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

Sudhanshu Bhoi [1] **Anurag Tripathi** [1] **Amir Raza** [1] **Mayank Jauhari** [1]

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 5,000 real-world invoices across 5 vendors, CACHE-ED2 achieves near-perfect extraction accuracy (0.99 average) with 1.0 extraction rate and perfect format detection across 50+ templates, matching or exceeding manually authored vendor-specific prompts, while reducing cumulative token consumption by $\sim$2.6$\times$ over 1,000 documents.

[1]Amazon, Bangalore, India. Correspondence to: Sudhanshu Bhoi <sudhbee@amazon.com>, Anurag Tripathi <tripaanu@amazon.com>.

Proceedings of the 43^{rd} International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026.

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) and document understanding spanning subtasks from multimodal extraction and tabular inference to forgery detection and cross-document reasoning, this per-instance paradigm is economically untenable at production scale. The field has progressed through OCR-based pipelines, Vision-Language Models (Xu et al., 2020; Kim et al., 2022), and now Agentic Document AI (Sapkota et al., 2025); yet a fundamental tension persists: specialized document transformers (100–300M parameters) offer computational efficiency and spatial fidelity but lack reasoning capacity, while frontier LLMs deliver superior semantic inference but still fall below 50% on complex OCR-Reasoning benchmarks (Huang et al., 2025). Our work embraces a hybrid philosophy: coupling the structural extraction efficiency of small specialized models with the high-order reasoning of LLMs, rather than replacing one with the other.

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; a service period derived as the invoice month's date range; 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 bypassing 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 & S, 2025), demonstrating 8% extraction improvement over industry baselines and 50% cost reduction via a single actor-critic agent pass. However, three critical limitations emerged. First, Template Representation graph matching relied on document parsers to produce deterministic graph structures, but parsers are sensitive to scan perturbations: minor skew or OCR misclassification produces different node structures for identical formats, causing frequent cache misses. Second, the LLM received only cryptic graph paths without the original document image, permanently cementing OCR errors into the representation with no visual context for correction. Third, the single actor-critic agent pass could not execute or verify its own corrections, requiring multiple document instances to converge and stagnating at suboptimal accuracy.

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.

On public benchmarks (CORD v2, FATURA) and 5,000 real-world invoices across 5 vendors, CACHE-ED2 achieves near-perfect extraction accuracy (0.99 average) with 1.0 extraction rate and perfect format detection across 50+ templates, matching or exceeding manually authored vendor-specific prompts, while reducing cumulative token consumption by $\sim 2.6\times$ over 1,000 documents.

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). Fine-tuned hybrids like DocLLM (Wang et al., 2024), LayoutLLM (Luo et al., 2024), and DocLayLLM (Liao et al., 2025) inject layout awareness into LLM attention, but continuous fine-tuning is economically burdensome and locks systems into architectural snapshots, whereas closed-box APIs guarantee access to improving frontier models without GPU overhead (Quantiphi, 2025). A further constraint is that LLMs cannot natively output spatial coordinates, requiring external grounding frameworks like ViG-LLM (Bhoi, 2026).

To inject domain knowledge, vendor-specific prompting provides visual SOPs indicating where and how to extract each field, analogous to set-of-mark prompting (Yang et al., 2023) and visual grounding techniques used in other domains, 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 & Kalai, 2024), retrieval-augmented generation (Lewis et al., 2020; Gao et al., 2024), and declarative pipeline optimizers like DSPy (Khattab 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.

Domain-Specific Languages (DSLs) offer a path to deterministic, reusable execution. DSLs have long served as foundational abstractions in CAD (OpenSCAD (OpenSCAD Contributors, 2019)), game AI (Gas & Jeuring, 2013), and video editing (Cao et al., 2023). Recent work integrates DSLs with LLMs: MetaGen (Makatura et al., 2025) generates metamaterial designs via a specialized DSL, and Spell (Ramos et al., 2026) synthesizes generalizable programmatic edits, leveraging LLMs' innate code proficiency (Chen et al., 2021; Rozière et al., 2024). 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.

Agentic frameworks for document understanding, including CACHE-ED (Bhoi & 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). Complementary to such learned validation, automated reasoning (Amazon Web Services, 2024) provides mathematical proof-based assurance of system correctness using formal logic techniques such as SAT and SMT solvers, deployed at scale within AWS for policy verification, network reachability analysis, and most recently for LLM output validation in Amazon Bedrock Guardrails (Backes et al., 2018; Amazon Web Services, 2025). We incorporate automated reasoning principles within the proposed framework to formally validate the correctness of values extracted by the DSL execution engine.

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.

Algorithm 1 CACHE-ED2 - Document Field Extraction Algorithm

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

$D_r \leftarrow \text{GenerateDocumentRepresentation}(d)$
$f \leftarrow \text{IdentifyFormat}(d)$
if f is present **then** ▷ Cache hit: reuse stored program
  $\mathcal{P} \leftarrow \text{StoreLookup}(\mathcal{M}, v, f)$
  $\mathbf{o} \leftarrow \text{ExecuteDSL}(\mathcal{P}, D_r)$
else ▷ Cache miss: generate via ReAct loop with DSL execution as tool
  $\mathcal{P}, \mathbf{o} \leftarrow \mathcal{A}_{\text{gen}}(d, \mathcal{Z})$
end if
$\mathbf{o}^* \leftarrow$ Value from Analyst (HITL)
if $\mathbf{o}$ matches $\mathbf{o}^*$ **then**
  Update field scores, $\text{StoreUpdate}(\mathcal{M}, v, f, \mathcal{P})$
  return $\mathbf{o}$
end if
Invoke Feedback Agent: $\Delta\mathcal{P} \leftarrow \mathcal{A}_{\text{hitl}}(\mathcal{P}, \mathbf{o}, \mathbf{o}^*, d, \mathbf{s})$ ▷ Critic with field-level scores
if $\Delta\mathcal{P} \neq$ `NO_CHANGES_NEEDED` **then**
  $\mathcal{P}, \mathbf{o} \leftarrow \mathcal{A}_{\text{gen}}(d, \mathcal{Z}, \Delta\mathcal{P})$
  Reset scores for changed fields
end if
$\text{StoreUpdate}(\mathcal{M}, v, f, \mathcal{P})$
return $\mathbf{o}$

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* → *Candidates* → *Disambiguation* → *Transforms* → *Validation*. Five strategy types are chained as ordered fallbacks: key-based matching (exact, contains, fuzzy, regex against `KEY→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`,

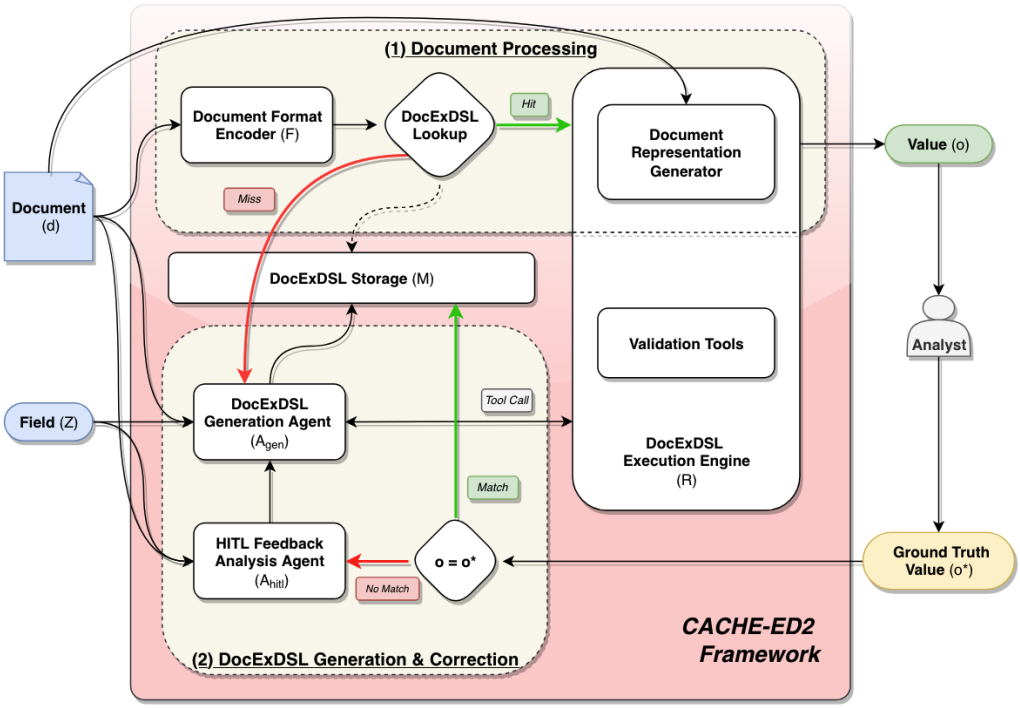


Figure 1. **CACHE-ED2 Framework.** On a cache *hit*, the stored DocExDSL program $\mathcal{P}_f$ is executed deterministically by $\mathcal{R}$ without LLM involvement. On a *miss*, $\mathcal{A}_{\text{gen}}$ synthesizes a new program via ReAct reasoning using $\mathcal{R}$ as a tool. Analyst corrections 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.

`contains`, and bidirectional `adjacency` edges (extending CACHE-ED1's representation (Bhoi & S, 2025)). The engine constructs 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 $\mathcal{C} = \{c_1, \ldots, c_K\}$ via cosine similarity; matches above threshold τ route to $\mathcal{M}[v, f]$, while unmatched documents trigger $\mathcal{A}_{\text{gen}}$. The model encodes the first page, as format is recognizable from it in the vast majority of enterprise documents.

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

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 $\Delta\mathcal{P} = \mathcal{A}_{\text{hitl}}(\mathcal{P}_f, \mathbf{o}, \mathbf{o}^*, d, \mathbf{s})$, such as adjusting anchor directions, adding derivation rules, or modifying disambiguation strategies. The field-level scores $\mathbf{s}$ guard against incorporating erroneous corrections: high-score fields have proven track records and are not overridden by single edits. If $\Delta\mathcal{P} \neq$ `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 & 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. Crucially, all CORD fields are extracted as-is from the document surface, with no SOP-based derivation required, isolating raw extraction capability from business-rule reasoning. We omit the vendor-SOP baseline as CORD carries no vendor-specific rules. Table 1(a) reports results across eight fields.

The single-pass LLM achieves 0.91 average accuracy but has no opportunity to correct errors. Both CACHE-ED variants incorporate a feedback loop and improve to 0.98 (CACHE-ED1) and 0.99 (CACHE-ED2), confirming the value of closed-loop refinement. Notably, CACHE-ED2 executes all extractions through the deterministic DocExDSL engine at inference, with no LLM in the loop, yet matches direct LLM extraction quality, demonstrating that compiling to DocExDSL preserves extraction fidelity.

Since CORD requires no SOP derivation, all methods perform well; real-world invoice processing demands business-rule reasoning that separates them. 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 performs well on easy fields but drops sharply on SOP-dependent fields (0.64 average accuracy) and fails to extract derived fields (59% extraction rate for Due Date), confirming that raw LLM extraction without business context is insufficient. Vendor-specific prompts close this gap (0.96 average on SOP-dependent fields) but remain static and cannot adapt to analyst corrections or handle multiple formats per vendor. CACHE-ED1 underperforms across all fields (0.81 average accuracy) because its text-only graph-path representation lacks the visual context needed for reliable spatial reasoning (Appendix D).

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 sending all format SOPs to the LLM simultaneously. Refer to Appendix F for agent-derived vs. human-authored SOP comparisons and Appendix E for representative DocExDSL 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. Table 2(a) compares CACHE-ED1's graph matching against CACHE-ED2's contrastive Document Format Encoder.

Table 1. Extraction quality on CORD v2 receipts and real-world invoices across four methods. Results reported as **Extraction Rate / Accuracy**; higher is better. Best results per field are 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 per document format on FATURA (50 layouts, grouped by visual variance) and real-world invoices (5 vendor formats); higher is better. CACHE-ED2's contrastive encoder achieves perfect detection across all groups.

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**

CACHE-ED1's hit rate degrades from 0.89 on the easiest group to 0.25 on the hardest, a 64 percentage point collapse, while CACHE-ED2 achieves perfect hit rate (1.0) across all five groups. We observe consistent results on production data across 5 vendor formats (Table 2(b)): CACHE-ED1 averages 0.47 due to two compounding fragilities: the document parser generates different graph representations for the same format under minor visual perturbations, and NER-based value masking fails to detect certain values, causing spurious new templates. CACHE-ED2 achieves perfect hit rate here as well (2.1× improvement), confirming that the learned embeddings generalize across dataset scale, domain, and document type. Refer to Appendix A for t-SNE visualizations of inter- and intra-class separation.

Training Details. We initialize the Document Format Encoder from the pre-trained LiLT-base checkpoint and attach a 3-layer MLP projection head mapping embeddings to a 512-dimensional vector. We train using triplet margin loss with margin $\alpha = 0.2$ and online semi-hard triplet mining on a dataset of vendor format pairs constructed from the top vendor formats.

4.3. Token Efficiency

We measure cumulative token consumption on the real-world invoice corpus as documents are processed sequentially (Figure 2). The single-pass LLM baseline consumes ∼13,360 tokens per document, totaling ∼13.4M 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 ∼5M tokens (∼62% reduction vs. baseline), while CACHE-ED1 consumes ∼9M tokens (∼31% reduction), ∼1.8× 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. The step-like plateaus in CACHE-ED2's curve reflect sustained cache-hit periods with zero LLM calls.

4.4. Learning Dynamics

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 improves gradually, plateauing well below acceptable levels. The slow learning stems from the single actor-critic agent pass requiring multiple document instances to accumulate sufficient signal,

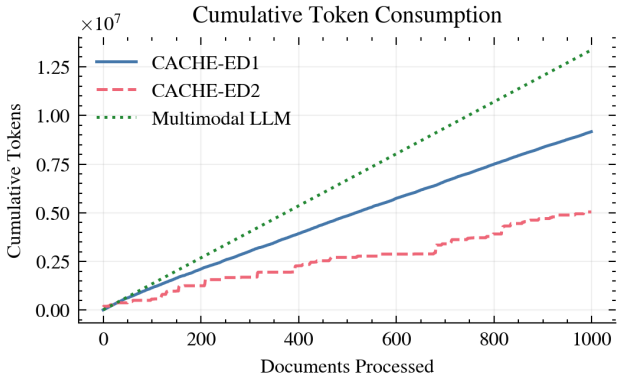


Figure 2. Cumulative token consumption over 1,000 documents; lower is better. The LLM baseline (dashed) grows linearly. CACHE-ED2 achieves ∼62% reduction; CACHE-ED1 achieves ∼31%.

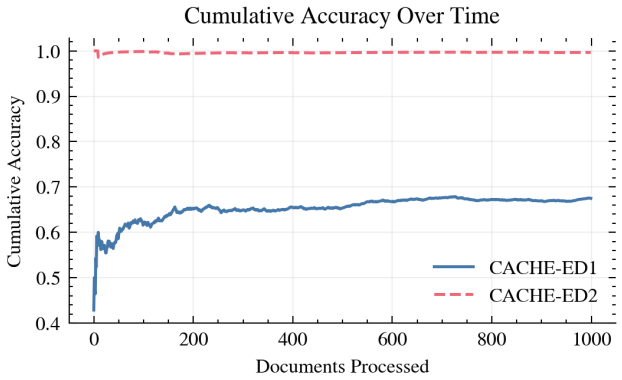


Figure 3. Cumulative accuracy over sequential documents for a single real-world vendor; higher is better. CACHE-ED2 reaches near-perfect accuracy from the first document; CACHE-ED1 converges slowly and plateaus at low accuracy.

compounded by the text-only representation limiting spatial reasoning. CACHE-ED2 achieves near-perfect accuracy from the first document instance and maintains it throughout: the ReAct agent's ability to iteratively test and correct DocExDSL programs via tool calls, combined with full document image visibility, enables accurate SOP derivation in a single encounter with a new format.

4.5. CACHE-ED1 Limitations

The experiments reveal three systematic limitations of CACHE-ED1. First, the graph-path-based document representation provides the LLM with cryptic textual paths rather than the full document image, limiting spatial reasoning for anchor-based and derived extractions. Second, Template Representation graph matching is fragile to OCR variance: minor differences in document parsing output produce different representations, causing cache misses that cascade into repeated LLM calls. Third, the single actor-critic agent pass lacks the ability to execute and verify its own corrections, requiring multiple document instances to converge. Detailed failure analyses are provided in Appendix D.

5. Conclusion

We presented CACHE-ED2, a framework that repositions the LLM from an instance-level data analyst to a system-level developer synthesizing reusable DocExDSL programs for visually rich document extraction. The framework advances its predecessor through four contributions: a contrastive Document Format Encoder achieving perfect cache hit rates across 50+ templates on both public (FATURA) and production data, a multimodal ReAct-based Generation Agent deriving extraction logic in a single encounter via iterative tool-use, a five-stage DSL with automated reasoning constraints ensuring deterministic, explainable, and bounding-box-grounded extraction, and a HITL Feedback Analysis Agent that refines cached programs from analyst corrections. On public benchmarks (CORD v2, FATURA) and 5,000 real-world invoices across 5 vendors, CACHE-ED2 achieves near-perfect extraction accuracy (0.99 average) with 1.0 extraction rate, matching or exceeding manually authored vendor-specific prompts, while reducing cumulative token consumption by ∼62% over 1,000 documents relative to the single-pass LLM baseline, a gap that widens further at scale.

Several directions remain. First, DocExDSL programs in $\mathcal{M}$ can be optimized through cross-vendor pollination, synthesizing common extraction patterns across formats within a vertical to reduce cold-start latency for new vendors. Second, the DSL itself can evolve: when $\mathcal{A}_{\text{gen}}$ identifies patterns inexpressible in the current language, these gaps can drive continuous capability expansion. Third, critic agent reasoning can be improved via reinforcement learning on human-provided SOPs from high-volume vendors, strengthening its ability to interpret analyst feedback. Finally, the Document Format Encoder currently operates on single-page text-and-layout input; future iterations can incorporate visual features (as in LayoutLMv3 (Huang et al., 2022)) and multi-page context for complex document types.

LLM/Agent Usage Disclosure

During the preparation of this manuscript, the authors used a large language model to improve the readability and grammatical clarity of author-written drafts. The authors thoroughly reviewed and edited all LLM-assisted content and take full responsibility for the content of this paper.

References

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

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

Backes, J., Bolignano, P., Cook, B., Dodge, C., Gacek, A., Luckow, K., Rungta, N., Tkachuk, O., and Varming, C. Semantic-based automated reasoning for AWS access policies using SMT. In *Proceedings of the 18th Conference on Formal Methods in Computer-Aided Design (FMCAD)*. IEEE, 2018. Introduced the Zelkova tool for policy analysis at AWS.

Bhoi, S. Vig-llm: Enhancing visual grounding capabilities in closed-box llms for document information extraction without ocr dependencies. 2026. URL https://www.amazon.science/publications/vig-llm-enhancing-visual-grounding-capabilities-in-closed-box-llms-for-document-information-extraction-without-ocr-dependencies.

Bhoi, S. and S, H. Y. V. Cache-ed: Redefining document entity extraction with graph-based templates, actor-critic agents & hil. 2025. URL https://www.amazon.science/publications/cache-ed-redefining-document-entity-extraction-with-graph-based-templates-actor-critic-agents-hil.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Siber, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pp. 1877–1901, 2020. arXiv:2005.14165. Introduced GPT-3 and the in-context learning paradigm.

Cao, L., Hao, G., and Liang, Y. Digital asset management tool for film animation and game development. *Computer-Aided Design and Applications*, 20(S8):89–99, 2023. doi: 10.14733/cadaps.2023.S8.89-99.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F. P., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W. H., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A. N., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I., and Zaremba, W. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021. Introduces Codex (GPT fine-tuned on GitHub code); powers GitHub Copilot.

Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., Wang, M., and Wang, H. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2024. Comprehensive survey covering RAG paradigms, components, and evaluation for LLMs.

Gas, S. and Jeuring, J. A DSL for describing the artificial intelligence in real-time video games. In *Proceedings of the 3rd International Workshop on Games and Software Engineering (GAS), co-located with ICSE 2013*, San Francisco, USA, 2013. IEEE.

Huang, P., Lai, S., et al. OCR-Reasoning benchmark: Unveiling the true capabilities of MLLMs in complex text-rich image reasoning. *arXiv preprint arXiv:2505.17163*, 2025. Accepted at ICLR 2026.

Huang, Y., Lv, T., Cui, L., Lu, Y., and Wei, F. 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, 2022. arXiv:2204.08387.

Khattab, O., Singhvi, A., Maheshwari, P., Zhang, Z., Santhanam, K., Vardhamanan, S., Haq, S., Sharma, A., Joshi, T. T., Mober, H., Shah, C., Potts, C., and Zaharia, M. DSPy: Compiling declarative language model calls into state-of-the-art pipelines. In *Proceedings of the 12th International Conference on Learning Representations (ICLR)*, 2024. arXiv:2310.03714.

Kim, G., Hong, T., Yim, M., Nam, J., Park, J., Yim, J., Hwang, W., Yun, S., Han, D., and Park, S. OCR-free document understanding transformer. In *European Conference on Computer Vision (ECCV)*. Springer, 2022. arXiv:2111.15664.

Kurowski, M. et al. Enabling the use of unstructured data for robotic process automation. *arXiv preprint arXiv:2507.11364*, 2025.

Lee, C.-Y., Li, C.-L., Dozat, T., Perot, V., Su, G., Hua, N., Ainslie, J., Wang, R., Fujii, Y., and Pfister, T. 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)*, pp. 3735–3754, Dublin, Ireland, 2022. Association for Computational Linguistics. arXiv:2203.08411.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., Riedel, S., and Kiela, D. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pp. 9459–9474, 2020. arXiv:2005.11401. Introduced the RAG framework combining parametric and non-parametric memory.

Li, Z., Abulaiti, A., Lu, Y., Chen, X., Zheng, J., Lin, H., Han, X., Jiang, S., Dong, B., and Sun, L. READoc: A unified benchmark for realistic document structured extraction. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 21889–21905, Vienna, Austria, 2025. Association for Computational Linguistics. arXiv:2409.05137.

Liao, H., RoyChowdhury, A., Li, W., Bansal, A., Zhang, Y., Tu, Z., Satzoda, R. K., Manmatha, R., and Mahadevan, V. DocTr: Document transformer for structured information extraction in documents. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE, 2023. arXiv:2307.07929.

Liao, L. et al. DocLayLLM: An efficient multi-modal extension of large language models for text-rich document understanding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2025. arXiv:2408.15045.

Limam, M., Dhiaf, M., and Kessentini, Y. FATURA: A multi-layout invoice image dataset for document analysis and understanding. *arXiv preprint arXiv:2311.11856*, 2023.

Luo, C., Cheng, Z., Huang, G., and Peng, Y. LayoutLLM: Layout instruction tuning with large language models for document understanding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2024. arXiv:2404.05225.

Makatura, L., Zhu, B., Sun, Y.-L., Tjandrasuwita, M., Guo, J., and Matusik, W. MetaGen: A DSL, database, and benchmark for VLM-assisted metamaterial generation. *arXiv preprint arXiv:2508.17568*, 2025. Introduces MetaDSL for compact, semantically rich metamaterial design; presented at NeurIPS 2025.

Microsoft. Document intelligence supported Markdown elements. https://learn.microsoft.com/en-us/azure/ai-services/document-intelligence/concept/markdown-elements, 2025. Azure AI Document Intelligence Layout API transforms complex PDFs into semantically structured Markdown.

OpenSCAD Contributors. OpenSCAD: The programmers solid 3D CAD modeller. https://openscad.org/, 2019. Google Season of Docs 2019. Script-based CSG modeler enabling parametric 3D design via declarative DSL.

Park, S., Shin, S., Lee, B., Lee, J., Surh, J., Seo, M., and Lee, H. CORD: A consolidated receipt dataset for post-OCR parsing. In *Document Intelligence Workshop at Neural Information Processing Systems*, 2019.

Quantiphi. From documents to insights: How multimodal LLMs elevate key information extraction (KIE). https://quantiphi.com/blog/from-documents-to-insights-how-multimodal-llms-el 2025. Accessed: 2026-04-16. Discusses closed-box API advantages for frontier model access without GPU overhead.

Ramos, D., Gamboa, C., Lynce, I., Manquinho, V., Martins, R., and Le Goues, C. SPELL: Synthesis of programmatic edits using LLMs. *arXiv preprint arXiv:2602.01107*, 2026. Distills latent migration knowledge from LLMs into structured, testable, repeatable programmatic logic.

Rozière, B., Gehring, J., Gloeckle, F., Sootla, S., Gat, I., Tan, X. E., Adi, Y., Liu, J., Sauvestre, R., Remez, T., Rapin, J., Kozhevnikov, A., Evtimov, I., Bitton, J., Bhatt, M., Ferber, C. C., Grattafiori, A., Xiong, W., Défossez, A., Copet, J., Azhar, F., Touvron, H., Martin, L., Usunier, N., Scialom, T., and Synnaeve, G. Code Llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2024. Family of code LLMs based on Llama 2; state-of-the-art open models for code generation.

Sapkota, H. et al. AI agents vs. agentic AI: A conceptual taxonomy, applications and challenges. *arXiv preprint arXiv:2505.10468*, 2025.

Stahlberg, F. et al. Problem solved? Information extraction design space for layout-rich documents using LLMs. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 17908–17927, Suzhou, China, 2025. Association for Computational Linguistics. arXiv:2502.18179.

Suzgun, M. and Kalai, A. T. Meta-prompting: Enhancing language models with task-agnostic scaffolding. *arXiv preprint arXiv:2401.12954*, 2024. Transforms a single LM into a conductor orchestrating multiple expert LM instances.

Wang, D., Raman, N., Sibue, M., Ma, Z., Babkin, P., Kaur, S., Pei, Y., Nourbakhsh, A., and Liu, X. DocLLM: A layout-aware generative language model for multimodal document understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8529–8548, Bangkok, Thailand, 2024. Association for Computational Linguistics. arXiv:2401.00908.

Wang, J., Jin, L., and Ding, K. 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)*, pp. 7747–7757. Association for Computational Linguistics, 2022. URL https://aclanthology.org/2022.acl-long.534.

Wang, W., Hu, H., Zhang, Z., Li, Z., Shao, H., and Dahlmeier, D. Document intelligence in the era of large language models. *arXiv preprint arXiv:2510.13366*, 2025. Survey on LLMs for Document AI.

Xu, Y., Li, M., Cui, L., Huang, S., Wei, F., and Zhou, M. LayoutLM: Pre-training of text and layout for document image understanding. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 1192–1200. ACM, 2020. arXiv:1912.13318.

Yang, J., Zhang, H., Li, F., Zou, X., Li, C., and Gao, J. Set-of-mark prompting unleashes extraordinary visual grounding in GPT-4V. *arXiv preprint arXiv:2310.11441*, 2023.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. ReAct: Synergizing reasoning and acting in language models. In *Proceedings of the 11th International Conference on Learning Representations (ICLR)*, 2023. URL https://arxiv.org/abs/2210.03629.

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 (∼125M parameters) and train only a 3-layer MLP projection head (768→512→256→256, ReLU, L2-normalized output) — 590K trainable parameters (0.47% of total). The model is trained with triplet margin loss (α=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 × 4 samples each. Training runs 30 epochs with AdamW (lr=1e-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 (τ=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.

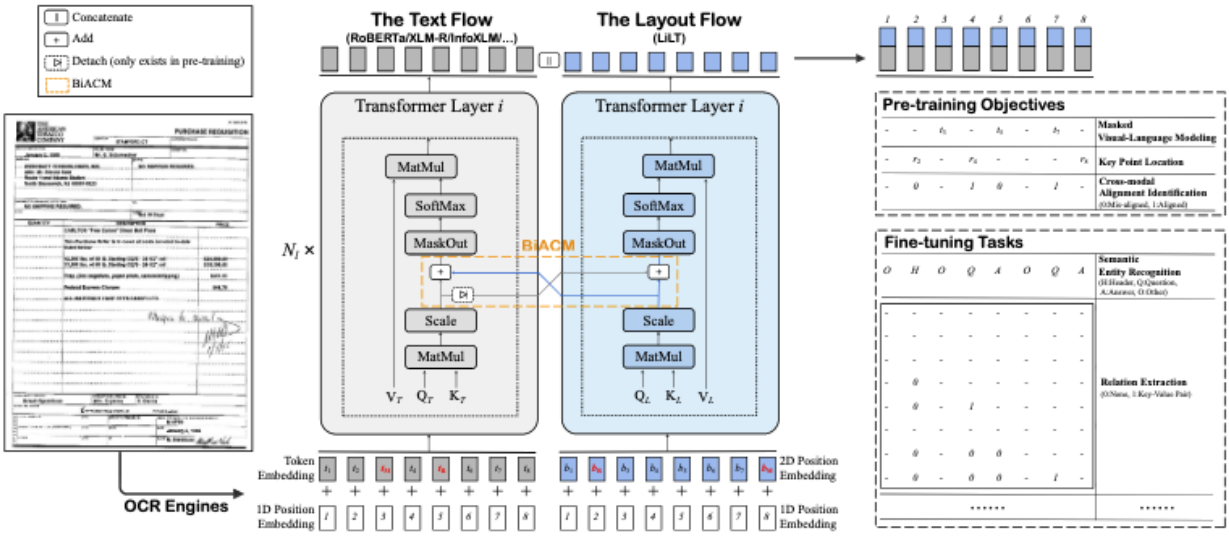


Figure 4. LiLT dual-stream architecture. The text flow (RoBERTa, 768-dim) and layout flow (192-dim) run in parallel with BiACM sharing attention scores bidirectionally at each layer.

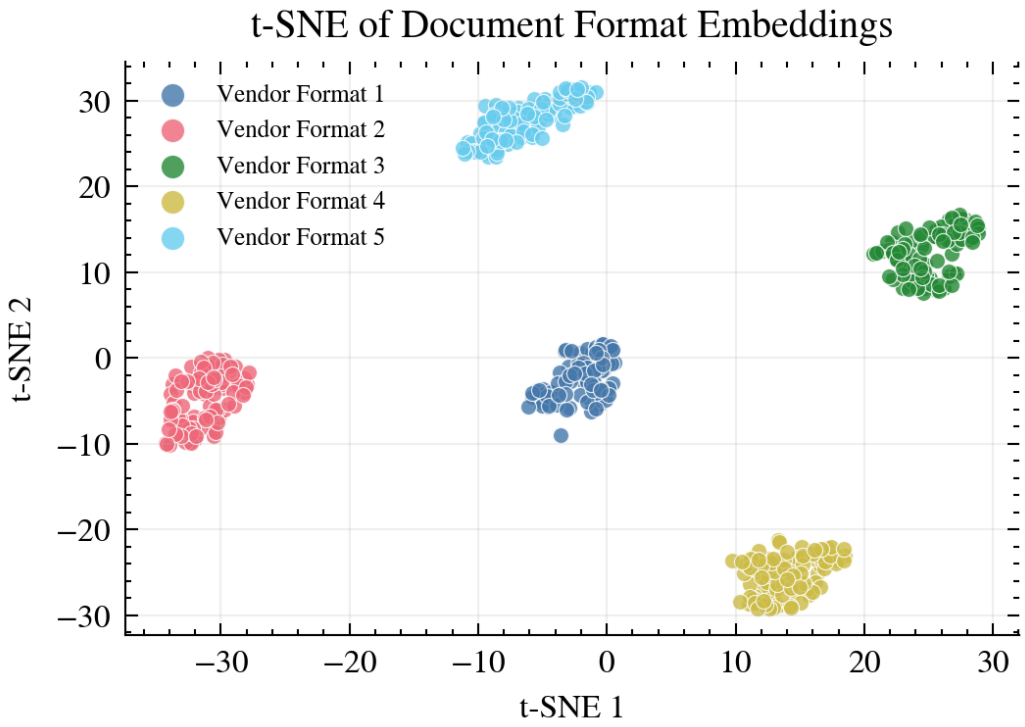


Figure 5. t-SNE projection of 256-dimensional format embeddings. Each color represents a vendor format. Intra-class cosine similarity averages 0.9954; inter-class averages 0.3704 (gap: 0.6250).

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 ReAct 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

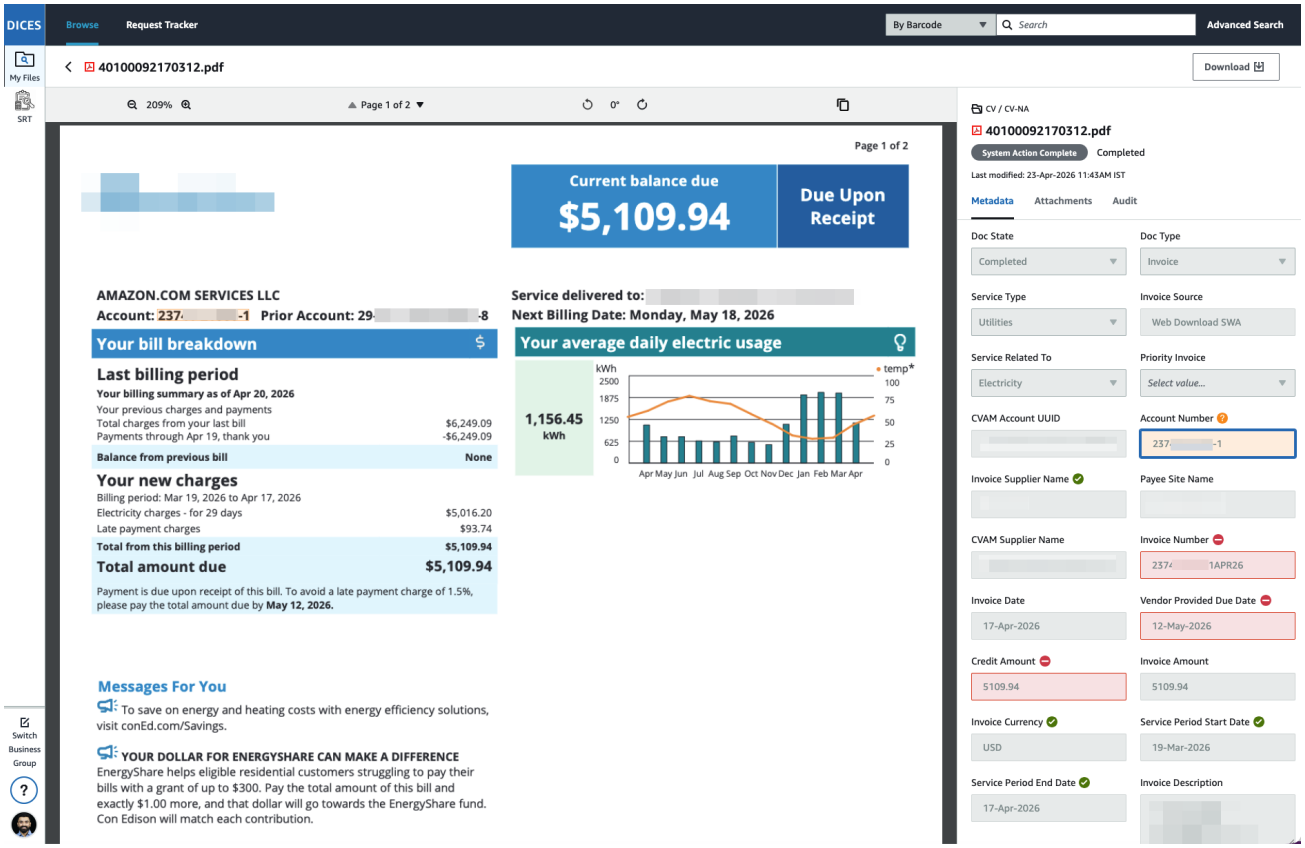


Figure 6. The human-in-the-loop validation interface used by analysts to verify and correct extracted values. Extracted fields are color-coded by confidence level to direct analyst attention, with bounding-box overlays providing spatial grounding for each extraction.

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. 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 $\leq$ 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 Section 4.5.

Template Fragility Under OCR Variance. CACHE-ED1 constructs templates by anonymizing VALUE nodes and computing structural fingerprints via graph isomorphism. Figure 7 shows two instances of the same vendor format where the document parser detects the site identifier as "`AMAZON SITE`" in one and "`AMAZON 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 8 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.

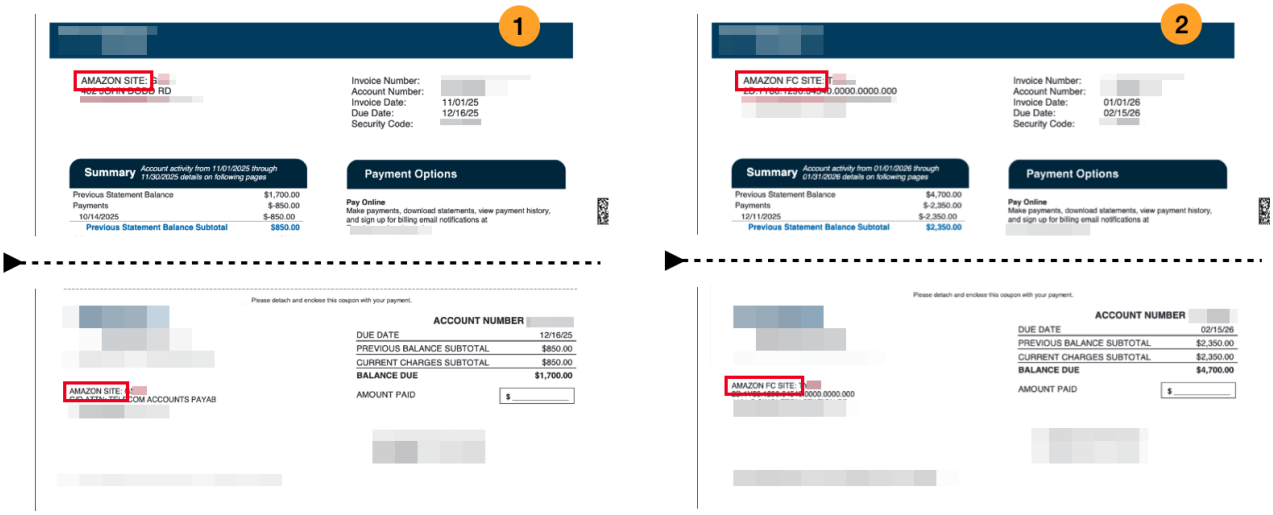


Figure 7. Template fragility under OCR variance. Two instances of the same format where a single-word KEY difference ("AMAZON SITE" vs. "AMAZON FC SITE") produces distinct anonymized graphs, causing a cache miss.

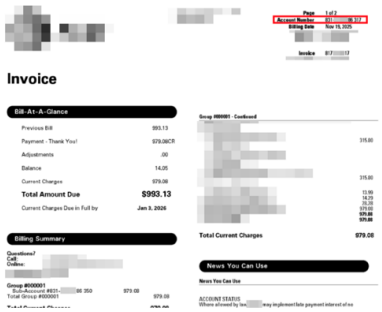
Document Image Sample	LLM Input	LLM Output	Expected Output
	`... 5. Key: XX 317 is associated with value(s): 831-NNN-NN86. There are surrounding key-value pairs are: Account Number -> , XX WA XX -> SEATTLE, 98108 to the above key that can provide more context if required. ... 79. Key: Account Number 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>317</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", suggesting the full account number context is "317" with sub-account "831-NNN-NN86 350".</explanation> </account_number>`	**831-NNN-NN86 317**

Figure 8. 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.

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 9 summarizes the stages.

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.

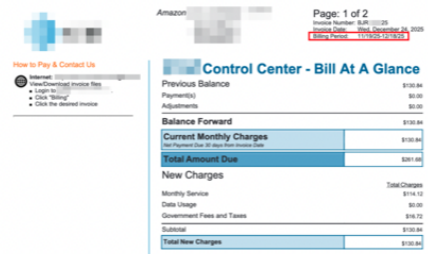
Figure 10 demonstrates the complementary case for the Stage 2. After 78 successful validations, the `vendor_provided_due_date` rule (`invoice_date + 45 days`) carries a score of 78. An analyst submits "3/14/2026" for invoice date "1/27/2026" — one day off from the computed "3/13/2026". The critic issues `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.

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 11 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.

Input (Document/HITL)	Agent	Output (DocExDSL/Feedback)	Extracted Value

	DocExDSL Generation Agent	`# 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"),`
			11/19/25

Dec 01, 2025	HITL Feedback Analysis Agent	`### 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.`	

	DocExDSL Generation Agent	`# 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"),`
			12/01/2025

Figure 9. 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.

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.

```
### 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.

Failure Mode	Human Curated SOP	CACHE-ED2 Derived SOP

No Label	To find **Account Number** use the highlighted area
	Eg. AZ[redacted]SX5
	 
		`Field("account_number", type="string", required=True)`
		`.extract_by_anchor("Amazon -", direction="nearest", offset=0.0, match="contains")`
		`.on_pages(2)`
		`.subset(pattern=r"[A-Z]{3}[A-Z0-9]+")`
		`.disambiguate("first_occurrence")`

Derived Information	To find **Invoice Number**
	1. Check if Invoice Number is explicitly mentioned in the document. Only follow below steps if NOT explicitly present. Else use Invoice Number present.
	2. Extract Account Number Eg. 408[redacted]635 6
	3. Extract Month and Year from Invoice Date Eg. Dec 1, 2025
	4. Get Invoice Number using the Account Number + Month Year of Invoice Date Eg. 408[redacted]6356DEC2025 (Note: no space between the month and account number)
	 
		`Field("invoice_month_abbr", type="string")`
		`.derive_from("invoice_date_raw")`
		`.subset(pattern=r"^[A-Za-z]{3}")`
		`.transform(case="upper"),`
		`Field("invoice_year_2digit", type="string")`
		`.derive_from("invoice_date_raw")`
		`.subset(pattern=r"\d{4}$")`
		`.subset(pattern=r"\d{2}$"),`
		`Field("invoice_number", type="string", required=True)`
		`.derive_from("account_number_digits", "invoice_month_abbr", "invoice_year_2digit")`
		`.transform(template="{account_number_digits}{invoice_month_abbr}{invoice_year_2digit}")`

Disambiguation	To find **Invoice Number**
	1. Consider the highlighted header level occurrence of Invoice Number and NOT the line level occurrences
	 
		`Field("invoice_number", type="string", required=True)`
		`.extract_by_key("INVOICE NUMBER", "invoice number", match="contains")`
		`.or_extract_by_anchor("INVOICE NUMBER:", direction="right", offset=0.3)`
		`.disambiguate("first_occurrence")`

Figure 11. Human-curated vs. CACHE-ED2 derived SOPs across three failure modes: unlabeled information, derived information, and disambiguation. The framework produces functionally equivalent extraction logic from analyst corrections alone.