

Separating Reading from Verifying in Automated Invoice Checking

Aliyan Faisal

InvoiceInspect

Companion to Test Report 1 (Baseline)

Abstract

Invoices are the last control before a payment is released, yet they are usually checked by eye. Automated checking is attractive but difficult to trust: general-purpose language models can mis-multiply, call a correct invoice wrong, and cannot show where a problem lies. We describe InvoiceInspect, an invoice checker built on the principle that *reading* a document, *verifying* its arithmetic and *explaining* the result are separate jobs that need different tools. Reading uses deterministic extraction from the PDF text layer (with OCR for scans and a vision model as a second, independent reading); verifying uses an exact rule engine; explanation comes last and cannot create or hide an error. Results are three-state (pass, fail, could not verify) and every finding carries its evidence. We evaluate the system with an evaluation harness on two corpora of fictitious invoices with known answers (30 independently written, 40 from the project's generator; 52 planted mistakes in total). Given correct input values, the rule engine found all 52 planted mistakes with no false alarms. With the real extractor, field accuracy was 99.9% on the generator corpus but 84.5% on the independent corpus, where 25 of 32 mistakes were found and 30 false alarms were raised, mostly "not found" warnings. We conclude that verification is reliable for the covered rules and that reading, which depends heavily on page layout, is where the remaining errors are.

Keywords: invoice verification, document information extraction, deterministic validation, OCR, evaluation harness, human-in-the-loop

1. Introduction

Accounts-payable teams and small businesses pay vendor invoices after a quick visual review. The errors that matter are small and arithmetic: a line total that is not quantity times unit price, a subtotal that differs from the sum of the lines, tax computed on the wrong base, a grand total that is off by a few units, a charge printed twice, a missing tax identifier. Such errors are easy to detect once the printed numbers have been read correctly, and easy to miss in a long document.

Asking a general AI assistant whether an invoice is correct yields a prediction rather than a check. For a payment decision an unexplained, non-reproducible answer is not usable. This motivates the research question:

RQ. *Can software read a vendor invoice well enough, and check it strictly enough, to tell a person which printed numbers disagree, without asking them to trust an AI's opinion?*

We split the question into two measurable parts: how accurately the system **reads** an invoice (RQ1) and how correctly it **verifies** what it has read (RQ2). The contributions of this report are:

- a design that separates reading, verifying and explaining, with three-state results and per-finding evidence (Section 3);
- an evaluation harness with an *oracle* extractor that isolates verification from reading, and two published corpora with answer keys (Section 4);
- a baseline measurement that includes the weak results and a ranked list of the causes (Sections 5 and 6).

2. Background and Related Approaches

We distinguish three common ways of checking invoices automatically and the limits of each for this task.

Manual review. Reliable when the reviewer is careful, but slow, inconsistent and not auditable; arithmetic slips are the typical miss.

Template and rule-based extraction with OCR. Deterministic and explainable, but a template must exist for each vendor layout, so coverage degrades on layouts that were not anticipated. Optical character recognition (for example Tesseract [1]) converts scanned pages to text but introduces its own read errors.

Language-model-only checking. Handles unfamiliar layouts, but arithmetic is performed by a model that is not guaranteed to be exact, outputs can vary between runs, and a claim of an error may not be traceable to a place on the page.

InvoiceInspect combines the strengths of these approaches: deterministic reading and exact rules as the basis, with models restricted to roles where a wrong output cannot silently change a verified result. This report does not claim to outperform any particular product; it measures one system against known answers.

3. Methodology: System Design

The design follows one principle: *reading, verifying and explaining are different jobs and use different tools* (Table 1, Figure 1).

Table 1. Division of labour between components.

Job	Tool	Rationale
Reading: PDF to fields, line items, totals	Deterministic extraction from the text layer; Tesseract OCR for pages without text; a vision model as an independent second reading	Layouts vary, so reading is the hard part; two independent readings can be cross-checked
Verifying: do the numbers agree?	Plain code: line math, subtotal, tax, discounts, grand total, required fields	Arithmetic must be exact, reproducible and explainable
Explaining and second-guessing	AI wording of findings; a "second look" that can only downgrade a finding it can prove is a misreading	Aids understanding; cannot create or hide an error on its own



Figure 1. Processing pipeline. The shaded stage is the deterministic rule engine.

3.1 Design rules

- **Three-state results.** Every check yields pass, fail or *could not verify*. A rule fails only on a high-confidence input; a missing or unclear value is reported as not found, never as invalid.
- **Evidence with every finding.** A finding states the expected value, the printed value, their difference and the page of origin; if evidence cannot be located the system says so.
- **Cross-validation.** If the vision reading and the text layer disagree, the value is marked low confidence. A value seen only by the vision model is kept only if its quoted text exists in the text layer.
- **Human confirmation.** Values read below 0.5 confidence are listed for the user to confirm or correct; a correction becomes confidence 1.0 and the rules re-run.
- **Privacy by default.** Free checks are processed in memory and the uploaded file is not stored; storage is opt-in on paid plans.

3.2 Stages

Inspect parses the PDF, classifies pages as text, scanned or mixed, and caps decoded data so a large scan cannot exhaust memory. **Read** applies money-token detection, document-level voting on number and date locale, table and column detection and totals-block detection; scanned pages are OCR'd and OCR confidence caps field confidence. **Verify** runs the rule engine; one printed mistake yields one finding. **Explain** words each finding, and any explanation containing a number absent from its input is discarded. Every extracted field carries a value, a confidence, the method that produced it and its evidence.

4. Experimental Setup

Harness. The evaluation harness runs an extractor over invoices with known answers and scores (i) field accuracy, the fraction of fields and line-item cells read correctly; (ii) planted mistakes found, where a mistake counts as found when the tool reports it; and (iii) false alarms, any finding that matches no planted mistake. Four extractors are compared: *rules* (the real extractor), *oracle* (handed the correct values, isolating the rule engine), *rules-ocr* (each invoice is scanned, then OCR'd) and *broken* (a deliberately faulty control that must score badly).

Corpora. All invoices are fictitious; the PDFs and answer keys are public (Table 2).

Table 2. Evaluation corpora.

Corpus	N	Composition	Planted mistakes
A: independent	30	Written separately from the extractor with different PDF software: 10 page structures, 8 currencies, 4 languages, tax types VAT, GST, sales tax, MwSt., TVA, IVA; 15 correct, 15 with one to three random mistakes	32
B: generator	40	The project's own generator, 10 visual designs, half correct and half with one mistake each (the easier set)	20

Procedure. Each extractor is run once per corpus in the same way, and the harness writes raw per-invoice results. The extractor was not modified for this baseline, so later reports can be compared with it directly.

5. Results

5.1 Verification (RQ2)

With correct values supplied (*oracle*), the rule engine found 32 of 32 planted mistakes on corpus A and 20 of 20 on corpus B, with zero false alarms on either corpus. The verification rules are therefore correct for the mistake types planted. The broken control scored poorly, confirming the harness detects bad reading.

5.2 Reading (RQ1)

Table 3. Baseline results.

Measure (rules extractor)	Corpus A (independent)	Corpus B (generator)
Field accuracy	84.5%	99.9%
Planted mistakes found	25 of 32	20 of 20
False alarms	30 (26 "not found", 4 arithmetic)	0
Correct invoices with no findings	3 of 15	20 of 20
Oracle: mistakes found / false alarms	32 of 32 / 0	20 of 20 / 0

The 15-point gap between the corpora shows that the extractor performs well on layouts it was developed beside and much worse on independent ones. Reading quality depends heavily on page structure (Figure 2): accuracy ranges from 99.5% on multi-page invoices to 37.0% on quantity-first layouts, where line items are not extracted.

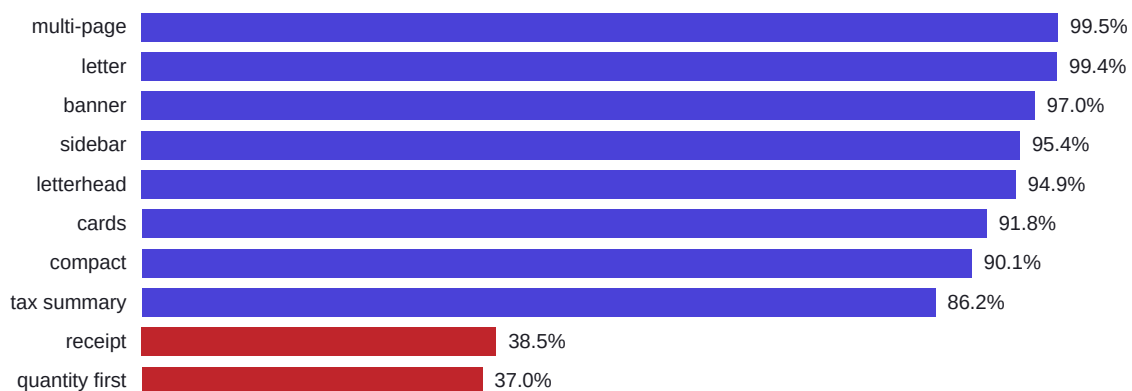


Figure 2. Field accuracy by page structure, corpus A (rules extractor).

Table 4. Planted mistakes found by type, corpus A.

Planted mistake type	Planted	Found
Subtotal	7	7
Missing tax ID	2	2
Tax	5	4
Grand total	3	2
Duplicate charge	6	4
Line total	9	6

Six of the seven missed mistakes occur on the receipt and quantity-first layouts. Of the 30 false alarms, 26 were "not found" warnings, chiefly tax IDs (read on 16 of 30 invoices) and invoice numbers (read on 22 of 30). The OCR path (rules-ocr) reached 66.5% field accuracy, found 22 of 32 mistakes and raised 55 false alarms.

5.3 Identified causes

Table 5. Causes of reading errors, ranked by effect on the results.

Cause	Evidence (corpus A)
Non-VAT tax IDs (ABN, EIN, GSTIN, NTN, UID, CIF) not recognised	Tax ID read on 16 of 30; 14 false "tax ID not detected" warnings
Line items of receipt and quantity-first layouts not extracted	Six of seven missed mistakes; layouts score 38.5% and 37.0%
Some invoice-number formats and labels missed	Read on 22 of 30; 7 false warnings
Customer name taken from a nearby label (e.g. "Date")	Read on 22 of 30
Currency inferred from "\$" when invoice says AUD or CAD	Read on 25 of 30; all 5 errors are AUD or CAD
A line or total row mistaken for shipping	7 invoices with a spurious shipping amount

6. Discussion

Separating the jobs shows where errors come from. Because the rules are exact on correct input, every remaining failure traces to the reading step. This gives a list of specific problems to fix (Table 5), which a single model-based checker cannot offer.

Self-built test data overstates accuracy. The extractor scores 99.9% on the corpus produced by the project's own generator and 84.5% on independent invoices. Measurement on generator output alone would have hidden the layout dependence, which is why the independent corpus exists.

A warning costs less than a wrong answer. Most false alarms are "not found" warnings, which cost a reader a glance; a confident wrong "error" would cost trust. The three-state design and the confirmation step are intended to keep misreadings in the first category, although the harness does not measure that effect.

7. Limitations and Threats to Validity

Sample size and realism. Both corpora are synthetic and small (30 and 40 invoices). The results are a comparison and a list of things to fix, not an estimate of real-world accuracy.

Tuning bias. Once the extractor is tuned on corpus A, its score there will be optimistic; the next evaluation must use a fresh, unseen set.

Author bias. The independent corpus was written by the same person who built the tool, though with different PDF software and without reference to the extractor.

Unmeasured components. The vision cross-check, AI explanations and second look are tested with simulated model responses but are not scored by the harness against a real model; no accuracy claim is made for them. The effect of confidence limits and the confirmation step on OCR false alarms is likewise unmeasured.

Scope. Inputs are PDFs with a text layer or a clean scan. Handwriting, skewed scans and languages beyond the corpus are not covered. The tool checks internal consistency only; it does not establish whether a price was agreed, goods were received or a tax rate is legally correct, and it is not accounting or legal advice.

8. Conclusion

Treating invoice checking as two problems, exact verification and layout-dependent reading, yields a system whose verification can be trusted and whose weaknesses can be measured. On the baseline, verification found every planted mistake without false alarms, while reading accuracy fell from 99.9% to 84.5% when invoice layouts were independent of the extractor. The identified causes form a ranked list for improvement, and later reports score the same invoices in the same way so that progress is measured rather than asserted.

Appendix A. Availability of Data

All test invoices, answer keys and raw result files are published on the Harness Reports page of the InvoiceInspect website, so every figure in this report can be checked against them. The four extractors (rules, oracle, rules-ocr and broken) and the way results are scored are described in the Evaluation Harness Guide.

References

- [1] R. Smith. An Overview of the Tesseract OCR Engine. Proceedings of the Ninth International Conference on Document Analysis and Recognition (ICDAR), 2007.
- [2] A. Faisal. InvoiceInspect: Evaluation Harness Guide. Published on the Harness Reports page.
- [3] A. Faisal. InvoiceInspect: Test Report 1 (Baseline). Published on the Harness Reports page.