stage use similar algorithms and differ mainly in what their rewards score. Table 12 places ours beside them, using each system’s own account of its reward. Beyond covering more of the page, our reward differs in how finely it scores and in its penalties.

An edit-distance reward is smooth, but it is computed over the whole page and cannot say which cell of a table is wrong. A binary unit test says exactly what it checks and gives nothing in between, so a nearly correct table and an absent one are worth the same. Infinity-Parser2 grades cropped tables with tree edit distance, but it scores a full page with an edit distance. Our terms are graded where a partial parse deserves partial credit, and binary where a page either carries a parseable table or does not.

Twelve of our thirty-four components are penalties. A reward built only from scores costs an output nothing for repeating itself, running past the end of the page or carrying formatting the page does not have. Two of the published rewards guard against repetition. olmOCR 2 rewards an output that ends with the end-of-sequence token, and LightOnOCR penalises repetition loops. None of them describes penalties for split or merged tables, unfinished tables or formatting the page does not have. Our reward needs the formatting penalty because the formatting tests measure recall alone, so without it, styling every line raises the score.

System	What the reward scores	Granularity	Penalties
■ PUBLISHED MODELS		as their authors describe them	
Infinity-Parser [41]	edit distance, paragraph count, reading order	whole page	none described
olmOCR 2 [30]	binary unit tests, on synthetic pages whose source markup gives the answer; an output that ends with the end-of-sequence token; document metadata	per test, binary	none described; the end-of-sequence reward guards against outputs that never stop
LightOnOCR [40]	the olmOCR 2 unit tests and formulas that render; in a separate variant, the overlap of the emitted image boxes with the true boxes	per test; per box	repetition loops, malformed formulas, missing and invented boxes
Infinity-Parser2 [13]	eight tasks jointly, each with its own metric: tree edit distance for cropped tables, value-and-label matching for charts converted to tables, edit distance for full pages	per task	none described
■ THIS WORK			
Ours	nine targets, from the transcription itself to layout attribution and output hygiene	per component, graded and binary	twelve of thirty-four components

Table 12 The reward of each reinforcement-trained document-parsing VLM, as its authors describe it. “None described” records that the published account does not mention penalties against degenerate output, not that the system has none.

B.4.4 Combining the terms

Variance reweighting. OCRia-1 is trained with hand-tuned panel weights, with variance reweighting as a safeguard. Let σ_j be the standard deviation of panel j within a group, averaged over the groups of a step. We keep a running estimate S_j and, after 15 warm-up steps, scale the panel weight by it:

$$S_j \leftarrow 0.9 S_j + 0.1 \sigma_j, \quad \omega_j \leftarrow \omega_j \cdot \min(1, \max(0.3, S_j/0.05)). \quad (6)$$

A panel whose rollouts no longer differ loses up to 70% of its weight. A panel can be exempted when its low variance means that the model is stuck on it and has not mastered it.

Balancing the terms. In a group-relative update, the influence of a term depends on how much it varies within the group as well as on its weight. A term with a small weight and a large variance can therefore dominate the gradient, while a term that barely varies, such as the content term, rarely decides the ranking. In one training run, the formatting term became the largest single contribution to the ranking within ten steps, the contribution of the table term fell by half, and the table scores regressed. Normalising the panel weights to sum to one also lowers the weight of every other panel when one target is split into two panels, and giving the formula term its own panel cut its effective weight by a factor of four. The published remedy is to normalise each component by its own within-group statistics before summing [14], but it did not work for our reward. Several of our components are one-sided guardrails, penalties that are zero when nothing is wrong, and normalising them to unit variance gives them as much influence as the terms they protect; on stored groups, it moved a share of the gradient onto them. In a training run with this normalisation, the

policy removed formatting markup to avoid the formatting penalty, the formatting score collapsed and the overall score fell with it. A revised normalisation leaves the guardrails out of the normalised set, puts a floor on the within-group standard deviation, clips the normalised score and blends the result with the ordinary advantage of the weighted reward. On the same stored groups, it removes the guardrails’ share of the gradient and raises the share of the chart and table terms, and Constrained GRPO [11] independently argues for combining standardised advantages over standardising a scalarised reward. We have tested it only offline, and OCRia-1 is not trained with it. Reweighting moves gradient between objectives but cannot add signal, so a chart page on which every rollout fails in the same way gives no signal with any weights. Because the gradients of the objectives are correlated, projecting one gradient out of another [47] also changes the others.

B.5 Pages with no training signal

Published remedies for pages whose rollouts agree are to filter batches by disagreement [46], to select prompts by their effect on learning [22], and to schedule difficulty as the model improves [6, 38]. We report results for two of them.

Filtering during training. We filtered batches by disagreement on one objective. This increased that objective’s share of the gradient but reduced the signal for the other objectives. The filter reads the current policy’s own disagreement, which changes as the policy changes.

Selecting the pages before training. This test removes, before training starts, the pages that the SFT model is unlikely to learn from. We sampled each candidate page eight times with the SFT model and set aside the 6,096 pages whose samples agreed. The SFT model had already transcribed 4,713 of them correctly and was uniformly poor on 1,215. On the remaining 168, the agreement came from a suspect reference. A page that the SFT model already transcribes is unlikely to divide the policy trained from it, so these pages are the least likely to carry signal later. Training on the remaining pages stayed ahead of the full corpus at both steps we measured, by a margin that narrowed, and the two runs improved on different dimensions. OCRia-1 is trained on the full corpus, because removing these pages gave mixed results (Section 3), and the corpus still contains 3,473 of them.

C Encoder experiments

This appendix studies where small text is lost between the page image and the transcription, and how the number of image tokens per page changes the scores.

The encoder of Qwen cuts a page image into 16×16 px patches, and the projector joins each 2×2 group of patches into one image token, so one image token covers 32×32 px of the page. At 150 dpi that is three to four characters of 10 pt text and about twenty characters of 4 pt text. We measured each stage between the page and the transcription separately, all on the Qwen3.5-4B model. A probe is a small classifier trained to name the character at a position from the features at that position. Its accuracy says how readable the character is at that stage, not whether the information is gone.

Small text is weakened before the decoder sees it, mostly by the resolution the encoder receives and further by the projector. Table structure crosses the projector almost unchanged, and the decoder reads what it is given. Fine-tuning changed the vision side little: character-identity probes before and after the projector are within 0.03 of the base model’s. Inside the decoder it changed more, with probes at layer 16 up to 0.06 above the base model’s.

More image tokens help and help fine text most. Table 15 serves olmOCR-bench to the same weights with different numbers of image tokens. Going from 1,240 to 2,268 tokens per page by enlarging the same 1,288 px render takes long tiny text from 92.3% to 93.9% and the overall score from 83.5% to 84.4%. Those extra tokens carry no new detail, and rendering the page at 1,822 px instead, for 2,451 tokens, adds only 0.3 points more on long tiny text, though 1.5 points more overall. For fine text, what limits the model is how many tokens each glyph gets, not the detail in the pixels. The model was trained on pages rendered at 1,024 px,

Stage	Measurement	Result
Rendering	image tokens per table cell, and type height	63 tokens per cell and 20 px type on pages up to 100 cells; 1.1 tokens per cell and 8 px above 1,000 cells
Vision encoder	probe accuracy for character identity as the page gets fewer image tokens	0.82 at 3,796 tokens, 0.40 at 1,024, 0.22 at 400, 0.11 at 144 (chance floor about 0.09)
Projector	probe accuracy before and after the projector	digits 0.84 $\rightarrow$ 0.76, letters 0.56 $\rightarrow$ 0.41; gutters, column boundaries and merged cells drop by at most 0.02
Decoder	activation patching on degraded pages	clean image tokens at the decoder input restore all of the lost quality; at layer 16 they restore 50% (95% CI 39–63%)

Table 13 Where small text is lost, stage by stage, on the Qwen3.5-4B model.

Cells	Pages	Table score	Image tokens per cell	Type height (px)	Reference length (tokens)	Abnormal end
1–100	259	0.96	63.0	19.6	596	0
101–300	142	0.92	12.6	18.6	2,368	0
301–600	61	0.82	5.0	14.6	5,111	1
601–1,000	28	0.88	3.2	13.2	9,196	0
Over 1,000	13	0.59	1.1	8.0	24,139	6

Table 14 ParseBench table pages by size (4B model, 150 dpi, output limit 24,576 tokens). The table score is the mean over pages; the other columns are medians.

about 800 image tokens, so every standard evaluation rendering here gives it more tokens per glyph than it saw in training.

Image tokens	Page image	olmOCR-bench	Long tiny text
266	1,288 px render (shrunk)	45.7	7.9
532	1,288 px render (shrunk)	67.2	36.9
1,240	1,288 px render	83.5	92.3
2,268	1,288 px render (enlarged)	84.4	93.9
2,451	1,822 px render	85.9	94.2

Table 15 olmOCR-bench for the 4B model as the image-token budget changes. Image tokens are the median per page.

Retraining the projector alone did not help. On the base model, fine-tuning only the projector raised probe accuracy by 0.08 and dropped olmOCR-bench from 71.0 to 48.0, so we did not adopt it.

Scope. Every benchmark number is a single run with greedy decoding. Every measurement is on the 4B model except the projector retraining, which was run on the base model.

D Evaluation details

This appendix gives the protocol of the evaluation in [Section 5](#), the results for each benchmark ([Appendix D.1](#)) and the cases where the scores are lowest ([Appendix D.2](#)).

Transcription benchmarks. ParseBench [48] grades a page on text content, text formatting, tables and charts. Text content, text formatting and charts are scored as the pass rate of rules written against the

page, and tables with the combination of GriTS and TableRecordMatch (GTRM); we report the mean of the four. Pages are rendered at 300 dpi, and the model’s image processor scales a letter page down to about 4.1 megapixels, about 210 dpi. olmOCR-bench [29] holds 1,403 pages and 7,010 unit tests (a string present or absent, two strings in order, a cell under its header, an equation rendering to the same symbols), rendered at 1,288 px on the long side and scored as the mean over its seven categories and the baseline test. PulseBench-Tab [31] holds 1,820 cropped tables in nine languages scored by the T-LAG grid metric. We send the crop capped at 2,560 px with a 16k-token budget, score the largest HTML table in the answer, and report the share of tables scored, since the protocol leaves tables without a parseable prediction out of the mean. fr-bench-pdf2md [35] holds 721 difficult French pages with 2,318 olmOCR-style tests, scored as the pass rate over all of them. We render its pages at 2,048 px on the long side.

Post-processing. The model is trained on our output format (Appendix A.1), which includes constructs that these benchmarks neither expect nor score, such as page headers and footers inside their own tags, a watermark tag, an image marker with a label, a Mermaid block for a diagram, and a horizontal rule for a printed one. Before scoring we remove these constructs from every candidate and normalize the markup inside table cells (emphasis, checkboxes, and superscripts and subscripts written back as the glyphs the page prints).

Rendering resolution. We serve each benchmark at its own standard rendering: olmOCR-bench pages at 1,288 px on the long side (about 1,240 image tokens for a letter page), and ParseBench pages at 300 dpi, which the image processor scales down to about 3,900 image tokens for a letter page. Both are above the 1,024 px, about 800 image tokens, that the model is trained on. Scores depend on this choice. With no change to the weights, giving the OCRia-1 model 2,451 image tokens per page instead of 1,240 raises its olmOCR-bench score by 2.4 points (Appendix C). Other models respond the same way. Chandra-OCR-2 and LightOnOCR-3 improve by 2 points on olmOCR-bench when their pages are rendered at 212 dpi.

A difference of 2--3 points between two systems can come from rendering alone, so we use each benchmark’s standard rendering for the systems we evaluate ourselves. On olmOCR-bench the other systems scored much lower at our rendering, so we report their published numbers and run OCRia-1 at the standard 1,288 px. Table 17 combines our runs at 1,288 px with published scores for Mistral OCR 4.1 and olmOCR-2.

Other systems. On ParseBench, PulseBench-Tab and fr-bench-pdf2md every other system is run by us at the same renderings and scored by the same scorers. All open models (Infinity-Parser2-Pro, LightOnOCR-3-4B, Chandra-OCR-2, dots.mocr, LightOnOCR-2-1B) are served with vLLM at each model’s own image budget. LightOnOCR-3-4B runs in its grounding mode, which scores higher than its plain transcription mode on tables and charts (0.75 against 0.61 on ParseBench). olmOCR-2 runs through its own pipeline, which renders the pages itself at the size we pass. Surya OCR 2 runs through its own package, on vLLM. Where a system labels page headers, footers and pictures, those blocks are dropped the way our own tags are, and pipe tables are converted to HTML where a scorer wants HTML.

D.1 Results by benchmark

Tables 16, 18 and 19 break down the results of Table 3 for olmOCR-bench, PulseBench-Tab and fr-bench-pdf2md.

Table 3 reports the mean of the Tables, arXiv math and Old scans math categories, the parts of olmOCR-bench that test tables and formulas. We report this part because tables and formulas are the elements of the page that our training data targets (Section 2), while olmOCR-bench has no test of the other two elements we focus on, charts and text formatting. Table 16 gives all eight categories and the official score, with the numbers that the other systems’ authors publish. Table 17 reports our runs alongside two published baselines. For our runs, pages are rendered at 1,288 px on the long side, the benchmark’s standard rendering, and models are served with vLLM at their own image budgets. We apply the same post-processing and the olmOCR-bench scorer to all outputs from these runs. Mistral OCR 4.1’s scores come from the LightOnOCR-3 post, and olmOCR-2’s from its model card, which reports the same 1,288 px rendering. In this comparison OCRia-1

Model	arXiv math	Old scans math	Tables	Old scans	Headers, footers	Multi-column	Long tiny text	Base	Overall
Infinity-Parser2-Pro	88.1	91.3	91.2	58.2	95.8	83.7	92.5	99.9	87.6
LightOnOCR-3-4B	91.1	89.1	90.2	52.5	89.2	85.6	93.2	99.5	86.3
olmOCR-2 (7B)	82.9	82.1	84.3	48.3	95.7	84.3	81.4	99.7	82.3
Chandra-OCR-2 (4B)	86.9	89.1	92.1	51.1	91.4	82.1	93.7	99.9	85.8
dots.mocr (3B)	85.9	85.5	90.7	48.2	94.0	85.3	81.6	99.7	83.9
Surya OCR 2 (0.7B)	88.3	81.4	86.6	41.8	92.5	82.4	93.7	99.7	83.3
LightOnOCR-2-1B	89.6	85.6	89.0	42.2	19.7	84.8	91.4	99.6	75.2
Mistral OCR 4.1	82.6	72.3	85.9	47.9	91.4	84.5	91.2	99.5	81.9
OCRia-1	89.6	89.1	92.9	48.9	81.4	81.6	93.7	99.9	84.6

Table 16 olmOCR-bench by category. Overall is the official average over the eight columns. The other systems’ rows are the numbers they publish (the LightOnOCR-3 post; olmOCR-2’s model card); the OCRia-1 row is our run. LightOnOCR-2-1B’s paper reports 83.2 with the headers-and-footers category left out; here its overall is recomputed over all eight categories, using the headers-and-footers score (19.7) from the same paper.

Model	arXiv math	Old scans math	Tables	Old scans	Headers, footers	Multi-column	Long tiny text	Base	Overall
Infinity-Parser2-Pro	82.9	85.2	87.7	51.0	92.1	82.8	84.4	99.9	83.2
LightOnOCR-3-4B	86.9	78.8	87.6	48.7	87.6	84.3	83.5	99.7	82.1
olmOCR-2 (7B)	82.9	82.1	84.3	48.3	95.7	84.3	81.4	99.7	82.3
Chandra-OCR-2 (4B)	80.3	84.3	86.7	45.8	90.8	80.7	81.2	99.9	81.2
dots.mocr (3B)	77.9	78.4	86.3	40.9	90.0	82.2	85.3	99.6	80.1
Surya OCR 2 (0.7B)	78.8	76.0	82.1	38.2	93.2	79.9	73.3	99.7	77.6
LightOnOCR-2-1B	88.1	87.3	85.7	40.3	22.4	83.8	80.3	99.5	73.4
Mistral OCR 4.1	82.6	72.3	85.9	47.9	91.4	84.5	91.2	99.5	81.9
OCRia-1	85.0	87.8	92.9	46.6	81.8	81.4	92.3	99.8	83.5

Table 17 olmOCR-bench: our runs at 1,288 px alongside published baselines. For our runs, pages are rendered at 1,288 px on the long side, models are served with vLLM at their own image budgets, and outputs receive the same post-processing and are scored by the olmOCR-bench scorer. Mistral OCR 4.1’s row is from the LightOnOCR-3 post and olmOCR-2’s from its model card at the same 1,288 px. Overall is the official average over the eight columns. Of the 440 arXiv-math tests that OCRia-1 fails, 143 render to the same formula as the reference but write the $\LaTeX$ in a form other than the KaTeX one the test expects; counting them as passed gives 89.9 on that category.

has the highest official score, 83.5 against 83.2 for Infinity-Parser2-Pro, and the highest mean on the three table and formula categories, 88.6 against 87.0 for LightOnOCR-2-1B. The six systems we served ourselves score 1.8 to 5.7 points lower here than in their published numbers, which is why the main table uses the published ones.

D.2 Areas for improvement

The scores are lowest on the hardest cases within the areas we worked on. Old scans are the weakest olmOCR-bench category, 48.9% for OCRia-1 against 81.4 to 99.9% for the other seven categories. The other systems in Table 16 score 41.8 to 58.2% on old scans, so these pages remain hard for current document-parsing VLMs. Tables in Arabic, Greek and Korean score 0.70 to 0.77 on PulseBench-Tab, against 0.86 to 0.90 for French, Chinese, Spanish, Russian and English. Text formatting is the lowest of the four ParseBench dimensions for OCRia-1, although its 0.83 is the highest score on that dimension in Table 4. Within it, bold passes 77% of its rules and italics 63%. Highlighting passes 35% and strikethrough 45%, and both are rare on real pages.

Figures 11 and 12 show two pages at the limits of the current model.

Model	Mean	Coverage %	Median	Perfect tables
Infinity-Parser2-Pro	0.82	96.8	0.93	419 (23.0%)
LightOnOCR-3-4B	0.82	96.6	0.93	431 (23.7%)
olmOCR-2 (7B)	0.80	92.4	0.90	354 (19.5%)
Chandra-OCR-2 (4B)	0.80	93.0	0.92	340 (18.7%)
dots.mocr (3B)	0.82	88.7	0.92	308 (16.9%)
Surya OCR 2 (0.7B)	0.71	93.7	0.82	224 (12.3%)
LightOnOCR-2-1B	0.81	89.6	0.90	354 (19.5%)
Mistral OCR 4.1	0.75	97.7	0.82	269 (14.8%)
OCRIA-1	0.84	99.1	0.95	485 (26.6%)

Table 18 Results on PulseBench-Tab, 1,820 tables. Mean and median T-LAG score over the tables scored; coverage is the share of tables with a parseable prediction.

Model	fr-bench-pdf2md, pass rate by category							
	Forms	Graphics	Handwritten	Long table	Long table 2	Multi-column	Tiny text	Overall
Infinity-Parser2-Pro	33.9	26.9	33.3	51.5	50.0	76.4	78.7	54.9
LightOnOCR-3-4B	49.6	68.3	41.2	80.6	70.7	83.6	83.2	71.4
olmOCR-2 (7B)	43.8	37.7	20.6	67.9	63.9	84.8	66.2	56.9
Chandra-OCR-2 (4B)	38.3	72.8	22.4	80.6	77.8	70.9	70.0	65.5
dots.mocr (3B)	40.1	26.1	13.9	56.7	59.8	80.6	79.3	56.8
Surya OCR 2 (0.7B)	22.6	26.6	1.8	74.6	66.1	73.3	64.5	50.9
LightOnOCR-2-1B	35.4	32.9	12.1	73.9	61.3	78.8	64.7	53.5
Mistral OCR 4.1	39.4	55.8	11.5	69.4	59.8	73.3	57.4	54.1
OCRIA-1	46.0	78.5	21.2	83.2	81.9	77.0	78.9	72.1

Table 19 Results on fr-bench-pdf2md, pass rate by category and overall, over 2,318 tests.

E Serving stack

This appendix describes the serving stack of [Section 4](#), how its speed is measured ([Appendix E.1](#)) and the draft model ([Appendix E.2](#)).

The served model is the bf16 checkpoint quantized to a mixed NVFP4 and FP8 map [1], calibrated on the model’s own transcriptions and chosen by a per-layer search that kept only maps within the noise of the bf16 model on ParseBench (-0.0039 , 95 % interval $[-0.0083, +0.0006]$ on 1,578 documents), with the vision encoder left in bf16. It runs on vLLM [18] with the draft of [Section 4](#) and five changes:

- a port of ReplaySSM [16], which keeps one checkpoint of each gated-delta-network state and replays the accepted tokens instead of snapshotting the state at every draft position, with its verify kernel rewritten in CUDA;
- fused activation-quantization kernels;
- the batch metadata built once per step instead of once per cache group;
- the prompt prefix shared by every request, captured once and injected per request;
- a draft length that switches from 31 tokens to 15 above thirteen concurrent requests, where the verify step becomes compute-bound.

Every change was kept only if its kernels produced the same bits as the ones they replaced or its ParseBench

score on 749 documents stayed within the noise of the previous served model.

E.1 Speed measurements

Concurrency sweep. Requests come from a client in the same region that keeps a fixed number of them in flight; the model decodes greedily under the production prompt on pages rendered at 4.1 megapixels (about 3,900 image tokens), and per-request decoding speed is a request’s output tokens divided by its own decoding time, as reported by AIPerf, the load generator we use. With a single request, the model decodes at 4,734 tokens per second, and the median request of about 1,300 output tokens takes 0.41 s from request to last token, time to first token included, about nine times the end-to-end speed of the bf16 checkpoint on stock vLLM (321 to 333 tokens per second on one page). With eight concurrent requests, the model decodes at 1,932 tokens per second per request on production pages; most of the difference from the single-request figure is the prefill for the other requests, which runs on the same GPU and stalls the decoding steps it overlaps. With 64 concurrent requests, the GPU parses 10.21 pages of production traffic per second with a p90 latency of 9.6 s (11.78 with every page rescaled to 1,200 output tokens), and Table 20 gives the levels in between. Figure 5 shows the single-stream speed.

Concurrent requests	Output tok/s per request	Aggregate output tok/s	Pages/s
1	4,734	–	2.16
8	1,932	9,964	7.42
16	1,103	12,740	8.67
32	566	14,833	9.77
48	400	15,240	9.98
64	302	15,557	10.21

Table 20 Speed of OCRia-1 on one B200 with the final serving stack of Appendix E, against the number of concurrent requests. Measured with AIPerf, each value is the mean of three runs. Aggregate output tokens per second is all output tokens of a run over the run’s wall-clock time. Appendix E.1 gives the protocol.

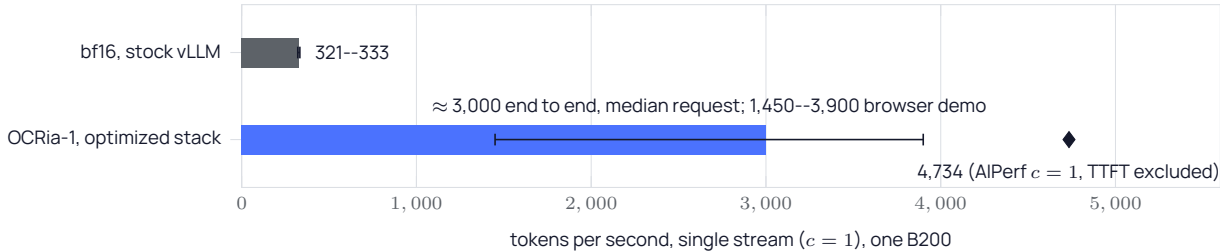


Figure 5 Single-stream decoding speed on one B200. The same model in bf16 on stock vLLM against the optimized stack (NVFP4 target, DFlash2 draft with 31 draft tokens, the kernels of Appendix E), greedy decoding, the production prompt, one page at a time. Bars are the end-to-end rate, time to first token included, of the median request at a client in the same region; the whisker is the range seen from a browser client over runs; the diamond is the AIPerf per-request decode rate on the concurrency ladder, which excludes the time to first token.

Cross-model comparison. We compare OCRia-1 with Nemotron-3.5-Parse, HunyuanOCR-1.5 with DFlash, SmolDocling-256M and Falcon-OCR in BF16, each on one B200 at one and eight concurrent requests. Table 21 gives the values plotted in Figure 4.

Model	Concurrent requests	Output tok/s per request	Pages/s
OCRia-1 (optimized)	1	4,734	2.17
Nemotron-3.5-Parse	1	1,395	0.80
HunyuanOCR-1.5 + DFlash	1	1,403	0.71
SmolDocling-256M	1	894	0.55
Falcon-OCR bf16	1	599	0.39
OCRia-1 (optimized)	8	1,932	7.83
Nemotron-3.5-Parse	8	963	4.75
HunyuanOCR-1.5 + DFlash	8	787	3.26
SmolDocling-256M	8	732	3.21
Falcon-OCR bf16	8	356	1.95

Table 21 Cross-model speed comparison on one B200 with one and eight concurrent requests. [Table 20](#) reports the separate concurrency sweep.

E.2 Speculator

This appendix gives the details of the draft model of [Section 4](#). Like other recent drafts [\[3, 12, 23\]](#) it reads the hidden states the target computed for the context rather than the context’s tokens: DFlash concatenates them from eight of the target’s layers, projects them and injects them as precomputed keys and values into every draft layer, and the draft writes a whole block of tokens in one pass, with mask tokens standing in for the block positions. The convolution lets each block position see the ones before it, and the selector keeps the sixteen most likely tokens at each position, scores every pair of consecutive candidates with a low-rank bilinear term, and walks one path through the block.

Training details. The training features are the target’s hidden states at the eight layers, captured once by running the quantized target of the time over training pages with greedy decoding and the image-only transcription prompt; the target stays frozen, and the served draft was trained against the checkpoint that preceded the released one. Training has two phases and then the selector. First only the projection, its normalisation and the convolutions are trained, 72 M of the 686 M parameters, with the cross-entropy of the target’s greedy tokens weighted by $\exp(-\text{position}/4)$, because the early positions decide whether a step accepts anything at all; on 16,384 pages this takes about ten hours on one B200 and raises the accepted length on held-out replay from 8.2 to 13.0 tokens per block. Then the whole draft is unfrozen and trained on 15,481 pages (18,142 steps on eight B200s, about four hours), warm-starting through 2k- and 8k-page stages; at equal data the unfrozen draft beats the frozen one (10.15 against 9.93 accepted tokens). The selector is fitted last on the frozen draft from its cached features, with a prefix cross-entropy over the sixteen candidates at 32 anchor positions per page, so the quantity optimized is the length of the prefix the target will accept in one verification step; it raises the held-out accepted length from 12.65 to 14.87 tokens, and more pages helped at every stage (10.2, 11.2 and 12.7 accepted tokens from 1k, 4k and 16k pages).

Path selection. We refine DFlash 2’s draft-token selection with dynamic programming. DFlash 2 greedily selects among sixteen candidates per position using scores conditioned on the previous choice. We normalise these scores into transition probabilities $P_i(c_i | c_{i-1})$, the probability that the target accepts candidate c_i at position i after c_{i-1} , and select the chain $c_1, \dots, c_K$ that maximises the expected accepted-prefix length:

$$\mathbb{E}[\text{accepted length}] = \sum_{l=1}^K \prod_{i \leq l} P_i(c_i | c_{i-1}), \quad (7)$$

where c_0 is the last verified token. Solving this objective exactly adds negligible cost and gives slightly more accepted tokens per step than the default greedy selection ([Figure 6](#)).

Setting	k	Tokens per step	Accepted per step	Condition
■ OCRIA-1, DFLASH 2 DRAFT, GREEDY		tokens emitted per verification step; accepted drafts of 31		
Tuning pages, c = 8	31	17.5	16.5	timed inside the server, B200
Production pages, 40-page sample, c = 1	31	12.5	11.5	timed inside the server, B200
Production traffic, 512 pages, c = 8	31	13.3	–	per-request mean, AIPerf ladder of Table 20
Fixed canary pages at container start, c = 1	31	14.2	–	B200
The same canary pages on a B300	31	12.9–13.2	–	B300 arithmetic differs from B200; 14.2 on B200
■ PUBLISHED SPECULATORS		authors’ figures, each with its own definition of accepted length		
DFlash, Qwen3-8B, block 16 [5]	16	6.5	–	text benchmarks, temperature 0
DFlash, Qwen3-4B, block 16 [5]	16	6.5	–	text benchmarks, temperature 0
DFlash 2, Qwen3.8-27B, block 8 [15]	8	4.1–5.5	–	vendor blog
HunyuanOCR-1.5 with DFlash, block 16 [20]	16	8.4	–	OmniDocBench, vLLM, batch 1 (8.9 in Transformers)
GLM-OCR, MTP, 10 tokens per step [10]	10	5.2	–	authors’ average
HPD-Parsing, P-MTP [44]	–	6.6	–	A800, vLLM
GravityOCR, diffusion self-speculation [17]	–	9.7	–	tokens per forward pass, drafting and verification counted as separate passes, SGLang

Table 22 Accepted tokens per verification step. Top group: the served DFlash 2 draft (6 layers, block 32, selector top-16) on the OCRIA-1 NVFP4 target at k=31, temperature 0; a step emits the accepted drafts plus one target token, so 17.5 tokens per step means 16.5 of 31 drafts accepted. Tuning pages are the eight pages the kernel work was tuned on; production pages are a 40-page production sample and the 512-page AIPerf ladder of [Table 20](#); the canary is a fixed page set run at every container start. Bottom group: published figures, each with its authors’ definition of accepted length.

Acceptance. Served with 31 draft tokens per step, each verification step emits 13 to 17 tokens, that is 12 to 16 accepted drafts plus the target’s own token, depending on the pages: 17.5 on the eight pages the kernel work was tuned on, 14.2 on the fixed set of pages run at every server start, and 13.3 on production traffic ([Table 22](#)); with 15 draft tokens the same draft emits 12.2 on the tuning pages. Published drafts with a 16-token block accept fewer: DFlash reports about 6.5 tokens per step on Qwen3-8B, the draft shipped with HunyuanOCR-1.5 about 8 to 9, and multi-token prediction heads about 5 [\[5, 10, 20\]](#). Acceptance does not depend on the batch size, so the draft weights are the same at every batch size from 1 to 64; only the number of draft tokens changes, from 31 to 15 above thirteen concurrent requests ([Appendix E](#)). [Figure 6](#) gives the acceptance by draft position: the probability that a draft token is right, given that the ones before it were, stays above 0.8 to the last position, so a block of 32 is not too long.

F Prompts

This is the system prompt for transcription.

■ PLAIN TRANSCRIPTION PROMPT

```
Transcribe this Page in markdown, be as accurate as possible in your transcription, do not describe/summarize.

Formatting:
- Use ** for bold text and * for italic text.
```

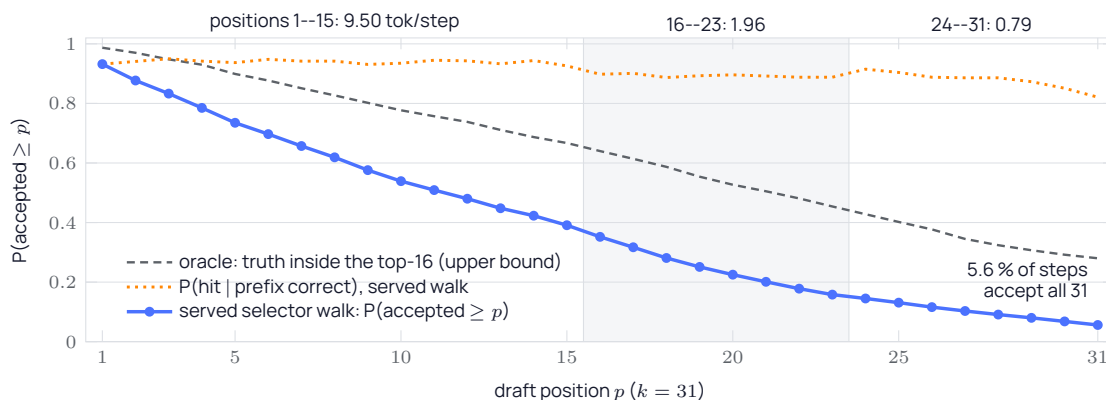


Figure 6 Acceptance by draft position. Probability that the target accepts at least the first p draft tokens, for the greedy walk over the draft’s candidates (the served dynamic-programming walk accepts slightly more) and for an oracle that picks the right token whenever it is among the draft’s sixteen candidates, with the hit rate at each position given the prefix was right; offline replay of 9,193 greedy draft calls on 100 pages with 31 draft tokens. The area under the walk curve is the accepted tokens per step.

- Use `~~ text ~~` for strikethrough text. Only use this for text that is visually stricken through.
- Use `<mark>` and `</mark>` tags for highlighted text, aka text to which a stabilo hilight was applied Only use to hilight sentence or words, not full block of text/tables. Use HTML color names for different colors of highlight, e.g. `<mark style="background-color: blue">text</mark>`. Do not use mark for first cell of a table, as it is often used for the header and should not be highlighted.
- Use `<u>` and `</u>` tags for underlined text.
- Use `\*` for bullet points and `-` for sub-bullets.
- Use `>` for blockquotes.
- Use ``` for inline code.
- Use ````` for code blocks.
- Use `[ ]` for empty checkboxes.
- Use `[x]` for checked checkboxes.
- Use `<sub>` and `</sub>` for subscript text.
- Use `<sup>` and `</sup>` for superscript text.
- Use `[...]` for empty fields to fill, if the field is already filled, represent them as field content in markdown.
- Use `$. . $.` for inline equations and `$$ . . . $$` for display equations. Do NOT use `\( . . . \)` or `\[ . . . \]` - the corpus uses dollar delimiters (43,604 inline occurrences against 35 in the parenthesis form).
- Chemical formulas are NOT LaTeX: write them as plain text with `<sub>` and `<sup>`, e.g. `K<sub>3</sub>Fe(CN)<sub>6</sub>` and `2H<sub>2</sub>O`, never `$K_3Fe(CN)_6$`. Reserve `$. . $.` for real mathematics.

Tables:

- Output tables in HTML when possible.
- Table should not be used for page formatting (column or grid like layout), but only for tabular data.
- Some table may be irregular, with merged cells, missing cells, or inconsistent formatting, or have an exotic structure (like a triangular table). Do your best to represent the table accurately with correct reading order.

Charts, Graphs, and Figures:

- IMPORTANT: Every chart or graph MUST be converted into an HTML `<table>`. This is non-negotiable. The table represent the chart data, not the chart layout. Focus on accurately capturing the underlying data and relationships in the table format.
- FORBIDDEN outputs for charts: `<iframe>`, `<embed>`, `<canvas>`, `<img>`, `<object>`, links, URLs, or plain-text descriptions. Using any of these for a chart is an automatic FAIL.
- MERMAID FALLBACK: For diagrams, figures and flowcharts that are better represented as a graph than as a table (architecture diagrams, flowcharts, org charts, entity relationships), use Mermaid syntax INSTEAD of an HTML table and output it between ````mermaid` and ````` tags. This exception applies ONLY to diagrams; a chart or graph plotting data values is still an HTML `<table>`.

Images:

- Transcribe any text printed inside the image as markdown.
- Reference the image itself as `![label: description]`, where label is exactly one of: logo, icon, photo, illustration, screenshot, map, signature, seal, code, drawing, image. Use the caption or wording the page itself prints for the description; do not invent one, and do not describe an image the page says nothing about.

Page furniture:

- Output the page header (if any) between `<page_header>` and `</page_header>` tags.
- Output the page footer (if any) between `<page_footer>` and `</page_footer>` tags.
- Output the printed page number (if any) between `<page_number>` and `</page_number>` tags, nested inside the header or footer band it sits in.

Watermarks:

- Output a watermark stamped across the page (DRAFT, OBSOLETE, CONFIDENTIAL) between <watermark> and </watermark> tags, not as body text.

Multi-column Text:

- Unroll multi-column text into a single column format.

Completeness:

- Ensure ALL content from the page is transcribed without omissions.
- As this is a transcription task, fidelity is of upmost importance. You should NEVER truncate or shorten tabular data. Any instance of table data that looks like "[...table data continues in rows n - m ...]" or omits rows or columns will result in an automatic FAIL.

Accuracy:

- Maintain text and formatting as close to the original as possible within the constraints of Markdown syntax.

Handwriting / Cursive Text:

- If the page contains handwriting or cursive text, ignore the OCR output for those sections and rely solely on the screenshot to transcribe that content.

Output result between ```markdown and ```

This appendix shows OCRa-1 on pages from ParseBench [48]. We chose the pages to show one capability each, so they are not a sample of the benchmark; Section 5 gives the scores over all pages. Each page is rendered at 150 dpi and transcribed with the prompts of Appendix F and greedy decoding. The right-hand panel of each figure renders what the model wrote, without editing: tables as the HTML the model produced, page headers and footers in grey, image markers in brackets, and formulas typeset from the model’s L^AT_EX. Where an output is much longer than the page image, the panel shows its beginning.

Page image

Two pages at the limits of the current model

The two pages below show where OCRia-1 reaches the limits of the current model. In both, the model gets the structure of the page right, and the error is in the last part of the work: the rows past the output budget, and the exact values of a chart that prints none.

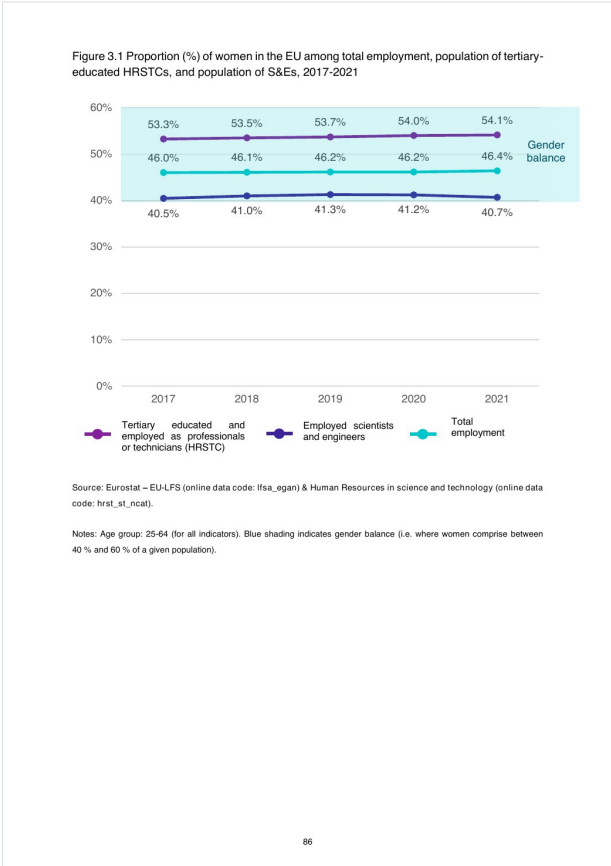


Figure 3.1 Proportion (%) of women in the EU among total employment, population of tertiary-educated HRSTCs, and population of S&Es, 2017-2021

Proportion (%) of women in the EU among total employment, population of tertiary-educated HRSTCs, and population of S&Es, 2017-2021. Blue shading indicates gender balance (i.e. where women comprise between 40 % and 60 % of a given population).

Year	Tertiary educated and employed as professionals or technicians (HRSTC)	Employed scientists and engineers	Total employment
2017	53.3	40.5	46.0
2018	53.5	41.0	46.1
2019	53.7	41.3	46.2
2020	54.0	41.2	46.2
2021	54.1	40.7	46.4

Source: Eurostat – EU-LFS (online data code: ifsa_egan) & Human Resources in science and technology (online data code: hrst_st_ncat).

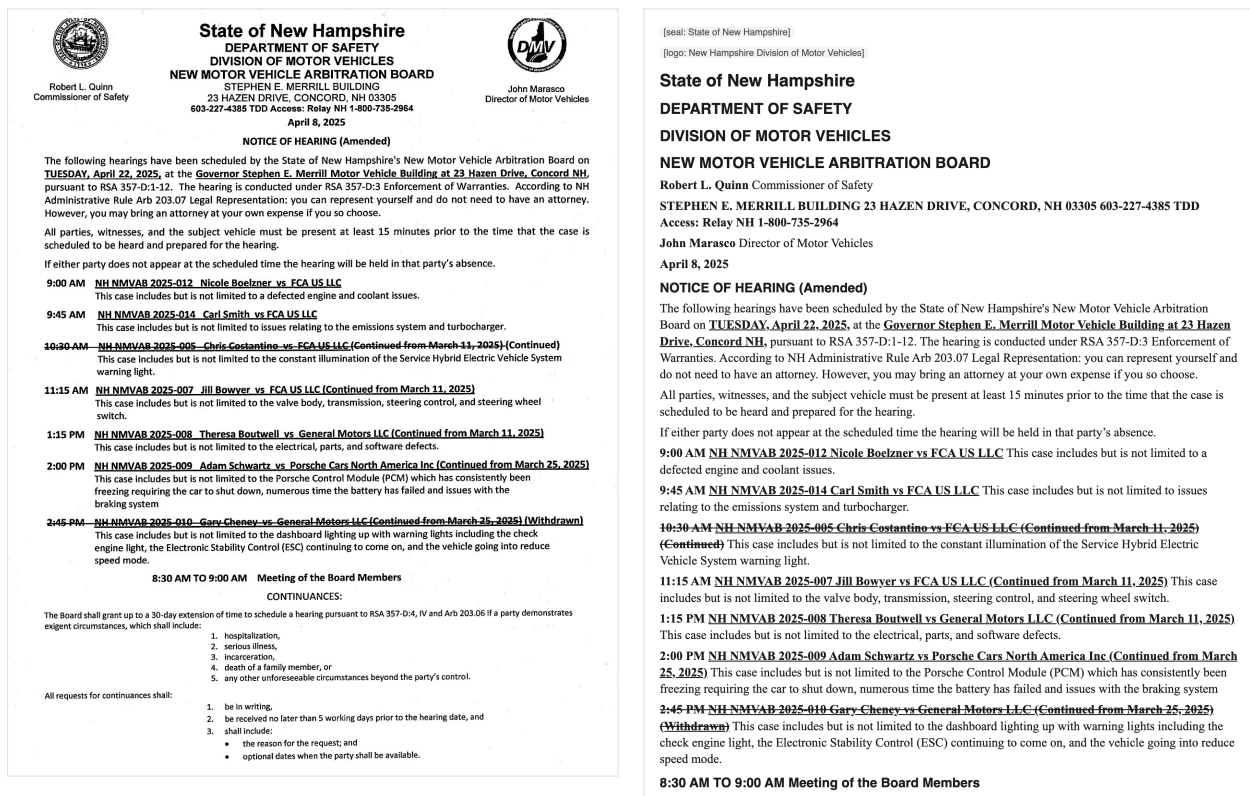
Notes: Age group: 25-64 (for all indicators). Blue shading indicates gender balance (i.e. where women comprise between 40 % and 60 % of a given population).

footer 86

OCRia-1 output, rendered

Page image

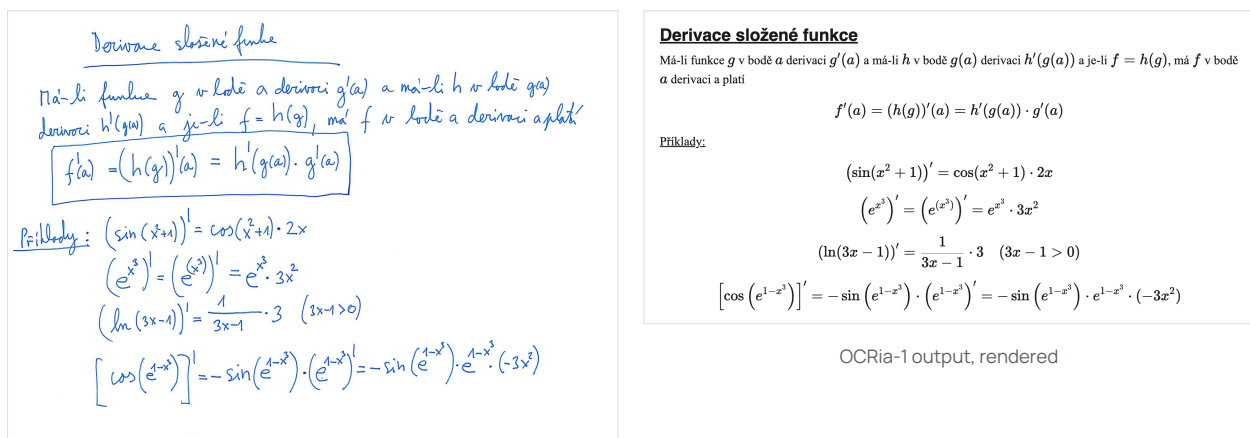
Figure 8 A chart transcribed as its data. A line chart with three series over five years. OCRia-1 writes the chart as a table of its 15 data points, one row per year and one column per series, and the values match the labels printed on the chart.



Page image

OCRia-1 output, rendered (beginning)

Figure 9 Text formatting. A hearing notice that marks a withdrawn and a continued case by striking them through, underlines the case names and sets the times in bold. OCRia-1 keeps all three: the struck-through entries as Markdown strikethrough, the case names as `<u>` and the times as bold.



Page image

OCRia-1 output, rendered

Figure 10 Handwritten formulas. A handwritten note in Czech on the derivative of a composite function. OCRia-1 transcribes the Czech text and writes each formula as $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$.

