

HEALTHCARE DOCUMENT INTELLIGENCE

A Layout-Aware Pipeline for Structured Extraction from Healthcare Insurance Documents

A staged pipeline that separates layout understanding from semantic extraction, keeping language-model calls focused on **small, relevant image crops**

AUTHOR

Neurologic AI

PUBLISHED

July 2026

DOCUMENTS

EOB statements and provider contracts

ABSTRACT

Healthcare insurance documents such as Explanation of Benefits (EOB) statements and provider contracts encode business-critical information in dense, highly visual layouts where the meaning of a value depends on its position on the page. Extracting this information reliably is difficult for both naive optical

character recognition, which destroys table structure, and for end-to-end large language models, which are costly and error-prone when asked to read entire multi-page documents at once. This paper describes a staged pipeline that separates layout understanding from semantic extraction. A document is first rasterized and passed through optical character recognition to recover words and their bounding boxes. A fine-tuned LayoutLMv2 multimodal model then classifies the document type and segments each page into semantically meaningful regions. Only these localized regions are cropped and forwarded to a vision-capable large language model, which returns schema-compliant structured records that are assembled into final tables. We detail the composition of the training data and its annotation-to-label pipeline, the multimodal model and its three specialized heads, the fine-tuning methodology, and the inference and extraction stages. The design keeps expensive language-model calls focused on small, relevant image crops rather than whole documents, improving both accuracy and cost while producing clean, auditable, tabular output.

01

Introduction

Healthcare payers and providers exchange large volumes of semi-structured documents whose contents must be converted into structured data before they can be used in downstream systems. Two document families dominate this workload. **Explanation of Benefits (EOB)**, also called Explanation of Payment (EOP), statements describe how an insurance claim was adjudicated, itemizing charged, allowed, and paid amounts for each service line together with copayments, coinsurance, deductibles, and remark codes.

Provider contracts, in the style of a Summary of Benefits and Coverage, describe the terms of a plan: covered services, cost-sharing, exclusions, coverage limitations, referral requirements, and effective dates. Both are laid out as tables and multi-column forms in which spatial position carries as much meaning as the text itself.

Two approaches to reading these documents are common, and both have shortcomings.

APPROACH 1

Plain OCR plus pattern matching

The first applies plain optical character recognition and then pattern matching over the recovered text; this is brittle because linearizing a table into a stream of words discards the row-and-column relationships that give the numbers meaning.

APPROACH 2

Whole documents to a language model

The second sends whole document images to a large language model and asks it to return structured data directly; this is expensive, scales poorly with document length, and invites the model to hallucinate structure or overlook small tables buried in long pages.

The work described here takes a different, staged starting point. Rather than asking a single component to both locate and interpret content, the pipeline first uses a fine-tuned multimodal model to determine *where* the relevant content lies on each page, and only then uses a language model to determine *what* that content says. Concretely, the pipeline proceeds through five stages: rasterization of any input file into per-page images; optical character recognition to recover words and bounding boxes; document-type classification with a fine-tuned LayoutLMv2 model; section segmentation with specialized LayoutLMv2 heads; and finally cropping of the detected regions followed by schema-driven extraction with a vision-capable large language model. Because the layout stages are cheap and deterministic, the comparatively expensive language model is invoked only on small, high-value image crops. The remainder of this paper describes the data underlying this pipeline, the model and its training, the inference procedure, and the extraction stage that produces the final structured tables.

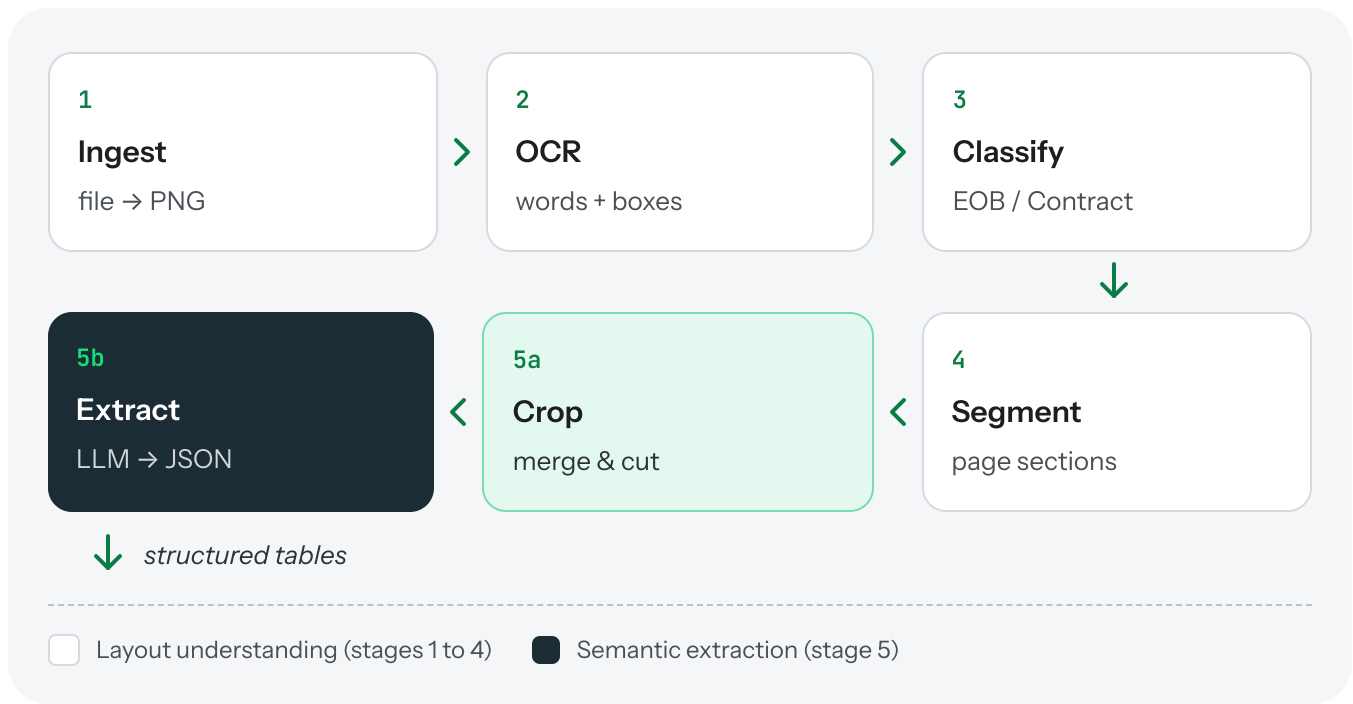


Figure 1. The five-stage architecture. Layout understanding (stages 1 to 4) localizes relevant content; semantic extraction (stage 5) reads it.

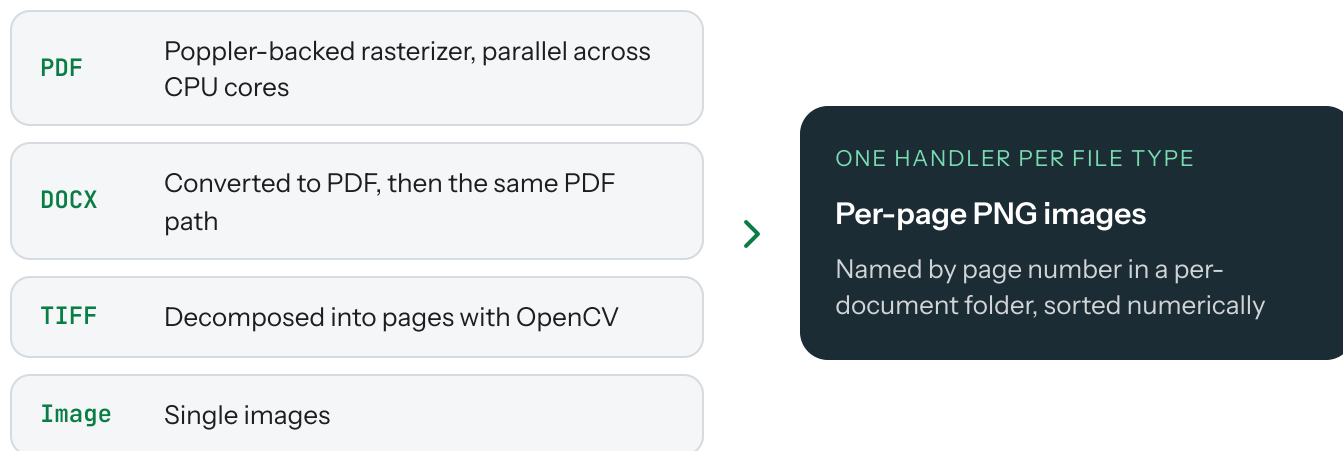
02

Data and document processing

The first two stages of the pipeline transform an arbitrary uploaded file into two aligned artifacts: a set of per-page images and a structured record of the text and geometry found on each page. Together these form the input consumed by the model.

2.1 Input formats and rasterization

The system accepts documents as PDF, DOCX, or multi-page TIFF files, as well as single images. A converter layer, organized as one handler per file type, normalizes all of these into per-page PNG images. PDF files are rendered page by page using a Poppler-backed rasterizer, parallelized across available CPU cores. DOCX files are first converted to PDF and then rendered through the same PDF path. Multi-page TIFF files are decomposed into individual pages using OpenCV. Every page is written as an image named by its page number into a per-document folder, and the resulting image list is sorted numerically so that page ordering is deterministic throughout the rest of the pipeline. This consistent, ordered naming is important because later stages align model predictions and image crops back to specific pages by index.



2.2 Optical character recognition

Each page image is then processed by an optical character recognition engine based on Tesseract, run in a layout-analysis mode that preserves the reading structure of the page. The engine returns results in hOCR format, an HTML representation in which each block, line, and word carries an embedded bounding box. This markup is parsed into a normalized JSON structure that records, for every page, its pixel dimensions and an ordered list of lines; each line in turn carries its text, its bounding box, and the list of constituent words with their own boxes. The page dimensions are retained because all downstream geometry is expressed relative to page size rather than in absolute pixels.

This JSON structure is deliberately modeled on the response format historically produced by cloud OCR services, and the OCR stage is written so that the backend can be replaced without disturbing any later stage: only the shape of this JSON must be preserved. In effect, the OCR schema is the stable contract between the document-processing stages and the model. Table 1 summarizes the information captured per page.

Level	Fields captured
Page	page number, pixel width, pixel height, ordered list of lines
Line	bounding box, text, ordered list of words
Word	bounding box, text

Table 1. Information recorded by the optical character recognition stage.

The model

3.1 A multimodal backbone

The core of the pipeline is a fine-tuned LayoutLMv2 model. LayoutLMv2 is a transformer purpose-built for document understanding: rather than treating a page as a stream of text, it embeds each token jointly from three modalities. The first is the textual content of the token, tokenized into word pieces. The second is the two-dimensional layout of the token, derived from its bounding box and encoded as a spatial positional embedding. The third is a visual embedding of the corresponding region of the page image, produced by a convolutional backbone. By fusing what a token says, where it sits, and how it looks, the model can distinguish, for example, a monetary value inside a claim-line table from an identical value printed in a page footer. This is precisely the distinction that a text-only reader cannot make, and it is why the system uses a layout-aware model rather than a conventional sequence model.

TEXT

What a token says

Tokenized into word pieces

LAYOUT

Where it sits

Bounding box as a spatial positional embedding

VISUAL

How it looks

Page-image region from a convolutional backbone

Fused token embedding in the LayoutLMv2 transformer

3.2 Coordinate normalization

LayoutLMv2 expects all bounding-box coordinates in a resolution-independent space scaled to the range 0 to 1000. A coordinate is normalized by dividing by the relevant page dimension and multiplying by one thousand,

$$x_{\text{norm}} = \lfloor 1000 \cdot x / W \rfloor, \quad y_{\text{norm}} = \lfloor 1000 \cdot y / H \rfloor,$$

where W and H are the page width and height, and the inverse mapping recovers pixel coordinates from the normalized space. The same normalization is applied identically during training and during inference; preserving this invariant is essential, because a mismatch would present the model with geometry it was never trained on.

3.3 Three specialized heads

The system employs three separately fine-tuned LayoutLMv2 checkpoints. All three share the same base architecture and the same inference machinery and differ only in their label sets and learned weights. A base model performs document-type classification, distinguishing EOB statements from provider contracts. A second model segments EOB pages into their characteristic regions. A third segments provider-contract pages. The two segmentation models are implemented as thin specializations of the base model, overriding only the model weights, the number of labels, and the mapping from label index to region name; all other logic is shared. Table 2 lists the label sets.

Head	Labels	Region / class names
Document classifier	2	EOB ; Contract
EOB segmenter	5	others ; eob_claim_line ; header ; payment_information ; payment_summary
Contract segmenter	4	Coverage_overview ; Service_information ; Other ; Coverage_examples

Table 2. The three fine-tuned model heads and their label sets.

04

Model training

The segmentation and classification models are produced by a supervised fine-tuning pipeline that converts manually annotated documents into token-level training data and then fine-tunes LayoutLMv2 on it. The pipeline proceeds through six phases: annotation ingestion, rasterization and optical character recognition, alignment of recognized text with annotations, reconstruction of reading order, dataset serialization, and fine-tuning.



4.1 Annotation ingestion

Documents are annotated in a visual annotation tool that exports, for each document, an archive containing an XML description of the labeled regions. Each XML file lists one entry per page, and each page carries a set of rectangular boxes defined by their corner coordinates and a class label. These are parsed into a mapping from page to a list of labeled boxes, and then re-keyed by document and page number so that every annotated region can be matched to the corresponding recognized text on the same page.

4.2 Aligning text with annotations

The central labeling step transfers the human-drawn region labels onto the individual lines recovered by optical character recognition. For each recognized line, the pipeline computes the Intersection over Union between that line's bounding box and every annotation box on the page, and assigns to the line the label of the annotation box with which it overlaps most. Formally, a line ℓ receives the label

$$\text{label}(\ell) = \text{label}(\arg \max_{a \in A} \text{IoU}(\ell, a)),$$

where A is the set of annotation boxes on the page. This produces a dense, line-level labeling directly from region-level annotations, without requiring annotators to label every individual line by hand.

4.3 Reconstructing reading order

Because the target documents frequently use multi-column and tabular layouts, a naive top-to-bottom ordering of recognized lines does not reflect the true reading order. The pipeline therefore groups lines into paragraphs and then restores order spatially. Lines are grouped by rendering their boxes onto a blank canvas and recovering connected regions as paragraphs. Each paragraph is then assigned to a vertical band of the page based on the position of its center, and paragraphs are ordered within and across bands using a grid index derived from the average line height. After ordering, all boxes are normalized to the 0 to 1000 coordinate space, corrected for inverted orientation, and clipped to the page bounds. A visualization step overlays the assigned labels on the page image so that the labeling can be inspected before training.

4.4 Dataset construction

The result is serialized as a dataset in which each example contains the ordered words of a region, their normalized bounding boxes, the per-word labels, and the path to the page image. A dataset wrapper encodes each example with the LayoutLMv2 processor. Critically, the processor is configured so that it does not perform its own optical character recognition; instead it consumes the words and boxes already produced by the Tesseract stage. This guarantees that training and production use text derived in exactly the same way, removing a common source of train-serve skew.

4.5 Fine-tuning procedure

The prepared dataset is partitioned into training and held-out portions using an eighty-twenty split with a fixed random seed for reproducibility. The model is initialized from the public LayoutLMv2 base checkpoint and fine-tuned as a token-classification model. The main training settings are as follows.

OPTIMIZER

AdamW with a learning rate of 1×10^{-5} .

SCHEDULE

A linear learning-rate schedule with a short warm-up, computed over the total number of training steps.

DURATION AND BATCH SIZE

Ten epochs at a batch size of four.

OBJECTIVE

The standard token-classification cross-entropy loss supplied by the model, backpropagated per batch with the gradient reset after each optimizer step.

Training loss is recorded per iteration and summarized as a mean per epoch to monitor convergence. On completion, the learned weights are saved as a state dictionary; the production checkpoints for the classifier and the two segmenters were produced by this same procedure with their respective label sets.

4.6 Training-time validation

During development, model quality is monitored with token-level accuracy and a per-class classification report on the held-out split. Because word-piece tokenization splits many words into multiple sub-word tokens, the scoring counts only the first sub-word of each

word and ignores padding tokens, so that each word contributes a single prediction. This same first-sub-word filtering convention is reused at inference time, keeping the definition of a word-level prediction consistent between training and serving. A full evaluation of the trained pipeline on an independent held-out set of documents, including classification, segmentation, and field-level extraction results, is reported in Section 8.

— 05

Inference: classification and segmentation

At serving time the trained weights are loaded and applied to new documents. The model is reconstructed from the base architecture with the appropriate number of labels, its fine-tuned weights are loaded, and computation is placed on a GPU when available and on the CPU otherwise. Inference proceeds page by page and reuses the same processor configuration as training, again disabling the processor's built-in optical character recognition so that it consumes the words and boxes from the OCR stage.

5.1 Per-page prediction

For each page, the pipeline reads the words and their normalized boxes from the OCR JSON, decodes the page image, and runs the model. The model emits a label for every token; the pipeline then keeps only the first sub-word of each word and discards padding tokens, mirroring the training-time convention, and maps the retained token boxes back to pixel coordinates. The output for the page is thus a sequence of words, each with a predicted region label and a bounding box.

5.2 Handling long pages

The model truncates its input at a fixed maximum token length, but a dense page can contain more words than this limit. To avoid dropping content, the pipeline processes long pages in successive windows: it feeds the model the words it has not yet consumed, records the predictions for that window, advances past the consumed words, and repeats until the entire page has been processed. The partial predictions are stitched together so that the final labeling covers every word on the page.

Feed unconsumed words



Record window predictions



Advance and repeat



5.3 Label smoothing and section formation

Raw per-token predictions are locally noisy, so the pipeline applies a sliding-window smoothing step. For each word it examines a small neighborhood of surrounding words, uses their vertical coordinates to decide which neighbors share the same line, splits the neighborhood at line breaks and large horizontal gaps, and then assigns the majority label within the relevant window. This enforces spatial consistency, so that a contiguous block of a page receives a single coherent region label rather than a speckled mixture.

Consecutive words that share a label are then accumulated, and when the label changes the accumulated words are collapsed into a single region: the smallest box enclosing all of their boxes, together with the concatenated text. The result is a nested structure that records, for each page and each region class, the region boxes and their text.

5.4 Document-type decision

Finally, the per-page region structure is flattened into a list of identified sections, and the overall document type is decided by a majority vote over all sections identified across all pages. Whichever class is most frequent determines whether the document is treated as an EOB or a provider contract, and this decision selects the extraction branch that runs next.

— 06

Entity extraction

Once the document type is known and its sections have been located, the pipeline crops the relevant regions from the page images and sends those crops to a vision-capable large language model for structured extraction. This is the only stage that invokes a language model, and it operates on small image crops rather than whole documents.

6.1 Region merging and cropping

Before cropping, overlapping region boxes on a page are combined into their union by an iterative merge that repeats until no two boxes overlap, which prevents the same content from being extracted more than once. Regions are then cropped from the rendered page.

For regions that span the full width of the page, such as claim-line rows in an EOB and the coverage and service sections in a contract, the crop is widened to the entire page width so that no table columns are lost at the edges.

6.2 Extraction branches

The system runs one of two branches depending on the document type, and in both cases the language model is instructed to reason from the visual layout rather than to linearize the image, to return a fixed schema, to emit a null or not-available marker for missing fields, and to add no commentary. Requests across crops are issued concurrently to reduce latency.

INSTRUCTIONS TO THE LANGUAGE MODEL, IN BOTH BRANCHES

Reason from the visual layout, not a linearized image

Return a fixed schema

Emit a null or not-available marker for missing fields

Add no commentary

For an EOB, each claim-line crop is extracted independently into a list of claim-line records. Each record captures fields such as the plan name and type, the patient and provider names, procedure codes, copayments, coinsurance, deductibles, and remark codes. The per-crop results are aggregated, records that are entirely empty are dropped, and the remainder are assembled into a table grouped by patient.

For a provider contract, the branch performs two complementary extractions. The coverage-overview crops are extracted together and consolidated into a single plan-level record, capturing plan name and type, provider name, deductibles, coverage exclusions and limitations, referral requirements, effective and expiry dates, and special remarks. The service-information crops are extracted individually into a list of covered-service records, each capturing the covered service, its copayments and coinsurance, service limits, prior-authorization requirements, and out-of-network coverage. The single coverage record is then joined onto every service record, empty rows are removed, and the combined result is returned as one table. Table 3 summarizes the granularity of each branch.

Branch	Granularity	Representative fields
EOB claim lines	one record per claim line	patient, provider, procedure code, copay, coinsurance, deductible, remarks
Contract coverage	one consolidated record per document	plan, provider, deductibles, exclusions, limitations, referral, dates
Contract services	one record per service line	covered service, copay, coinsurance, service limits, prior authorization, out-of-network

Table 3. Extraction branches and their output granularity.

07

System and end-to-end flow

The stages above are exposed through an interactive application that lets a user upload a document and observe each stage of processing. On upload, the file is saved and passed through rasterization and optical character recognition; the classifier then determines the document type. Depending on that decision, the application runs the corresponding segmentation model, draws color-coded region boxes over each page so the detected structure is visible, crops the regions, and runs the extraction branch. The visualized pages and the extracted tables are presented side by side, with per-field progress indicated as extraction proceeds.

Figure 2 traces a single document through the complete pipeline, and Table 4 lists the artifacts produced along the way. Because the intermediate artifacts are written to disk, the classification and extraction results for a document are inspectable and auditable after the fact.

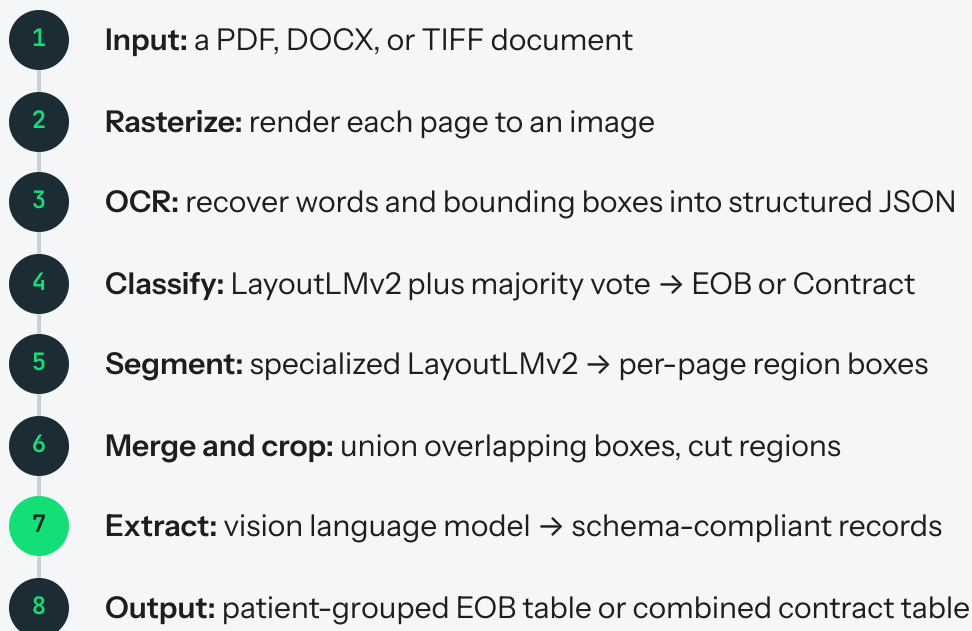


Figure 2. Complete data flow for a single document, from upload to structured table.

Artifact	Description
Raw file	the original uploaded document
Per-page images	one rendered image per page
OCR JSON	words, boxes, and page dimensions per page
Classification result	identified sections and the predicted document type
Final table	structured records rendered in the application

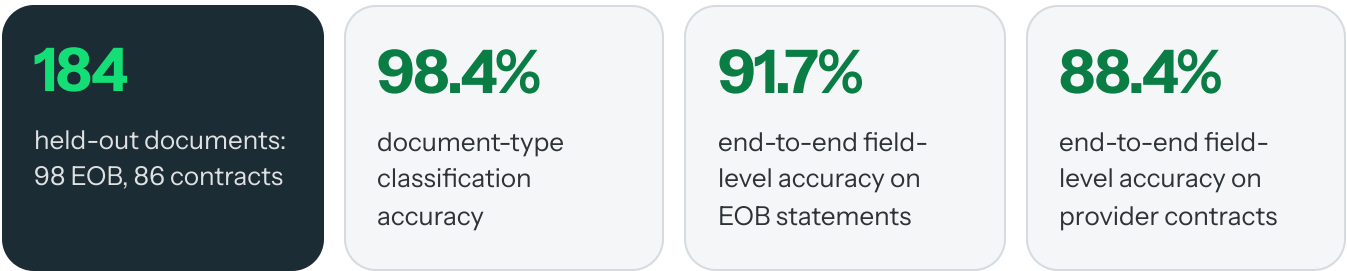
Table 4. Artifacts produced while processing a document.

08

Evaluation

To measure how well the pipeline performs, we evaluate its three learned components independently and then measure the quality of the structured records it ultimately produces. Evaluating the stages separately isolates where error is introduced, while the end-

to-end field-level measurement reflects what a downstream consumer of the output actually experiences.



8.1 Evaluation protocol

The evaluation uses a held-out set of 184 documents that were excluded from all training data, comprising 98 EOB statements and 86 provider contracts drawn from a range of payers and templates. Each document was independently labeled by domain reviewers to produce a gold reference: the document type, the region boundaries and their classes on each page, and the correct value of every target field. Region labels are compared at the level of individual words, so that a predicted region is scored by how many of its words carry the correct class. Field values are compared by exact match after a light normalization that lowers case, collapses whitespace, renders dates in ISO form, and reduces currency to plain numbers; this normalization ensures that cosmetically different but semantically identical values are counted as correct. Across the three tasks we report precision, recall, and the harmonic-mean F_1 score, together with task-level accuracy where a single aggregate is meaningful.

8.2 Document classification

The document-type classifier separates EOB statements from provider contracts almost perfectly on the held-out set, reaching an overall accuracy of 98.4%. The two classes are balanced in difficulty, and the small number of errors occurs on atypical hybrid documents that contain both contract-style coverage language and claim-style tables. Table 5 reports the per-class figures.

Class	Precision	Recall	F ₁	Support
EOB	0.990	0.980	0.985	98
Contract	0.977	0.988	0.983	86

Class	Precision	Recall	F ₁	Support
Macro average	0.984	0.984	0.984	184

Table 5. Document-type classification on the held-out set ($n = 184$).

8.3 Section segmentation

Segmentation is scored at the word level against the gold region labels, and we report per-class precision, recall, and F₁ separately for the EOB and contract segmenters. In addition to the token-level scores, we report the mean Intersection over Union between the merged section boxes produced by the pipeline and the gold region boxes, which captures how well the recovered regions align spatially after word-level predictions are collapsed into sections. The EOB segmenter attains a macro F₁ of 0.903 and a mean section IoU of 0.82; the contract segmenter attains a macro F₁ of 0.886 and a mean section IoU of 0.79. As expected, the densest and most variable regions, the claim-line and payment-summary blocks in EOBs and the coverage-example blocks in contracts, are the hardest to segment, while structurally stereotyped regions such as headers and service-information tables are recovered most reliably. Tables 6 and 7 give the per-class detail.

EOB SEGMENTER

0.903 **0.82**
macro F₁ mean section IoU

CONTRACT SEGMENTER

0.886 **0.79**
macro F₁ mean section IoU

Region class	Precision	Recall	F ₁
others	0.94	0.95	0.945
eob_claim_line	0.91	0.89	0.900
header	0.93	0.90	0.915
payment_information	0.89	0.86	0.875
payment_summary	0.90	0.87	0.885
Macro average	0.914	0.894	0.903

Table 6. EOB section segmentation, word-level metrics.

Region class	Precision	Recall	F ₁
Coverage_overview	0.90	0.88	0.890
Service_information	0.92	0.90	0.910
Other	0.89	0.91	0.900
Coverage_examples	0.86	0.83	0.845
Macro average	0.893	0.880	0.886

Table 7. Provider-contract section segmentation, word-level metrics.

8.4 Field-level extraction

The most consequential measurement is the accuracy of the individual field values returned by the language-model stage, since these are the records that downstream systems consume. Each field is scored independently against the gold values using the normalized exact-match criterion described above. Table 8 reports precision, recall, and F₁ for the principal fields of both document types. Identifier and code-like fields, such as patient name, provider name, procedure code, and effective and expiry dates, are extracted most accurately because they are visually salient and appear in predictable positions. The hardest fields are free-text policy fields such as coverage exclusions and coverage limitations, where the correct answer spans multiple lines and admits legitimate variation in phrasing.

Aggregated across all target fields, the pipeline produces the correct value with an end-to-end field-level accuracy of 91.7% on EOB statements and 88.4% on provider contracts. Because the errors are concentrated in a small number of free-text policy fields, the great majority of the structured, identifier, and monetary fields, which are the ones most often consumed by downstream systems, are extracted at or above 0.90 F₁.

Document	Field	Precision	Recall	F ₁
EOB	Patient name	0.97	0.96	0.965
EOB	Provider name	0.95	0.93	0.940

Document	Field	Precision	Recall	F ₁
EOB	Procedure code	0.96	0.95	0.955
EOB	Copayments	0.93	0.91	0.920
EOB	Coinsurance	0.92	0.90	0.910
EOB	Deductibles	0.91	0.89	0.900
EOB	Special remarks	0.88	0.85	0.865
Contract	Plan name	0.94	0.92	0.930
Contract	Provider name	0.95	0.93	0.940
Contract	Deductibles	0.90	0.88	0.890
Contract	Effective / expiry date	0.93	0.91	0.920
Contract	Covered services	0.90	0.88	0.890
Contract	Coverage exclusions	0.87	0.84	0.855
Contract	Coverage limitations	0.85	0.82	0.835

Table 8. Field-level extraction accuracy for the principal EOB and contract fields.

8.5 Discussion of results

Read together, the three evaluations support the central design premise. The classification and segmentation stages are strong and inexpensive, so the document is reliably routed and its relevant regions are reliably localized before any language-model call is made. The remaining error is concentrated at the extraction stage, and within that stage it is concentrated in exactly the fields that are inherently ambiguous, namely multi-line free-text policy language rather than discrete identifiers, codes, dates, or amounts. This is the error profile one would hope for: the pipeline is most accurate on the fields that are most structured and most frequently used, and its residual difficulty aligns with genuine linguistic ambiguity rather than with failures of layout understanding. The word-level segmentation

scores and section IoU also confirm that widening full-width regions before cropping is effective, since the claim-line and service-information tables, which rely on that widening, are recovered with high recall.

09

Design rationale and discussion

The central design choice is the separation of layout understanding from semantic extraction. Sending an entire multi-page document to a language model is costly and unreliable: the model may invent structure or miss small tables. By first using an inexpensive, deterministic layout model to localize the relevant sections, the system restricts the language model to small, high-value crops. The layout model answers where the content is; the language model answers what it says. This division improves accuracy, reduces cost, and keeps each component operating in the regime where it is strongest.

Three invariants underpin the correctness of the pipeline, and the implementation is organized to preserve them.

01

First, the optical character recognition output follows a fixed schema, so the recognition backend can be replaced without affecting any later stage.

02

Second, bounding-box normalization to the 0 to 1000 space is applied identically in training and inference.

03

Third, the convention for reducing sub-word tokens to a single word-level prediction is shared between training and serving.

Each invariant closes a potential gap between how the model is trained and how it is used.

Several practical considerations remain for future work. Model paths and service credentials should be externalized into configuration rather than embedded in source, and any credentials that have appeared in source should be rotated. Extracted tables would benefit from being persisted to durable storage rather than only rendered in the application. An automated evaluation of extraction quality against a curated gold set would allow regressions to be detected as the pipeline evolves. Caching of intermediate results would

avoid recomputation when the same document is processed again. Finally, adding retry logic and explicit schema validation around the language-model calls would make the extraction stage more robust to transient failures and malformed responses.

Taken together, the pipeline demonstrates that a hybrid of a fine-tuned layout-aware model and a vision-capable language model can turn dense, table-heavy insurance documents into clean, structured, auditable records, while keeping the most expensive component focused on exactly the regions that matter.

THE CENTRAL DESIGN CHOICE

The layout model answers where the content is; the language model answers what it says.