Prompt-Optimized OCR for Production: GEPA Shows OCR is Steerable for Business Document Pipelines

Greg Miller (greg@intrinsic-labs.ai)
Jon Slemp (jon@intrinsic-labs.ai)
Intrinsic Labs Research Team
https://intrinsic-labs.ai/

October 2025

Abstract

This whitepaper evaluates prompt steerability in optical character recognition (OCR) pipelines and demonstrates that modern language models are now capable of improving these systems autonomously through reflection. We find that OCR *is* meaningfully steerable in production-like settings—particularly for business workflows built on recurrent, family-similar documents such as invoices, forms, and utility bills. Using the Generative Error–Prompt Alignment (GEPA) optimizer within a forked DSPy framework on the Omni benchmark, we observed consistent gains of 3.3–3.6 percentage points across Gemini models and a Pareto frontier approaching 97 percent on structured extraction subsets. These improvements concentrate in the language stage (Markdown→JSON), where schema discipline, read-order, and reconciliation policies matter most; perception-stage errors still define the upper bound on poor scans. More broadly, this study shows that frontier models are now *intelligent enough to analyze their own failures and iteratively optimize their prompts*—a shift that makes self-improving, reflection-driven pipelines not just possible, but practical for production AI systems.

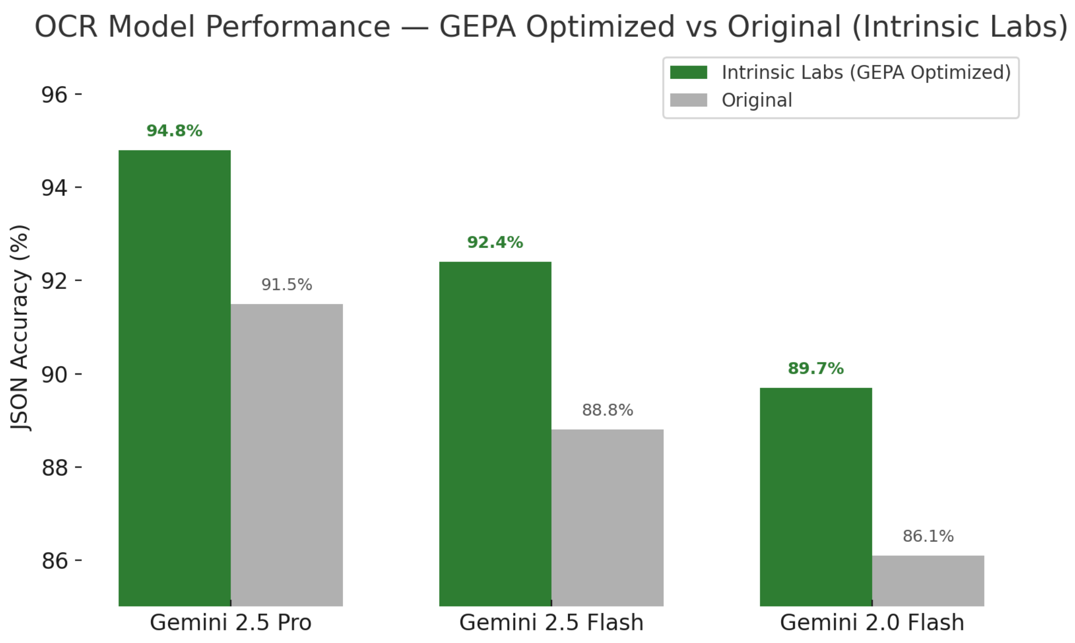

Figure 1: GEPA improved JSON extraction accuracy from baseline to optimized performance. Source: Studio-Intrinsic / benchmarking-ocr-gepa.

1 Introduction

Intrinsic Labs builds production AI systems for businesses, with a focus on reliability, cost, and operational efficiency. We believe modern AI pipelines should *self-improve* as models advance—primarily through prompt optimization and reflective feedback loops—reserving fine-tuning for cases where it is demonstrably necessary.

Document pipelines suffer from the compounding error problem present in non-determenistic systems: five stages at 90% each yield $(0.9)^5 \approx 0.59$ end-to-end reliability, making them effectively useless. Our question was practical: can an automatic prompt optimization layer materially shift outcomes *without* retraining, and under what conditions?

Key takeaways

- OCR is **steerable in practice** for business pipelines with **family-similar** inputs, producing consistent accuracy gains and more stable outputs.
- Gains concentrate in the language stage (Markdown→JSON), where policy, structure, and consistency dominate.
- Perception-stage limits persist on poor scans; however, for typical enterprise documents, prompt optimization yields production-relevant improvements.

2 Motivation

2.1 Why revisit OCR steerability now

Older models made prompt optimization brittle. Frontier models can reflect and follow procedural specifications with much higher fidelity, enabling reliable control over extraction logic, schema adherence, and hallucination discipline. This shift makes prompt optimization a first-class lever for production teams.

2.2 Business relevance

Most enterprise workflows process *recurring* document families: monthly utility bills, tax forms, carrier invoices, statements. In these contexts, marginal accuracy gains are amplified by volume, and consistency reduces human exception handling. A steerable, self-improving extraction layer therefore compounds value over time.

3 Benchmark and Pipeline

3.1 Task and dataset

We evaluated on the Omni OCR Benchmark, using 1,000 real-world documents (invoices, receipts, utility bills, forms) with ground-truth JSON. We held out 20% for validation.

3.2 Two-stage architecture

1. **Image → Markdown** (vision): Gemini 2.0 Flash extracts textual content and coarse structure.

2. **Markdown → JSON** (language): An LLM maps text to a strict schema with field-level accuracy guarantees.

This mirrors typical production flows and isolates where prompting has leverage (language stage) versus where perception quality dominates (vision stage).

3.3 Metric

We used Omni's TypeScript evaluator:

$$\text{Accuracy} = 1 - \frac{\text{additions} + \text{deletions} + \text{modifications}}{\text{total fields}}.$$

It symmetrically penalizes missing, spurious, and incorrect fields and reflects business constraints (e.g., totals must reconcile, dates must be valid).

3.4 Baselines

- **Gemini 2.0 Flash:** 86.1%
- **Gemini 2.5 Flash:** 88.8%
- **Gemini 2.5 Pro:** 91.5%

4 Optimization Framework

4.1 GEPA with DSPy: reflective prompt evolution

We began with **GEPA (Genetic–Pareto)**, a framework for optimizing *textual components* of any system—such as prompts, code, or configuration strings—through reflective text evolution. GEPA uses large language models (LLMs) not merely as solvers but as *reflectors*: they analyze execution traces, identify weaknesses, and propose textual improvements that move the system closer to its objective.

Unlike reinforcement learning or brute-force grid search, GEPA operates through a lightweight evolutionary process:

1. Run the current candidate (prompt or program) on a small evaluation set and collect structured traces (outputs, metrics, and errors).
2. Pass these traces to a *reflection model* that explains what went wrong and proposes specific textual edits.
3. Evaluate the edited candidates and update a *Pareto frontier* balancing task accuracy with prompt simplicity.
4. Repeat until convergence or budget exhaustion.

This approach effectively "pre-computes" reasoning and policy into the prompt, yielding a configuration that generalizes across future documents. The official implementation of GEPA integrates directly with `DSPy` through the `dspy.GEPA` API, making it easy to treat prompts or entire DSPy programs as optimizable components.

4.2 Minimal fork for system-prompt control and image context

We used `dspy.GEPA` as the base optimizer and created a minimal fork of DSPy to support two extensions required for document-based tasks:

- **System-prompt overrides:** allowed per-run injection of custom system instructions so GEPA could test variations in framing, tone, and structural guidance (coverage, schema adherence, read-order, and hallucination policy) without modifying the underlying program code.
- **Image-URL visibility:** exposed document image URLs to the reflection model (*GPT-5*) so it could interpret the same visual context that the Gemini executor used, enabling more targeted reflections about layout, coverage gaps, or mis-parsed regions.

4.3 Optimization flow

Our optimization loop proceeded as follows:

1. **Seed and evaluate.** Start from an initial prompt governing the OCR→Markdown and Markdown→JSON stages. Execute on a curated validation slice using **Gemini 2.0 Flash** as the task model. Capture JSON diff metrics and metadata about missing fields or mis-ordered content.
2. **Reflect.** Provide execution traces and image URLs to the **GPT-5** reflection model. GPT-5 analyzed failure clusters (e.g., omitted footers, totals misalignment, hallucinated values) and proposed minimal, testable edits.
3. **Select.** Evaluate new candidates, track per-field accuracy, and update the Pareto frontier over accuracy and prompt complexity. Shorter, more procedural edits were favored.
4. **Iterate.** Continue until convergence or diminishing returns, exporting the top-performing candidate as the optimized prompt.

4.4 Curating the optimization slice

To keep the optimization budget efficient and signal dense, we optimized on a **difficult but solvable** subset of the Omni benchmark:

- Documents with baseline accuracy between 70–90%, where the model had partial understanding but systematic extraction errors.
- Coverage across invoices, receipts, and multi-column forms exhibiting common real-world failure modes (e.g., dropped totals, merged fields, unreadable line items).

This “teachable band” yielded frequent, interpretable feedback for the reflection model and avoided wasting compute on unsalvageable scans or trivially correct samples.

4.5 Execution and cross-model evaluation

The optimization itself used:

- **Reflection model:** *GPT-5-High*, guiding prompt evolution through analysis and critique.
- **Executor:** *Gemini 2.0 Flash*, providing rapid, consistent evaluations during search.

Once the optimized prompt converged, we froze it and evaluated the full benchmark using:

- Gemini 2.0 Flash (optimization target),
- Gemini 2.5 Flash, and
- Gemini 2.5 Pro.

This measured transferability across model scales. Gains generalized strongly—roughly 80–90% of the improvement carried forward—supporting a practical pattern: *optimize once on a small model, deploy broadly.*

4.6 Qualitative evolution of the prompt

Across iterations, the reflective model independently converged toward best-practice extraction policies familiar to experienced OCR engineers:

- Explicit coverage instructions ("include everything visible," "preserve reading order").
- Rules against hallucination ("do not invent," "use `unreadable` for illegible text").
- Schema discipline ("each label/value on its own line," "use Markdown tables; HTML only for merged cells").
- Fidelity to layout ("parse line items before totals," "keep captions adjacent to figures").

By the end of optimization, GEPA had effectively written a *policy prompt*—a deterministic specification for reliable document transcription.

4.7 Reproducibility

We fixed random seeds for data splits, versioned evaluator configurations, and logged every candidate and score. The forked DSPy code (system-prompt overrides, image-URL hooks) and all experiment scripts are open-sourced at:

https://github.com/Studio-Intrinsic/benchmarking-ocr-gepa

5 Results and Analysis

5.1 Aggregate performance

Model	Baseline	Config-Only	Final Optimized
Gemini 2.0 Flash	86.1%	86.6%	**89.7% (+3.6)**
Gemini 2.5 Flash	88.8%	89.3%	**92.4% (+3.6)**
Gemini 2.5 Pro	91.5%	–	**94.8% (+3.3)**

Table 1: OCR pipeline accuracy across models before and after GEPA optimization. "Config-Only" reflects evaluator/model-choice hygiene (no search).

We also observed a Pareto frontier approaching ∼97% on structured subsets, indicating headroom where document families are consistent and policies are clear.

5.2 Where steerability shows up

Language stage (steerable). GEPA reliably improved:

- Schema anchoring and *null vs. hallucinate* policy,
- Normalization (dates, currency/cent precision),
- Reconciliation (subtotal equals sum of line items),
- Read-order and block grouping (e.g., captions, headers).

Vision stage (bounded by input). Missed tokens and corrupt text rarely improved via prompting; quality limits persist on low-resolution or artifacted scans.

5.3 Cross-model transfer

Prompts optimized on Gemini 2.0 Flash transferred 80–90% of their gains to 2.5 Flash and Pro with light edits—supporting a cost-efficient strategy: optimize on lower-cost models, then port.

5.4 Steerability conditions for business

OCR steerability is strongest when:

- Documents are **family-similar** (recurring vendor templates, forms),
- The output schema is **strict** and validated,
- Policies (read-order, null handling, reconciliation) are explicit,
- Perception quality is reasonable (print, scan, column bleed under control).

These conditions describe most enterprise pipelines.

6 Resulting Optimized Prompt

6.1 Final prompt produced by GEPA

```
Convert the provided document image to Markdown. Return only the Markdown with no exp

Rules
1) Coverage and reading order
```

```
   - Include everything visible: titles, headings, subheadings, headers/footers, page
   - Preserve the original reading order (top-to-bottom, left-to-right). Keep related
   - Do not omit or invent any content.

2) Headings and hierarchy
   - Render the main document/page title as H1 (# ...) exactly as printed.
   - Use H2/H3 (##, ###) for subordinate section labels according to visual prominenc
   - Keep heading text exactly as shown (punctuation, casing, typos unchanged).
   - Separate logical blocks with a single blank line.

3) Text blocks, labels, and line breaks
   - Preserve meaningful line breaks. Do not arbitrarily merge or split lines.
   - Keep each labeled field on its own line unless the source prints multiple labels
   - Preserve punctuation, casing, hyphenation, and spacing exactly as printed.
   - Preserve typographic emphasis that conveys structure: use Markdown bold/italic o

4) Tables
   - Use a Markdown table for standard tabular data with clear columns/rows.
   - Use HTML <table> only when needed to preserve merged cells or nested tables.
   - Do not convert simple form layouts into tables. Transcribe as labeled lines.
   - Preserve column order, header labels, and units exactly as printed.
   - If a cell is blank, output an empty cell.

5) Numbers and units
   - Copy values exactly, including currency symbols, separators, decimals, and units

6) Charts and infographics
   - Add one concise, factual alt-text line using Markdown image syntax without a URL
   - Include readable data tables only when printed data is clearly visible.
   - Do not invent or estimate values.

7) Logos, watermarks, stamps, and page numbers
   - Transcribe visible text exactly as printed.

8) Checkboxes
   - Render unchecked as  and checked as . Keep each adjacent to its label.

9) Unreadable or ambiguous content
   - Do not guess. Use 'unreadable' in place of illegible text.
```

```
10) Consistency and formatting
   - Maintain one coherent Markdown style across the document.
   - Preserve meaningful line breaks and ordering.
   - Do not transform text into lists unless explicitly printed.

11) Prohibitions
   - Do not add explanations, commentary, or metadata.
   - Do not correct spelling or formatting.
   - Do not fabricate content.
```

6.2 Why this prompt matters

The prompt encodes coverage, ordering, schema discipline, and non-hallucination policies—*exactly* the factors that drive production reliability. GEPA "rediscovered" best practices human engineers use, and made them explicit and testable. It is a reusable, policy-level artifact.

7 Practical Guidance for Deployment

7.1 Where to apply GEPA

- **Best fit:** Schema-constrained extraction (Markdown→JSON), recurring vendor/forms, and workflows where reconciliation rules matter.
- **Prerequisites:** Reasonable scan quality; clear schema; validation and post-processing checks.

7.2 Operational checklist

- Define a strict JSON schema, null policy, and reconciliation checks.
- Establish a held-out validation split with per-field metrics.
- Run configuration hygiene (model choice, scaffolding) before GEPA.
- Optimize teachable subsets (0.70–0.90 baseline) to control cost.
- Add runtime guardrails (schema validator; subtotal sum checks).

- Monitor vendor/form drift; refresh prompts on distribution shift.

7.3 Risk and compliance

- Ensure PII handling aligns with policy; redact where needed.
- Enforce “no invention”—prefer `null` over guessed values.
- Consider a clause to ignore instructions embedded in scanned text.

8 Discussion

8.1 What we learned

We find OCR is a *steerable* target for prompt optimization in the settings most businesses care about: recurring, family-similar documents with strict output schemas. Gains accrue where policy and structure dominate (language stage), while perception continues to bound the ceiling on degraded scans.

8.2 Why this matters for system design

This supports a layered pattern: stabilize perception with strong OCR backbones and preprocessing; then add a reflective, prompt-optimized extraction head. As models advance, the optimization layer captures capability gains without retraining—turning reliability into a moving frontier that improves over time.

9 Limitations and Future Work

9.1 Limitations

- Results pertain to document OCR/structured extraction; unconstrained generation differs.
- Prompts can overfit to document families; continuous monitoring is recommended.
- Severe perception defects (low-res scans, heavy artifacting) still dominate failure modes.

9.2 Future work

- **Adaptive runtime selection:** Choose prompts per document via similarity search.
- **Layout-aware cues:** Add lightweight spatial features (zones, columns) to prompts.
- **Human-in-the-loop:** Convert reviewer edits into GEPA seeds for faster convergence.
- **Perception advances:** Pair prompt-optimized extraction with improved vision backbones.

10 Conclusion

OCR is a *steerable* task for most business pipelines. In contexts with family-similar documents and strict schemas, an automatic prompt optimization layer delivers reliable, repeatable gains (3.3–3.6 pts here) and reduces exception handling. While perception bounds outliers, policy-focused prompting turns "almost right" into production-ready—no retraining required. We believe this design pattern is the most practical path to self-improving document AI systems.

Implementation Reference

Full code, scripts, and evaluator details:

https://github.com/Studio-Intrinsic/benchmarking-ocr-gepa

References

- Omni OCR Benchmark — `https://github.com/getomni-ai/benchmark`
- DSPy Framework — `https://github.com/stanfordnlp/dspy`
- GEPA Optimizer — `https://github.com/gepa-ai/gepa`
- Studio-Intrinsic Implementation — `https://github.com/Studio-Intrinsic/benchmarking-ocr-g`