
Recovering structure from PDF

Key takeaways

1. A PDF paints glyphs, lines, and images at coordinates. Words, rows, columns, spans, and reading order are not in the file; any tool that reports them inferred them.
2. Different tables can leave the same marks on a page. Base each structural decision on a mark that settles it, and record unknown or conflict when none does.
3. Keep the source bytes, each reader's candidate, and a page address for every cell apart. Markdown cannot express a span or an unknown, so it is a view, never the record.

Common misunderstandings of recovering tables from PDF

Which claim would you defend right now? Choose before the evidence.

“A PDF stores a document’s text and tables; extraction mostly means parsing them out.”

“If the extracted Markdown table looks right, the recovery is right.”

“When two extraction tools agree on a cell, the cell is probably correct.”

Keep your choice. Look for the mechanism or counterexample that would make it fail.

Instapoll

Instapoll · Review · Oct 20 A

A LaTeX paper's PDF shows a table whose header "Latency (ms)" spans two columns. What does the PDF file itself record about that header?

- A. A table cell holding the text, with a column span of two
- B. The text, tagged as the header of the two columns beneath it
- C. Glyphs painted at page coordinates, and a short line drawn under them
- D. The LaTeX command that produced the span, kept for later editing

A PDF page is a drawing program

pdflatex's own content stream for the table at right, numbers rounded

Text at coordinates, and lines

```
BT /F32 9.96 Tf 231.4 395.2 Td
% one operator paints a whole row
[(System)-6005(p50)-2714(p99)] TJ
ET
% a 60-point line: the partial rule
q 1 0 0 1 321.6 406.5 cm
0.299 w 0 0 m 60.0 0 l S Q
BT /F32 9.96 Tf 231.4 378.3 Td
% kerning numbers, not spaces
[(Lo)-28(c)28(k-free)-2208(ring)] TJ
% down a line, before the numbers
0 -11.96 Td [(queue)] TJ
93.92 11.96 Td [(1.2)-2992(8.4)] TJ
ET
```

Table 1: Queue latency under load.

		Latency (ms)	
System		p50	p99
Lock-free	ring	1.2	8.4
queue			
Mutex		1.9	31.0

Nothing in the file names a row, a column, a cell, or a span. Tagged PDF can add a structure tree; default pdflatex output has none.

Structure is inferred, one rung at a time

An extractor climbs from painted glyphs to a logical table, guessing at each rung

1 **Glyph → character**
the font's Unicode map; a ligature or math glyph may map to nothing

2 **Characters → words**
a space is a gap wider than some threshold

3 **Words → lines**
shared baselines, broken by superscripts and subscripts

4 **Lines → reading order**
stream order is the producer's, not the reader's

5 **Lines → cells, rows, columns**
alignment, plus rules when there are any

6 **Pages → one logical table**
continuations and repeated headers

An error low on the ladder surfaces higher up as a wrong table.

Three tools, three plausible tables

The same specimen through two extractors and one OCR engine

tool	what it returned	where it went wrong
pdfplumber, rule-based	no table	booktabs draws no vertical rules to find
pdfplumber, whitespace-based	a 7×3 grid	“Latency” and “(ms)” in separate columns; “queue” a row of its own; two blank rows where the rules were
PyMuPDF	no table; text in stream order	“... ring, queue, 1.2, 8.4 ...”: the wrapped word arrives before its row’s numbers
Tesseract OCR on the image	“System po0 p99”	“p50” read as “po0”; the plain-text output drops the confidence it computed

Each answer is plausible, each is wrong somewhere different, and none says where it is unsure.

The marks allow more than one table

Rendering is many-to-one, so recovery is an inverse problem

Reading A: what the author wrote

3 columns

row 0	·	Latency (ms), span 2
row 1	System	p50 p99
row 2	Lock-free ring queue	
		1.2 8.4
row 3	Mutex	1.9 31.0

Reading B: also consistent with the marks

4 columns

row 0	·	·	Latency	(ms)
row 1	System	·	p50	p99
row 2	Lock-free	ring	1.2	8.4
row 3	queue	·	·	·
row 4	Mutex	·	1.9	31.0

Many logical tables paint the same page, and the page alone does not always say which one it came from. A human looking at the image can be fooled too: the justified line makes “ring” look like a column.

Find the mark that settles each decision

Coordinates in points, read from the specimen

decision	the mark that settles it	in the specimen
“Latency (ms)” spans two columns	its extent covers both columns, and a partial rule sits under it	text 322.6–380.6 runs from p50’s left edge to p99’s right; rule 321.6–381.6
“ring” is not a column	no other row has marks there, and the line ends at the cell’s edge	“ring” ends at 310.6, exactly 1.1 in past the cell’s start; x 293–311 is empty elsewhere
“queue” continues its cell	geometry cannot say: the line pitch equals the row pitch	12.0 pt below “Lock-free”; “Mutex” is 11.9 below “queue”

When no mark settles a decision, record unknown or conflict. A confident guess is the error nobody finds later.

The text layer is a claim, too

Corroboration needs a second, independent path to the page

No text layer

A scanned or rasterized page is one image. OCR proposes text, with errors like “po0” for “p50”.

A wrong text layer

A font without a Unicode map extracts as garbage; a ligature or a minus sign (U+2212) can come out as something else.

Someone else's guess

The invisible OCR layer a scanner adds looks like text, but it is an earlier engine's reading.

Two tools reading one text layer are one observation, not two. Check the text against the rendered pixels.

A page break does not end a table

Joining pages is a decision with evidence, not concatenation

Pages 4 and 5, concatenated

```
System      p50    p99
Lock-free   1.2    8.4
- 4 -
Table 3 (continued)
System      p50    p99
Mutex       1.9    31.0
```

```
# a page number as a row;
# the header twice, as data
```

One logical table

```
header  System | p50 | p99
        evidence: page 4 row 0,
                        page 5 row 0
rows     Lock-free (page 4)
          Mutex     (page 5)
join     4 → 5: the header
          repeats; captions match
```

A repeated header is more evidence for the same header cells, not new rows. An invented join is as wrong as a missed one.

One wrong table, two confident agents

A vision model turned a chart into rows; two agents then treated the rows as ground truth

stage	what happened	why it looked trustworthy
vision model	estimated overlapping chart series	returned precise rows and numbers
reviewer agent	argued that the gap opens at high load	cited the generated table
paper-QA agent	claimed a 14.7% throughput advantage	independently reused the same artifact
human check	compared claims with the source image	saw overlap rather than separation

Agreement downstream was correlated error: both agents read one generated file, and nothing in it pointed back to the figure.

Keep the page apart from your guesses

Four kinds of data, four lifetimes

layer	example	mutation rule
source	the PDF's bytes, addressed by SHA-256	immutable
candidate	one reader's table, with its method and version	append-only; a rerun adds, never overwrites
decision	the reading accepted, or unknown/conflict, with its evidence	a new decision supersedes; nothing is erased
view	Markdown or CSV for people	disposable; rebuilt from the records

A better extractor next month is a new candidate, not a replacement. If its run overwrites the last one, you can no longer tell whether it improved anything.

Every cell needs an address in the source

An address names the bytes, the page, the box, and the coordinate system

One cell of Reading A

```
{  
  "source_sha256": "b414f7d8...",  
  "row": 0, "col": 1, "col_span": 2,  
  "text": "Latency (ms)", "status": "known",  
  "evidence": [{  
    "page": 1,  
    "bbox": [322.6, 372.0, 380.6, 382.0],  
    "origin": "top-left", "units": "pt"}],  
  "method": "text layer + rule geometry"  
}
```

The stream put this word's baseline at $y = 412.0$, measured up from the bottom; pdfplumber reports its top edge at 372.0, measured down from the top. "Table 3" is only a locator: a revised paper rennumbers it.

Empty, unknown, and conflict are distinct

Four states a recovered cell can be in

```
{"text": "", "status": "known"}  
# the cell is blank on the page
```

```
{"text": null, "status": "unknown"}  
# the marks are illegible, or no mark settles the reading
```

```
{"candidates": ["p50", "po0"], "status": "conflict"}  
# two independent readers disagree
```

```
# no record at all: nothing has examined this region yet
```

A CSV writer turns all four into an empty field. Define each state in the schema, and let no default erase the difference.

Markdown cannot say what the table said

A right table and a wrong one render equally tidy

Reading A, as Markdown

```
| System | Latency (ms) p50 | Latency (ms) p99 |
|---|---|---|
| Lock-free ring queue | 1.2 | 8.4 |
| Mutex | 1.9 | 31.0 |
```

the span is copied into two headers

pdfplumber's grid, as Markdown

```
| | Latency | (ms) |
|---|---|---|
| System | p50 | p99 |
| Lock-free ring | 1.2 | 8.4 |
| queue | | |
| Mutex | 1.9 | 31.0 |
```

A pipe table has one header row, no spans, no unknown, and no place for evidence. Discarded structure and provenance cannot be reconstructed from it later, so it can be a view but never the record.

Load–modify–save is a migration

Even for one field: a typed schema filters out what it does not declare

Typed rewrite drops what it cannot represent

```
@dataclass
class Cell:
    row: int
    col: int
    text: str

raw = json.loads(path.read_text())
cell = Cell(**{k: raw[k]
               for k in ("row", "col", "text")})
cell.text = normalize(cell.text)
path.write_text(json.dumps(asdict(cell)))

# status, col_span, evidence vanish
```

Preserving edit

```
raw = json.loads(path.read_text())
before = deepcopy(raw)
raw["text"] = normalize(raw["text"])
assert_unknown_fields_preserved(
    before, raw, changed={"text"}
)
atomic_write(path, json.dumps(raw))

# or a schema-aware migration
```

A text normalizer that never heard of evidence erased it. Test every round trip with a field the stage does not know; Protocol Buffers keeps unknown fields for exactly this reason.

Key Takeaways

1. A PDF page is a drawing program: it paints glyphs, lines, and images at coordinates. Words, rows, columns, spans, and reading order are not stored, so every extractor infers them, and an error low in that chain surfaces as a wrong table.
2. Different tables can leave the same marks. Base each structural decision on a mark that settles it — a partial rule, an aligned extent, a repeated header — and record unknown or conflict when no mark does.
3. Keep source bytes, candidates, decisions, and views apart, and give every cell an address in the source. Markdown cannot say span, unknown, or where, and a stage that rewrites a record it only partly understands drops the rest.

Recover a partly readable table

10 min

Words and rules (width in pt), top to bottom

mark	x0	x1	top
rule 0.8	86.9	223.6	72.4
Mops/s	173.8	206.3	75.9
rule 0.3	161.5	218.7	89.4
Workload	92.9	134.7	92.7
1 thr	162.5	184.1	92.7
8 thr	196.0	217.7	92.7
rule 0.5	86.9	223.6	106.3
Read-mostly	92.9	147.6	109.6
4.1	171.4	184.1	109.6
29.8	199.9	217.7	109.6
Write-heavy	92.9	145.8	121.6
3.9	171.4	184.1	121.6
Mixed 50/50	92.9	150.5	133.5
3.2	171.4	184.1	133.5
11.0	200.0	217.7	133.5
scan	92.9	111.8	145.5
rule 0.8	86.9	223.6	159.2

- A text-layer extractor reported these marks for one table, in points from the page's top-left. There are no vertical rules.
- Alone, 3 min: write the logical table — its dimensions, any spans, and each cell's status: known, empty, unknown, or conflict.
- In threes, 5 min: cite the coordinates that settle each structural decision. A decision no mark settles becomes unknown or conflict.
- Deliverable: the table with statuses, one citation per decision, and the one decision you could not settle.