PARSED PAGE EXPLORER (PPX): A TOOL TOWARDS AGENTIC HUMAN KNOWLEDGE DISTILLATION

Levi Willms *
Computer Science, Faculty of Science
Ontario Tech University
Oshawa, ON
levi.willms@ontariotechu.net

Ken Pu (SUPERVISOR)
Computer Science, Faculty of Science
Ontario Tech University
Oshawa, ON
ken.pu@ontariotechu.ca

April, 2026

ABSTRACT

Navigating the boundaries between vast document spaces — textbooks, papers, articles — and the abstract concepts they express is a grand challenge of agentic knowledge distillation. To help humans learn faster, AI agents need specialized tools to traverse the layers between raw information and actionable knowledge. A critical missing link in this stack is the exact alignment between raw source documents and the structured text (Markdown) that downstream agents consume. Without such provenance, agents that translate raw data into concepts cannot verify, cite, or retrieve the evidence behind their own claims.

This thesis addresses that bottleneck through the Parsed Page eXplorer (ppx), a foundational tool that establishes strict alignment between a document page and its Markdown transcription. ppx treats a document as two complementary representations: a hierarchical Layout Tree $\mathbf{LT}$ of labeled bounding boxes produced by a layout analyzer, and a Markdown document $\mathbf{MD}$ produced by OCR over the same page. The load-bearing contribution is a mapping between them that assigns each visual node a contiguous span of $\mathbf{MD}$ together with its semantic provenance. This mapping lets agents move seamlessly between documents and information without losing track of where a given fragment came from.

The thesis develops this idea in three parts. First, we give a set-theoretic formulation of multimodal alignment that separates the four-level Layout Tree from the Markdown AST and casts alignment as a pair of partial injective functions scored by semantic similarity. Second, we realize the model as a composable, embedding-based hierarchical aligner — max-weight bipartite matching over cosine similarities of Sentence-BERT embeddings, solved via the Jonker–Volgenant shortest-augmenting-path algorithm at both block and line granularities — and package it as a type-safe Python implementation on PaddleOCR / PPStructure V3 with a client/server architecture for interactive AI-assisted document reading. Third, we quantify alignment quality on a 130-page corpus across six academic documents, measure degradation under controlled OCR-noise injection, and characterize a staircase structure of precision / recall / $F1$ collapse with increasing corruption. Together these contributions provide the navigation mechanism between documents and information that agent-driven knowledge distillation requires.

Keywords agentic knowledge distillation · document understanding · layout analysis · OCR · sequence alignment · multimodal document processing · PaddleOCR · Parsed Page eXplorer

1 Introduction

The grand challenge: agentic knowledge distillation. Human knowledge lives across a vast document space — textbooks, scientific papers, technical manuals, and online articles — that a single human reader can only ever traverse a

*This document is a partial fulfillment of the requires for the Undergraduate honours thesis in Computer Science.

small fraction of. The ambition of agentic knowledge distillation is to enlist AI agents as collaborators in this traversal: agents that read documents on our behalf, extract the information they contain, and refine that information into the concepts and abstractions a human learner can act on. The payoff is compounding: if an agent can reliably move from a page of text to a citable fact, and from a collection of facts to a coherent summary, the rate at which humans turn raw documents into understanding is no longer bounded by how quickly any one of us can read.

Three layers an agent must cross. The obstacle to this program is that agents must continually traverse the boundaries between three qualitatively different representations of the same content. At the bottom sits the document: a page image or PDF, with visual structure (titles, paragraphs, tables, figures) but no addressable text. In the middle sits information: a linearized textual transcription — typically Markdown produced by an OCR pipeline — that is directly consumable by a language model but has lost its spatial grounding. At the top sits concepts: abstractions and claims formed from the information but disconnected from the documents that justify them. An agent that can freely cross these boundaries can cite back to the page that supports a claim, verify a fact against a figure, or refuse an answer when the evidence is thin. An agent that cannot remains a sophisticated guesser.

The missing rung: document $\leftrightarrow$ information. The weakest rung on this ladder is the one between documents and information — specifically, the exact correspondence between the visual structure of a page and the Markdown extracted from it. Optical character recognition reduces a document image to a linearized text stream, discarding the spatial structure that a reader uses to navigate a page Rahman et al. [2025]. Document layout analysis, in parallel, recovers that spatial structure as a hierarchy of labeled bounding boxes Sun et al. [2025], Wang et al. [2024], but says nothing about the textual content those boxes contain. Modern toolkits such as PaddleOCR / PPStructure V3 Cui et al. [2025] produce both views of the same page, yet offer no principled correspondence between them. Downstream agentic tasks — retrieval-augmented generation over documents, AI-assisted reading, table and formula grounding — demand exactly that correspondence, because it is precisely what lets an agent point to where in the document a given piece of information came from.

The Parsed Page eXplorer. This thesis introduces the Parsed Page eXplorer (ppx), a foundational tool built explicitly to be the navigation mechanism between documents and information. Given a Layout Tree $\mathbf{LT}$ produced by a layout analyzer and a Markdown document $\mathbf{MD}$ produced by an OCR engine, ppx assigns to each visual token in $\mathbf{LT}$ a contiguous span of $\mathbf{MD}$ that reproduces its transcription, together with the similarity score that justified the match. The resulting data structure — a hierarchical mapping from visual nodes to Markdown spans with attached confidence — is the semantic provenance an agent needs: every fragment of extracted information remains tethered to the visual region of the source document it was distilled from. We are deliberate in framing this as a mechanism, not a model. ppx does not attempt to generate the concepts and abstractions at the top of the ladder; it ensures that whichever agent does can always trace a concept back through information to its original document.

The layout–text alignment problem. Stripped of the agentic framing, the technical problem ppx solves is the layout–text alignment problem: given $\mathbf{LT}$ and $\mathbf{MD}$, produce a mapping $\phi : \mathcal{N} \to \mathcal{S}_{\mathbf{MD}}$ that respects the four-level containment structure of $\mathbf{LT}$ and the character-span semantics of $\mathbf{MD}$. Neither modality alone carries enough information to drive structured document understanding; the alignment is the load-bearing representation that makes both useful together. Despite its practical importance, the problem has lacked both a crisp formal statement and an established evaluation protocol in the literature.

What this thesis contributes.

- A **set-theoretic formulation** of multimodal alignment (section 3) that separates the four-level Layout Tree from the Markdown AST and poses alignment as a pair of partial injective functions scored by semantic similarity.
- A **composable pipeline** (sections 4 and 5) — the ppx system — realizing the model through sentence-embedding encoding of both modalities, hierarchical max-weight bipartite matching via the Jonker–Volgenant algorithm at block and line granularities, and a type-safe client / server architecture for interactive AI-assisted document reading.
- An **empirical benchmark** (section 6) measuring alignment quality and robustness over a 130-page, six-document corpus, including a noise-injection study that exposes a staircase degradation pattern in precision, recall, and $F1$ as OCR noise increases.

Scope. We restrict attention to English-language academic PDFs rendered by PaddleOCR / PPStructure V3; multi-script, handwritten, and historical material are out of scope. Likewise, the upper rung of the ladder — concepts and

abstractions formed from the extracted information — is the domain of whichever agent consumes ppx's output and is outside the scope of this thesis.

Organization. Section 2 surveys OCR, layout analysis, reading-order recovery, and the small prior literature on cross-modal alignment of visual tokens and text. Section 3 formalizes the alignment problem set-theoretically. Section 4 presents the embedding-based hierarchical aligner and its two constituent algorithms. Section 5 describes the three-package implementation (`ppx-ocr`, `ppx-align`, `ppx-svelte`) that makes up the `ppx` toolkit. Section 6 reports empirical alignment quality, noise robustness, and precision / recall / $F1$ degradation. Section 7 concludes with limitations and future work.

2 Related Work

The ppx toolkit sits at the intersection of four research threads: optical character recognition as the text backbone, document layout analysis as the structural backbone, reading-order and structure extraction as adjacent end-to-end approaches, and the comparatively small literature on cross-modal alignment between visual tokens and text. We survey each in turn and close by naming the gap ppx occupies.

2.1 Optical Character Recognition

Modern OCR is the product of a long transition from rule-based template matching to deep-learning detectors and sequence decoders. Classical engines such as `Tesseract` Smith [2007] began with per-glyph templates and progressed to LSTM-based recognition in their later releases; contemporary systems pair fully-convolutional text detectors with attention-based or transformer-based recognizers. The breadth of this progression is cataloged in recent systematic reviews of the field Rahman et al. [2025].

For ppx the backbone of interest is the PaddleOCR family. PP-OCRv3 Li et al. [2022] is the lightweight detector-recognizer pair that supplies line- and word-level transcriptions. PaddleOCR 3.0 Cui et al. [2025] extends this core with the PP-StructureV3 layout analyzer and a Markdown-first document parsing front-end, which together give our pipeline both of the representations it needs to align. PaddleOCR-VL Li et al. [2025] is an ultra-compact vision-language model for multilingual document parsing, relevant as a possible alternative backend behind the same ppx adapters but not currently adopted.

An orthogonal family of systems skips the OCR pipeline decomposition altogether. Donut Kim et al. [2022] and Nougat Blecher et al. [2023] are encoder-decoder models that emit structured text — JSON and Markdown respectively — directly from the page image, without producing intermediate bounding boxes. These approaches are complementary rather than competing in our setting: they produce the Markdown we would want to align against, but they do not expose the aligned visual tokens that our bipartite-matching pipeline requires on the other side.

2.2 Document Layout Analysis

Datasets. The large-scale study of document layout owes much to two dataset efforts. **PubLayNet** Zhong et al. [2019] automatically aligned roughly 360,000 PubMed Central pages with their companion XML to produce what was at the time the largest public layout-analysis corpus. **DocLayNet** Pfitzmann et al. [2022] contributed $80{,}863$ human-annotated pages across eleven categories and diverse domains; its annotation effort (32 annotators over roughly three months) stands as the practical scale benchmark for any manually-supervised document corpus and is the reference point against which we compare the modest 130-page corpus of section 6.

Detectors. On the model side, PP-DocLayout Sun et al. [2025] is the detector behind PP-StructureV3 and is the direct source of the region- and block-level labels ppx consumes. DLAFormer Wang et al. [2024] formulates layout analysis end-to-end as a set-prediction transformer, representative of the contemporary move from pipeline-style detection to unified sequence-to-set models.

Multimodal pre-trained models. A parallel line of work builds pre-trained transformers that jointly ingest text, layout, and image features. LayoutLMv3 Huang et al. [2022] is notable for including a word-patch alignment objective during pre-training; UDOP Tang et al. [2023] unifies vision, text, and layout into a single generative transformer capable of document understanding and generation. Both are natural neighbors to our work because both operate over text-with-layout inputs, but both express alignment as a model-internal pre-training signal rather than as an externally-visible mapping between visual tokens and Markdown spans. The mapping we produce is designed to be

inspected, cited, and consumed by external agents — a different object from the one these architectures compute inside their encoders.

2.3 Reading Order and Structure Extraction

Reading-order algorithms. Reading-order recovery is a long-running sub-problem of document analysis. Classical XY-cut Meunier [2005] remains the archetypal projection-profile algorithm and the baseline against which newer methods are measured. LayoutReader Wang et al. [2021] recast the problem as seq2seq prediction and released the companion ReadingBank dataset, enabling learning-based approaches at scale. More recent work Zhang et al. [2024] casts reading order as an ordering relation learnt directly from layout signals, moving away from explicit sequence decoding. Reading order enters ppx indirectly: PP-StructureV3 supplies a reading-order index for each block, and our bipartite-matching formulation uses that order to keep the per-level alignment sub-problems tractable and well-defined.

Integrated parsers. Several recent systems merge detection, recognition, and reading-order into a single model. Éclair Martin and Chen [2025] predicts text, bounding boxes, and semantic class in reading order as one end-to-end forward pass. Together with Nougat Blecher et al. [2023], these are the closest single-shot analogs to our decomposed pipeline. The trade-off is familiar: integrated models can be optimized jointly but expose fewer well-typed intermediates, while a decomposed pipeline (section 5) is easier to inspect, swap components in, and benchmark stage-by-stage.

Structural benchmarks. Two recent benchmarks frame the end-to-end problem of turning a PDF into structured Markdown. READoc Li et al. [2024] evaluates PDF-to-Markdown conversion over 2,233 arXiv and GitHub documents with an explicit reading-order component; OmniDocBench Ouyang et al. [2025] extends benchmarking to diverse PDF parsing with comprehensive per-element annotations. Neither benchmark measures token-level alignment between visual tokens and the extracted Markdown — both operate at the end-to-end output level — which leaves the noise-injection regime of section 6 complementary rather than redundant to this line of work.

2.4 Cross-Modal Alignment of Visual Tokens and Text

The direct neighbors of our problem are the few lines of prior work that explicitly address an alignment between visual and textual representations of a document. Each intersects our setting along one axis but not all of them.

Token-patch alignment inside pre-trained models. The word-patch alignment objective of LayoutLMv3 Huang et al. [2022] is the closest conceptual neighbor we know of: during pre-training the model is encouraged to map masked words to their corresponding image patches. Two differences matter. First, this is a pre-training signal internal to a single model, not a standalone aligner one can invoke at inference time on an arbitrary OCR output. Second, it operates at word / patch granularity, whereas ppx aligns at the block- and line-level structure of the AST. LayoutLMv3's alignment is a representation-learning mechanism; ours is an output artifact.

Scene-text alignment. ODM Duan et al. [2024] proposes a further text-image alignment objective aimed at scene-text detection and spotting. The setting — unconstrained outdoor imagery with short text strings and no layout hierarchy — is structurally different from the paginated-document case we study, and the alignment problem reduces to per-instance matching rather than to hierarchical span assignment.

Post-OCR string alignment. The post-OCR correction literature, exemplified by work such as Martinek et al. [2019], aligns noisy OCR output against an independent ground-truth transcript using edit distance or fuzzy string matching. This is character- or word-level, assumes a reference transcript exists, and aims to correct the OCR output rather than to establish a correspondence between two co-produced representations of the same page. Structurally it sits one rung below our problem: it aligns an OCR string against a gold string, not a layout-tree against a Markdown AST.

Positioning. To the best of our knowledge, no prior work (i) aligns four-level layout tokens to a structured Markdown AST produced by the same OCR pipeline, or (ii) does so as max-weight bipartite matching over sentence-embedding cosine similarities at both block and line granularities. The formulation of section 3 and the algorithms of section 4 therefore occupy a gap between the model-internal alignment objectives of the Document-AI literature and the string-level alignment of the post-OCR correction tradition. Framing ppx as the navigation mechanism between documents and information (section 1) is intended partly as a research-program statement: the alignment artifact itself, not the model that produced it, is the primary object of study.

3 Problem Formulation

We give a set-theoretic account of the two representations of a page and cast multimodal alignment as a mapping between them. Throughout this section, Λ denotes a finite set of layout labels and Σ the Unicode character set.

3.1 Document Representations

Definition 3.1 (Raster Document)**.** A raster document is a finite sequence of page images $D = \langle d_1, d_2, \ldots, d_P \rangle$, where each page $d_p \in \mathbb{R}^{H \times W \times 3}$ is an RGB image. All subsequent definitions apply to a single page; the page index p is dropped for brevity.

Definition 3.2 (Bounding Box)**.** A bounding box over label set Λ is a tuple

$$b = (x_0,\, y_0,\, x_1,\, y_1,\, \lambda) \;\in\; \mathbb{N}^4 \times \Lambda,$$

with top-left corner (x_0, y_0), bottom-right corner (x_1, y_1), and semantic label $\lambda \in \Lambda$ (e.g. `paragraph`, `figure`, `table`). The area of b is $\mathrm{area}(b) = (x_1 - x_0)(y_1 - y_0)$.

Definition 3.3 (Containment)**.** Bounding box b' is contained in b, written $b' \sqsubseteq b$, if and only if

$$\frac{\mathrm{area}(b \cap b')}{\mathrm{area}(b')} \;\geq\; \tau,$$

for a fixed tolerance $\tau \in (0, 1]$, where $b \cap b'$ denotes the rectangular intersection. We use $\tau = 0.8$ throughout.

3.2 Visual Tokens and the Layout Tree

The layout analyzer Sun et al. [2025], Cui et al. [2025] parses a page into a four-level hierarchy of labeled bounding boxes.

Definition 3.4 (Visual Token Layers)**.** A page is analyzed into four ordered token layers:

- $R = \{r_i\}$ — region bounding boxes (coarsest granularity)
- $K = \{k_j\}$ — block bounding boxes, each carrying a reading-order index $\mathrm{ord}(k_j) \in \mathbb{N}$ Zhang et al. [2024]
- $\mathcal{L} = \{\ell_m\}$ — line bounding boxes, sorted by y_0
- $W = \{w_n\}$ — word bounding boxes, grouped by line membership

Each visual token v is additionally annotated with its OCR transcription $\mathrm{content}(v) \in \Sigma^*$.

Definition 3.5 (Layout Tree)**.** The Layout Tree **LT** is a rooted tree with node set $\mathcal{N} = R \cup K \cup \mathcal{L} \cup W$ and edges given by containment:

$$\mathrm{parent}(v) = \arg \max_{u \in \mathrm{level}(v)-1} \frac{\mathrm{area}(u \cap v)}{\mathrm{area}(v)},$$

where $\mathrm{level}(v) \in \{0, 1, 2, 3\}$ denotes region, block, line, and word. Children of each internal node are sorted by reading order. See fig. 1.

3.3 Text Tokens and the Markdown Document

The OCR engine Cui et al. [2025], Martin and Chen [2025] produces a single Markdown string $\mathbf{MD} \in \Sigma^*$. Parsing $\mathbf{MD}$ with a Markdown parser yields a structured view suitable for alignment.

Definition 3.6 (Markdown Document and AST)**.** A parsed Markdown document is a tuple

$$\mathbf{MD} = (\mu,\, \mathcal{A},\, \sigma,\, \omega),$$

where

- $\mu \in \Sigma^*$ is the raw Markdown string;
- $\mathcal{A} = \langle a_0, a_1, \ldots, a_{N-1} \rangle$ is the ordered list of leaf AST nodes (paragraphs, list items, headings, code blocks, etc.) in document order;
- $\sigma : \mathcal{A} \to \mathbb{N} \times \mathbb{N}$ assigns to each AST node a_i a character span $\sigma(a_i) = [s_i, e_i) \subseteq [0, |\mu|)$;

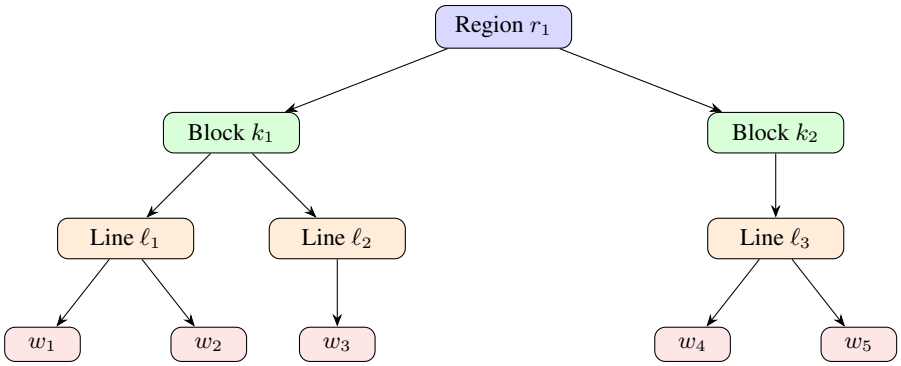

Figure 1: Schematic of the Layout Tree **LT**. Colors denote hierarchy levels: region ⊃ block ⊃ line ⊃ word. Children within each level are ordered by reading order.

- $\omega : \mathcal{A} \to (\mathbb{N} \times \mathbb{N})^*$ assigns to each AST node a_i an ordered list of word spans $\omega(a_i) = \langle [p_{i,1}, q_{i,1}), \ldots \rangle$ relative to the node's content.

See fig. 2.

Definition 3.7 (AST-Node Range)**.** An AST-node range is a half-open integer interval $[i, j) \subseteq [0, N)$ selecting the contiguous sub-sequence $\mathcal{A}[i{:}j]$ of leaf nodes. Its induced character span is $\widetilde{\sigma}([i, j)) = [s_i,\ e_{j-1}) \subseteq [0, |\mu|)$. Denote by $\mathcal{S}_{\mathcal{A}}$ the set of all such AST-node ranges.

Definition 3.8 (Word-Bounded Character Sub-Span)**.** For any AST-node range $[i, j)$, a word-bounded sub-span is a character interval $[c_s, c_e) \subseteq \widetilde{\sigma}([i, j))$ whose endpoints coincide with word-span boundaries in $\bigcup_{k=i}^{j-1} \omega(a_k)$. Denote by $\mathcal{S}_W$ the set of all such sub-spans.

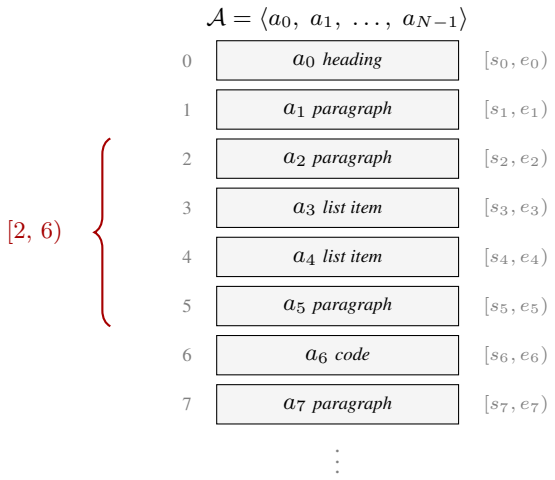

Figure 2: The AST-node sequence $\mathcal{A}$ obtained by parsing the OCR-generated Markdown **MD**. Nodes are indexed $0 \ldots N{-}1$ (left) and each carries a character span $[s_i, e_i) \subseteq [0, |\mu|)$ (right). The red brace marks an example AST-node range $[2, 6)$, inducing the contiguous character span $[s_2, e_5)$ of μ.

3.4 The Alignment Problem

The alignment problem pairs visual tokens with Markdown at two granularities: blocks to AST-node ranges, and lines to word-bounded character sub-spans within each block's range. We formalize both as partial functions with a rejection regime governed by a similarity threshold.

Definition 3.9 (Semantic Embedding)**.** A semantic embedding is a function $\varepsilon : \Sigma^* \to \mathbb{R}^d$ that maps a string to a d-dimensional unit vector. We realize ε with a Sentence-BERT encoder Reimers and Gurevych [2019], Wang et al. [2020].

Definition 3.10 (Similarity Function)**.** Given embedding ε, the similarity between a visual token v with transcription $\mathrm{content}(v)$ and a text fragment $y \in \Sigma^*$ is the cosine similarity of their embeddings:

$$\mathrm{sim}(v,\, y) \;=\; \frac{\varepsilon(\mathrm{content}(v)) \cdot \varepsilon(y)}{\|\varepsilon(\mathrm{content}(v))\|\,\|\varepsilon(y)\|} \;\in\; [-1, 1].$$

Definition 3.11 (Block Alignment)**.** Let $K \subseteq \mathcal{N}$ be the set of block visual tokens and $\mathcal{S}_{\mathcal{A}}$ the AST-node ranges of **MD**. A block alignment is a partial injective function

$$\phi_K \;:\; K \;\rightharpoonup\; \mathcal{S}_{\mathcal{A}}.$$

Given threshold $\theta \in [-1, 1]$, ϕ_K is valid if, for every $k \in \mathrm{dom}(\phi_K)$, $\mathrm{sim}(k,\, \mu[\widetilde{\sigma}(\phi_K(k))]) \geq \theta$.

Definition 3.12 (Line Alignment)**.** For each block $k \in \mathrm{dom}(\phi_K)$, let $L_k \subseteq \mathcal{N}$ be its line children in **LT** and let $\mathcal{S}_W^{(k)}$ be the set of word-bounded sub-spans of the character range $\widetilde{\sigma}(\phi_K(k))$. A line alignment conditioned on k is a partial injective function

$$\phi_\ell^{(k)} \;:\; L_k \;\rightharpoonup\; \mathcal{S}_W^{(k)},$$

valid under threshold θ analogously to definition 3.11.

Definition 3.13 (Multimodal Alignment)**.** The multimodal alignment problem is, given **LT**, **MD**, embedding ε, and threshold θ, to compute the pair $(\phi_K^*,\, \{\phi_\ell^{(k)*}\}_{k \in \mathrm{dom}(\phi_K^*)})$ that maximizes the total similarity score

$$\sum_{k \in \mathrm{dom}(\phi_K)} \mathrm{sim}(k, \mu[\widetilde{\sigma}(\phi_K(k))]) \;+\; \sum_{k} \sum_{\ell \in \mathrm{dom}(\phi_\ell^{(k)})} \mathrm{sim}(\ell,\, \mu[\phi_\ell^{(k)}(\ell)])$$

over all valid alignments. See fig. 3.

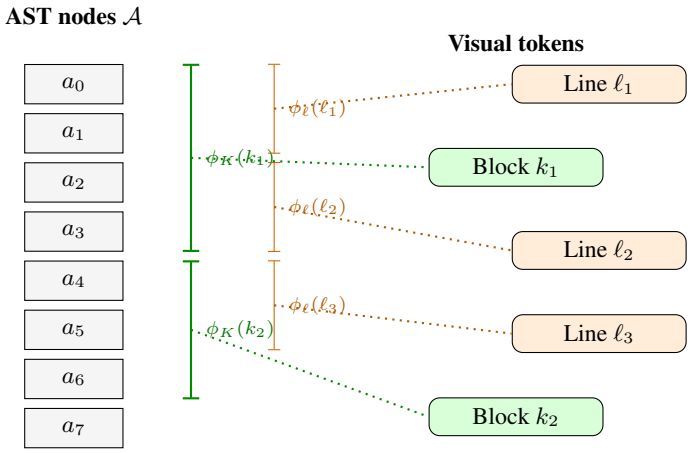

Figure 3: The multimodal alignment (ϕ_K, ϕ_ℓ) as a pair of mappings. At the block level, each visual block is matched to an AST-node range of $\mathcal{A}$ (green brackets). At the line level, each line within a matched block is matched to a word-bounded character sub-span inside that block's range (orange brackets). Dotted arrows depict the assignment. The alignment is partial: a visual token with no candidate scoring above θ is left unmatched.

Remark 3.1. Unlike interval-partition formulations, ϕ_K and $\phi_\ell^{(k)}$ are neither required to cover their parent span nor to assign every visual token: thresholding admits rejection, and bipartite injectivity prevents reuse of the same text fragment. Each level reduces to a max-weight bipartite matching problem, solved in section 4.

4 Alignment Algorithms

This section operationalizes the multimodal alignment problem of definitions 3.11 and 3.12 as a sequence of concrete algorithms. The pipeline proceeds in three logical stages. First, an encode stage lifts visual tokens and Markdown fragments into a shared semantic vector space through a Sentence-BERT encoder. Second, a match stage builds a bipartite cost matrix from pairwise cosine similarities at each hierarchy level and solves the associated max-weight assignment with the Jonker–Volgenant shortest-augmenting-path algorithm. Finally, a compose stage stacks a block-level matching over the whole page with a line-level matching conditioned on each matched block's AST-node range, so that line alignment inherits the containment structure of the Layout Tree without enforcing it through a hard constraint. The remainder of this section fills in each stage.

4.1 Semantic Embeddings of Visual and Textual Tokens

Both modalities are encoded with a single shared Sentence-BERT model Reimers and Gurevych [2019], instantiated as the distilled `all-MiniLM-L6-v2` checkpoint built on the MiniLM architecture Wang et al. [2020]. Sharing one encoder across both sides is essential: cosine similarity is only meaningful between vectors drawn from the same embedding space, and any mismatch between a separate "vision" and "text" encoder would have to be bridged by an explicit projection we would then have to calibrate. Using the same encoder on OCR transcriptions and Markdown candidates sidesteps that problem — both inputs are strings over Σ from the encoder's perspective.

Formally, let $\varepsilon : \Sigma^* \to \mathbb{R}^d$ denote the encoder ($d = 384$ for `all-MiniLM-L6-v2`), producing a unit-norm dense vector for any input string. On the visual side, each visual token $v \in \mathcal{N}$ supplies its OCR transcription $\mathrm{content}(v)$ (recovered by PaddleOCR in the pipeline of section 5) and receives the embedding $\mathbf{e}_v = \varepsilon(\mathrm{content}(v))$. On the textual side, each candidate Markdown fragment $y \in \Sigma^*$ receives $\mathbf{e}_y = \varepsilon(y)$. At block granularity these candidates are AST-node ranges $[i, j)$ materialized as the Markdown slice $\mu[\widetilde{\sigma}([i, j))]$; at line granularity they are the word-bounded sub-spans enumerated in section 4.4.

Encoding is both batched and memoized. A single `functools.cache`'d `get_model` factory returns the shared `SentenceTransformer` instance, so model-loading cost is paid exactly once per process. Within a page, every visual-token content string and every candidate fragment is collected into a single list before calling `.encode`, letting the encoder exploit GPU-friendly batched inference rather than issuing a forward pass per string.

4.2 Cosine Similarity and Max-Weight Bipartite Matching

Cosine similarity matrix. Given query embeddings $A \in \mathbb{R}^{m \times d}$ for the visual tokens at some level and candidate embeddings $B \in \mathbb{R}^{n \times d}$ for the corresponding Markdown fragments, the similarity matrix $S \in [-1, 1]^{m \times n}$ records the cosine similarity of every visual-token / candidate pair:

$$S_{ij} \;=\; \frac{A_i \cdot B_j}{\|A_i\| \, \|B_j\|}.$$

In practice, because the MiniLM checkpoint emits unit-norm embeddings, S reduces to the dot product $AB^\top$. Our implementation computes it uniformly as $S = 1 - \mathrm{cdist}(A, B, \mathrm{cosine})$ via SciPy Virtanen et al. [2020], which is robust to the rare case in which the encoder returns non-unit vectors and costs $O(mnd)$ floating-point multiplies.

From similarity to bipartite matching. At each hierarchy level the alignment sub-problem is: select a partial injective function $\phi : \{0, \ldots, m{-}1\} \rightharpoonup \{0, \ldots, n{-}1\}$ that maximizes the total similarity $\sum_i S_{i,\phi(i)}$. Injectivity encodes the requirement from definitions 3.11 and 3.12 that no Markdown fragment be reused for two different visual tokens; partiality accommodates the rejection regime introduced by the similarity threshold below. This is precisely the linear assignment problem over the rectangular cost matrix $C = -S$, a classical combinatorial-optimization problem solvable in polynomial time.

We solve it with SciPy's `linear_sum_assignment`, which implements the Jonker–Volgenant shortest-augmenting-path algorithm Jonker and Volgenant [1987], Crouse [2016]. The routine returns two index vectors (I, J) such that the pairs $\{(I_k, J_k)\}$ form an optimal assignment. The worst-case time complexity is $O(\max(m, n)^3)$; in our setting $n \gg m$ (many candidate fragments against a handful of visual tokens), so the assignment cost is effectively cubic in n. In practice the runtimes we observe are far below the worst case because the cost matrix is dense but dominated by a small number of high-similarity entries per row, a regime where shortest-augmenting-path methods terminate early.

Thresholded rejection. After the optimum has been selected, we apply a similarity threshold $\theta \in [-1, 1]$: for each pair (i, j) returned by the solver, keep the edge only if $S_{i,j} \geq \theta$. This realizes the partial-function semantics of definitions 3.11 and 3.12: a visual token for which no candidate scores above θ is left unmatched rather than forced into a low-confidence assignment. Crucially, thresholding happens after global optimization, so it can only prune weak edges; it never swaps a high-scoring edge for a weaker one. The role of θ as a precision / recall knob is discussed in section 4.5.

Algorithm 1 states the complete matching primitive. It is the single procedure invoked at both hierarchy levels — only the embedding matrices and the threshold change.

4.3 Block-Level Alignment over AST-Node Ranges

Candidate generation. Let $K = \{k_1, \ldots, k_m\} \subseteq \mathcal{N}$ be the block-level visual tokens of a page's Layout Tree. The block-alignment stage must decide, for each k_i, which contiguous chunk of the Markdown AST transcribes it.

Algorithm 1 Max-weight bipartite matching with thresholding

Require: query embeddings $A \in \mathbb{R}^{m \times d}$, candidate embeddings $B \in \mathbb{R}^{n \times d}$, threshold θ
Ensure: list of triples (i, j, s) with $s \geq \theta$
$S \leftarrow 1 - \mathrm{cdist}(A, B, \text{cosine})$ ▷ pairwise cosine similarity, $S \in \mathbb{R}^{m \times n}$
$(I, J) \leftarrow \textsc{LinearSumAssignment}(-S)$ ▷ Jonker–Volgenant, maximize total similarity
$M \leftarrow \emptyset$
for $(i, j) \in \mathrm{zip}(I, J)$ **do**
  if $S_{i,j} \geq \theta$ **then**
    $M \leftarrow M \cup \{(i,\ j,\ S_{i,j})\}$
  end if
end for
return M

Because the OCR pipeline does not expose block boundaries in the Markdown stream directly, we do not know in advance which ranges are plausible candidates; we therefore enumerate every contiguous AST-node range. Concretely, for all $0 \leq i < j \leq N$ the pair $[i, j)$ identifies one candidate, yielding $\binom{N+1}{2} = O(N^2)$ candidates per page. For each range we materialize the induced Markdown slice $\mu[\widetilde{\sigma}([i, j))]$ as a string, record the pair (i, j), and hand the string to the encoder. The recorded index pair is what lets us translate the matcher's numerical output back into a `BlockAlignmentTarget` with the appropriate `ast_start` and `ast_end` fields.

Matching. With candidates in hand, the matching step is a direct application of algorithm 1: encode the m block transcriptions into $A \in \mathbb{R}^{m \times d}$, encode the $O(N^2)$ candidate strings into B, and invoke the algorithm with block-level threshold θ_K. Each returned triple (p, q, s) assigns block K_p to the recorded range $\mathrm{rng}[q] = [i^*, j^*)$ with similarity score s, which the implementation wraps into a `BlockAlignmentTarget` with `ast_start` $= i^*$, `ast_end` $= j^*$, and `score` $= s$.

Complexity. The costs of the block stage decompose across three subroutines. Encoding is $O((m + N^2)\, C_\varepsilon)$ where C_ε is the per-string forward-pass cost; the N^2 term dominates since the number of candidates grows quadratically while the number of visual blocks grows linearly. The similarity matrix is a straightforward $O(mN^2 d)$ computation. The assignment solver runs in $O(\max(m, N^2)^3)$ worst case, which is $O(N^6)$ in the regime $N \gg m$ that holds for any realistic page. The $O(N^2)$ candidate blow-up is the principal scalability concern at the block level; without mitigation it would dominate everything else.

Granularity as the mitigating design choice. The constant N in the analysis above is not the number of characters, words, or OCR tokens in the Markdown, but the number of AST leaf nodes — paragraphs, headings, list items, code blocks, and similar block-level constructs returned by `markdown-it-py`. Anchoring the nested enumeration at this coarse, semantically meaningful granularity is a deliberate design choice in `get_doc_range_embeddings`. Typical pages in our corpus yield $N \approx 10$–50, so $N^2 \approx 10^2$–10^3 candidates — comfortably within a single batched SBERT call and a sub-second Jonker–Volgenant solve. An alternative tokenization at the word level would push N into the hundreds or low thousands and inflate N^2 by three to six orders of magnitude, making block matching computationally intractable; a character-level tokenization would do so by several additional orders of magnitude.

The mitigation is therefore a large constant-factor reduction rather than an asymptotic improvement: the $O(N^2)$ candidate count and the derived $O(N^6)$ worst-case assignment cost remain. Pages with many short AST leaves — fragmented equations, long itemized lists, numerous figure captions — can still stress the block stage in practice, which is one of the reasons the pipeline decomposes line matching onto already-matched blocks rather than attempting a single global matching (section 4.5).

4.4 Line-Level Alignment over Word-Bounded Sub-Spans

Conditioning on a matched block. Line-level matching is performed one matched block at a time rather than globally across the page. For each block $k \in \mathrm{dom}(\phi_K)$ returned by section 4.3, we restrict attention to the AST-node range $[i^*, j^*) = \phi_K(k)$ that was assigned to it, together with the set L_k of line children of k in **LT**. Conditioning on (i^*, j^*) has two consequences. First, it dramatically prunes the candidate space: character sub-spans need only be enumerated inside the matched range, not across the whole Markdown. Second, it imports the containment structure of the Layout Tree as a soft constraint — a line cannot be matched to text that its parent block was not matched to, so the alignment is consistent with the hierarchy by construction rather than by explicit enforcement.

Length-filtered word-bounded candidates. Inside the block's AST range, we collect every word span reported by the Markdown tokenizer, $W_k = \bigcup_{p=i^*}^{j^*-1} \omega(a_p)$, annotating each span with the AST-node index it came from. Enumerating every pair (p, q) with $p < q \leq |W_k|$ gives an $O(|W_k|^2)$ set of word-bounded character sub-spans $[c_s, c_e)$ — candidates whose endpoints align to token boundaries rather than to arbitrary characters. Word-boundedness avoids producing candidates that cut a word in half, which would both look nonsensical in the viewer of section 5.5 and destabilize the cosine score.

Without pruning, the word-level candidate count can still be large. We therefore apply a length filter keyed to the observed line lengths in L_k. Let $\ell_{\min} = \min_{\ell \in L_k} |\text{content}(\ell)|$ and $\ell_{\max} = \max_{\ell \in L_k} |\text{content}(\ell)|$ be the shortest and longest line transcriptions in the block. A candidate sub-span $[c_s, c_e)$ is retained only if $\ell_{\min}/2 \leq c_e - c_s \leq 2\,\ell_{\max}$. The $\frac{1}{2}\times$ contraction and $2\times$ expansion give the filter enough slack to tolerate OCR-introduced length noise — a hyphenation break, a spurious whitespace insertion, a missed punctuation mark — while still aggressively pruning candidates that are clearly too short or too long to transcribe any line in the block.

Matching. The filtered candidate strings are encoded into $B \in \mathbb{R}^{n \times d}$, the line transcriptions into $A \in \mathbb{R}^{|L_k| \times d}$, and algorithm 1 is invoked with line-level threshold θ_ℓ. Each returned triple (p, q, s) is translated into a `CharAlignmentTarget` carrying the AST-node indices and character offsets recovered from the retained (p, q) pair:

`CharAlignmentTarget`(ast_start, char_start, ast_end, char_end, s).

These targets are the line-level portion of the final `DocAlignment` payload consumed by the server and viewer of section 5.5.

4.5 Hierarchical Composition and Thresholding

Two-stage composition. The full ALIGNTREE procedure (algorithm 2) composes the block and line stages as a dependent pair. The block stage runs once per page and returns ϕ_K; then, for each $k \in \text{dom}(\phi_K)$, the line stage runs with k's matched AST range as the candidate domain, yielding the block-specific line alignment $\phi_\ell^{(k)}$. The final `DocAlignment` bundles ϕ_K with the family $\{\phi_\ell^{(k)}\}$. A `blocks_only` mode exposed through the CLI skips the line stage entirely — useful when only coarse-grained alignment is required (for example, feeding downstream retrieval) or when ablating the two stages against one another.

Algorithm 2 Hierarchical multimodal alignment (ALIGNTREE)

Require: Layout tree $\mathbf{LT}$, parsed Markdown $\mathbf{MD}$, thresholds θ_K, θ_ℓ
Ensure: Pair $(\phi_K,\ \{\phi_\ell^{(k)}\}_k)$

$K \leftarrow$ block-level nodes of $\mathbf{LT}$
$\phi_K \leftarrow \text{ALIGNBLOCKS}(K, \mathcal{A}, \theta_K)$ ▷ section 4.3 + algorithm 1
$\Phi_\ell \leftarrow \emptyset$
for all $k \in \text{dom}(\phi_K)$ **do**
    $[i^*, j^*) \leftarrow \phi_K(k)$
    $L_k \leftarrow$ line children of k in $\mathbf{LT}$
    $\phi_\ell^{(k)} \leftarrow \text{ALIGNLINES}(L_k,\ \mathcal{A}[i^*{:}j^*],\ \theta_\ell)$ ▷ section 4.4 + algorithm 1
    $\Phi_\ell \leftarrow \Phi_\ell \cup \{(k, \phi_\ell^{(k)})\}$
end for
return $(\phi_K,\ \Phi_\ell)$

Role of the threshold θ. The thresholds θ_K (block) and θ_ℓ (line) are the sole knobs controlling the precision / recall trade-off of the pipeline. A low θ admits many matches, raising recall at the cost of spurious pairs; a high θ admits only confident matches, raising precision at the cost of leaving some visual tokens unaligned. The implementation exposes both thresholds through the `ppx-align build` CLI and defaults them to $\theta_K = \theta_\ell = 0.2$. Because thresholding happens after Jonker–Volgenant has selected a globally optimal assignment, it can only suppress weak edges — it cannot ever swap a strong edge for a weaker one. This distinguishes our approach from algorithms that embed a threshold in a greedy or forward-pass matcher, where a weak early commit can lock out stronger later matches.

Why hierarchical decomposition matters. A flat, page-global line-to-word matching would have to enumerate word-bounded sub-spans over the entire Markdown stream rather than inside a single matched block. The candidate count would scale as $O(N_{\text{words}}^2)$, which is two to three orders of magnitude larger than the per-block $O(|W_k|^2)$ enumeration

we actually perform; combined with a similarity matrix of corresponding size and a cubic assignment solve, the flat version is combinatorially prohibitive for full pages. Block-level conditioning localizes each line-matching sub-problem to a single matched block; combined with the length filter, the line stage's total cost scales roughly linearly in the number of matched blocks per page.

Conditioning also contributes a structural guarantee: no line can be matched to text outside its parent block's AST range, because candidates outside that range never enter the solver. The containment constraint of the Layout Tree is thus enforced by construction at the line level, without the explicit interval-nesting constraints that a flat formulation would require. This is the reason fig. 4 places the block matcher upstream of the line matcher: ϕ_K is not merely an output of the pipeline but also the input that shapes the search space for every subsequent line-level sub-problem.

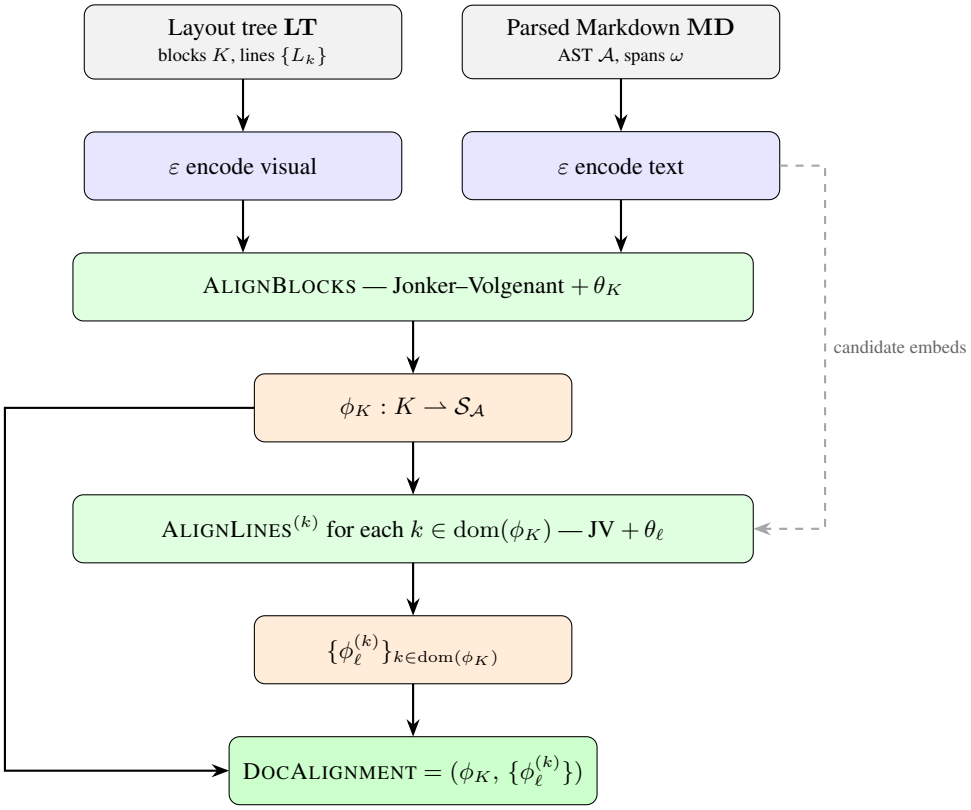

Figure 4: Pipeline view of hierarchical alignment. Visual and textual content are independently encoded by ε (blue). ALIGNBLOCKS (green) produces the block-level partial mapping ϕ_K, which conditions ALIGNLINES on each matched block's AST-node range. Both matchers share algorithm 1 — Jonker–Volgenant assignment followed by θ-thresholding (orange). The final DOCALIGNMENT bundles ϕ_K and the per-block line alignments.

5 System Implementation

The algorithms of section 4 are realized as a three-package Python and TypeScript system: `ppx-ocr` produces per-page visual tokens and Markdown from a raster PDF, `ppx-align` builds the layout tree, runs embedding-based alignment, and exposes the result over HTTP, and `ppx-svelte` is an interactive browser client that consumes the HTTP API to display synchronized views of the page image and the Markdown. fig. 5 gives the top-level picture.

5.1 PaddleOCR and PaddlePaddle Integration

PaddleOCR is a Python OCR toolkit built on the PaddlePaddle deep-learning framework Cui et al. [2025], and it supplies ppx with the two representations the alignment pipeline consumes. We invoke two of its high-level APIs at complementary granularities. `PaddleOCR` Li et al. [2022] is the text-recognition entry point: given a page image it returns line- and word-level bounding boxes together with their OCR transcriptions. We disable document unwarping

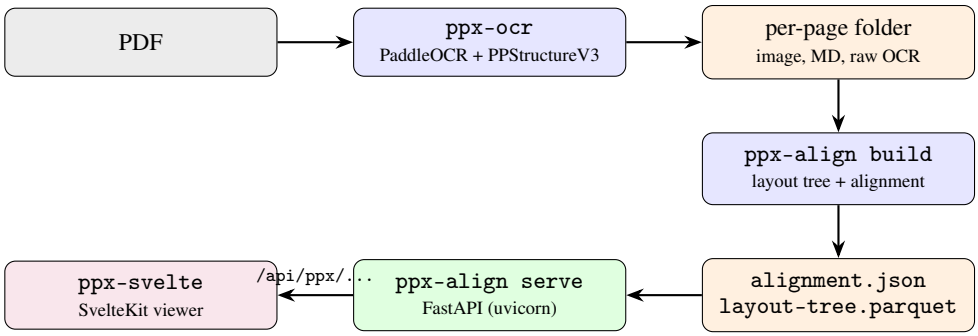

Figure 5: System architecture. Solid arrows denote data flow; orange nodes are on-disk artifacts. `ppx-ocr` runs the PaddleOCR / PPStructure V3 pipeline and writes one folder per page. `ppx-align build` ingests those folders, constructs the Layout Tree and Markdown AST, and persists `alignment.json` and `layout-tree.parquet`. `ppx-align serve` is a FastAPI backend that streams images, Markdown, tree rows, and alignments over JSON to the SvelteKit client.

because page rectification has already been handled upstream by the PDF renderer. `PPStructureV3` Sun et al. [2025] is the document-structure analyzer: it produces region- and block-level bounding boxes labeled with semantic categories (titles, paragraphs, tables, figures, formulas) and returns a Markdown rendering of the page. Both models share the PaddlePaddle runtime, and `ppx-ocr` instantiates each exactly once per process, caching them with `functools.cache` so that the GPU model-load cost is paid once and amortized across every page.

Keeping the two models behind separate adapters also makes the pipeline portable. `ppx-ocr` constructs two intermediate layer bundles — `VisualTokenLayers` for the lines and words from `PaddleOCR` and `LayoutLayers` for the regions, blocks, and formulas from `PPStructureV3` — and merges them into a single `VisualLayers` dataclass consumed downstream. The split isolates each Paddle model behind a well-typed boundary, so a future backend change (for example to the vision-language PaddleOCR-VL Li et al. [2025]) touches one adapter only rather than leaking through the rest of the pipeline.

Listing 1: Caching PaddlePaddle model instances (`ppx-ocr/core/ocr.py`).

```
from functools import cache
from paddleocr import PaddleOCR, PPStructureV3

@cache
def get_ocr_model():
    return PaddleOCR(use_doc_unwarping=False)

@cache
def get_structure_model():
    return PPStructureV3()
```

5.2 OCR and Layout Analysis Pipeline

The `ppx-ocr` command-line driver Cui et al. [2025] converts a PDF into a directory structure keyed by page index, with one subfolder per page containing all artifacts downstream stages need. The canonical artifacts include the rasterized RGB page image, the Markdown rendering produced by `PPStructureV3`, the region- and block-level bounding-box tables with PP-DocLayout labels Sun et al. [2025], the line- and word-level OCR detection tables, and auxiliary outputs such as figure crops, table HTML, and LaTeX-rendered formulas when they are present on the page. Page directories are atomic units: the driver writes into a per-page subfolder and, by default, skips pages whose folder already exists. The ingest stage is therefore safe to resume after a crash and cheap to re-run incrementally as new pages are added to a document.

5.3 Type-Safe Data Representations

A recurring source of bugs in document-analysis pipelines is silent schema drift between stages: a column rename in OCR output that goes unnoticed until an alignment metric mysteriously drops several weeks later. ppx guards against this with two complementary layers of validation, one for tabular artifacts and one for structured values.

Tabular artifacts — bounding-box tables, word tokens, layout rows — are declared as subclasses of `pandera.DataFrameModel` with explicit columns, dtypes, and nullability constraints. Schemas such as `LineTokenSchema`, `WordTokenSchema`, `LayoutSchema`, `BlockSchema`, and `LayoutTreeSchema` are validated at every stage boundary, so a type mismatch between `ppx-ocr` output and `ppx-align` input surfaces as a structured Pandera error rather than as a silent numerical regression downstream.

Non-tabular values — `ParsedDocument`, `BlockAlignmentTarget`, `CharAlignmentTarget`, `DocAlignment` — are Pydantic `BaseModels`. The alignment results in particular are wire-format-ready: `DocAlignment` serializes directly to the JSON consumed by the HTTP API (section 5.5), closing the loop from algorithmic output to network response with no ad-hoc marshaling code.

Listing 2: Pydantic models for alignment targets (`ppx-align/core/types.py`).

```python
class BlockAlignmentTarget(BaseModel):
    ast_start: int
    ast_end: int
    score: float

class CharAlignmentTarget(BaseModel):
    ast_index_start: int
    char_start: int
    ast_index_end: int
    char_end: int
    score: float

class DocAlignment(BaseModel):
    block_alignments: dict[str, BlockAlignmentTarget]
    line_alignments:  dict[str, CharAlignmentTarget]
```

5.4 Composable Pipeline Design

Every stage of the pipeline — OCR, structure analysis, layout-tree construction, Markdown parsing, block alignment, line alignment — is a pure function from one validated artifact to another. The types of section 5.3 are the glue that makes this composition safe: a stage advertises its input and output as Pandera or Pydantic types, and a mismatched upstream raises at the boundary rather than silently corrupting downstream results. Representative signatures include `build_parsed_doc(MarkdownDocument) -> ParsedDocument` for the Markdown side, `align_blocks` returning a mapping from block node ids to `BlockAlignmentTarget`, and `align_tree` returning the full `DocAlignment` that the server consumes. No stage mutates its inputs; each returns a new artifact that can be cached, persisted, or fed to multiple downstream stages. The `ppx-align` encoder runs the `SentenceTransformer` forward pass on a CUDA device when one is available, with a CPU fallback preserved for development.

The Markdown side of the pipeline deserves particular attention because it bridges the OCR output and the AST indices the formulation of section 3 is built on. `build_parsed_doc` (in `ppx-align/core/md.py`) parses the OCR-generated Markdown with `markdown-it-py` in GFM-like mode, extracts the top-level syntax-tree children as the AST leaf sequence $\mathcal{A}$, and derives character spans σ from the `node.map` line indices combined with an offset table computed from `splitlines(keepends=True)`. Word spans ω are produced by NLTK's `TreebankWordTokenizer.span_tokenize` applied to each AST node's content. A single helper function, `get_content`, dispatches over an AST range, an AST index, a `BlockAlignmentTarget`, or a `CharAlignmentTarget` to recover the Markdown slice backing any alignment artifact, giving viewers and debuggers a uniform retrieval API without re-implementing span arithmetic.

The `ppx-align` CLI orchestrates these stage functions in dependency order and writes `layout-tree.parquet` and `alignment.json` into each page directory. It supports common partial-execution modes — single-page runs, a blocks-only mode, and the noise-variant benchmark runs used in section 6 — through optional flags; we refer the reader to the project repository for the full command-line surface. The benchmark-variant runs are worth naming explicitly because they are how section 6's results come into being: a single invocation can produce auxiliary artifacts named `markdown_{level}.md` and `alignment_{level}.json` alongside the canonical artifacts, one pair per noise level, which `ppx-bench` later consumes directly without re-running the OCR stage.

5.5 Client/Server Architecture for AI-Assisted Reading

The alignment artifacts are only useful if a human — or an LLM agent — can explore them. To that end, `ppx-align serve` exposes a thin FastAPI facade over the on-disk output directory, and `ppx-svelte` consumes it in the browser. The backend streams JSON endpoints for document listing, per-page page images and thumbnails, the raw and parsed Markdown, the layout tree rows, and the `DocAlignment` itself; the viewer is a SvelteKit / Vite application that reaches the backend through the Vite dev proxy in development and a reverse proxy in production. The project repository is the authoritative reference for the exact route surface.

The viewer renders a two-panel layout — page image on the left, Markdown on the right — and fetches each document's tree and alignment lazily on page change. Hovering a visual-token overlay highlights the aligned Markdown span and scrolls the right panel to it; toggling Raw exposes the Markdown source for reverse selection, in which case the selected character range is displayed as an `ast_index:char_offset` span that can be copied into downstream tooling. Figure 7 shows the viewer in use on two documents from the benchmark corpus, with full-size versions collected in appendix A (figs. 12 and 13). The left panel of each screenshot renders the original source PDF with the hovered visual token's bounding box highlighted, while the right panel shows the corresponding span in the raw OCR-generated Markdown. Both examples expose real OCR artifacts — collapsed table structure in the CLIP example, raw `$...$` equation delimiters in the ResNet example — yet the alignment still resolves to the correct span, which is consistent with the content-type resilience quantified in section 6.2.

Separating `build` from `serve` is a deliberate architectural choice. Alignment is GPU-heavy and batch-oriented; serving is I/O bound and latency-sensitive. Splitting them lets the heavy stage run once per corpus update — producing the typed artifacts the server then reads directly — while the server runs as a lightweight sidecar. The server deliberately has no model dependencies (no PaddlePaddle, no PyTorch), which keeps its container footprint small and lets it be deployed behind an authenticating reverse proxy without exposing the model runtime or its GPU weights to the network.

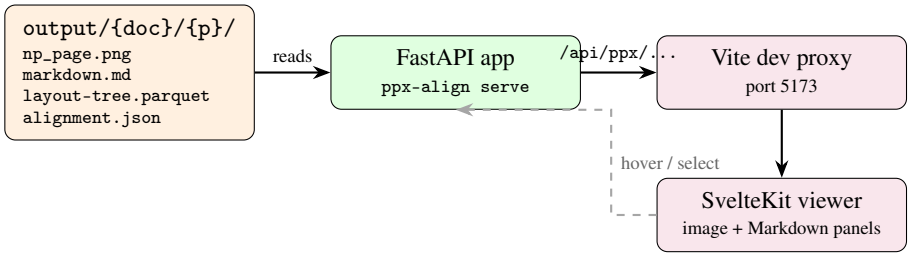

Figure 6: Client/server topology. `ppx-align serve` reads per-page artifacts from disk and exposes them as JSON and static resources. The SvelteKit viewer reaches the backend through the Vite dev proxy in development and a reverse proxy in production. The dashed arrow denotes user interactions (hover highlights, reverse text selection) that issue additional API requests.

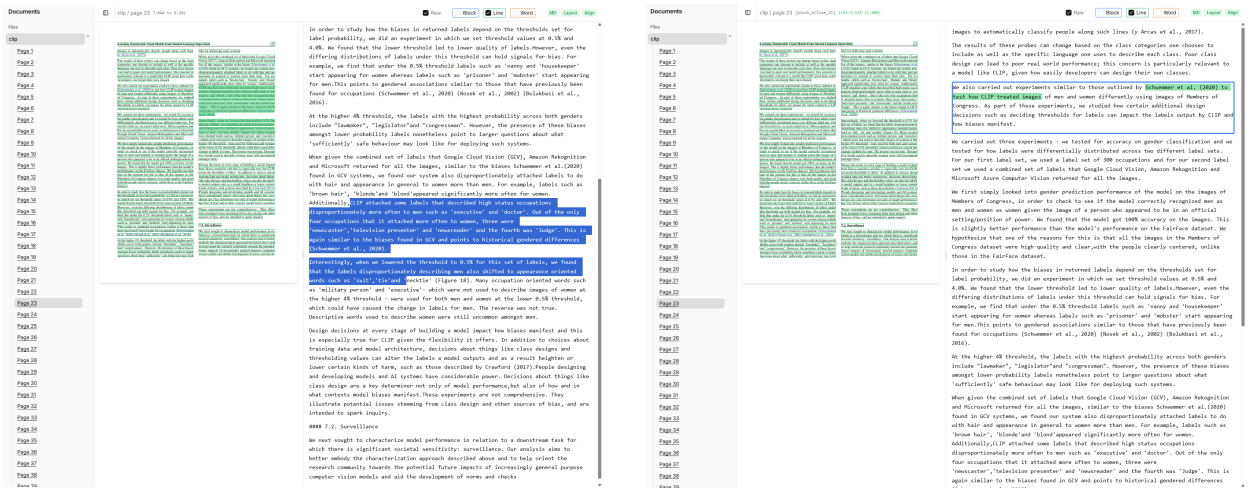

Figure 7: Thumbnails of `ppx-svelte` viewing the CLIP paper (left) and the ResNet paper (right). In each screenshot the left-hand panel of the viewer shows the original source PDF with the hovered visual token's bounding box highlighted, and the right-hand panel shows the corresponding span in the raw OCR-generated Markdown. OCR artifacts — collapsed tables in the CLIP example, raw `$...$` equation delimiters in the ResNet example — are preserved intentionally: the alignment resolves to the correct span regardless of downstream text quality. Click either thumbnail in an interactive PDF viewer to jump to the full-size version in appendix A (figs. 12 and 13).

6 Benchmark

We quantify the pipeline's behavior along three axes: alignment quality on uncorrupted inputs, degradation under controlled OCR-noise injection, and precision / recall / F1 relative to the uncorrupted reference. All experiments are implemented in the `ppx-bench` package.

6.1 Setup

Corpus. We run the benchmarks over a corpus of six academic documents — `PubLayNet`, `clip`, `deeplearning`, `paddleocrv3`, `resnet`, and `rl` — spanning 130 pages after processing through the `ppx-ocr` and `ppx-align` pipeline of section 5. Together the corpus yields 1,505 block-level visual tokens and 13,908 line-level visual tokens, a scale sufficient to expose both aggregate trends and per-label subpopulations. The per-document composition is summarized in table 1.

The six documents were selected as a convenience sample spanning several machine-learning subdomains (layout analysis, vision–language, vision, reinforcement learning, and a technical report), with a bias toward well-known papers whose OCR output we could sanity-check by reading. The sample is not intended as a representative corpus and its small size is an acknowledged limitation; the primary purpose is to exercise the pipeline across varied page structures.

	pages	blocks	lines
PubLayNet	8	95	891
clip	48	536	5,062
deeplearning	10	114	1,076
paddleocrv3	24	270	2,459
resnet	12	148	1,340
rl	28	342	3,080
total	**130**	**1,505**	**13,908**

Table 1: Benchmark corpus composition. All six documents are academic papers with the exception of `paddleocrv3`, which is a technical report. Block and line counts are the totals reported by `ppx-align` after building the Layout Tree for each page.

Block-label distribution. The PP-DocLayout categories assigned to each block determine which parent-label bucket each line inherits, and — as section 6.2 shows — are the strongest predictor of alignment quality. table 2 lists the

distribution; `text`, `paragraph_title`, `figure_title`, and `number` dominate, with a long tail of formula-, table-, and image-type blocks. The 541 lines carrying the sentinel `unknown` parent label correspond to structural gaps in the OCR output where a line was detected but not assigned to any block.

label	blocks	block mean	lines	line mean
text	718	0.97	5,114	0.96
paragraph_title	144	0.82	149	0.96
figure_title	119	0.80	492	0.97
number	118	0.02	1	0.00
header	77	0.39	64	0.55
formula	62	0.98	268	0.75
formula_number	55	0.18	1	0.00
table	43	0.90	2,035	0.99
image	37	0.37	1,291	0.42
reference	31	1.00	1,416	0.96
chart	30	0.41	2,137	0.45
footer	26	0.02	26	0.02
footnote	14	0.45	43	0.52
algorithm	8	1.00	204	0.95
aside_text	8	0.32	14	0.58
abstract	6	1.00	99	0.96
doc_title	5	0.90	6	1.00
vision_footnote	3	1.00	3	1.00
reference_content	1	0.00	4	0.00
`unknown` (lines only)	—	—	541	0.00

Table 2: Block-label distribution across the corpus, together with the mean alignment score of each label at block and line granularity. Line counts exceed block counts for `text`, `table`, `reference`, `image`, and `chart` because those blocks contain multiple OCR line tokens.

Three benchmark suites. We report three complementary studies, each implemented as a separate entry point in `ppx-bench`. The vanilla benchmark (section 6.2) measures alignment-score distributions at block and line granularity on the uncorrupted corpus; it establishes the baseline and exposes the strong label-dependent structure of alignment quality. The noise benchmark (section 6.3) re-runs the alignment pipeline on Markdown perturbed at corruption fractions $\{0.00, 0.01, 0.02, 0.04, 0.05, 0.20\}$ and tracks the resulting shift in mean line score. The precision/recall benchmark (section 6.4) treats the noise-0 alignment as a gold-standard proxy and computes per-line precision, recall, and $F1$ at 13 noise levels densely sampled in $[0, 0.2]$, isolating the transition from intact behavior to catastrophic failure.

Corruption model. The corruption routine in `ppx-bench.core.corrupt` operates on the Markdown string directly. For each line alignment target, it recovers the line's absolute character span and replaces a `noise_level` fraction of those characters with a random draw from `ascii_letters` $\cup$ `digits` $\cup$ `punctuation` $\cup \{$`space`$\}$. Characters outside any line span — structural Markdown such as heading markers, blank lines, HTML table markup — are preserved, so corruption affects only the portions of the document the aligner actually consumes. Overlapping spans are de-duplicated to prevent double-corruption of shared characters. By operating per span rather than uniformly over the Markdown, the corruption rate seen by each aligned line matches `noise_level` in expectation, and propagates proportionally to the block level (each block being a union of its line spans).

Per-line precision and recall. For each line ℓ with a gold-standard alignment, we first check whether the parent block's AST-node range agrees between the gold and noisy alignments. If it does not, we classify ℓ as a block error and record $(P, R) = (0, 0)$: the line cannot possibly land correctly because its containing block has been reassigned. When the parent block matches, we compute absolute character spans $[g_s, g_e)$ and $[n_s, n_e)$ for the gold and noisy alignments in each document's own Markdown coordinate system (corruption can shift AST boundaries when structural characters change), and set

$$P = \frac{|[g_s, g_e) \cap [n_s, n_e)|}{|[n_s, n_e)|}, \qquad R = \frac{|[g_s, g_e) \cap [n_s, n_e)|}{|[g_s, g_e)|}, \qquad F1 = \frac{2PR}{P + R}.$$

Lines without a gold alignment are excluded from evaluation; lines with a gold alignment but no noisy alignment are counted as $(0, 0)$. All aggregates reported below are unweighted means over the evaluated lines.

6.2 Vanilla Alignment Quality

Aggregate scores. On the uncorrupted corpus, the alignment pipeline achieves a mean block overlap of 0.79 with a cross-document standard deviation of 0.07, and a mean line overlap of 0.80 with $\sigma = 0.08$ (table 3). The small cross-document spread across six structurally diverse sources indicates that the pipeline generalizes: no single document pulls the aggregate in either direction.

	mean	**cross-doc σ**
Blocks	0.79	0.07
Lines	0.80	0.08

Table 3: Aggregate alignment-score statistics on the uncorrupted corpus. Means are computed over all 1,505 blocks and 13,908 lines; σ is across the six documents.

The per-document kernel-density estimates of line score all share a single peak near $\text{score} = 1.0$, with `rl` the one visibly flatter curve; the remaining five documents track one another closely.

Structural bimodality by block label. The aggregate hides substantial structural variation. Grouping by block label reveals a strong bimodal split between text-like content, which aligns near-perfectly, and non-textual or sparse-signal content, which aligns poorly or not at all. table 4 summarizes the groups.

label group	**line mean**	**interpretation**
`text`, `paragraph_title`, `table`, `reference`, `figure_title`, `algorithm`, `abstract`	0.95–0.99	text-like content aligns near-perfectly
`formula`	0.75	block matches well (block mean 0.98) but character spans inside the formula only partially overlap Markdown nodes
`chart`, `image`	0.45, 0.42	non-textual OCR content has no semantic anchor in the Markdown stream
`header`	0.55	running heads are repetitive across pages; distinctive tokens keep individual lines alignable even though the block-level score (block mean 0.39) is low
`footer`	0.02	page numbers and short footers; low-signal for sentence embeddings
`unknown`	0.00	541 lines with no block label; a structural gap in the OCR output
`formula_number`, `number`	— (n=1 each, trivial)	short numeric content fails at the block level (block means 0.18 / 0.02); nearly no lines assigned at the line level

Table 4: Line-level mean alignment score by block-label group. Values in the middle column are line means; block-level context is given in-row where it clarifies the behavior. Text-like content and non-textual content sit on opposite ends of the distribution, with almost no mass in the middle.

The stacked histogram of fig. 8 makes the bimodality concrete. Mass at $\text{score} = 1.0$ is almost entirely contributed by `text`, `table`, and `reference` lines; mass at $\text{score} = 0$ is `unknown` and `image`. The middle of the range is largely empty.

Interpretation. Alignment quality is a function of content type rather than a uniform "good/bad" attribute of the pipeline. The gap between the aggregate (0.80) and a hypothetical ceiling of 1.0 is driven almost entirely by the 20–30% of lines whose block label is `image`, `chart`, `number`, `header`, or `footer` — categories for which the OCR transcription carries little to no semantic content that an embedding-based matcher could anchor to. We interpret this as a fundamental property of the formulation (cosine similarity of Sentence-BERT embeddings requires semantic content on both sides) rather than a regression that further algorithmic tuning could close.

Per-document outliers. `rl` sits meaningfully below the rest of the corpus with a line-level mean of 0.64. Inspection reveals dense figure content and short, symbol-heavy labels, both of which push more of its lines into the low-mean label groups above. `PubLayNet` and `resnet` lead the corpus at approximately 0.88, reflecting longer, text-heavy paragraphs with few auxiliary blocks.

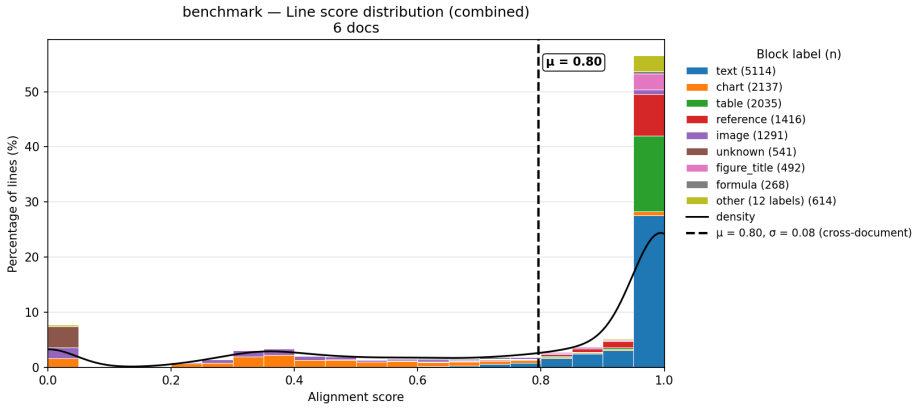

Figure 8: Uncorrupted line-score histogram stacked by block label. Mass at score=1.0 is almost entirely `text` / `table` / `reference`; mass at score=0 is `unknown` plus `image`. Structural bimodality across content types is concrete rather than a tail artifact.

6.3 Noise Robustness

Score degradation with noise. Injecting character-level corruption into each line's Markdown span causes the mean line-level alignment score to fall with the corruption fraction, but the fall is not strictly monotone near the origin. table 5 lists the 13 sampled levels along with the corresponding mean line scores and relative drops from baseline. At noise 0.005 the mean score actually rises by 0.6%; between 0.005 and 0.007 it barely moves. This apparent stability is misleading, as we explain below: the line-score metric measures how well a line matches wherever it ended up, not whether it ended up in the right place, and by the time the score begins its sustained decline the block-error rate of section 6.4 has already tripled. From noise 0.009 onward the descent is cleanly monotone, ending at 0.469 at noise 0.20 — a 40.5% drop from baseline.

noise	mean line score	drop from baseline
0.000	0.787	—
0.005	0.792	+0.6%
0.006	0.782	−0.6%
0.007	0.784	−0.4%
0.008	0.774	−1.7%
0.009	0.760	−3.5%
0.010	0.750	−4.8%
0.020	0.734	−6.8%
0.030	0.702	−10.8%
0.040	0.690	−12.4%
0.050	0.669	−15.0%
0.100	0.590	−25.1%
0.200	0.469	−40.5%

Table 5: Mean line-level alignment score across the 13 noise levels sampled by the noise benchmark. Degradation is monotonic in the large but the line-score metric is indifferent to whether a line ended up in its correct block: at noise 0.005–0.007 the score barely changes (and slightly rises at 0.005), while the block-error rate reported in section 6.4 has already begun to climb.

Distributional shift. fig. 9 visualizes the same data as per-noise kernel density estimates. The score-1.0 peak visible at noise 0 progressively shifts leftward and flattens as corruption grows, with probability mass moving from the unit-peak toward the middle of the range. The shape of the shift is consistent with a gradual deterioration of individual matches

rather than a single catastrophic drop at a critical noise level — a picture that the denser sampling of section 6.4 will refine into a staircase.

Parallel per-document trajectories. All six documents share the same decay slope. `PubLayNet` and `resnet` — the strongest baselines — remain on top at every noise level ($0.89 \to 0.49$ and $0.88 \to 0.58$ between noise 0 and 0.20). `rl` remains at the bottom ($0.64 \to 0.45$). No document "breaks" catastrophically before others; the per-document ordering at noise 0 is preserved throughout the sampled range. Cross-document differences are expressed as a vertical offset on the score axis rather than as differential sensitivity to noise.

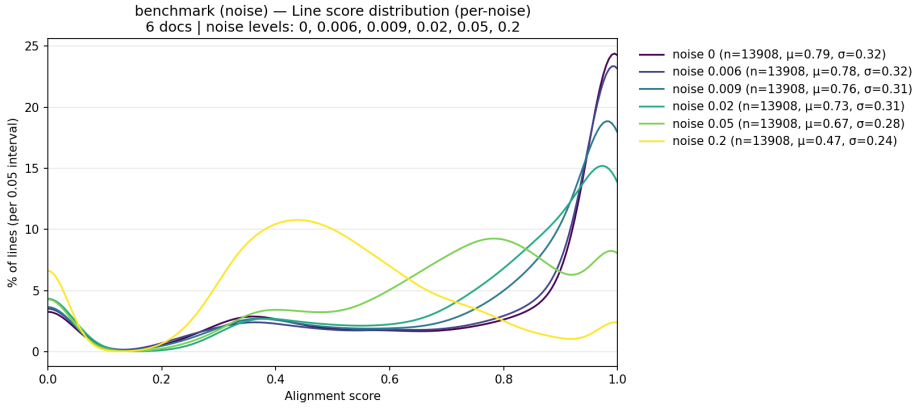

Figure 9: Per-noise KDE of line scores. Each curve is the distribution at one corruption level; the score-1.0 peak progressively shifts leftward and flattens as noise increases, with mass moving from the unit-peak toward the middle of the range.

6.4 Precision, Recall, and F1 under Noise

Headline results. Densely sampling noise levels in $[0, 0.2]$ and computing precision / recall / F1 against the noise-0 alignment resolves the smooth curve of section 6.3 into a staircase with two discrete step-changes and an earlier shoulder. table 6 reports the 13-point sweep.

noise	P	R	F1	block err	regime
0.000	1.00	1.00	1.00	0%	intact
0.005	0.90	0.90	0.90	3%	early shoulder
0.006	0.85	0.85	0.85	10%	block-error triples
0.007	0.83	0.82	0.82	12%	smooth
0.008	0.81	0.81	0.81	11%	smooth
0.009	0.73	0.72	0.72	18%	**step #1**
0.01	0.70	0.68	0.68	22%	smooth
0.02	0.64	0.61	0.62	26%	smooth
0.03	0.53	0.50	0.51	36%	**step #2**
0.04	0.50	0.46	0.47	39%	smooth
0.05	0.45	0.41	0.42	41%	smooth
0.10	0.32	0.29	0.29	48%	tail
0.20	0.14	0.12	0.13	64%	tail

Table 6: Mean precision, recall, $F1$, and block-error rate at each of the 13 sampled noise levels. Block-error rate is the fraction of lines whose parent block is reassigned under corruption, each of which contributes $(P, R) = (0, 0)$ to the averages.

fig. 10 plots the same data against a continuous noise axis. The staircase structure — otherwise invisible in the smooth summary of section 6.3 — appears as visible bends in the descent.

Staircase degradation. Between the transitions, each metric decays gently; at the steps, a discrete cohort of blocks flips in a single doubling of the noise level. Step #1, between noise 0.008 and 0.009, is sharp: $F1$ drops from 0.81 to

0.72 and the block-error rate jumps from 11% to 18%. Step #2, between 0.02 and 0.03, is of comparable magnitude: $F1$ from 0.62 to 0.51, block-error from 26% to 36%. An earlier shoulder between 0.005 and 0.006 triples the block-error rate (3% → 10%) even though $F1$ moves by only 0.05 — the damage registers first on blocks and only later propagates into line-level scores. We hypothesize that the staircase reflects content-length variation in block embeddings: short, low-distinctiveness blocks lose their similarity margin earlier than long, text-rich blocks, so increasing noise knocks out successive cohorts at distinct thresholds. Validating this mechanism per label group is left to future work.

Document-specific thresholds. Drilling into the transition zone exposes heterogeneity across documents. At noise 0.006, `paddleocrv3` and `rl` have already crossed 25% block-error, whereas `PubLayNet`, `resnet`, `clip`, and `deeplearning` remain below 10% through noise 0.008 and only step up at 0.009. The aggregate step observed at 0.009 is thus the average of documents that have already crossed their own first step and documents that cross it here; per-document staircases are sharper and shifted along the noise axis. Weaker-baseline documents cross the first step at lower noise, consistent with — though not a direct proof of — the content-distinctiveness hypothesis above.

Precision, recall, and $F1$ move together. At every sampled noise level the three metrics stay within 0.03 of each other (table 6). This reflects the symmetry of the character-span overlap metric combined with the fact that when a line matches within its correct block, the matched span is almost always exact — partial-overlap cases contributing asymmetric numerator and denominator are rare.

Bimodal precision distribution. fig. 11 shows the per-noise distribution of line-level precision. At every noise level the distribution has two peaks, near 0 and near 1. Increasing noise does not smear the middle of the range; it shifts mass from the 1-peak to the 0-peak. Alignment therefore behaves as a binary outcome per line — either fully correct or fully wrong — with very little partial credit.

Cross-document variance peaks in the transition zone. Cross-document σ on $F1$ is near zero at noise 0 (by definition), rises to approximately 0.08 at noise 0.005, widens to 0.10–0.14 across 0.02–0.05, and shrinks back to ~ 0.04 at 0.20. Documents diverge most in the middle of the descent because content-specific resilience is most visible there; at the extremes all documents are either intact or largely broken, and variance collapses.

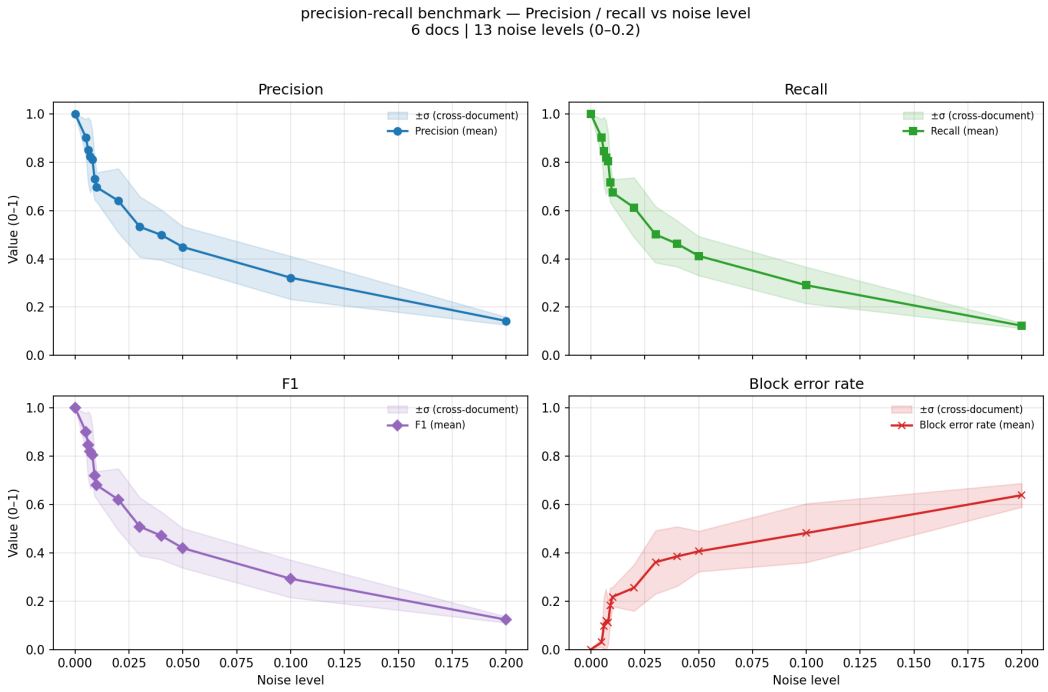

Figure 10: Mean precision, recall, $F1$, and block-error rate plotted against a continuous noise axis with cross-document σ envelopes. The staircase structure appears as visible bends in the otherwise-smooth descents around noise 0.009 and 0.03.

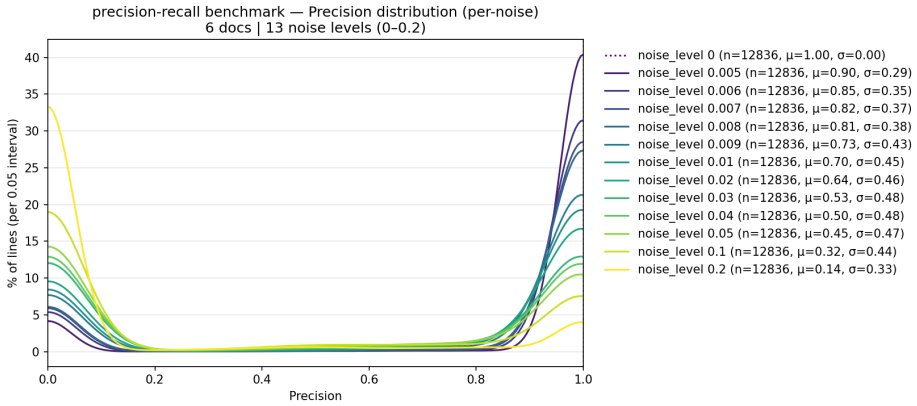

Figure 11: Per-noise precision distribution. Every curve is bimodal with peaks near 0 and 1 at every noise level; increasing noise shifts mass between the two peaks rather than smearing the middle of the range.

6.5 Cross-Cutting Observations

Taken together, the three benchmark suites expose five properties of the pipeline that matter for downstream use and that motivate the limitations and future-work directions of section 7.

Alignment is binary per line, gated by block-level correctness. The bimodal precision distribution of fig. 11, together with the block-error rule built into the metric, implies that line-level success depends almost entirely on whether the parent block is correctly matched. When the parent block lands in the right AST-node range, the line's character-level span is usually exact; when it does not, the line contributes 0 to every aggregate regardless of whether the local matcher would have produced a reasonable span. There is little partial-credit behavior in between.

Block matching is the fragile link. Consequently the block-alignment stage of section 4.3 is where noise causes the pipeline to fail. Embedding-based block matching flips when a small number of distinguishing tokens change, and the flip cascades into line-level failure even though line-level matching (when it runs at all) remains accurate. This observation motivates both the per-label failure analysis and the block-alignment diagnostics listed under Future Work.

Non-textual content is a structural blind spot. table 4 shows that `image`, `chart`, `number`, `header`, and `footer` blocks score poorly at noise 0 and cannot be repaired by any improvement to the alignment algorithm — their OCR content is not a semantic handle the Markdown provides. This is a property of the problem formulation, not of the pipeline, and any downstream system that requires coverage of these categories must supply an auxiliary matching strategy.

Uniform degradation across documents. All six documents share the same decay trajectory under noise. Differences between `rl` and `PubLayNet` are expressed as a vertical offset on the score axis rather than as differential sensitivity; the per-document ordering at noise 0 is preserved throughout the sampled range. Document-level generalization is therefore robust, even as per-document staircases differ in where along the noise axis their step-changes land.

Block-error rate is the earliest-warning metric. At noise 0.005 the mean line score actually rises by 0.6% (to 0.792), and at noise 0.006 it has fallen only to 0.782 — a 0.6% drop from baseline. In the same range the block-error rate triples from 3% to 10% and precision drops from 0.90 to 0.85. Monitoring line-level scores alone would therefore understate early-stage degradation by a wide margin; operational deployments should treat block-error rate as the primary health metric and line score as a secondary signal.

7 Conclusion

This thesis set out to address the weakest rung on the agentic knowledge-distillation ladder of section 1: the correspondence between a document and the Markdown produced from it. We introduced the Parsed Page eXplorer (`ppx`) as a foundational tool for that correspondence, developed it across three chapters, and evaluated it on a modest but diverse

benchmark. The remainder of this section summarizes what was contributed, what the contributions do not resolve, and the directions for which the present work lays the groundwork.

7.1 Summary of Contributions

The thesis made three contributions, each spanning one of the paper's core chapters. First, section 3 gave a **set-theoretic formulation** of multimodal document alignment that cleanly separates the four-level Layout Tree from the Markdown AST and casts alignment as a pair of partial injective functions (one at block granularity, one at line granularity) scored by semantic similarity. The formulation's value is not only as a precise statement of what `ppx` computes but also as a shared vocabulary that future systems — whether decomposed pipelines or integrated end-to-end models — can use to describe their own outputs in terms compatible with ours.

Second, sections 4 and 5 presented a **composable pipeline** that realizes the formulation. The algorithmic core is embedding-based hierarchical matching: Sentence-BERT encodings of visual transcriptions and Markdown candidates are fed into a max-weight bipartite matcher solved by the Jonker–Volgenant shortest-augmenting-path algorithm, first at block granularity over AST-node ranges, then within each matched block at line granularity over word-bounded character sub-spans. The system incarnation ships as three cooperating packages — `ppx-ocr`, `ppx-align`, and `ppx-svelte` — with type-safe intermediate artifacts, a FastAPI HTTP surface, and a SvelteKit viewer built to make alignments inspectable case-by-case.

Third, section 6 provided an **empirical benchmark** quantifying the pipeline's behavior. On a 130-page, six-document corpus we measured aggregate alignment quality under clean inputs, characterized the strong bimodality of per-label scores (text-like content aligns near-perfectly; non-textual content does not), and exercised the pipeline under controlled OCR-noise injection. The most distinctive finding is the staircase degradation of precision, recall, and $F1$ as noise increases: two discrete step-changes at noise levels of roughly 0.009 and 0.03 separate a gentle decay from a catastrophic collapse, with block-level matching identified as the fragile link that fails first.

7.2 Limitations

The preceding contributions come with several honest limitations, which we state explicitly so that downstream users of `ppx` can reason about when its outputs should and should not be trusted.

Content-type sensitivity. Alignment quality is a strong function of block label rather than a uniform pipeline attribute. Text-like content (`text`, `table`, `reference`, `paragraph_title`) aligns near-perfectly on clean inputs, whereas non-textual blocks — images, charts, raw page numbers, headers, and footers — have no semantic anchor in the Markdown and score poorly regardless of noise level or algorithmic tuning. This is a property of the cosine-similarity formulation rather than of any implementation choice: embedding-based matching fundamentally requires semantic content on both sides.

Block-level fragility. The benchmark identifies block matching as the dominant source of failure under noise. Once a line's parent block has been reassigned to the wrong AST range, the line's character-span metric collapses to zero regardless of how well the local line matcher would have behaved. The pipeline's effective precision is therefore bounded by the block stage's robustness, and small changes in block-level similarity margins translate into discrete cohort flips at the observed noise thresholds.

Domain scope. The pipeline and benchmark are tuned to English-language academic PDFs rendered by PaddleOCR / PPStructure V3. Behavior on multi-script documents, handwritten material, historical typography, and born-digital formats with unusual layout conventions is untested and should not be assumed to transfer.

Dependence on the upstream layout analyzer. `ppx` is a consumer of the Layout Tree produced by PP-StructureV3; errors upstream propagate downstream. A block that the detector mis-segments cannot be recovered by any alignment algorithm operating on its output, so `ppx`'s own alignment-quality metrics should be read as conditional on the quality of the layout analysis that fed them.

7.3 Future Work

The findings of section 6 and the limitations above suggest several directions for extending `ppx`.

A natural first step is a **per-label failure analysis**. The staircase thresholds identified at the aggregate level are almost certainly the averages of sharper per-label staircases, with short and low-distinctiveness block labels losing their

similarity margin earlier than long, text-rich ones. Grouping precision, recall, and block-error rate by `block_label` at the critical noise levels would turn the current qualitative hypothesis into a concrete taxonomy of which content types are least robust to corruption.

Closely related is **block-alignment diagnostics**. If a small number of tokens dominate each block's match, then targeted perturbation of those tokens may explain both the discrete step structure and the binary per-line success / failure pattern seen in the precision distribution. Diagnostics of this kind would also inform the choice of threshold θ and suggest when to escalate a low-margin block to a fallback matcher.

At the encoder level, **domain-adapted embeddings** are a plausible lever for the content-type sensitivity limitation. Replacing `all-MiniLM-L6-v2` with a model fine-tuned on Markdown / scientific text pairs should improve behavior on equations and tables, where general-purpose sentence embeddings are least informative. The encode stage of section 4 is deliberately isolated behind a single function so that such a swap is mechanical.

At the post-processing level, **structural repair passes** could enforce soft containment (no child span exceeds its parent's character range) and soft monotonicity (sibling spans cross only when scores justify it) on the final `DocAlignment`. These are structural constraints that the current matcher does not enforce globally, and they would absorb several of the silent-error patterns our inspection tooling currently surfaces manually.

At the matching-formulation level, **larger-context matching** would replace sentence-level embeddings with cross-attention between each visual token and a context window of surrounding AST nodes, so that disambiguation across near-duplicate blocks exploits the layout-tree neighborhood rather than discarding it. This would push `ppx` closer in spirit to LayoutLMv3's word-patch objective Huang et al. [2022], but applied as an inference-time matcher producing an externally-visible alignment rather than as a model-internal pre-training signal.

Extending to concept-space alignment. The five directions above all sharpen the document $\leftrightarrow$ information rung of the agentic ladder introduced in section 1. The natural extension is to climb to the next rung: align information to the concepts and claims it supports. The formal model of section 3 transfers in structure if not in detail. One would replace the Markdown AST $\mathcal{A}$ with a concept graph — entities, relations, and supporting claims — and replace the Sentence-BERT embedding ε with a model whose geometry reflects semantic equivalence of propositions rather than surface-form similarity of text. The pair of partial injective functions $(\phi_K, \phi_\ell^{(k)})$ would generalize to a pair of concept-attachment functions that assign each Markdown span to the concept(s) it evidences, with a containment analog that says a child span's concept-set is contained in its parent's. The technical obstacles are substantive: concept identity is not crisp the way AST indices are, so the injectivity requirement relaxes to a tolerance over equivalence classes; and the candidate space is open-ended, so the block-level enumeration strategy of section 4.3 does not transfer verbatim. Prior work on entity linking, claim extraction, and knowledge-graph construction supplies the individual primitives, but composing them behind a `ppx`-style typed interface that preserves provenance across the document–information–concept stack is, as far as we know, an open problem.

Agentic consumption and evaluation. The other direction the agentic framing of section 1 invites is building on top of `ppx` rather than underneath it. Three classes of agent workflow are immediate consumers of the alignment artifact. The first is citation-constrained retrieval-augmented generation: an agent that answers a question must return, along with its answer, the layout-node span that justifies each claim, and must refuse to answer when no span scores above a confidence threshold. The second is document-grounded question answering with span-level anchors: a reader-style agent that highlights the precise region of a page image that supports each answer, using `ppx`'s mapping to translate from Markdown-space reasoning back to page-space display. The third is provenance-preserving summarization: condensing a document while attaching, to each sentence of the summary, the visual-token set it distilled. All three workflows share a common need that the current literature does not systematically evaluate: a benchmark for how well an agent uses provenance, not just whether its final answer is correct. Designing such a benchmark — with metrics for span-level citation accuracy, refusal calibration, and faithfulness of highlighted regions to claimed evidence — is itself a research direction that `ppx` is a pre-requisite for, and we see it as the most direct way the work of this thesis feeds back into the agentic knowledge-distillation program it was motivated by.

Taken together, these directions split into two groups. The first five sharpen the document $\leftrightarrow$ information rung of the agentic ladder without re-architecting the pipeline, and all rest on the typed boundaries that section 5 already exposes — they are targeted upgrades that the existing scaffold is designed to absorb. The last two, concept-space alignment and agentic consumption, climb off that rung onto new ones and open research programs rather than closing them: they are the directions in which the foundation this thesis provides becomes a prerequisite for work rather than a replacement for it. In that sense the contributions of this thesis are intended less as a final system than as a scaffold: a formal model, a benchmark that identifies where the scaffold bends, and an implementation designed to absorb targeted upgrades and to

carry new rungs above it. The navigation mechanism between documents and information that section 1 posited as the weakest rung of the agentic knowledge-distillation ladder is, we hope, one step sturdier for it.

A Full-size Viewer Screenshots

The thumbnails in fig. 7 link here. Each figure below is the full-resolution screenshot.

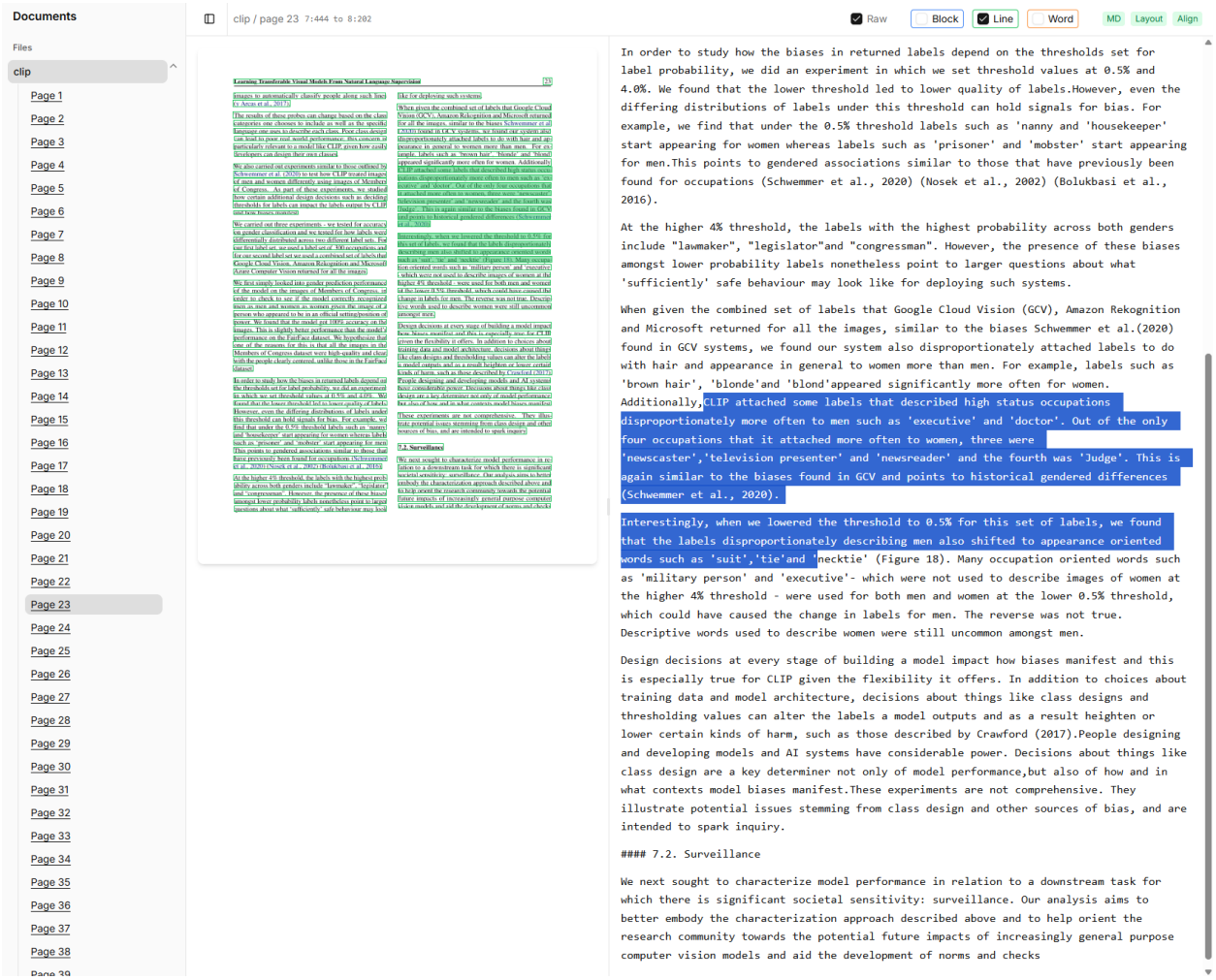

Figure 12: Full-size `ppx-svelte` viewing the CLIP paper. Left panel: original source PDF with hovered visual token highlighted. Right panel: raw OCR-generated Markdown with the aligned span selected. The collapsed-table artifacts in the Markdown panel are a known OCR limitation; the alignment still identifies the correct region of the page image. Linked from fig. 7.

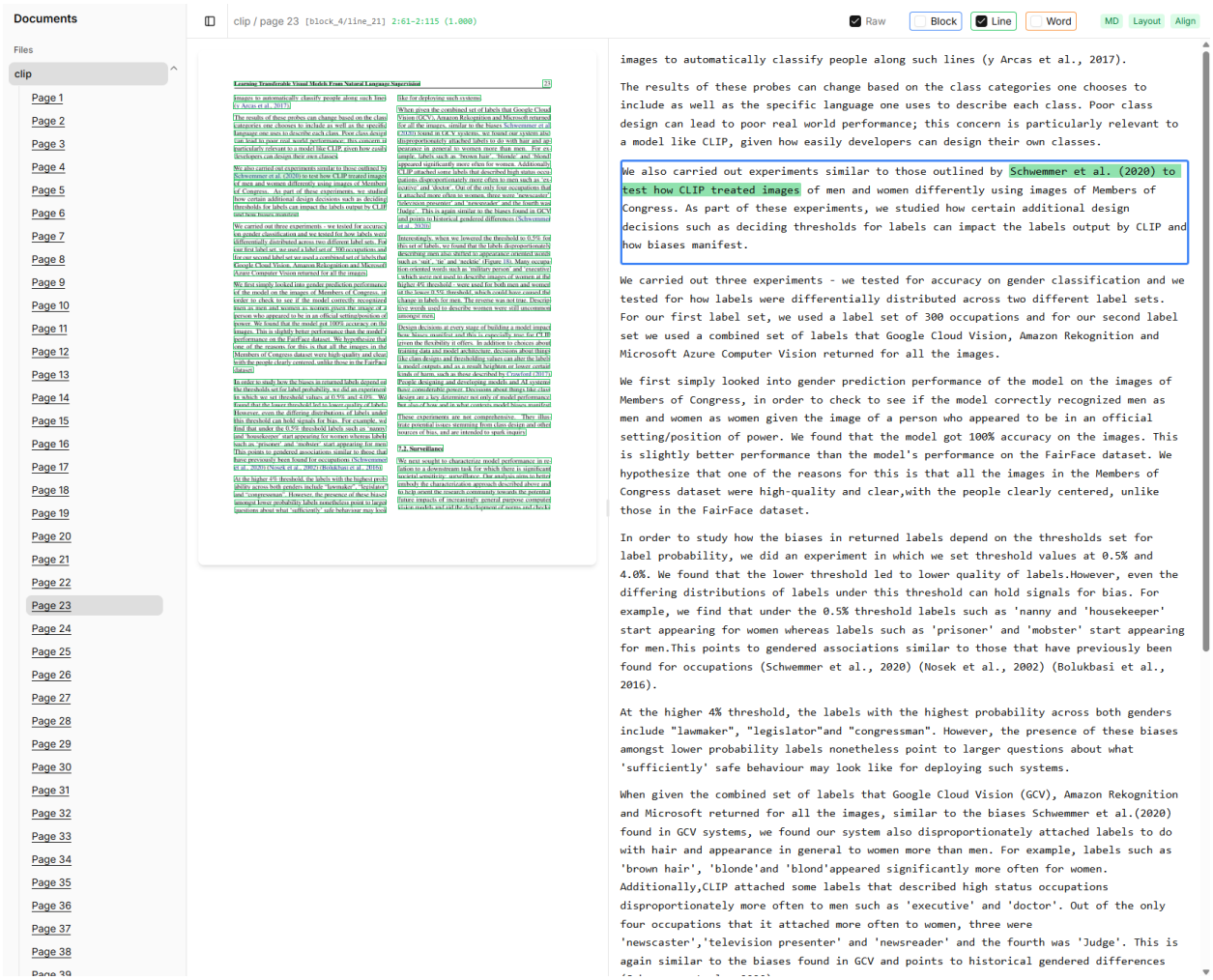

Figure 13: Full-size `ppx-svelte` viewing the ResNet paper. Left panel: original source PDF with hovered visual token highlighted. Right panel: raw OCR-generated Markdown with the aligned span selected. Equation blocks appear as raw `$...$` fragments because `PPStructureV3` emits them as LaTeX-delimited Markdown; the alignment pairs each equation block with its correct bounding box regardless of this surface form. Linked from fig. 7.

References

M. Rahman, S. Ali, et al. Systematic literature review of machine learning models and applications for text recognition. IEEE Access, 2025. doi:10.1109/ACCESS.2025.

Ting Sun, Cheng Cui, Yuning Du, and Yi Liu. Pp-doclayout: A unified document layout detection model to accelerate large-scale data construction. arXiv preprint arXiv:2503.17213, 2025.

J. Wang, K. Hu, and Q. Huo. Dlaformer: An end-to-end transformer for document layout analysis. In Proceedings of the International Conference on Document Analysis and Recognition (ICDAR), 2024. arXiv:2405.11757.

Cheng Cui, Ting Sun, Manhui Lin, Tingquan Gao, Yubo Zhang, et al. Paddleocr 3.0 technical report. arXiv preprint arXiv:2507.05595, 2025.

Ray Smith. An overview of the Tesseract OCR engine. In Proceedings of the International Conference on Document Analysis and Recognition (ICDAR), 2007. doi:10.1109/ICDAR.2007.4376991.

Chenxia Li, Weiying Liu, Ruoyu Guo, Xiaoting Yin, Kaiwen Jiang, Yongkun Du, Yuning Du, Lingfeng Zhu, Baohua Lai, Xiaoguang Hu, Dianhai Yu, and Yanjun Wang. Pp-ocrv3: More attempts for the improvement of ultra lightweight ocr system. arXiv preprint arXiv:2206.03001, 2022.

X. Li, M. Zhang, et al. Paddleocr-vl: Boosting multilingual document parsing via a 0.9b ultra-compact vision-language model. arXiv preprint arXiv:2510.14528, 2025.

Geewook Kim, Teakgyu Hong, Moonbin Yim, JeongYeon Nam, Jinyoung Park, Jinyeong Yim, Wonseok Hwang, Sangdoo Yun, Dongyoon Han, and Seunghyun Park. OCR-free document understanding transformer. In European Conference on Computer Vision (ECCV), 2022. arXiv:2111.15664.

Lukas Blecher, Guillem Cucurull, Thomas Scialom, and Robert Stojnic. Nougat: Neural optical understanding for academic documents. arXiv preprint arXiv:2308.13418, 2023.

Xu Zhong, Jianbin Tang, and Antonio Jimeno-Yepes. PubLayNet: Largest dataset ever for document layout analysis. In Proceedings of the International Conference on Document Analysis and Recognition (ICDAR), 2019. arXiv:1908.07836.

Birgit Pfitzmann, Christoph Auer, Michele Dolfi, Ahmed S. Nassar, and Peter Staar. DocLayNet: A large human-annotated dataset for document-layout analysis. In Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD), 2022. doi:10.1145/3534678.3539043. arXiv:2206.01062.

Yupan Huang, Tengchao Lv, Lei Cui, Yutong Lu, and Furu Wei. LayoutLMv3: Pre-training for document AI with unified text and image masking. In Proceedings of the 30th ACM International Conference on Multimedia (MM), 2022. arXiv:2204.08387.

Zineng Tang, Ziyi Yang, Guoxin Wang, Yuwei Fang, Yang Liu, Chenguang Zhu, Michael Zeng, Cha Zhang, and Mohit Bansal. Unifying vision, text, and layout for universal document processing. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2023. arXiv:2212.02623.

Jean-Luc Meunier. Optimized XY-cut for determining a page reading order. In Proceedings of the International Conference on Document Analysis and Recognition (ICDAR), 2005. doi:10.1109/ICDAR.2005.182.

Zilong Wang, Yiheng Xu, Lei Cui, Jingbo Shang, and Furu Wei. LayoutReader: Pre-training of text and layout for reading order detection. In Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2021. arXiv:2108.11591.

C. Zhang, Y. Tu, Y. Zhao, et al. Modeling layout reading order as ordering relations for visually-rich document understanding. In Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2024.

P. Martin and L. Chen. Éclair: Extracting content and layout with integrated reading order for documents. arXiv preprint arXiv:2502.04223, 2025.

Zichao Li et al. READoc: A unified benchmark for realistic document structured extraction. arXiv preprint arXiv:2409.05137, 2024.

Linke Ouyang, Yuan Qu, Hongbin Zhou, Jiawei Zhu, Rui Zhang, et al. OmniDocBench: Benchmarking diverse PDF document parsing with comprehensive annotations. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2025. arXiv:2412.07626.

Chen Duan, Pei Fu, Shan Guo, Qianyi Jiang, and Xiaoming Wei. ODM: A text-image further alignment pre-training approach for scene text detection and spotting. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2024. arXiv:2403.00303.

Jiri Martinek, Ladislav Lenc, and Pavel Kral. Building an efficient OCR system for historical documents with little training data. In Proceedings of the International Conference on Document Analysis and Recognition Workshops (ICDAR-W), 2019. doi:10.1007/978-3-030-22871-2_58.

Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 3982–3992. Association for Computational Linguistics, 2019. arXiv:1908.10084.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. MiniLM: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. In Advances in Neural Information Processing Systems (NeurIPS), 2020. arXiv:2002.10957.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, et al. SciPy 1.0: Fundamental algorithms for scientific computing in Python. Nature Methods, 17:261–272, 2020. doi:10.1038/s41592-019-0686-2.

Roy Jonker and Anton Volgenant. A shortest augmenting path algorithm for dense and sparse linear assignment problems. Computing, 38(4):325–340, 1987. doi:10.1007/BF02278710.

David F. Crouse. On implementing 2D rectangular assignment algorithms. IEEE Transactions on Aerospace and Electronic Systems, 52(4):1679–1696, 2016. doi:10.1109/TAES.2016.140952.