

CACHE-ED2: Compiling LLM Reasoning into Reusable Extraction Programs for Document Extraction at Scale

Sudhanshu Bhoi and Anurag Tripathi and Amir Raza and Mayank Jauhari

Amazon, Bengaluru, India

{sudhbee, tripaanu, amirraz, jauharim}@amazon.com

Abstract

Extracting structured information from visually rich documents at enterprise scale demands both the reasoning capability of large language models and the efficiency of deterministic execution. Current approaches either deploy LLMs as instance-level analysts, incurring per-document inference costs that are prohibitive for repetitive templates, or rely on manually authored vendor-specific prompts that do not scale beyond a handful of high-volume vendors. We present CACHE-ED2, a framework that repositions the LLM as a system-level developer: given a novel document format, a multimodal ReAct-based agent synthesizes a reusable Document Extraction DSL (DocExDSL) program encoding the complete extraction logic (strategies, disambiguation, transforms, and automated reasoning validation), then caches it for deterministic, LLM-free execution on all subsequent documents of the same format. A contrastive Document Format Encoder routes incoming documents to cached programs with perfect hit rates, and a HITL Feedback Analysis Agent incorporates analyst corrections to refine programs over time. On public benchmarks (CORD v2, FATURA) and production streams spanning seven business verticals (580,000+ documents, 11,500+ vendors), CACHE-ED2 achieves near-perfect extraction accuracy (~ 0.99) with a 1.0 extraction rate on document-present fields and perfect format matching across 50+ benchmark templates, matching or exceeding manually authored vendor-specific prompts, while reducing cumulative token consumption by $\sim 2.6\times$ over first 1,000 documents.

1 Introduction

Large Language Models (LLMs) have evolved into trillion-parameter reasoning engines with zero-shot generalization, multi-step deduction, and agentic task orchestration capabilities (Wang et al., 2025). In document information extraction (IE), the task of

converting unstructured documents into structured, schema-conforming data, LLMs are predominantly deployed as instance-level analysts: each document is individually queried for field values, incurring 3–5 seconds latency and \$4–5 per page at inference (Bhoi, 2026). With over 80% of enterprise data in unstructured formats (Kurowski et al., 2025), this per-instance paradigm is economically untenable at production scale.

Two problems motivate this work. First, document templates follow a skewed vendor-volume distribution: few high-volume vendors dominate while thousands of tail-end vendors each contribute sparse instances. Manual SOPs scale for head vendors but are infeasible for the tail, and invoking an LLM per document across millions of monthly instances is a massive resource misallocation. Second, production extraction is not straightforward reading; analysts follow business-specific SOPs requiring derivation of information absent from the document. A due date may be computed as invoice date plus agreed payment terms; an “account number” may refer not to a bank account but to an internal business identifier for vendor resolution. Such domain-specific reasoning (deriving, disambiguating, and overriding based on business context) demands LLM inferential capacity and Human-in-the-Loop (HITL) learning that smaller models cannot provide.

We propose repositioning the LLM from instance-level analyst to system-level developer that synthesizes deterministic extraction code in a Document Extraction DSL (DocExDSL). Given a novel template, the LLM reasons once over the document’s visual and semantic structure, incorporates HITL-provided SOPs, and generates an executable DocExDSL program. After validation through automated reasoning constraints, the program is cached against the template identifier; all subsequent matching instances are processed by the deterministic execution engine, completely by-

passing the LLM, reducing cost and latency to compiled-code execution amortized across the entire document volume per template.

Template-based caching was introduced in CACHE-ED (Bhoi and S, 2025), demonstrating 8% extraction improvement over industry baselines and 50% cost reduction. However, three critical limitations emerged in production: fragile graph matching causing frequent cache misses under minor scan perturbations, a text-only representation without visual context that cemented OCR errors, and a single-pass agent unable to self-verify, requiring multiple instances to converge.

We present CACHE-ED2, resolving these limitations through four contributions:

1. **DocExDSL: A Document Extraction DSL (§3.1).** The first imperative, executable DSL for document IE, encoding extraction strategies, derived field computation, and automated reasoning validation as a deterministic code artifact that executes at sub-second latency without model inference.
2. **Contrastive Document Format Encoder (§3.3).** A learned encoder replacing fragile Template Representation graph matching with format-level embeddings, achieving perfect cache hit rate, a $2.1\times$ improvement over CACHE-ED, and eliminating the primary failure mode of template-based caching.
3. **Multimodal ReAct-based DocExDSL Generation Agent (§3.5).** A code generation agent that synthesizes DocExDSL programs through iterative self-debugging: generating, executing via the deterministic engine, and verifying against its visual reading of the full document image. This execution-in-the-loop design achieves near-perfect extraction from the first encounter with a new format, eliminating CACHE-ED’s multi-instance convergence requirement.
4. **HITL Feedback Analysis Agent (§3.6).** A critic agent translating analyst corrections into program-level changes rather than instance-level overrides, enabling continuous DocExDSL refinement while filtering transient human errors from systematic SOP updates.

2 Related Work

Document IE has evolved from grid-based and GNN methods through Document Transformers

(DocTr (Liao et al., 2023)) and Vision-Language Models (LayoutLMv3 (Huang et al., 2022), Donut (Kim et al., 2022), FormNet (Lee et al., 2022)) to frontier LLMs (Wang et al., 2025). Specialized models offer computational efficiency but lack cross-page reasoning, while frontier LLMs close this gap at significantly higher inference cost (Huang et al., 2025; Stahlberg et al., 2025).

To inject domain knowledge, vendor-specific prompting provides visual SOPs indicating where and how to extract each field (Yang et al., 2023), but requires manual creation per vendor and is unscalable for tail-end distributions. Automated alternatives, including in-context learning (Brown et al., 2020), meta-prompting (Suzgun and Kalai, 2024), retrieval-augmented generation (Lewis et al., 2020; Gao et al., 2024), and declarative pipeline optimizers like DSPy (Khatab et al., 2024), improve per-call quality but share a fundamental limitation: every method still invokes the LLM for each document instance, producing no reusable extraction artifact for recurring templates.

Within document IE, Markdown has emerged as a quasi-standard intermediary (Microsoft, 2025; Li et al., 2025), but it is declarative: it lacks looping, conditionals, and executable traversal logic. We bridge this gap with an imperative, executable DSL for document IE, where the LLM synthesizes deterministic extraction programs rather than merely formatting content. Synthesizing extraction programs in a DSL originates in programming-by-example systems such as FlashExtract (Le and Gulwani, 2014), which learn text-extraction programs from user-highlighted examples; we extend this lineage to visually rich documents with zero-example, LLM-driven synthesis and production HITL repair. Template-level regularity has also been exploited for structure recovery: TWIX (Lin et al., 2025) infers the visual template underlying templated documents to reconstruct their records at high accuracy and low cost, and is complementary to our approach, which synthesizes schema-driven extraction programs over exactly such recovered structure. Agentic frameworks for document understanding, including CACHE-ED (Bhoi and S, 2025), have demonstrated the value of multi-agent orchestration with actor-critic mechanisms and HITL feedback for production-scale extraction (Sapkota et al., 2025).

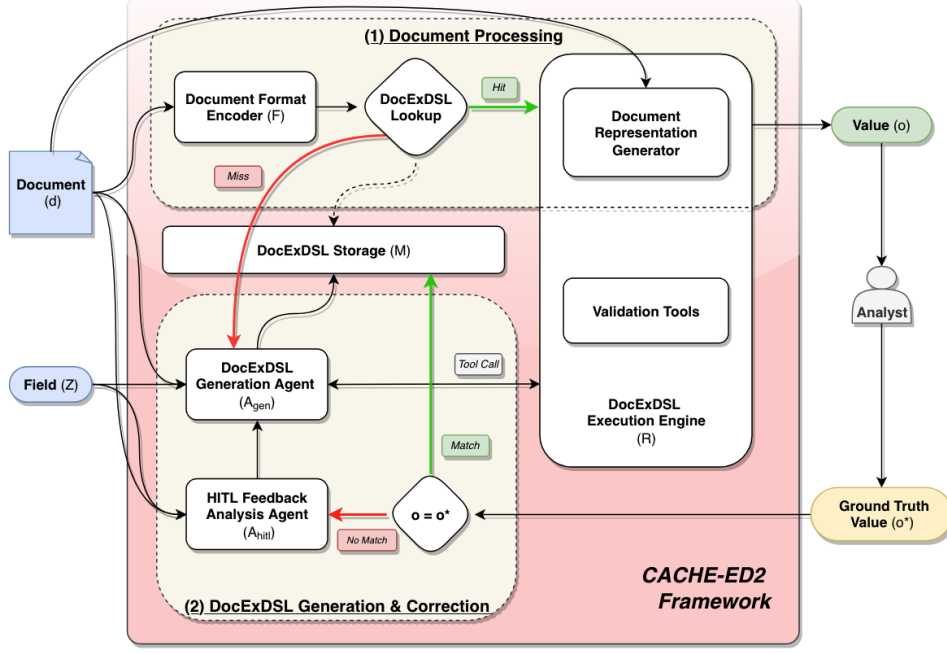


Figure 1: **CACHE-ED2 Framework.** On a cache *hit*, the stored DocExDSL program $\mathcal{P}_f$ is executed deterministically by $\mathcal{R}$ without LLM involvement. On a *miss*, $\mathcal{A}_{\text{gen}}$ synthesizes a new program via ReAct reasoning using $\mathcal{R}$ as a tool. Analyst corrections trigger $\mathcal{A}_{\text{hitl}}$, which recommends program changes fed back to $\mathcal{A}_{\text{gen}}$. Validated programs are cached in $\mathcal{M}$ for deterministic reuse.

3 Approach

CACHE-ED2 comprises six components: a purpose-built extraction language (*DocExDSL*, §3.1), a deterministic *Execution Engine* $\mathcal{R}$ (§3.2), a *Document Format Encoder* $\mathcal{F}$ (§3.3), a persistent *DocExDSL Store* $\mathcal{M}$ (§3.4), and two agents: a *DocExDSL Generation Agent* $\mathcal{A}_{\text{gen}}$ (§3.5) for program synthesis, and a *HITL Feedback Analysis Agent* $\mathcal{A}_{\text{hitl}}$ (§3.6) for incorporating human corrections. The complete pipeline is formalized in Algorithm 1 and illustrated in Figure 1.

3.1 DocExDSL: Document Extraction DSL

DocExDSL makes extraction a code artifact: a human-readable, machine-executable program capturing *how* extraction is performed, including derived information, anchor-based spatial search, disambiguation, and automated verification. Implemented as a Python-embedded language with Pydantic models and fluent method chaining, it leverages the innate Python proficiency of frontier reasoning models (Chen et al., 2021; Rozière et al., 2024), enabling generation from schema specifications without fine-tuning.

A program $\mathcal{P}_f$ is a Schema of Field declarations, each passing through a five-stage pipeline: *Strategies* $\rightarrow$ *Candidates* $\rightarrow$ *Disambiguation* $\rightarrow$

Transforms $\rightarrow$ *Validation*. Five strategy types are chained as ordered fallbacks: key-based matching (exact, contains, fuzzy, regex against KEY $\rightarrow$ VALUE associations), anchor-based spatial search, table extraction, region-based extraction, and derived fields. Disambiguation resolves multiple candidates via single-pick (e.g., highest_confidence, nearest_to_anchor) or aggregate methods (e.g., sum, concat). Transforms apply post-processing (date arithmetic, substring extraction, string formatting) before the validation layer enforces automated reasoning constraints: regex patterns, range checks, and cross-field consistency assertions (Amazon Web Services, 2024, 2025). Derived fields are resolved after all direct fields via topological ordering over the dependency DAG G_{dep} . Refer to Appendix C for the complete specification and Appendix F for examples.

3.2 DocExDSL Execution Engine

The engine $\mathcal{R}$ deterministically executes a program $\mathcal{P}_f$ on a Document Representation D_r organized as a typed document graph $G = (V, E)$ with KEY, VALUE, TEXT, TABLE, and CELL nodes connected by association, contains, and bidirectional adjacency edges (extending CACHE-ED1’s representation (Bhoi and S, 2025)). The engine con-

Algorithm 1 CACHE-ED2 - Document Field Extraction Algorithm

Require: Document d , Extraction Schema $\mathcal{Z}$, Business Vertical v

- 1: $D_r \leftarrow \text{GenerateDocumentRepresentation}(d)$
- 2: $f \leftarrow \text{IdentifyFormat}(d)$
- 3: **if** f is present **then** $\triangleright$ Cache hit: reuse stored program
- 4: $\mathcal{P} \leftarrow \text{StoreLookup}(\mathcal{M}, v, f)$
- 5: $\mathbf{o} \leftarrow \text{ExecuteDSL}(\mathcal{P}, D_r)$
- 6: **else** $\triangleright$ Cache miss: generate via ReAct loop with DSL execution as tool
- 7: $\mathcal{P}, \mathbf{o} \leftarrow \mathcal{A}_{\text{gen}}(d, \mathcal{Z})$
- 8: **end if**
- 9: $\mathbf{o}^* \leftarrow \text{Value from Analyst (HITL)}$
- 10: **if** $\mathbf{o}$ matches $\mathbf{o}^*$ **then**
- 11: Update field scores,
- $\text{StoreUpdate}(\mathcal{M}, v, f, \mathcal{P})$
- 12: **return** $\mathbf{o}$
- 13: **end if**
- 14: Invoke Feedback Agent: $\Delta\mathcal{P} \leftarrow \mathcal{A}_{\text{hitl}}(\mathcal{P}, \mathbf{o}, \mathbf{o}^*, d, s) \triangleright$ Critic with field-level scores
- 15: **if** $\Delta\mathcal{P} \neq \text{NO_CHANGES_NEEDED}$ **then**
- 16: $\mathcal{P}, \mathbf{o} \leftarrow \mathcal{A}_{\text{gen}}(d, \mathcal{Z}, \Delta\mathcal{P})$
- 17: Reset scores for changed fields
- 18: **end if**
- 19: $\text{StoreUpdate}(\mathcal{M}, v, f, \mathcal{P})$
- 20: **return** $\mathbf{o}$

structs the dependency DAG G_{dep} , topologically orders fields, and for each field executes the fallback strategy chain, disambiguation, transforms, and validation, achieving sub-second processing entirely without model inference. When invoked as a tool by $\mathcal{A}_{\text{gen}}$ during program synthesis, $\mathcal{R}$ returns structured per-field diagnostics, including extracted value, strategy outcome, disambiguation reasoning, and validation errors, providing actionable signals for iterative DSL refinement.

3.3 Document Format Encoder

The Document Format Encoder $\mathcal{F}$ produces format embeddings $\mathbf{e}_d = \mathcal{F}(d) \in \mathbb{R}^m$ to determine which cached program applies. Built on LiLT (Wang et al., 2022) with a lightweight projection head $h(\cdot)$ trained via margin-based triplet margin loss (Equation 1) with online semi-hard triplet mining, the encoder maps same-format documents into tight clusters. At inference, $\mathbf{e}_d$ is compared against cluster centroids $\mathcal{C} = c_1, \dots, c_K$ via cosine similar-

ity; matches above threshold τ route to $\mathcal{M}[v, f]$, while unmatched documents trigger $\mathcal{A}_{\text{gen}}$. Because vendors and formats arrive continuously in production, this matching must remain reliable on vendor formats outside the encoder’s training distribution, without per vendor or per-format retraining. We achieve this by keeping the pretrained LiLT backbone frozen, so its general text-and-layout representations transfer to unseen vendors, while the lightweight projection head shapes them into a format-discriminative embedding space. Since every cached format is represented by a centroid built from the documents that created its cache entry, the encoder’s contribution is out-of-distribution generalization to vendors unseen in training, rather than recognition of formats not yet cached: the first document of an uncached format correctly falls below τ against all existing centroids and triggers program synthesis, rather than false-matching an existing program. The model encodes the first page, as format is recognizable from it in the vast majority of enterprise documents.

$$\mathcal{L}_{\text{triplet}} = \frac{1}{|\mathcal{T}|} \sum_{(a,p,n) \in \mathcal{T}} \max(0, \delta_p - \delta_n + \alpha) \quad (1)$$

where $\delta_p = \|h(x_a) - h(x_p)\|_2$, $\delta_n = \|h(x_a) - h(x_n)\|_2$, $h(\cdot)$ is the projection head, (x_a, x_p, x_n) are anchor-positive-negative document triplets drawn from the semi-hard set $\mathcal{T} = \{(x_a, x_p, x_n) : \delta(x_a, x_p) < \delta(x_a, x_n) < \delta(x_a, x_p) + \alpha\}$ and α is the margin.

3.4 DocExDSL Store

The store $\mathcal{M}$ indexes validated programs by format, partitioned by business vertical v to prevent cross-contamination of extraction SOPs. Each field p_j in a stored program carries a *document-processed score* $s_j^{(\mathcal{P}_f)} \in \mathbb{Z}^+$, incremented per successful extraction match against human-validated output. This score provides $\mathcal{A}_{\text{hitl}}$ with a confidence signal, ensuring that high-score fields are not overridden by single potentially erroneous corrections, and enables accuracy degradation detection over time.

3.5 DocExDSL Generation Agent

The agent $\mathcal{A}_{\text{gen}}$ is a ReAct-based (Yao et al., 2023) *code generator*, not a value extractor: it writes programs describing extraction logic and never extracts values directly. It delegates extraction entirely to $\mathcal{R}$ via tool calls, using the

full-resolution document image $\mathcal{I}$ as visual reference to judge correctness. Given d (including its image $\mathcal{I}$) and schema $\mathcal{Z}$, the agent iteratively generates a candidate $\mathcal{P}_f$, invokes $\mathcal{R}$ to obtain $\mathbf{o}$ with per-field diagnostics, compares against its visual reading of $\mathcal{I}$, and refines mismatched fields, effectively debugging its own code. Beyond direct extraction, the agent derives business-specific SOPs not explicitly present in the document: for instance, synthesizing a service period date range as `start_of_month/end_of_month` of the invoice date via DocExDSL’s `derive_from` and `date_component` transforms. The agent also accepts feedback $\Delta\mathcal{P}$ from $\mathcal{A}_{\text{hitl}}$ to regenerate specific fields via $\mathcal{P}'_f = \mathcal{A}_{\text{gen}}(d, \mathcal{Z}, \Delta\mathcal{P})$. Refer to Appendix E for example and Appendix B for prompt structure.

3.6 HITL Feedback Analysis Agent

The critic agent $\mathcal{A}_{\text{hitl}}$ analyzes mismatched fields $\Delta_{\text{fields}} = \{j \mid o_j \neq o_j^*\}$ between extracted output $\mathbf{o}$ and human-corrected output $\mathbf{o}^*$, producing SOP-level program changes $\Delta\mathcal{P} = \mathcal{A}_{\text{hitl}}(\mathcal{P}_f, \mathbf{o}, \mathbf{o}^*, d, s)$, such as adjusting anchor directions, adding derivation rules, or modifying disambiguation strategies. The field-level scores s guard against incorporating erroneous corrections: high-score fields have proven track records and are not overridden by single edits. If $\Delta\mathcal{P} \neq \text{NO_CHANGES_NEEDED}$, changes are forwarded to $\mathcal{A}_{\text{gen}}$ for resynthesis and scores for changed fields are reset. Unlike CACHE-ED1, verification logic is now embedded in DocExDSL’s validation layer, enabling fully deterministic, LLM-free validation even during cache-hit processing. Refer to Appendix B for prompt structure.

4 Experiments

We evaluate CACHE-ED2 against three baselines: (i) a *multimodal LLM* without extraction SOPs, mimicking mid-to-tail vendor behavior, (ii) *multimodal LLM with vendor-specific prompts* with manually authored per-vendor SOPs, and (iii) *CACHE-ED1* (Bhoi and S, 2025). All methods use Claude Sonnet 4.6 for consistency.

4.1 Extraction Quality

We first evaluate on CORD v2 (Park et al., 2019), a public receipt-parsing benchmark of 11,000+ images with structured field annotations across totals, subtotals, and payment-type breakdowns. Since

all CORD fields require no SOP-based derivation, this isolates raw extraction capability; we omit the vendor-SOP baseline accordingly. Table 1(a) reports results across eight fields. Both CACHE-ED variants improve over the single-pass LLM (0.91 average) through closed-loop refinement, with CACHE-ED2 achieving 0.99 average accuracy entirely through the deterministic DocExDSL engine at inference, demonstrating that compiling extraction logic to DocExDSL preserves fidelity without LLM involvement.

For context, models fine-tuned on CORD’s training split report entity-level F1 of 96.56 (LayoutLMv3_{BASE}) to 97.46 (LayoutLMv3_{LARGE}) (Huang et al., 2022) and 97.64 (UDOP-Dual) (Tang et al., 2023), while the OCR-free Donut reports 84.1 field-level F1 (Kim et al., 2022). These figures are not directly comparable to ours: they measure F1 after supervised fine-tuning on CORD, and none publishes per-field results, whereas CACHE-ED2 reports per-field value accuracy with no CORD-specific training and no gradient updates at any point. The operational differentiator is the maintenance path: corrections update a cached program rather than requiring re-annotation and fine-tuning.

Real-world invoice processing demands business-rule reasoning that separates the methods. We curate 5,000 production invoices (1,000 per vendor, 5 vendors, single vertical) and select seven fields spanning three difficulty tiers: *easy* fields (Invoice Number, Invoice Date, Invoice Currency), *SOP-dependent* fields where business rules override the raw value (Account Number, Service Period Start/End Date), and *derived* fields computed from other fields (Vendor Provided Due Date). Table 1(b) reports extraction rate and accuracy per method. The multimodal LLM drops sharply on SOP-dependent fields (0.64 average accuracy) and derived fields (0.59 extraction rate for Due Date), confirming that business context is essential.

CACHE-ED2 achieves near-perfect performance across all fields (0.99 average accuracy, 1.0 extraction rate). On derived fields, CACHE-ED2 surpasses vendor prompts by 9 percentage points on Due Date (0.97 vs. 0.88) for two reasons: the agent initially absorbs minor human variations (e.g., ± 1 day adjustments in date derivations) and learns the correct SOP over time as the scoring mechanism stabilizes, and the format-aware caching handles multiple vendor formats independently rather than

Table 1: Extraction quality on CORD v2 receipts and real-world invoices. Results reported as *Extraction Rate / Accuracy*; best per field in **bold**.

(a) CORD v2 Receipts								
Method	[Total] Price	[Total] Cash Price	[Total] Credit Card	[Total] E-Money	[Subtotal] Price	[Subtotal] Tax	[Subtotal] Service Charge	[Subtotal] Discount
Multimodal LLM	1.0 / 0.90	0.99 / 0.87	0.97 / 0.86	1.0 / 1.0	1.0 / 0.92	0.88 / 0.89	1.0 / 0.89	1.0 / 0.92
CACHE-ED1	1.0 / 0.97	1.0 / 0.95	0.99 / 0.97	1.0 / 1.0	1.0 / 0.98	0.99 / 0.97	1.0 / 0.96	1.0 / 1.0
CACHE-ED2 (Ours)	1.0 / 0.99	1.0 / 0.98	1.0 / 0.97	1.0 / 1.0	1.0 / 1.0	0.99 / 0.97	1.0 / 1.0	1.0 / 1.0

(b) Real-World Invoices							
Method	Invoice Number	Account Number	Invoice Date	Vendor Provided Due Date	Service Period Start Date	Service Period End Date	Invoice Currency
Multimodal LLM	1.0 / 0.99	0.99 / 0.54	1.0 / 0.95	0.59 / 0.92	0.97 / 0.69	0.97 / 0.69	0.82 / 1.0
+ Vendor SOP	1.0 / 0.98	1.0 / 0.97	1.0 / 0.99	1.0 / 0.88	1.0 / 0.96	1.0 / 0.96	1.0 / 1.0
CACHE-ED1	0.96 / 0.87	0.96 / 0.89	0.90 / 0.84	0.89 / 0.66	0.87 / 0.69	0.91 / 0.70	0.87 / 1.0
CACHE-ED2 (Ours)	1.0 / 1.0	1.0 / 0.97	1.0 / 0.99	1.0 / 0.97	1.0 / 0.99	1.0 / 0.98	1.0 / 1.0

Table 2: Cache hit rate on FATURA (50 layouts, grouped by visual variance) and real-world invoices (5 vendor formats). Higher is better.

Format	CACHE-ED1	CACHE-ED2 (Ours)
(a) FATURA (50 templates, grouped by difficulty)		
Group 1 (Easiest)	0.89	1.0
Group 2	0.72	1.0
Group 3	0.57	1.0
Group 4	0.44	1.0
Group 5 (Hardest)	0.25	1.0
(b) Real-World Invoices		
Vendor Format 1	0.14	1.0
Vendor Format 2	0.29	1.0
Vendor Format 3	0.70	1.0
Vendor Format 4	0.70	1.0
Vendor Format 5	0.52	1.0

sending all format SOPs to the LLM simultaneously. These performance gains are consistent across the entire vendor-volume distribution, including long-tail vendors with sparse instances (Appendix G). Refer to Appendix F for SOP comparisons and Appendix E for representative programs.

4.2 Format Detection

We first evaluate on FATURA (Limam et al., 2023), a public benchmark of 10,000 invoice images across 50 distinct layouts. We rank templates by intra-class visual variance, group them into five difficulty tiers of 10 templates each, and report average cache hit rate per group. All hit rates follow a vendor-held-out protocol: the evaluated vendors are excluded from encoder training, each format’s centroid is built from its own documents, dis-

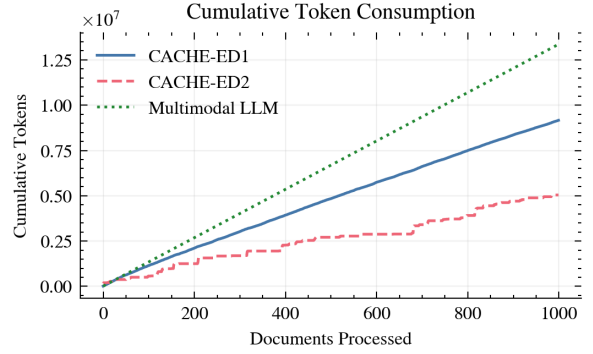


Figure 2: Cumulative token consumption over 1,000 documents (lower is better). CACHE-ED2 achieves ~62 reduction; CACHE-ED1 achieves ~31.

joint from those scored, and matching runs against all cached-format centroids (threshold sensitivity and near-miss margins: Appendix A). Table 2(a) compares CACHE-ED1’s graph matching against CACHE-ED2’s contrastive Document Format Encoder.

CACHE-ED1 averages 0.47 due to fragile graph matching under minor visual perturbations and NER-based value masking failures. CACHE-ED2 achieves perfect hit rate here as well ($2.1\times$ improvement), confirming that learned embeddings generalize across dataset scale, domain, and document type. Refer to Appendix A for t-SNE visualizations.

4.3 Token Efficiency and Learning Dynamics

We measure cumulative token consumption on the real-world invoice corpus as documents are processed sequentially (Figure 2). The single-pass

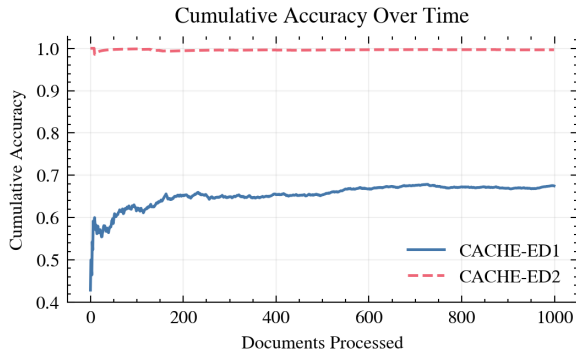


Figure 3: Cumulative accuracy for a single real-world vendor (higher is better). CACHE-ED2 reaches near-perfect accuracy from the first document.

LLM baseline consumes $\sim 13,360$ tokens per document, totaling $\sim 13.4\text{M}$ tokens for 1,000 documents. Both caching methods surpass the baseline within the first ~ 50 documents, after which cached formats are served at zero token cost. At 1,000 documents, CACHE-ED2 consumes $\sim 5\text{M}$ tokens ($\sim 62\%$ reduction vs. baseline), while CACHE-ED1 consumes $\sim 9\text{M}$ tokens ($\sim 31\%$ reduction), $\sim 1.8\times$ higher than CACHE-ED2. The gap is driven by CACHE-ED1’s lower cache hit rate: frequent misses trigger repeated program generation for formats that should already be cached. At production scale (36,093 documents), cached serving cuts LLM spend $6\times$ versus invoking the synthesis agent per document (Appendix I).

We track cumulative accuracy over time for a single real-world vendor to isolate learning behavior (Figure 3). CACHE-ED1 starts at low accuracy and plateaus well below acceptable levels, requiring multiple document instances to converge. CACHE-ED2 achieves near-perfect accuracy from the first document instance and maintains it throughout, as the ReAct agent’s execution-in-the-loop design enables accurate SOP derivation in a single encounter with a new format.

4.4 Production-Scale Evaluation

Beyond the five-vendor comparative study, we evaluate CACHE-ED2 across seven business verticals in three regions, on production document streams totaling more than 580,000 documents from over 11,500 vendors. The streams exhibit the heavy-tailed vendor concentration typical of enterprise document processing: per-vertical vendor counts range from 50 to 5,220, the top vendor alone contributes between 4.3% and 94.5% of a vertical’s volume, and covering 90% of volume requires any-

Table 3: Production-scale evaluation across seven anonymized business verticals (ordered by volume).

Vertical	Docs	Vendors	Ext. (%)	Acc. (%)
A	321,197	5,220	99.89	99.58
B	135,971	4,049	99.12	96.84
C	92,692	710	99.41	94.52
D	12,318	50	99.99	99.96
E	9,196	295	98.67	98.94
F	6,249	637	99.96	99.01
G	3,630	599	99.87	97.98
All	581,253	11,560	99.56	98.12

where from a single vendor to 639. Table 3 reports the measured per-vertical results. Correcting for audited annotation deltas (ground-truth entries that are mis-keyed or contain values absent from the source document) yields a 100% extraction rate on document-present fields and approximately 99% field accuracy. For a per-field deep dive and Pareto volume analysis over one vertical’s full vendor distribution, see Appendix G. These results address the scope of generalization directly: the mechanism is format-level, and its economics improve with every additional format cached.

Component ablations. Leave-one-out ablations on a 1,000-document production cohort (96 vendors, 115 formats) separate each component’s contribution: removing visual grounding collapses completion to 0.270 of the full system (the synthesis agent fails to converge without page images), removing execution-in-the-loop or HITL feedback lowers accuracy retention to 0.947 and 0.829 respectively, and the contrastive encoder guards the 39.7% of documents from multi-format vendors that vendor-level routing can misroute. Appendix H details the protocol and full results.

5 Conclusion

We presented CACHE-ED2, a framework that repositions the LLM from an instance-level analyst to a system-level developer synthesizing reusable DocExDSL programs for document extraction. By coupling a contrastive Document Format Encoder, a multimodal ReAct-based Generation Agent with execution-in-the-loop, and a score-guarded HITL Feedback Agent, the framework achieves near-perfect extraction accuracy while eliminating LLM inference at serving time and reducing cumulative token consumption by $\sim 62\%$ over 1,000 documents.

Limitations

Two limitations of the current framework warrant discussion. First, each new vendor format requires independent program synthesis from scratch; the framework does not yet leverage cross-template transfer. A concrete path is warm-start synthesis: retrieving the nearest cached program via the Document Format Encoder’s embedding space and supplying it to $\mathcal{A}_{\text{gen}}$ as a first-guess draft, which would cut ReAct iterations and cold-start cost precisely where the vendor long tail makes per-format synthesis most expensive. Second, DocExDSL’s grammar is fixed at design time: when $\mathcal{A}_{\text{gen}}$ encounters extraction patterns inexpressible in the current language (e.g., recursive table structures or conditional multi-page joins), these remain unresolvable until manual DSL expansion. An automated capability discovery mechanism that identifies such gaps from agent failures and proposes grammar extensions would allow the extraction language to grow alongside production complexity.

Ethics Statement

The real-world corpora comprise enterprise invoices processed by the authors’ organization’s production document pipelines, used for this research under internal data-use authorization. All multimodal LLM inference (Claude Sonnet) was executed inside a contractually isolated Amazon Bedrock enterprise environment within the organization’s secure cloud perimeter, under the service’s documented commitments that prompts and outputs are not stored, not shared with the model provider, and not used for model training, mitigating rather than eliminating third-party processing risk; no document images or extracted values were sent to public API endpoints. Human-in-the-loop signals are field-level corrections produced by professional analysts within their standard verification workflow; no additional annotation labor was commissioned for this work. All figures reproducing document content are uniformly de-identified, with account identifiers, names, addresses, and derived values masked.

References

Amazon Web Services. 2024. What is automated reasoning? <https://aws.amazon.com/what-is/automated-reasoning/>. Accessed: 2026-04-21.

Amazon Web Services. 2025. Automated reasoning checks in Amazon Bedrock Guardrails. <https://aws.amazon.com/about-aws/whats-new/2025/08/automated-reasoning-checks-amazon-bedrock-guardrails>. Applies formal verification to validate AI outputs against encoded business rules; delivers up to 99% accuracy in detecting hallucinations.

Sudhanshu Bhoi. 2026. [Vig-llm: Enhancing visual grounding capabilities in closed-box llms for document information extraction without ocr dependencies](#).

Sudhanshu Bhoi and Harish Y V S. 2025. [Cache-ed: Redefining document entity extraction with graph-based templates, actor-critic agents & hil](#).

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, and 12 others. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 1877–1901. ArXiv:2005.14165. Introduced GPT-3 and the in-context learning paradigm.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*. Introduces Codex (GPT fine-tuned on GitHub code); powers GitHub Copilot.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. 2024. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*. Comprehensive survey covering RAG paradigms, components, and evaluation for LLMs.

Penghao Huang, Songxuan Lai, and 1 others. 2025. OCR-Reasoning benchmark: Unveiling the true capabilities of MLLMs in complex text-rich image reasoning. *arXiv preprint arXiv:2505.17163*. Accepted at ICLR 2026.

Yupan Huang, Tengchao Lv, Lei Cui, Yutong Lu, and Furu Wei. 2022. LayoutLMv3: Pre-training for document AI with unified text and image masking. In *Proceedings of the 30th ACM International Conference on Multimedia (MM '22)*. ACM. ArXiv:2204.08387.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Mober, Cyrus Shah, Christopher Potts,

- and Matei Zaharia. 2024. DSPy: Compiling declarative language model calls into state-of-the-art pipelines. In *Proceedings of the 12th International Conference on Learning Representations (ICLR)*. ArXiv:2310.03714.
- Geewook Kim, Teakgyu Hong, Moonbin Yim, Jeongyeon Nam, Jinyoung Park, Jinyeong Yim, Wonseok Hwang, Sangdoo Yun, Dongyoon Han, and Seunghyun Park. 2022. OCR-free document understanding transformer. In *European Conference on Computer Vision (ECCV)*. Springer. ArXiv:2111.15664.
- Mateusz Kurowski and 1 others. 2025. Enabling the use of unstructured data for robotic process automation. *arXiv preprint arXiv:2507.11364*.
- Vu Le and Sumit Gulwani. 2014. [FlashExtract: A framework for data extraction by examples](#). In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pages 542–553.
- Chen-Yu Lee, Chun-Liang Li, Timothy Dozat, Vincent Perot, Guolong Su, Nan Hua, Joshua Ainslie, Renshen Wang, Yasuhisa Fujii, and Tomas Pfister. 2022. FormNet: Structural encoding beyond sequential modeling in form document information extraction. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 3735–3754, Dublin, Ireland. Association for Computational Linguistics. ArXiv:2203.08411.
- Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 9459–9474. ArXiv:2005.11401. Introduced the RAG framework combining parametric and non-parametric memory.
- Zichao Li, Aizier Abulaiti, Yaojie Lu, Xuanang Chen, Jia Zheng, Hongyu Lin, Xianpei Han, Shanshan Jiang, Bin Dong, and Le Sun. 2025. READoc: A unified benchmark for realistic document structured extraction. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 21889–21905, Vienna, Austria. Association for Computational Linguistics. ArXiv:2409.05137.
- Haofu Liao, Aruni RoyChowdhury, Weijian Li, Ankan Bansal, Yuting Zhang, Zhuowen Tu, Ravi Kumar Satzoda, R. Manmatha, and Vijay Mahadevan. 2023. DocTr: Document transformer for structured information extraction in documents. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE. ArXiv:2307.07929.
- Mahmoud Limam, Marwa Dhiaf, and Yousri Kessentini. 2023. FATURA: A multi-layout invoice image dataset for document analysis and understanding. *arXiv preprint arXiv:2311.11856*.
- Yiming Lin, Mawil Hasan, Rohan Kosalge, Alvin Cheung, and Aditya G. Parameswaran. 2025. [Visual template inference for data extraction from documents](#). Preprint, arXiv:2501.06659.
- Microsoft. 2025. Document intelligence supported Markdown elements. <https://learn.microsoft.com/en-us/azure/ai-services/document-intelligence/concept/markdown-elements>. Azure AI Document Intelligence Layout API transforms complex PDFs into semantically structured Markdown.
- Seunghyun Park, Seung Shin, Bado Lee, Junyeop Lee, Jaeheung Surh, Minjoon Seo, and Hwalsuk Lee. 2019. CORD: A consolidated receipt dataset for post-OCR parsing. In *Document Intelligence Workshop at Neural Information Processing Systems*.
- Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferber, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, and 7 others. 2024. Code Llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*. Family of code LLMs based on Llama 2; state-of-the-art open models for code generation.
- Himanshu Sapkota and 1 others. 2025. AI agents vs. agentic AI: A conceptual taxonomy, applications and challenges. *arXiv preprint arXiv:2505.10468*.
- Felix Stahlberg and 1 others. 2025. Problem solved? Information extraction design space for layout-rich documents using LLMs. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 17908–17927, Suzhou, China. Association for Computational Linguistics. ArXiv:2502.18179.
- Mirac Suzgun and Adam Tauman Kalai. 2024. Meta-prompting: Enhancing language models with task-agnostic scaffolding. *arXiv preprint arXiv:2401.12954*. Transforms a single LM into a conductor orchestrating multiple expert LM instances.
- Zineng Tang, Ziyi Yang, Guoxin Wang, Yuwei Fang, Yang Liu, Chenguang Zhu, Michael Zeng, Cha Zhang, and Mohit Bansal. 2023. Unifying vision, text and layout for universal document processing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Jiapeng Wang, Lianwen Jin, and Kai Ding. 2022. [LiLT: A simple yet effective language-independent layout transformer for structured document understanding](#). In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7747–7757. Association for Computational Linguistics.
- Weishi Wang, Hengchang Hu, Zhijie Zhang, Zhaochen Li, Hongxin Shao, and Daniel Dahlmeier. 2025. Document intelligence in the era of large language mod-

els. *arXiv preprint arXiv:2510.13366*. Survey on LLMs for Document AI.

Jianwei Yang, Hao Zhang, Feng Li, Xueyan Zou, Chunyuan Li, and Jianfeng Gao. 2023. Set-of-mark prompting unleashes extraordinary visual grounding in GPT-4V. *arXiv preprint arXiv:2310.11441*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. *ReAct: Synergizing reasoning and acting in language models*. In *Proceedings of the 11th International Conference on Learning Representations (ICLR)*.

A Document Format Encoder: LiLT Architecture and Training

This appendix details the architecture and training of the Document Format Encoder introduced in Section 3.3.

A.1 LiLT: Language-Independent Layout Transformer

The format encoder builds on LiLT (Wang et al., 2022), a dual-stream Transformer with separate text and layout flows connected via Bi-directional Attention Complementation (BiACM). The text flow initializes from RoBERTa at 768 hidden dimensions; the layout flow uses a compact 192-dimensional representation, keeping layout-only parameters to 6.1M. BiACM shares attention scores between flows at every layer, enabling cross-modal interaction without stream fusion. Pre-trained on 11M English documents (IIT-CDIP) using Masked Visual-Language Modeling, Key Point Location (7×7 grid), and Cross-modal Alignment Identification, LiLT has been fine-tuned for entity recognition, relation extraction, and document classification across eight languages on FUNSD, CORD, EPHOIE, RVL-CDIP, and XFUND. Its lightweight layout encoding, joint text-layout awareness, and availability of a public checkpoint (nielsr/lilt-roberta-en-base) make it a natural backbone for our format similarity task. Figure 4 illustrates the architecture.

A.2 Training Configuration

We freeze the full LiLT encoder (~ 125 M parameters) and train only a 3-layer MLP projection head ($768 \rightarrow 512 \rightarrow 256 \rightarrow 256$, ReLU, L2-normalized output) — 590K trainable parameters (0.47% of total). The model is trained with triplet margin loss ($\alpha=0.2$) using online semi-hard mining, which selects negatives harder than the positive but within the margin boundary. A balanced sampler constructs batches of 4 format classes $\times$ 4 samples

each. Training runs 30 epochs with AdamW ($\text{lr}=1\text{e-}3$), linear warmup and decay. Inputs are a commercial OCR engine-extracted words with bounding boxes normalized to $[0, 1000]$, tokenized via RoBERTa, truncated to 512 tokens with dynamic padding.

A.3 Embedding Quality

Documents are encoded into 256-dimensional L2-normalized vectors and matched against stored format centroids via cosine similarity ($\tau=0.95$). Held-out evaluation yields intra-class similarity of 0.9954 (std=0.0109) versus inter-class similarity of 0.3704 (std=0.2004) — a gap of 0.6250 — confirming tight, well-separated format clusters. Figure 5 visualizes these clusters.

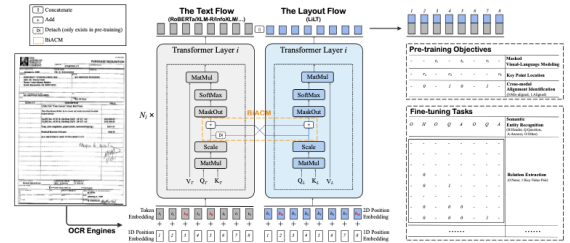


Figure 4: LiLT dual-stream architecture. The text flow (RoBERTa, 768-dim) and layout flow (192-dim) run in parallel with BiACM sharing attention scores bidirectionally at each layer.

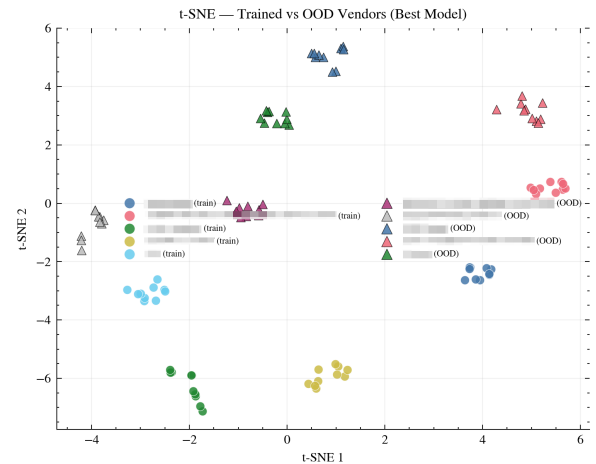


Figure 5: t-SNE projection of 256-dimensional format embeddings for ten sampled vendors (ten documents each): five drawn from the training split (circles) and five from the held-out document split (triangles). Clusters remain tight and well separated in both halves.

Routing Threshold Sensitivity. Routing is precision-critical: a false match silently sends a

document to the wrong cached program, whereas a miss merely triggers synthesis, so τ is set to maximize precision at the expense of recall. Figure 6 shows pairwise cosine similarities over the test embeddings; format-level labels are unavailable, so vendor identity labels the pairs (8,550 same-vendor, 8,524 different-vendor). Different-vendor pairs are reliably different formats, making precision and false-match measurements sound; same-vendor pairs can cross formats, making measured recall a pessimistic bound (the low-similarity same-vendor tail in the figure). The F1-optimal threshold is $\tau = 0.594$ (precision 0.985, recall 0.942); we operate at the stricter $\tau = 0.95$, where the false-match rate falls to 0.05% (four pairs) at precision 0.999, at the cost of pair-level recall (0.440). The band just below the operating point, $[0.93, 0.95]$, holds 11.1% of same-vendor pairs but only two different-vendor pairs. Pair-level recall understates the deployed hit rate twice over: same-vendor pairs may cross formats, and production routing matches each document against format centroids rather than individual documents; deployed hit rates are reported in the main text.

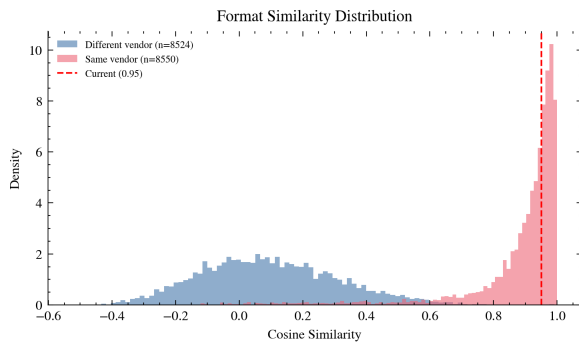


Figure 6: Pairwise cosine-similarity distributions of same-vendor and different-vendor pairs over the test embeddings, with the routing operating point $\tau = 0.95$ marked.

B DocExDSL Generation Agent and HITL Feedback Analysis Agent: Prompt Architecture

This appendix details the prompt structures of the two LLM agents: the Generation Agent (actor) that synthesizes DocExDSL programs, and the HITL Feedback Analysis Agent (critic) that diagnoses extraction errors and recommends program-level refinements.

B.1 Generation Agent (Actor) Prompt

The actor prompt operates in two modes — initial generation and revision — sharing a common structure with an optional feedback section activated during revision cycles.

System Instructions. The LLM acts as a code generation agent writing declarative DSL logic — never extracting values directly. It follows a Re-Act loop (generate, execute, inspect, revise) with a strict no-hardcoding constraint ensuring all programs generalize across same-format documents.

Document Context. Multi-page document images (base64-encoded) enable visual inspection of layout, labels, and spatial relationships — the multimodal grounding absent in CACHE-ED1 that enables accurate SOP derivation from a single instance.

Field Definitions. Each target field is specified with name, data type, description, valid and invalid label identifiers, and derivation hints for computed fields, scoped to required outputs plus intermediates necessary for derivation chains.

DSL Specification. The complete DocExDSL language reference is provided in-context: five extraction strategies (key-based, anchor-based, derived, table, region) as ordered fallback chains, disambiguation, transforms, and validation. This leverages frontier LLMs’ code proficiency without fine-tuning.

Strategy Selection Guidelines. Explicit guidelines map layout observations to strategy choices — label-adjacent values to key-based, spatially referenced values to anchor-based, computed values to derivation, tabular data to table extraction, fixed-position values to region-based — with fallback chains encouraged for robustness.

Revision Context (Optional — Critic Feedback). When critic feedback is available, the previous DSL code and per-field diagnostics are injected. The agent reasons through each flagged field’s root cause, maps the recommendation to a DSL change, and leaves unflagged fields untouched. This targeted revision prevents regression in proven logic while correcting only deficient fields.

Tool-Calling Workflow. An `execute_dsl` tool parses generated code, runs extraction against the document graph, and returns per-field diagnostics: extracted value, strategy outcome, candidate count,

and validation errors. The agent must invoke this after every generation, inspect for null or implausible outputs, and iterate — this self-debugging loop drives first-encounter accuracy.

Output Format. A Python code block containing DSL imports, Schema definition with extraction logic, strategy rationale as inline comments, and (during revision) documented reasoning for each change.

B.2 HITL Feedback Analysis Agent (Critic) Prompt

The critic is a single-turn LLM call (no tool-calling) producing structured diagnostic feedback. It analyzes extraction errors and provides generalizable recommendations — never extracting values or writing DSL code.

System Instructions. The critic diagnoses mismatches between extracted and human-provided values by reasoning about underlying business logic, not surface-level value matching. All recommendations must be generalizable (no hardcoded values; only relative operations). `NO_CHANGES_NEEDED` is a valid verdict when existing DSL is correct.

Rule Confidence Calibration. Each DSL rule’s validation score — documents successfully processed — calibrates trust. Score 0 assigns high weight to human feedback; scores above 5 warrant skepticism toward contradicting corrections, as the human may be wrong or the document anomalous. This score-weighted mechanism prevents erroneous corrections from corrupting stable programs.

Document Context. Multi-page images enable visual verification of where values appear and what labels surround them, grounding the critic’s diagnosis in the actual document layout.

Extraction Comparison. A structured table presents each field with human value, extracted value, and match status. The critic focuses exclusively on incorrect fields, determining whether the issue lies in DSL logic or the human correction itself.

Current DSL Code. The actor’s existing code is provided so the critic identifies the specific strategy or parameter needing adjustment rather than reasoning abstractly.

Failure Pattern Taxonomy. Ten common failure patterns guide diagnosis: wrong keys, untagged values, derived value errors, wrong disambiguation, subset extraction, wrong spatial direction, table extraction issues, wrong page scope, date/transform errors, and business logic overrides. This taxonomy constrains output to actionable categories mapping directly to DSL modifications.

Output Format. Two-part response: (1) overall verdict (`NO_CHANGES_NEEDED` with reasoning, or `CHANGES_RECOMMENDED`), and (2) per-field diagnostics with the business rule revealed, root cause category from the taxonomy, and a generalizable DSL fix. Instance-specific mismatches are marked `human_likely_wrong` with no DSL change.

C DocExDSL: Detailed Specification

This appendix supplements Section 3.1 with the complete DocExDSL specification — the document graph substrate, strategy parameters, disambiguation, transforms, validation, and the structured result model enabling the Generation Agent’s self-debugging loop (Section 3.5).

Document Graph Substrate. The DSL operates on a typed directed graph $G = (V, E)$ built from OCR output, with coordinates normalized to $[0, 1]$. Nodes are typed as `ROOT`, `KEY`, `VALUE`, `TEXT`, `TABLE`, and `CELL` — each carrying text, confidence, page, and bounding-box geometry. Edges encode association (`KEY`→`VALUE` pairings), contains (`TABLE`→`CELL` hierarchy), and bidirectional adjacency (spatial proximity, threshold 0.1). This abstraction decouples the DSL from any specific OCR provider.

Strategy Parameters. Key-based extraction accepts label strings with match modes (exact, contains, fuzzy at `SequenceMatcher` ≥ 0.7 , regex) and optional confidence thresholds. Anchor-based extraction takes reference text, directional search (right, left, above, below, nearest), maximum offset distance, and filterable node types. Table extraction identifies columns by header matching or explicit row/column index. Region-based extraction targets an absolute bounding box on a specified page, collecting all nodes whose centers fall within.

Fallback Chains. Strategies chain as ordered fallback sequences — the engine tries each in declaration order and uses the first returning candidates.

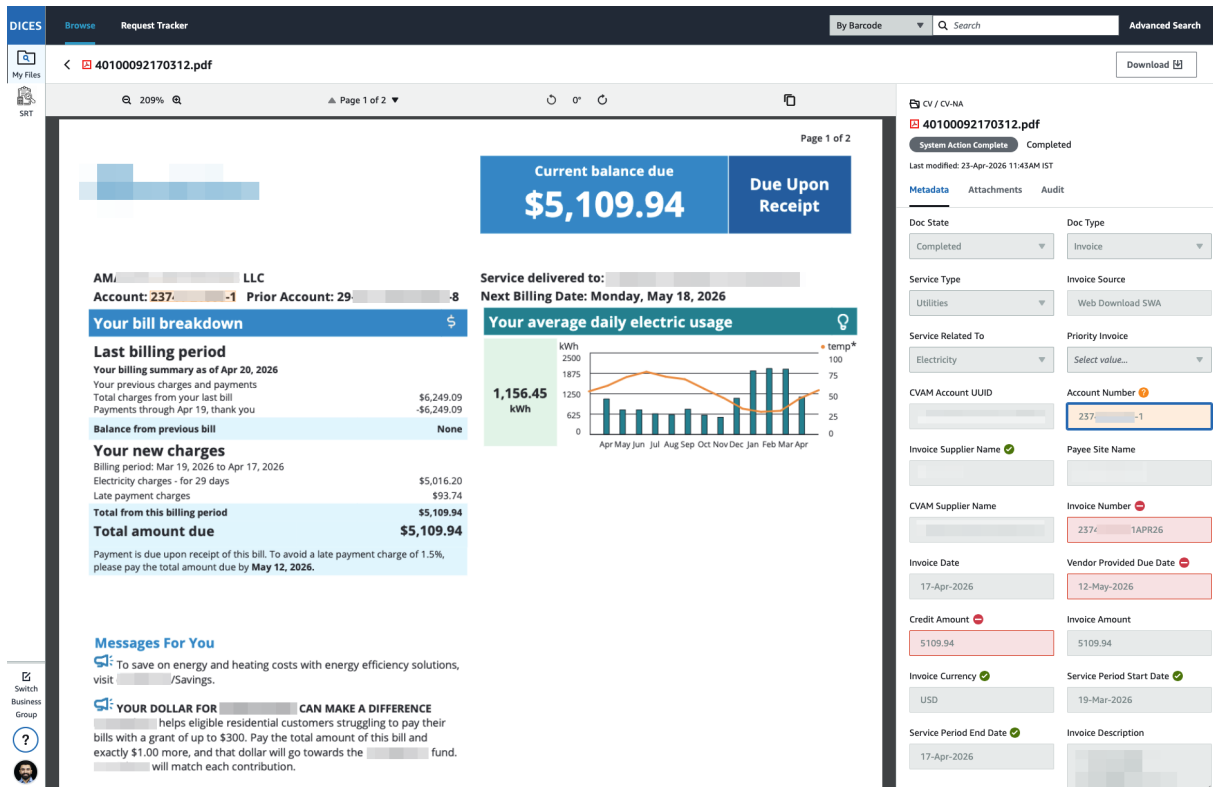


Figure 7: The human-in-the-loop validation interface used by analysts to verify and correct extracted values. Extracted fields are color-coded by confidence level to direct analyst attention, with bounding-box overlays providing spatial grounding for each extraction.

This is central to OCR robustness: broken key-value associations fall back to anchor-based spatial search; truncated anchor text falls back to fuzzy key matching.

Disambiguation. Eight single-pick methods (highest_confidence, largest_area, first/last_occurrence, nearest_to_anchor, longest, shortest, most_frequent) and five aggregate methods (sum, average, min, max, concat) resolve multiple candidates, with optional tiebreakers.

Transforms. Transforms execute in fixed order after disambiguation: Subset (regex, split, line selection) → Date (offset arithmetic, component extraction like start/end_of_month) → String (templates with schema context, case conversion, find-replace) → Custom (arbitrary callables). Chain order enables multi-step post-processing.

Validation. Five constraint types: pattern (regex), length (min/max), range (numeric bounds with automatic number extraction from formatted strings), allowed_values (enumeration), and custom (arbitrary predicates). Cross-field checks — line totals summing to header total, issue

date ≤ due date — are custom validators referencing other schema fields. Failed validations flag fields for review without halting execution.

Scope Filtering. Fields restricted via on_pages or in_region (normalized bounding box) constrain *all* strategies on that field — both source nodes (KEYs, anchors) and candidates (VALUES, CELLS). Essential for multi-page documents where identical labels appear across pages.

Structured Result Model. Each field returns an ExtractionResult exposing: final value, all pre-disambiguation candidates (with geometry and confidence), disambiguation reasoning, validation status with errors, and the successful strategy. This diagnostic output is what enables the Generation Agent’s ReAct loop — the agent inspects candidates, strategy outcomes, and validation errors to iteratively refine DSL code without ground-truth labels.

D CACHE-ED1 Failure Analysis: Template Fragility and Cryptic Representations

This appendix provides concrete evidence for the CACHE-ED1 limitations discussed in Sections 4.1 and 4.2.

Template Fragility Under OCR Variance. CACHE-ED1 constructs templates by anonymizing VALUE nodes and computing structural fingerprints via graph isomorphism. Figure 8 shows two instances of the same vendor format where the document parser detects the site identifier as “AMAAAA SITE” in one and “AMAAAA FC SITE” in the other. This single-word KEY difference produces structurally distinct anonymized graphs, triggering a cache miss and redundant program generation. Such variations are the root cause of CACHE-ED1’s 0.47 average cache hit rate on production data and its degradation from 0.89 to 0.25 across increasing layout difficulty on FATURA (Table 2). CACHE-ED2’s learned format encoder (Section 3.3) absorbs these into tight embedding clusters, achieving perfect hit rates across both datasets.

Cryptic Graph Paths Obstructing LLM Reasoning. CACHE-ED1 passes linearized graph paths to the LLM without the original document image. Figure 9 illustrates the consequence: the document displays “831-NNN-NN86 317” as a single account number, but the parser splits it into two entities due to whitespace — associating “Account Number” with only “317”. The LLM extracts this partial value and encodes the error into the cached program, with no visual context to detect that both segments are spatially adjacent. CACHE-ED2 provides the full document image to the Generation Agent (Section 3.5), enabling visual verification and correct program synthesis across OCR-fragmented regions.

E Sample End-to-End Document Extraction

This appendix traces a single document through the CACHE-ED2 pipeline (Algorithm 1), illustrating how the Generation Agent, HITL Feedback Analysis Agent, and field-level scoring collaborate to achieve correct extraction. Figure 11 summarizes the stages.

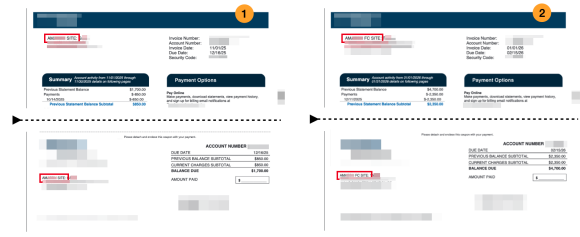


Figure 8: Template fragility under OCR variance. Two instances of the same format where a single-word KEY difference (“AMAAAA SITE” vs. “AMAAAA FC SITE”) produces distinct anonymized graphs, causing a cache miss.

Document Image Sample	LLM Input	LLM Output	Expected Output
	... 5. Reg: XX 317 is associated with value(s): 831-NNN-NN86. There are surrounding key-value pairs are Account Number -> , XX MA XX -> SEATTLE, 98108 to the above key that can provide more context if required. ... 79. Key: Account Number is associated with value(s): 317. There are surrounding key-value pairs are Previous Bill -> 993.13, News You Can Use -> to the above key that can provide more context if required. ...	<account_number> <value>227</value> <explanation>Found in key "Account Number" (entry 79) with value "317", further supported by entry 5 where "XX 317" is associated with "831-NNN-NN86", supporting the full account number context is "317" with sub-account "831-NNN-NN86 317" </explanation> </account_number>	831-NNN-NN86 317

Figure 9: Cryptic path representation failure. The parser splits “831-NNN-NN86 317” into two entities; the LLM extracts only “317” from the direct KEY association, missing the full value apparent from the document image.

Stage 1: Initial Extraction. On encountering a new vendor format, the Generation Agent synthesizes a DocExDSL program via its ReAct loop. For service_period_start, the agent extracts the raw billing period “Nov 19 – Dec 18” as a single value and derives the start date as “Nov 19” using a substring transform. The execution engine returns “Nov 19”.

Stage 2: Human Correction and Critic Analysis. The analyst corrects service_period_start to “Dec 1” in the Verifier UI. The HITL Feedback Analysis Agent receives this correction alongside the current DSL and document image, and identifies an underlying SOP: the service period start is the first day of the month of the billing period end date. It issues CHANGES_RECOMMENDED, advising replacement of the key-based strategy with a derived field applying start_of_month to the billing period end date.

Stage 3: Program Regeneration and Score-Guarded Stability. The Generation Agent incorporates the critic’s feedback, regenerating the DSL as: derive_from(billing_period_end).transform(date_component="start_of_month"). The engine now correctly extracts “Dec 1”. The field score resets to 0

and accumulates with each subsequent match.

Figure 10 demonstrates the complementary case for the Stage 2. After 78 successful validations, the `vendor_provided_due_date` rule (`invoice_date + 45 days`) carries a score of 78. An analyst submits “3/14/2026” for invoice date “1/27/2026” — one day off from the computed “3/13/2026”. The critic issues `NO_CHANGES_NEEDED`: adjusting to +46 days would match this correction but break 78 validated documents. The scoring mechanism correctly identifies a likely human error over a systematic SOP change, preserving program stability.

```
### Verdict: NO_CHANGES_NEEDED
- **Reasoning**: The current DSL derives `vendor_provided_due_date`
as `invoice_date + 45 days`, which correctly reflects the "Net 45"
payment terms printed on the document. Mathematically, 1/27/2026 + 45
days = 3/13/2026, which matches the extracted value. The human-
provided value of 3/14/2026 appears to be an off-by-one error on the
human's part. This rule has been validated successfully against 78
documents with a 78/78 success rate, making it highly reliable.
Changing the offset to `+46 days` to match this single human
correction would break the logic for all other documents where the
rule has been validated correctly. The current DSL is correct and
generalizable.
```

Figure 10: Critic verdict: `NO_CHANGES_NEEDED`. The `vendor_provided_due_date` rule (`invoice_date + 45 days`) has a 78/78 validation record. A one-day discrepancy (3/13 vs. 3/14) is identified as a likely human error; changing the offset would break all prior validations.

F Agent-Derived vs. Human-Curated SOPs Across Failure Modes

This appendix supplements Section 4.1 by comparing CACHE-ED2’s autonomously derived SOPs against human-curated vendor-specific instructions across three failure modes where business logic must be inferred. Figure 12 presents the full comparison.

Context. A significant class of extraction fields cannot be read directly from a labeled value — analysts instead follow vendor-specific SOPs encoding rules for locating unlabeled values, deriving absent fields, or disambiguating duplicate labels. These SOPs are traditionally authored manually per vendor. CACHE-ED2 derives equivalent logic from analyst corrections alone, without SOP documentation as input.

Unlabeled Information. When the target field (e.g., account number) lacks an explicit label, the human SOP locates it by position relative to known landmarks. CACHE-ED2’s Generation Agent identifies the same spatial relationship via document

image visibility and encodes it as an anchor-based strategy with directional search — producing equivalent logic without prior instruction.

Derived Information. When a field must be constructed from other fields (e.g., invoice number derived by concatenating account number with month/year of invoice date), the human SOP specifies the formula explicitly. The HITL Feedback Analysis Agent infers this rule from analyst corrections and recommends a derived field with appropriate string and date transforms — matching the human-authored derivation.

Disambiguation. When multiple values share the same label (e.g., invoice number in both header and line items), the human SOP specifies which instance to use. CACHE-ED2 resolves this through scope filtering and disambiguation (e.g., `first_occurrence` with page restriction), arriving at the same selection logic from corrections alone.

Across all three modes, CACHE-ED2 derived functionally equivalent SOPs, demonstrating that the actor-critic loop with visual grounding can autonomously discover vendor-specific business rules without explicit SOP authoring.

G Vendor Distribution and Long-Tail Performance

This section studies the performance of CACHE-ED2 on a real-world invoice distribution spanning the full vendor spectrum for a representative business vertical, including long-tail vendors with sparse document instances.

Corpus Composition. The evaluation corpus comprises 11,774 documents across 159 distinct vendors spanning a single business vertical. Figure 13 illustrates the cumulative volume distribution: the top 10 vendors account for 48.5% of all documents, the top 25 for 66.1%, and the top 50 for 81.0%, confirming the sharp Pareto concentration typical of enterprise invoice streams. The remaining 109 vendors collectively contribute 19% of volume, with individual counts ranging from single-digit to low double-digit instances per vendor.

Aggregate Performance. Table 4 reports extraction rate and accuracy across all seven header fields evaluated over the full 159-vendor corpus. CACHE-ED2 achieves 100% extraction rate across

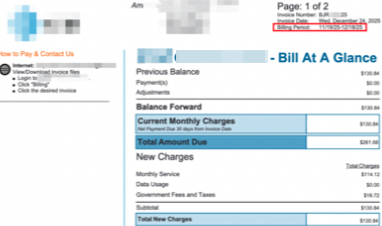
Input (Document/HITL)	Agent	Output (DocExDSL/Feedback)	Extracted Value
	DocExDSL Generation Agent	<pre># billing_period_raw: intermediate - extracts "11/19/25-12/18/25" from "Billing Period:" Field("billing_period_raw", type="string") # intermediate: needed to derive service_period_start_date and service_period_end_date .extract_by_key("Billing Period", "Billing Period:", match="contains") .disambiguate("first_occurrence"), # service_period_start_date: first part of billing period before "-" Field("service_period_start_date", type="date", required=True) .derive_from("billing_period_raw") .subset(split_by="-", split_index=0) .format("MM/DD/YYYY"),</pre>	11/19/25
Dec 01, 2025	HITL Feedback Analysis Agent	<pre>### Field: service_period_start_date - **Human Value**: Dec 01, 2025 - **Extracted Value**: Nov 19, 2025 - **Business Logic**: The billing period printed on the document is "11/19/25-12/18/25", but the human expects Dec 1, 2025 - the first day of the month containing the billing period end date. This indicates a business rule: the service period start date should be the start of the calendar month of the billing period end date, not the literal start date from the billing period string. - **Root Cause**: needs_derivation (business logic override - derive start-of-month from billing period end date) - **Recommended Fix**: Derive 'service_period_start_date' from 'billing_period_raw' by extracting the end date portion (after "-"), parsing it as a date, then applying a 'date_component('start_of_month')' transform. This generalizes to any billing period where the service month is defined by the end date's calendar month.</pre>	
	DocExDSL Generation Agent	<pre># billing_period_end_raw: intermediate - extracts end date portion after "-" Field("billing_period_end_raw", type="date") # intermediate: needed to derive service_period_start_date and service_period_end_date .derive_from("billing_period_raw") .subset(split_by="-", split_index=1), # service_period_start_date: start of calendar month of billing period end date Field("service_period_start_date", type="date", required=True) .derive_from("billing_period_end_raw") .transform(date_component="start_of_month") .format("MM/DD/YYYY"),</pre>	12/01/2025

Figure 11: End-to-end extraction flow through the CACHE-ED2 pipeline. Stage 1: Generation Agent extracts initial values. Stage 2: Human correction triggers critic analysis revealing an SOP. Stage 3: Agent regenerates DSL incorporating the learned derivation rule.

Table 4: CACHE-ED2 extraction performance across 159 vendors (11,774 documents). All fields achieve 100% extraction rate.

Field	Ext. Rate	Accuracy
Account Number	1.0	0.9797
Invoice Currency	1.0	1.0
Invoice Date	1.0	0.9931
Invoice Number	1.0	0.9932
Service Period Start	1.0	0.9978
Service Period End	1.0	0.9980
Vendor Provided Due Date	1.0	0.9975
Aggregate	1.0	0.9942

all fields and 99.42% aggregate accuracy, with no field falling below 97.97%. This performance is maintained uniformly across the vendor-volume distribution: tail vendors with fewer than 10 documents receive the same cached program quality as head vendors, since the ReAct-based Generation Agent synthesizes a correct DocExDSL program on first encounter regardless of expected future volume.

H Per-Component Ablation Study

We quantify each component’s marginal contribution with leave-one-out ablations on a real-world cohort, removing exactly one component per configuration while holding everything else fixed.

Protocol. The cohort is drawn from one production vertical by vendor-volume scaling filtration: each vendor’s monthly volume v_i is scaled to $\lfloor v_i \cdot s \rfloor$ documents, vendors retaining fewer than two documents are dropped, and s is the largest factor for which the retained total stays within a 1,000-document budget. This preserves the shape of the vendor volume distribution (96 vendors retained; per-vendor counts from 2 to 115) while guaranteeing every vendor contributes at least one post-cold-start document. Documents replay chronologically within each vendor, and every configuration, including the unablated reference, runs against its own empty program cache under identical conditions. Ablated arms: visual grounding off (no page images in any agent prompt), execution-in-the-loop off (single-pass generation without the execute-verify-refine tool loop), HITL feedback off (the first synthesized program is final), and encoder routing off (one program per vendor, ignoring for-

encoder’s near-unit retention masks substantial exposure: 21 of the 87 vendors completing documents, including the cohort’s largest, expose multiple formats and hold 39.7% of completed documents; a vendor-level program still serves the dominant format, and the oscillation it invites across the remaining formats concentrates in production’s high-volume head, where the main text measures the encoder’s contribution. All results come from one vertical’s cold-start cohort under a fail-fast step cap; in production, across seven verticals with differing extraction requirements and far larger per-format volumes, each effect compounds, paid on every subsequent document of the affected format, with non-completing documents served by retry and analyst fallback.

I End-to-End Cost and Latency

All calculations use three classes of inputs. Measured inputs come from a 36,093-document production run: serving rates (hit 0.960, miss 0.040, repair 0.313), per-path token means, and the zero-LLM property of the hit path, verified as zero LLM tokens across all 23,937 clean-hit documents. Priced inputs are on-demand list rates, accessed: the production model (Bedrock Claude Sonnet, \$3/\$15 per million input/output tokens) and OCR (Amazon Textract forms analysis, \$0.05 per page; the run’s documents average 3.5 Textract-billed pages, measured over 23,481 completed documents under the pipeline’s ten-page processing cap, raw mean 3.8 pages, median 3).¹ Latency figures are serialized single-document anchors and are order-of-magnitude. Structurally, LLM cost arises only on the miss path: program synthesis is paid once per format, and HITL repair is paid per correction event, and both produce program-level artifacts reused by every subsequent document of the format.

Discussion. Every LLM expense is format-level work while document volume recurs: the \$0.46 synthesis event amortizes to \$0.00015 per document at the observed 3,008 documents per format, and repair amortizes likewise, since a repaired program serves all subsequent documents of its format and the 31% event rate of this onboarding-heavy window declines as programs mature. OCR never

Table 6: Per-document cost and latency by serving path. Token counts (input/output) are measured means over the production run; latency figures are serialized single-document anchors (order of magnitude); costs use the production model’s list prices; OCR is priced at the measured mean of 3.5 billed pages per document.

Path / component	Latency	Tokens	Cost
<i>Cache hit (the 96.0% path)</i>			
OCR (3.5 pages)	seconds	n/a	\$0.174
Encoder inference	< 1 s	n/a	< \$0.01
Deterministic execution	~1 s	0	~\$0
End to end	~12 s	0	\$0.174
<i>Cache miss with synthesis (new format)</i>			
ReAct synthesis (~8 calls)	80–128 s	117.6K/7.2K	\$0.461
End to end	2–4 min	117.6K/7.2K	\$0.634
<i>HITL repair event</i>			
Critic analysis	10–16 s	8.5K/0.7K	\$0.036
Regeneration (~5 calls)	50–80 s	32.7K/3.1K	\$0.145
Repair total	1–1.6 min	41.2K/3.8K	\$0.181

amortizes: \$0.17 per document at the measured page mix, 70% of expected cost against 23% for repair and 7% for synthesis. The registry was already warm: only 3.8% of misses required live synthesis, so the average miss incurred \$0.02 of LLM cost against \$0.46 for a genuinely new format (the table reports the latter). The amortized expected cost is \$0.249 per document (\$249 per 1,000; an independent bucket-mixture computation agrees within \$0.01), 38× below the \$9.40 manual invoice-processing average, and the steady-state cache-hit document (\$0.17, zero LLM) is 54× below. Running the agentic path on every document at the same verification rigor would cost \$0.46 of LLM per document, six times the cached system’s \$0.075, which falls toward zero as formats mature, the compounding mechanism of Section 4.3. Provider choice moves the result only within \$0.18 to \$0.25 per document because OCR, not model price, dominates. The remaining levers split: LLM-side, steady-state repair rates, batch and prompt-caching discounts (up to 50% and 90% of list prices), and smaller or fine-tuned synthesis and repair models (DocExDSL constrains their output space); OCR-side, lighter analysis tiers (text detection at \$0.0015 per page versus \$0.05 forms analysis) and page-selective processing.

¹Provider public pricing pages (AWS Bedrock, AWS Textract) and the Ardent Partners 2025 accounts-payable benchmark, accessed 2026-08-29; on-demand list rates without batch or prompt-caching discounts.