CogExtract: Test-Time Search with Execution-Grounded Verification for Web Information Extraction on Dynamic Pages

Anonymous authors
Paper under double-blind review

Abstract

Web information extraction systems often generate reusable extraction rules from a representative page and apply them to related pages, but such rules can silently fail under local structural variations. Existing approaches provide limited mechanisms for cross-page verification and targeted recovery once generated rules fail. We present **CogExtract**, which reframes wrapper generation as test-time search with execution-grounded verification and failure-conditioned repair. CogExtract generates diverse XPath candidates and evaluates them using deterministic execution signals from a seed page and an unlabeled sibling verification page. When no candidate passes verification, the same execution evidence is used to diagnose the failure and guide targeted candidate regeneration. We evaluate CogExtract on LiveWeb-IE, covering 14 websites, 312 page groups, and 50,524 URLs, across 11 model backbones. CogExtract consistently improves performance across all backbones, outperforming VGS by 14.5 F1 points on average with only a 13% increase in model calls; with GPT-5, it reaches 76.6 F1. Execution-grounded verification also improves candidate discrimination over LLM self-evaluation (AUC 0.840 vs. 0.640; best-selection P@1 96.2% vs. 77.9%). On an audited subset covering perceptual and rule-formulation failures, structured diagnosis over execution evidence achieves 91.0% accuracy, compared with 59.0% for direct LLM judgment. These results show that execution feedback provides an effective signal for both candidate selection and targeted failure recovery in dynamic web information extraction. Our code[1] is publicly available.

1 Introduction

Web information extraction (WIE) aims to convert unstructured web content into structured data in response to user-specified information needs (Wang et al., 2023a). Traditional WIE systems rely on hand-crafted or automatically learned wrappers that exploit recurring DOM structural patterns across related pages, enabling efficient large-scale extraction but requiring substantial effort to construct and maintain (Crescenzi et al., 2002; Arasu & Garcia-Molina, 2003). Recent multimodal large language models (MLLMs) have substantially improved semantic understanding and visual grounding on web pages. (Qi et al., 2024; He et al., 2024; Zheng et al., 2024; Khalighinejad et al., 2025). A particularly relevant line of work uses foundation models to generate reusable extraction scripts or wrappers from representative pages, which can then be executed deterministically across related pages, combining the flexibility of foundation models with the efficiency of rule-based extraction. Recent work such as LiveWeb-IE and its Visual Grounding Scraper (VGS) (Yang et al., 2026) demonstrates the effectiveness of visual grounding for generating reusable wrappers on live, structurally evolving web pages.

Figure 1(A) illustrates a central limitation of this *generate-and-reuse* paradigm. A wrapper that succeeds on the representative seed page may silently fail on a sibling page when its local DOM structure varies, even though the pages belong to the same nominal template. In the example, the generated XPath correctly extracts the price from the seed page and several related pages, but returns an empty result on a sibling page when a local structural variation breaks its anchor. Seed-page success therefore does not imply cross-page reliability. Because reusable-script and wrapper-generation methods

[1] `https://anonymous.4open.science/r/984C/`

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

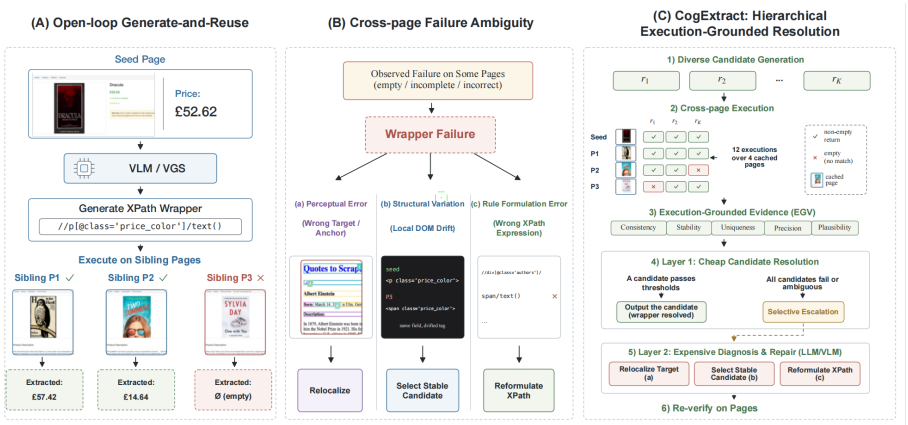


Figure 1: **Motivation and overview of CogExtract.** (A) A wrapper generated from a seed page may silently fail on structurally related sibling pages. (B) Similar observed failures may correspond to different repair-relevant failure modes and therefore provide different evidence for subsequent repair. (C) CogExtract first searches over diverse wrapper candidates using cross-page execution-grounded verification, and selectively escalates unresolved cases to evidence-conditioned diagnosis and targeted candidate regeneration.

often provide limited cross-page execution feedback after rule generation, such failures may remain undetected during deployment.

Detection alone, however, is insufficient for recovery. As shown in Figure 1(B), similar observed symptoms—such as an empty, incomplete, or incorrect extraction—may correspond to different *repair-relevant failure modes*. A perceptual error may ground the wrapper to an inappropriate target or anchor (Zheng et al., 2024; He et al., 2024); cross-page structural variation may make a seed-specific anchor unstable on related pages; and a rule formulation error may leave the grounding approximately correct while expressing the XPath too broadly, too narrowly, or otherwise improperly (Wang et al., 2023a). These modes are operationally important because they provide different evidence about how the failed candidate space should be revised. The challenge is therefore not only to detect that a wrapper has failed, but also to use cross-page evidence to determine how subsequent search should be redirected.

Figure 1(C) presents our solution, **CogExtract**, which introduces an execution-grounded feedback loop for reliable wrapper generation. Instead of treating execution as a post-hoc validation step, CogExtract leverages cross-page execution behavior as feedback throughout the test-time inference process. Given a seed-page grounding, CogExtract first explores a diverse candidate space and evaluates candidate wrappers according to their execution consistency across the seed page and a sibling verification page. This execution evidence enables Layer 1 resolution, where reliable candidates can be selected directly without additional model-based diagnosis. When none of the candidates satisfies the cross-page verification criteria, CogExtract enters Layer 2 and reuses the accumulated execution evidence to identify repair-relevant failure modes and guide targeted candidate regeneration. The newly generated candidates are then returned to the same execution-grounded verification process, forming a closed execution–repair loop. Concretely, Diversity-Conditioned Generation (DCG) constructs diverse wrapper hypotheses, Execution-Grounded Verification (EGV) provides deterministic signals for candidate evaluation, and Failure-Conditioned Reflection (FCR) converts execution failures into structured feedback for subsequent search. Therefore, candidate diversity serves not only as a source of alternative solutions but also as a basis for diagnosing failures and expanding the search space beyond blind regeneration.

We evaluate CogExtract on LiveWeb-IE, a dynamic web extraction benchmark spanning 14 real-world domains, 312 page groups, and 50,524 extraction instances across four task types. Across 11 model backbones, CogExtract improves over VGS by $+14.5$ F1 on average, with GPT-5 reaching 76.6 F1. Mechanism-level analyses show that 41.7% of audited initial failures are resolved through candidate selection without invoking FCR, and that EGV substantially outperforms LLM self-evaluation in candidate discrimination (AUC 0.840 vs. 0.640). Ablations further demonstrate complementary roles for DCG, EGV, and FCR in candidate-space construction, execution-grounded selection, and targeted recovery, respectively.

Our contributions are threefold.

1. We identify cross-page failure after wrapper generation as a key limitation of the generate-and-reuse paradigm, and formulate wrapper repair as an execution-grounded process that uses cross-page evidence for both candidate selection and targeted recovery.
2. We introduce CogExtract, an execution-grounded framework consisting of Diversity-Conditioned Generation (DCG), Execution-Grounded Verification (EGV), and Failure-Conditioned Reflection (FCR). The framework leverages execution feedback to construct, evaluate, and refine wrapper hypotheses.
3. We evaluate CogExtract extensively on LiveWeb-IE across 14 domains, 50,524 extraction instances, and 11 VLMs. CogExtract improves over VGS by $+14.5$ F1 on average, while GPT-5 reaches 76.6 F1.

2 Method

CogExtract performs execution-grounded wrapper resolution through a two-layer feedback process. Layer 1 uses cross-page execution evidence to evaluate and select candidate wrappers, while Layer 2 reuses the same evidence to diagnose failures and guide targeted candidate regeneration.

As shown in Figure 2, CogExtract consists of four stages: **(1) Visual Anchoring** localizes the target element and constructs the local context for rule synthesis; **(2) Diversity-Conditioned Generation (DCG)** generates a diverse set of XPath hypotheses using different structural strategies; **(3) Execution-Grounded Verification (EGV)** executes and evaluates the candidates across the seed and verification pages using deterministic cross-page signals; and **(4) Failure-Conditioned Reflection (FCR)**, triggered only when no candidate passes verification, uses the accumulated execution evidence to diagnose a repair-relevant failure mode and guide targeted candidate regeneration. The newly generated candidates are then returned to EGV for re-evaluation, closing the execution–diagnosis–repair loop. Visual Anchoring provides the shared initialization, Stages (2) and (3) constitute Layer 1, and Stage (4) forms the selectively invoked Layer 2.

2.1 Task Definition

Given a natural-language query q and a family of related pages that share a common template while allowing local structural variation, the goal is to generate, for each target attribute $a \in \mathcal{A}$, a reusable XPath rule r_a that generalizes across the family. The ideal objective,

$$r_a^* = \arg\max_{r_a \in \mathcal{R}} \mathbb{E}_{p \sim \mathcal{D}_\mathcal{P}}[F_1(r_a(p), y_a(p))], \tag{1}$$

is unattainable at test time: ground-truth values $y_a(p)$ are unavailable, $\mathcal{R}$ is combinatorially large, and only a small number of pages are observed. CogExtract therefore performs test-time candidate exploration over a finite candidate space using unlabeled execution-grounded evidence as a proxy for extraction quality. DCG constructs an initial candidate set $\mathcal{C}_a^{(0)} = \{r_a^{(1)}, \ldots, r_a^{(K)}\} \subset \mathcal{R}$. EGV evaluates each candidate on a seed page p_{seed} and a verification page p_{val} using a deterministic cross-page score $S(r_a; p_{\text{seed}}, p_{\text{val}})$ (§2.4). Among candidates that pass the verification criterion, the system selects $\hat{r}_a = \arg\max_{r_a \in \mathcal{C}_a^{(t)}:\text{Pass}(r_a)=1} S(r_a; p_{\text{seed}}, p_{\text{val}})$. This constitutes Layer 1 and requires no explicit failure diagnosis. When no candidate passes verification, the system enters Layer 2. FCR examines the execution traces of the failed candidates together with the local DOM context, diagnoses a repair-relevant failure mode $\hat{c}_a \in \{\text{PE}, \text{SV}, \text{RF}\}$, and uses this diagnosis to generate targeted new candidates $\mathcal{C}_a^{\text{new}}$. The candidate space is expanded to $\mathcal{C}_a^{(t+1)} = \mathcal{C}_a^{(t)} \cup \mathcal{C}_a^{\text{new}}$ and re-enters EGV. Thus, CogExtract reformulates wrapper generation as *candidate-space construction, execution-grounded selection, and failure-conditioned search-space expansion*.

2.2 Visual Anchoring

Before constructing XPath candidates, CogExtract localizes the page elements corresponding to the requested attributes. Direct search over the full DOM yields a large and noisy hypothesis space; we therefore adopt the visual grounding procedure of VGS as the front-end localization module, progressively reducing the observation space from the rendered page to a compact local structural context. This stage consists of three steps.

Attribute identification. Given a natural-language query q, the LLM decomposes it into a structured set of target attributes $\mathcal{A} = \{a_1, \ldots, a_m\}$. Each attribute is then processed independently in

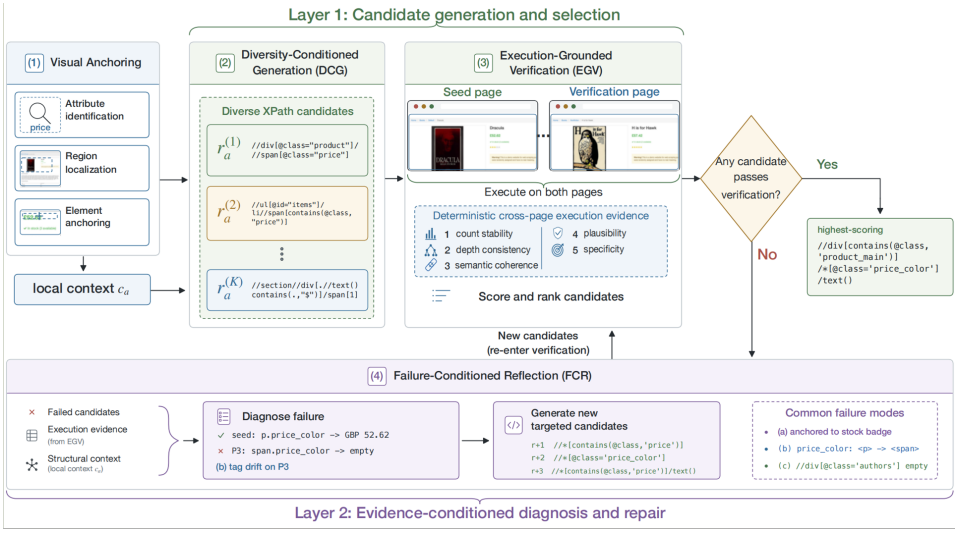


Figure 2: **CogExtract pipeline.** Visual anchoring produces the local context for XPath synthesis. Layer 1 generates diverse candidates with DCG and selects a verified rule using deterministic cross-page evidence from EGV. If no candidate passes verification, Layer 2 invokes FCR to diagnose the failure and generate targeted candidates, which return to EGV for re-evaluation.

the subsequent grounding and wrapper-generation stages. **Region localization.** The rendered page is divided into a sequence of vertical viewport regions $\mathcal{V} = \{v_1, \ldots, v_k\}$, each represented by a screenshot. For each target attribute a, the VLM evaluates these regions and selects the most relevant one, v_a^*, restricting subsequent grounding to a focused portion of the page. **Element anchoring.** Within the selected region v_a^*, modality-specific heuristics construct a set of candidate elements. For textual attributes, candidates are obtained from visually identified text elements, whereas image and link attributes are additionally constrained by their corresponding HTML tags. Set-of-Mark prompting assigns numbered visual markers to the candidates, and the VLM selects the subset $\mathcal{E}_a$ that best corresponds to attribute a. The associated DOM nodes, together with their neighboring HTML structure, form the local context $c_a = \text{Context}(\mathcal{E}_a, d)$, where d denotes the DOM neighborhood radius around the anchored elements. The resulting context c_a serves as the input to Diversity-Conditioned Generation in Section 2.3.

2.3 DCG:Diversity-Conditioned Generation

Generating only one XPath leaves the system with neither alternative hypotheses for candidate resolution nor comparative evidence for diagnosing failures. CogExtract therefore constructs a diverse candidate space rather than committing to a single rule. The resulting candidates serve two purposes: they provide alternative hypotheses for direct selection in Layer 1, and, when none passes verification, their different execution outcomes provide comparative evidence for subsequent failure diagnosis. Given the local context c_a obtained from §2.2, Diversity-Conditioned Generation (DCG) prompts the VLM to generate K XPath candidates in a single model invocation, $\mathcal{C}_a^{(0)} = \text{VLM}(c_a, \text{prompt}_{\text{modal}}) = \{r_a^{(1)}, \ldots, r_a^{(K)}\}$, where $\text{prompt}_{\text{modal}}$ is selected according to the attribute modality and explicitly encourages structural diversity across the generated rules. In particular, candidates are encouraged to use different anchoring strategies, such as class-based selectors, positional relations, and attribute matching. Generating the candidates jointly in a single VLM invocation avoids the cost of K independent model calls while providing a broader candidate space for subsequent verification.

2.4 Execution-Grounded Verification

Pages from the same page family generally preserve substantial structural regularity despite local DOM variations. A reusable XPath should therefore exhibit stable extraction behavior across related pages. However, non-empty execution alone does not guarantee correctness, as an XPath may consistently match irrelevant or overly broad elements. Execution-Grounded Verification (EGV) evaluates candidates using deterministic cross-page execution evidence without requiring ground-truth labels. Given a seed page p_{seed} and a verification page p_{val}, each candidate $r_a^{(k)}$ is executed on both pages.

EGV consists of a binary cross-page eligibility gate followed by a five-signal ranking function. A candidate is considered eligible only if it produces non-empty extraction results on both pages. This criterion ensures execution feasibility, but not semantic correctness. Each eligible candidate is then scored by a weighted aggregation of five complementary execution-grounded signals:

$$S(r_a^{(k)}; p_{\text{seed}}, p_{\text{val}}) = \sum_{i=1}^{5} w_i s_i(r_a^{(k)}; p_{\text{seed}}, p_{\text{val}}).$$

The highest-scoring eligible candidate is selected. The five signals capture complementary aspects of cross-page behavior: extraction-count stability, DOM-depth consistency, value-form coherence, attribute-specific plausibility, and extraction specificity. All signals are computed deterministically from execution outputs and DOM structures without additional LLM calls. Their formal definitions and implementation details are provided in Appendix L. If no candidate satisfies the eligibility gate, EGV passes the candidate-level execution evidence to FCR, triggering the evidence-conditioned repair process described in Section 2.5.

2.5 Failure-Conditioned Reflection

When no candidate passes the cross-page eligibility check of Section 2.4, CogExtract escalates to Failure-Conditioned Reflection (FCR). Rather than treating unsuccessful executions merely as rejection signals, FCR reuses the accumulated execution evidence to diagnose a repair-relevant failure mode and guide targeted candidate regeneration. FCR is invoked only when the current candidate space contains no cross-page eligible candidate; candidates that remain non-empty on both pages are handled within Layer 1 by EGV ranking. We distinguish three operationally distinct failure modes: **(a) perceptual error**—the target or anchor is mislocalized, requiring re-grounding; **(b) structural variation**—a seed-specific anchor fails to generalize, requiring a more cross-page-stable anchoring strategy; and **(c) rule formulation error**—the grounding is approximately correct but the XPath fails cross-page verification, requiring reformulation. These modes imply different directions for revising the current extraction hypothesis. For the failed candidate set $\mathcal{C}_a^{\text{fail}}$, FCR constructs deterministic diagnostic evidence from execution outcomes, match-count profiles, DOM depths, shared structural anchors, and relevant local HTML context, yielding $\mathcal{E}_a = \Phi(\mathcal{C}_a^{\text{fail}}, p_{\text{seed}}, p_{\text{val}})$. The procedure $\Phi(\cdot)$ deterministically constructs the diagnostic representation and contains no model-based attribution. The structured evidence is then provided to an LLM for failure-mode attribution, $\hat{c}_a = \text{LLM}_{\text{diag}}(\mathcal{C}_a^{\text{fail}}, \mathcal{E}_a, \text{prompt}_{\text{diag}}) \in \{\text{PE}, \text{SV}, \text{RF}\}$. Although the attribution itself is model-based, its input is a structured diagnostic representation constructed from deterministic execution evidence rather than unconstrained model self-assessment. The diagnosed mode is deterministically mapped to a repair strategy g_a: PE triggers target relocalization, SV invokes stable-anchor selection using cross-page structural evidence, and RF reformulates the XPath while preserving the current grounding. The VLM then expands the candidate space with new candidates conditioned on this repair direction, namely $\mathcal{C}_a^{\text{new}} = \text{VLM}(\mathcal{C}_a^{\text{fail}}, \mathcal{E}_a, \hat{c}_a, g_a, \text{prompt}_{\text{reflect}})$. The expanded space $\mathcal{C}_a^{(t+1)} = \mathcal{C}_a^{(t)} \cup \mathcal{C}_a^{\text{new}}$ then re-enters EGV for verification.

Overall, execution evidence serves two roles in the hierarchy: as a selection signal in Layer 1 and, when no eligible candidate remains, as diagnostic evidence for directing candidate-space expansion in Layer 2.

3 Experiments

Dataset. We evaluate on LiveWeb-IE, a real-world web information extraction benchmark covering diverse domains including academic repositories, e-commerce platforms, governmental portals, and structured reference databases. After excluding one inaccessible website with anti-scraping mechanisms, our evaluation contains 14 websites, 312 page groups, and 50,524 URLs. The benchmark covers four task types: single-attribute extraction targets one scalar field per page; multi-attribute extraction targets multiple scalar fields; list extraction returns repeated values for one field; and paired extraction returns aligned values for related fields. Each URL, together with its associated set of target attributes, is counted as one extraction instance. Attribute-level statistics reported in later analyses are obtained by expanding each instance over its constituent target attributes. Detailed dataset composition is provided in Appendix E.1. **Protocol.** For each page group, one page is used as the seed page for rule generation and one sibling page as verification. Both pages are sampled from the evaluated URLs following a transductive evaluation protocol. The verification page is used without

Table 1: Overall performance across 11 models (F1/P/R). ΔF1 is the gain over the strongest competing method.

	VGS			CoT			Reflexion			CogExtract			ΔF1
Model	F1	P	R	F1	P	R	F1	P	R	F1	P	R	
GPT-5	63.0	58.5	68.3	45.3	43.6	47.1	43.2	41.5	45.1	76.6	68.4	87.0	+13.6
qwen3.7-max	58.2	48.8	72.0	43.5	43.2	44.5	43.4	43.0	44.6	72.5	63.8	83.9	+14.3
GPT-4o	45.0	40.2	51.1	24.5	22.6	26.8	26.3	25.3	27.4	72.1	62.6	85.0	+27.1
qwen3.7-plus	56.2	48.0	67.8	35.4	35.6	36.5	34.7	35.2	36.1	71.8	66.5	78.0	+15.6
kimi-k2.5	52.0	46.1	59.6	39.3	39.9	39.5	40.0	39.9	40.4	68.5	64.7	72.8	+16.5
qwen3.5-35b-a3b	38.7	31.3	50.8	28.7	27.2	30.4	25.4	22.7	28.8	59.6	51.8	70.1	+20.9
MiniMax-M2.5	35.7	31.7	40.8	25.6	25.1	27.4	27.1	23.9	31.3	50.4	42.2	62.5	+14.7
qwen3-vl-32b	17.1	12.6	26.5	24.0	22.7	25.5	22.3	18.9	27.1	37.1	30.2	48.1	+13.1
qwen3-vl-30b	22.6	18.6	28.8	13.9	12.8	15.2	12.8	10.2	17.2	30.5	25.5	37.9	+7.9
Gemma3-12b	24.6	17.7	40.1	16.5	14.2	19.7	14.3	12.1	17.4	28.5	22.6	38.5	+3.9
qwen3-vl-8b	15.9	13.2	25.8	10.7	8.9	13.4	9.6	7.3	13.9	21.4	15.4	35.1	+5.5

ground-truth labels and provides only execution and structural evidence during test-time verification and repair. After execution-grounded verification and possible failure-conditioned repair, the selected rule is evaluated over all pages in the corresponding page group. Further protocol details are described in Appendix E.2. **Models.** We evaluate 14 vision-language model variants covering both closed-source and open-source families. The main comparison reports 11 models, while additional thinking variants are analyzed separately. All models use the same visual anchoring, candidate generation, and XPath execution pipeline, differing only in the backbone model. Complete model configurations are listed in Appendix E.3. **Baselines.** We compare against three representative approaches: VGS (Yang et al., 2026), Reflexion (Shinn et al., 2023), and Chain-of-Thought (CoT) (Wei et al., 2022). VGS directly generates a single XPath from visual grounding without cross-page execution verification. Reflexion augments generation with iterative self-reflection, while CoT prompts explicit structural reasoning before rule generation. Complete prompt templates for all methods are provided in Appendix F. **Metrics.** For each page-attribute pair, predicted values are compared with annotated ground truth to compute precision, recall, and F1. Multi-valued attributes are evaluated as sets, while single-valued attributes are treated as singleton sets. Standard text normalization and fuzzy matching are applied before scoring. Final results are macro-averaged over all samples and attributes. Additional metric details are provided in Appendix E.5.

3.1 Overall Results

Performance Across Backbones. Table 1 reports the overall extraction performance across 11 VLM backbones. CogExtract achieves the highest F1 for every evaluated backbone and improves over VGS by **+14.5 F1** on average, showing that the gains generalize across models with substantially different baseline capabilities. GPT-5 achieves the highest overall performance at **76.6 F1**, followed by Qwen3.7-Max (72.5) and GPT-4o (72.1). Notably, CogExtract improves GPT-5 from 63.0 under VGS to 76.6 (+13.6 F1), indicating that substantial gains remain even with a strong underlying VLM. The improvements are not restricted to the strongest backbones. GPT-4o improves from 45.0 F1 with VGS to 72.1 with CogExtract (+27.1), while Qwen3-VL-32B improves from 17.1 to 37.1 (+20.0). Gains are also observed for smaller models: Qwen3-VL-8B increases from 15.9 to 21.4 (+5.5), and Gemma3-12B from 24.6 to 28.5 (+3.9). Although the magnitude varies across backbones, CogExtract consistently outperforms the corresponding VGS, CoT, and Reflexion configurations. Across CogExtract configurations, recall is consistently higher than precision. For example, Qwen3.7-Plus achieves 78.0 recall versus 66.5 precision, while Kimi-K2.5 achieves 72.8 recall versus 64.7 precision. This pattern suggests that CogExtract favors extraction coverage, while execution-grounded verification helps control the precision loss that could otherwise arise from searching over multiple candidate rules. We analyze the contributions of candidate diversity and verification more directly in §3.2.

Performance Across Task Types. The gains also extend across extraction structures. On Qwen3.7-Plus, CogExtract improves over VGS on all four task types, with gains of **+26.8 F1** for multi-attribute extraction, **+15.4** for single-attribute extraction, **+6.6** for paired extraction, and **+6.0** for list extraction. Thus, the improvement is not confined to a particular extraction format, although the smaller gains on paired and list tasks indicate that structured grouping and alignment remain more challenging. Detailed precision, recall, and F1 results by task type are provided in Appendix B.

Computational Efficiency. Finally, the performance gains come with a modest increase in inference cost. Averaged across the 11 backbones, CogExtract achieves 53.5 F1 with 5.64 LLM/VLM calls per

Table 2: Progressive ablation of DCG, EGV, and FCR across five models.

Model	VGS $K{=}1$	+DCG $K{=}3$	+EGV DCG+EGV	Full DCG+EGV+FCR	−DCG EGV+FCR	−EGV DCG+FCR
qwen3.7-plus	56.2	58.0 (+1.8)	60.8 (+4.6)	**71.8** (+15.6)	66.5 (+10.3)	53.9 (−2.3)
kimi-k2.5	52.0	58.6 (+6.6)	61.1 (+9.1)	**68.5** (+16.5)	64.8 (+12.8)	55.2 (+3.2)
MiniMax-M2.5	35.7	40.7 (+5.0)	43.3 (+7.6)	**50.4** (+14.7)	46.9 (+11.2)	38.5 (+2.8)
qwen3-vl-32b	17.1	19.0 (+1.9)	23.1 (+6.0)	**37.1** (+20.0)	30.2 (+13.1)	20.3 (+3.2)
qwen3-vl-8b	15.9	14.7 (−1.2)	18.3 (+3.6)	**21.4** (+5.5)	19.8 (+3.9)	15.1 (−0.8)

attribute, compared with 39.0 F1 and 5.00 calls for VGS. Thus, the average gain of **+14.5 F1** requires only a **13% increase in model calls**. Detailed efficiency results and accuracy–cost comparisons are provided in Appendix C.

Thinking Models. We further test whether stronger internal reasoning reduces the benefit of external execution-grounded search. Across three Qwen3-VL backbones, CogExtract remains consistently beneficial under thinking variants. For example, on Qwen3-VL-32B, the CogExtract–VGS gap increases from $+20.0$ F1 in the instruct setting to $+44.0$ under thinking. These results suggest that stronger internal reasoning and execution-grounded search provide complementary gains. Full results are provided in Appendix K.

3.2 Ablation and Contribution Analysis

We examine the contribution of each component by progressively adding DCG, EGV, and FCR to the VGS baseline across five representative backbones. **VGS** uses a single candidate ($K = 1$) without cross-page verification or reflection. **+DCG** expands the candidate set to $K = 3$ through diversity-conditioned generation, while keeping verification and reflection disabled. **+DCG+EGV** further introduces execution-grounded verification over the $K = 3$ candidates, without invoking reflection. **Full** combines DCG, EGV, and FCR, corresponding to the complete CogExtract framework. The final two columns of Table 2 provide complementary removal ablations of the full system. **−DCG** retains EGV and FCR but reduces the candidate set to $K = 1$, removing candidate diversity. **−EGV** retains DCG and FCR but replaces cross-page verification with a seed-page-only execution check, removing cross-page execution evidence from verification.

The ablation results reveal a consistent pattern across the five backbones. DCG alone yields modest and model-dependent gains, while introducing EGV provides additional improvements by enabling cross-page discrimination among candidate wrappers. Adding FCR on top of DCG and EGV produces the largest further gain, with the full system outperforming VGS by $+5.5$ to $+20.0$ F1 across the evaluated backbones. The counterfactual controls reinforce this trend: removing either candidate diversity or cross-page verification degrades the full system. In particular, removing EGV causes substantial performance drops on most backbones, suggesting that cross-page execution evidence is useful not only for selecting among existing candidates but also for providing reliable feedback to the subsequent repair stage. Overall, these results indicate that DCG, EGV, and FCR play complementary roles in candidate-space construction, execution-grounded selection, and failure-conditioned search-space expansion, respectively. In particular, the substantial degradation under −EGV shows that cross-page execution evidence is important not only for selecting existing candidates but also for supporting effective downstream repair.

3.3 Diversity Evidence (DCG)

We first examine whether DCG actually constructs a structurally diverse candidate space. Across generated candidate groups, the strategy entropy is 0.941 and the average normalized pairwise edit distance is 0.856; only 2.1% of groups contain highly similar candidates with normalized distance below 0.3. A controlled study in Table 6 further separates the effect of explicit structural diversity from ordinary sampling variation. Removing the diversity instruction while changing the sampling temperature reduces strategy entropy from 0.941 to 0.879 or 0.861, with corresponding group-mean F1 changes of -1.39 and -1.90; neither difference is large. In contrast, forcing all candidates to follow a single structural strategy reduces entropy to 0.101 and decreases group-mean F1 by 11.63 points, a substantially larger drop. The forced single-strategy condition also increases the FCR trigger rate from 21.8% to 31.3%, suggesting that structural diversity increases the chance of resolving failures within the initial candidate space. These results suggest that the benefit of DCG depends more on maintaining structurally distinct hypotheses than on stochastic variation alone. We additionally analyze the candidate-set size K. Using a single-pass $K = 8$ generation and evaluating

prefix subsets, $K = 3$ already captures 92.5% of the candidate coverage obtained with $K = 8$. We therefore use $K = 3$ as the default setting, balancing candidate coverage against generation cost. The full saturation analysis is provided in Figure 7 (Appendix I).

3.4 Discriminative Power of Verification (EGV)

The value of cross-page verification depends on whether its scores can distinguish correct from incorrect candidates. Taking seed-page extraction $F_1 \geq 0.8$ as a proxy for candidate correctness, Figure 3 compares EGV with LLM self-evaluation under the same backbone, candidate set, evaluation criteria, and inputs. EGV achieves an AUC of 0.840, compared with 0.640 for self-evaluation; equivalently, EGV assigns the higher score to a randomly drawn correct candidate rather than an incorrect one 84% of the time. The group-level result in Figure 3 shows the same pattern: EGV attains a best-selection P@1 of 96.2%, versus 77.9% for self-evaluation. Thus, self-evaluation would select the wrong candidate in roughly one in five groups, causing subsequent reflection to start from an inferior candidate.

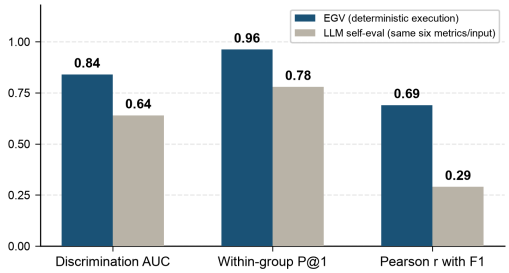


Figure 3: EGV versus model self-evaluation on 688 candidates from 229 groups using Qwen3.7-Plus.

Additional analyses reinforce the conclusion illustrated in Figure 3. Under a stricter cross-page label, which requires correct extraction on both the seed and verification pages, EGV's AUC rises to 0.861 whereas self-evaluation falls to 0.618. In particular, EGV systematically demotes the 23 candidates that succeed on the seed page but fail on its sibling page—precisely the failure mode that cross-page verification is designed to detect. The result is also insensitive to score weighting: equal weights yield an AUC of 0.843, nearly identical to the default 0.840. Re-ranking the same 229 fully labeled candidate groups using equal weights or 100 random Dirichlet weight vectors likewise leaves selection nearly unchanged. Best-selection P@1 is 96.2% under both the default and equal-weight schemes; about 70% of the random schemes preserve this value, and the remainder differ by less than one percentage point. Together, these results indicate that the discriminative signal and selection quality arise from execution-grounded evidence rather than from hand-tuned weights.

3.5 Hierarchical Resolution and FCR

We next examine how CogExtract resolves failures across its two-layer hierarchy. Among 326 audited initial failure cases, 136 (41.7%) are resolved directly at Layer 1 through candidate selection, without invoking FCR. This shows that execution-grounded verification can resolve a substantial fraction of failures by selecting an alternative hypothesis already present in the candidate space, rather than immediately resorting to reflection. FCR is invoked only when no candidate satisfies the cross-page eligibility criterion. On Qwen3.7-Plus, FCR is triggered for 22.5% of evaluated attributes; among these cases, 94.2% generate at least one new candidate that subsequently passes EGV. A paired no-diagnosis control on the same trigger set—identical evidence and budget, with only the diagnosis removed—lowers recovery to 78.8% and more than halves the correct-repair rate (9.1% vs. 19.5%), confirming that recovery stems from diagnosis-conditioned repair rather than regeneration alone (Appendix G). These results support the intended hierarchy: execution evidence first serves as a selection signal over the existing candidate space and, only when that space contains no viable solution, is repurposed as diagnostic feedback to guide targeted candidate regeneration. Detailed resolution statistics and failure-diagnosis analyses are provided in Appendix G.

3.6 Residual Error Analysis

Despite the overall gains, substantial residual errors remain. Using Qwen3.7-Max, the strongest non-GPT-5 backbone in our evaluation, 22,727 of 67,216 page–attribute evaluations (33.8%) have extraction F1 below 0.8. This threshold is used only for post-hoc analysis and is independent of the EGV eligibility criterion and the triggering condition of FCR. Manual inspection reveals several recurring sources of error, including visual or element mislocalization, boundary and alignment errors in structured extraction, XPath formulation failures, and text contamination or formatting artifacts. These cases indicate that the remaining bottlenecks do not arise solely from cross-page structural variation. Some failures originate from upstream grounding errors, while others reflect limitations

in representing complex extraction structures with the current rule space. Consequently, expanding or repairing XPath candidates alone cannot resolve all residual failures. A detailed breakdown of residual error types and representative examples is provided in Appendix A.

4 Related Work

Web information extraction (WIE) has evolved from rule-based wrapper induction and DOM pattern matching (Kushmerick et al., 1997; Muslea et al., 1999; Laender et al., 2002), through sequence labeling with CRFs (Lafferty et al., 2001) and visual features (Cai et al., 2003), to extraction programs synthesized via code generation (Lockard et al., 2020) and, more recently, data-driven and foundation-model-based approaches. LLMs enable extraction directly from HTML or simplified web representations (Wadhwa et al., 2023; Dong et al., 2024; Qi et al., 2024; Yang et al., 2024), while recent VLM methods further exploit rendered pages for visual grounding and extraction (He et al., 2024; Zheng et al., 2024; Khalighinejad et al., 2025; Bohra et al., 2025). VGS similarly generates reusable XPath wrappers from visually grounded elements. Despite stronger perception and grounding, these methods largely follow a generate-and-reuse paradigm: a wrapper synthesized from a representative page is applied to related pages without an execution-grounded feedback loop. When a wrapper silently fails on sibling pages due to local DOM variation, such failures are difficult to detect and systematically repair from the seed-page generation process alone. CogExtract addresses this gap by introducing cross-page execution verification and hierarchical, evidence-conditioned repair into the wrapper generation pipeline.

More broadly, inference-time methods improve generation through multiple hypotheses, verification, or iterative refinement, from chain-of-thought prompting (Wei et al., 2022) to explicit hypothesis search (Yao et al., 2024) and process-level reward modeling (Lightman et al., 2024). Self-Consistency (Wang et al., 2023b), Self-Refine (Madaan et al., 2023), Reflexion (Shinn et al., 2023), and Chain-of-Verification (Dhuliawala et al., 2024) rely on model-generated reasoning, verbalized confidence (Kadavath et al., 2022), or self-evaluation, which can be unreliable when the model commits to the same structural misconception (Huang et al., 2024). In code generation, execution-based methods use externally observable outcomes to evaluate and repair generated solutions (Chen et al., 2024; Ni et al., 2023). Recent work on agent failure attribution, such as AgenTracer (Zhang et al., 2025) and Aegis (Kong et al., 2025), focuses primarily on localizing which agent or step contributes to a failure. Our setting instead requires a repair-oriented diagnosis that determines how the current extraction hypothesis should be revised. CogExtract uses deterministic cross-page execution signals rather than model-internal feedback as the basis for verification, and converts execution failures into structured, repair-relevant diagnostic feedback that directly conditions subsequent candidate generation. Execution thus serves not merely as post-hoc validation, but as the signal that governs both candidate selection and targeted search-space expansion. This control loop is also related to dual-process views of cognition (Kahneman, 2011) and recent work on artificial metacognition (Khandelwal et al., 2025; Shakarian, 2026); in contrast to such general-reasoning settings, CogExtract grounds its monitoring and evaluation signals in deterministic execution evidence from the target pages.

5 Conclusion

We presented **CogExtract**, which reframes web extraction rule generation from single-pass decoding as test-time search with execution-grounded verification. DCG expands the candidate space, EGV provides deterministic execution-based selection signals, and FCR selectively redirects the search through structured diagnosis when no cross-page eligible candidate is available. On the LiveWeb-IE benchmark, covering 14 real-world sites, 312 page groups, and 50,524 URLs, CogExtract achieves consistent gains across 11 open- and closed-source models, improving over the visual baseline VGS by 14.5 F1 on average with only a 13% increase in model calls. CogExtract with GPT-5 reaches 76.6 F1. Results with thinking variants further suggest that stronger internal reasoning and execution-grounded search provide complementary gains.

Residual error analysis also reveals limitations of XPath-based extraction, particularly for complex list boundaries (c_2, 22.4%) and CSS pseudo-element text contamination (c_3, 12.5%). Future work will explore richer rule representations and adaptive text normalization to improve the robustness of fully automatic large-scale web extraction.

Use of Large Language Models in Paper Preparation

During the preparation of this paper, we used LLM assistants only to improve phrasing and grammar and to help screen candidate related work. All scientific content—method, implementation, experiments, analysis, and conclusions—was developed and verified by us.

References

Arvind Arasu and Hector Garcia-Molina. Extracting structured data from web pages. In *SIGMOD*, 2003.

Ankit Bohra et al. WebLists: Extracting structured information from complex interactive websites using executable LLM agents. *arXiv preprint arXiv:2504.12682*, 2025.

Deng Cai, Shipeng Yu, Ji-Rong Wen, and Wei-Ying Ma. VIPS: A vision-based page segmentation algorithm. Technical Report MSR-TR-2003-79, Microsoft Research, 2003.

Xiang Chen, Mingda Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. In *ICLR*, 2024.

Valter Crescenzi, Giansalvatore Mecca, and Paolo Merialdo. RoadRunner: Towards automatic data extraction from large web sites. In *VLDB*, 2002.

Shehzaad Dhuliawala et al. Chain-of-verification reduces hallucination in large language models. In *ICLR*, 2024.

Qingxiu Dong et al. A survey on in-context learning. In *EMNLP*, 2024.

Gemma Team. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025.

Hongliang He et al. WebVoyager: Building an end-to-end web agent with large multimodal models. In *ACL*, 2024.

Jie Huang et al. Large language models cannot self-correct reasoning yet. In *ICLR*, 2024.

Saurav Kadavath et al. Language models (mostly) know what they know. *arXiv preprint arXiv:2207.05221*, 2022.

Daniel Kahneman. *Thinking, Fast and Slow*. Farrar, Straus and Giroux, 2011.

Ghazal Khalighinejad et al. MatViX: Multimodal information extraction from visually rich articles. In *NAACL*, 2025.

Vedant Khandelwal, Francesca Rossi, Karthik Murugesan, Eric Miehling, Mark Campbell, Rayid Horesh, et al. Language models coupled with metacognition can outperform reasoning models. *arXiv preprint arXiv:2508.17959*, 2025.

Kimi Team. Kimi K2.5: Visual agentic intelligence. *arXiv preprint arXiv:2602.02276*, 2026.

Fanqi Kong, Ruijie Zhang, Huaxiao Yin, Guibin Zhang, Xiaofei Zhang, Ziang Chen, Zhaowei Zhang, Xiaoyuan Zhang, Song-Chun Zhu, and Xue Feng. Aegis: Automated error generation and attribution for multi-agent systems. *arXiv preprint arXiv:2509.14295*, 2025.

Nicholas Kushmerick, Daniel S. Weld, and Robert Doorenbos. Wrapper induction for information extraction. In *IJCAI*, 1997.

Alberto H. F. Laender, Berthier A. Ribeiro-Neto, Altigran S. da Silva, and Juliana S. Teixeira. A brief survey of web data extraction tools. *SIGMOD Record*, 31(2), 2002.

John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*, 2001.

Hunter Lightman et al. Let's verify step by step. In *ICLR*, 2024.

Colin Lockard, Prashant Shiralkar, Xin Luna Dong, and Hannaneh Hajishirzi. EVAPORATE: Code generation as a prompt learning problem. In *VLDB*, 2020.

Aman Madaan et al. Self-refine: Iterative refinement with self-feedback. In *NeurIPS*, 2023.

MiniMax. The MiniMax-M2 series: Mini activations unleashing max real-world intelligence. *arXiv preprint arXiv:2605.26494*, 2026.

Ion Muslea, Steven Minton, and Craig A. Knoblock. A hierarchical approach to wrapper induction. *Autonomous Agents and Multi-Agent Systems*, 1999.

Fangyu Ni, Jixuan Wang, Hongcheng Yin, Jianshu Li, and Percy Liang. LEVER: Learning to verify language-to-code generation with execution. In *ICML*, 2023.

OpenAI. Hello GPT-4o. `https://openai.com/index/hello-gpt-4o/`, 2024.

OpenAI. GPT-5 system card. Technical report, OpenAI, 2025.

Yunjia Qi, Hao Peng, Xiaozhi Wang, Bin Xu, Lei Hou, and Juanzi Li. ADELIE: Aligning large language models on information extraction. In *EMNLP*, 2024.

Qwen Team. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a.

Qwen Team. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*, 2025b.

Qwen Team. Qwen3.5: Towards native multimodal agents. `https://qwen.ai/blog?id=qwen3.5`, 2026a.

Qwen Team. Qwen3.7: The agent frontier. `https://qwen.ai/blog?id=qwen3.7`, 2026b.

Paulo Shakarian. Toward artificial metacognition. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, 2026.

Noah Shinn et al. Reflexion: Language agents with verbal reinforcement learning. In *NeurIPS*, 2023.

Aman Wadhwa et al. Revisiting information extraction from structured data with language models. In *EMNLP*, 2023.

Qifan Wang, Jingang Wang, Xiaojun Quan, Fuli Feng, Zenglin Xu, Shaoliang Nie, Sinong Wang, Madian Khabsa, Hamed Firooz, and Dongfang Liu. MUSTIE: Multimodal structural transformer for web information extraction. In *ACL*, pp. 2405–2420, 2023a.

Xuezhi Wang et al. Self-consistency improves chain of thought reasoning in language models. In *ICLR*, 2023b.

Jason Wei et al. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*, 2022.

Ke Yang et al. AgentOccam: A simple yet strong baseline for LLM-based web agents. *arXiv preprint arXiv:2410.13825*, 2024.

Seungbin Yang, Jihwan Kim, Jaemin Choi, Dongjin Kim, Soyoung Yang, ChaeHun Park, and Jaegul Choo. LiveWeb-IE: A benchmark for online web information extraction. In *ICLR*, 2026.

Shunyu Yao et al. Tree of thoughts: Deliberate problem solving with large language models. In *ICLR*, 2024.

Guibin Zhang, Junhao Wang, Junjie Chen, Wangchunshu Zhou, Kun Wang, and Shuicheng Yan. AgenTracer: Who is inducing failure in the LLM agentic systems? *arXiv preprint arXiv:2509.03312*, 2025.

Boyuan Zheng et al. GPT-4V(ision) is a generalist web agent, if grounded. In *ICML*, 2024.

A Detailed Residual Error Analysis

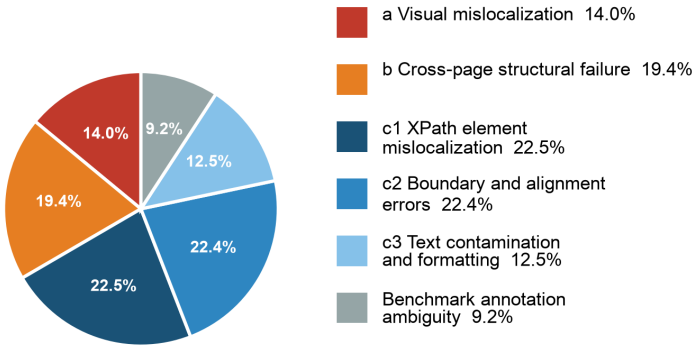


Figure 4: Distribution of residual error types for Qwen3.7-Max under CogExtract. Residual cases are defined as page–attribute evaluations with extraction F1 below 0.8. This threshold is used only for post-hoc analysis and is independent of EGV and FCR.

A.1 Residual Error Distribution

We conduct a post-hoc analysis of residual extraction errors using Qwen3.7-Max under CogExtract. Among 67,216 page–attribute evaluations, 22,727 (33.8%) have extraction F1 below 0.8. We use this threshold only to define the subset for error analysis; it is independent of the EGV eligibility criterion and does not affect whether FCR is triggered.

Figure 4 summarizes five recurring error types observed in these cases. The distribution indicates that residual errors arise from multiple stages of the extraction pipeline rather than from a single failure source. Some originate from incorrect visual or element grounding, while others involve structural boundary and alignment errors, XPath formulation failures, or contamination of otherwise relevant extracted text. This heterogeneity suggests that further improvements require addressing not only cross-page rule robustness, but also upstream grounding and the representation of structured extraction targets.

A.2 Representative Error Cases

We further examine representative examples of the recurring error types. These cases illustrate where failures arise in the extraction pipeline and help distinguish errors that may be addressed through candidate search and repair from those that originate outside the current XPath hypothesis space.

Visual mislocalization. In some cases, the visually grounded element does not correspond to the intended extraction target. Because subsequent XPath generation is conditioned on this grounding result, an incorrect anchor can constrain candidate search to an irrelevant DOM region. Generating additional XPath variants around the same anchor may therefore be insufficient to recover the intended target.

XPath element mislocalization. Even when the relevant page region is correctly identified, a generated XPath may select a nearby but semantically incorrect DOM element. This often occurs when neighboring elements share similar local structures or attributes, making multiple XPath hypotheses structurally plausible but semantically distinct.

Boundary and alignment errors. Structured list and paired extraction pose an additional challenge: a rule may identify the correct repeated region while failing to preserve item boundaries or cross-field alignment. In such cases, the extracted elements may be individually plausible, yet their grouping or correspondence is incorrect.

Table 3: Breakdown by task type for qwen3.7-plus (F1 / P / R)

	VGS			CogExtract			
Type	F1	P	R	F1	P	R	ΔF1
Type 1 (single-attribute)	53.5	50.2	65.3	68.9	65.5	79.8	+15.4
Type 2 (multi-attribute)	55.6	51.7	71.0	82.4	79.1	95.7	+26.8
Type 4 (paired)	60.4	60.0	60.9	67.0	65.4	71.7	+6.6
Type 3 (list)	46.3	45.3	48.2	52.3	52.0	52.9	+6.0
Overall	56.2	48.0	67.8	71.8	66.5	78.0	+15.6

Text contamination and formatting artifacts. Some rules correctly localize the intended element but return additional labels, presentation text, or formatting artifacts together with the target value. Here, structural localization is largely correct, but the returned text does not precisely match the annotated target.

Benchmark ambiguity. We also observe cases in which multiple extraction targets are semantically plausible, while the benchmark annotation specifies only one of them. We distinguish these cases from clear system failures because the observed discrepancy may reflect annotation ambiguity rather than an error in grounding or rule generation.

Representative screenshots and extraction outputs for these cases are provided below.

B Task-Type Breakdown

We provide a detailed breakdown of extraction performance across the four task types in LiveWeb-IE. Table 3 reports precision, recall, and F1 for VGS and CogExtract using Qwen3.7-Plus.

CogExtract improves F1 across all four task types. The largest gain is observed for multi-attribute extraction, where F1 increases from 55.6 to 82.4 (+26.8), accompanied by an increase in recall from 71.0 to 95.7. Single-attribute extraction also improves substantially, from 53.5 to 68.9 F1 (+15.4).

The gains are smaller for the more structurally specialized paired and list tasks. Paired extraction improves from 60.4 to 67.0 F1 (+6.6), while list extraction improves from 46.3 to 52.3 (+6.0). These results suggest that CogExtract benefits different extraction structures, but complex grouping and alignment requirements remain challenging. The precision–recall patterns also vary across task types. For multi-attribute extraction, CogExtract achieves 79.1 precision and 95.7 recall, indicating a particularly strong improvement in coverage. In contrast, list extraction yields more balanced precision and recall (52.0 and 52.9, respectively). Thus, the coverage-oriented behavior observed in the aggregate results is more pronounced for single- and multi-attribute extraction than for structurally specialized tasks.

C Computational Efficiency

We further analyze the inference cost of CogExtract in terms of the average number of LLM/VLM calls per extracted attribute. Figure 5 compares this cost with the average extraction F1 across the 11 backbones.

CoT and Reflexion require 1.00 and 1.33 calls per attribute, respectively, but achieve relatively low average F1 scores of 27.9 and 27.2. VGS requires 5.00 calls and reaches 39.0 F1, whereas CogExtract requires 5.64 calls and achieves 53.5 F1. Relative to VGS, CogExtract therefore improves average F1 by 14.5 points while increasing the number of model calls by only 13%.

The additional cost of CogExtract primarily comes from the selective invocation of FCR when Layer 1 fails to identify an eligible candidate. In contrast, DCG generates multiple XPath candidates jointly within a single model invocation, while EGV evaluates candidates using deterministic execution signals without additional LLM/VLM calls. Consequently, the test-time search procedure does not require a proportional increase in model inference as the candidate space expands.

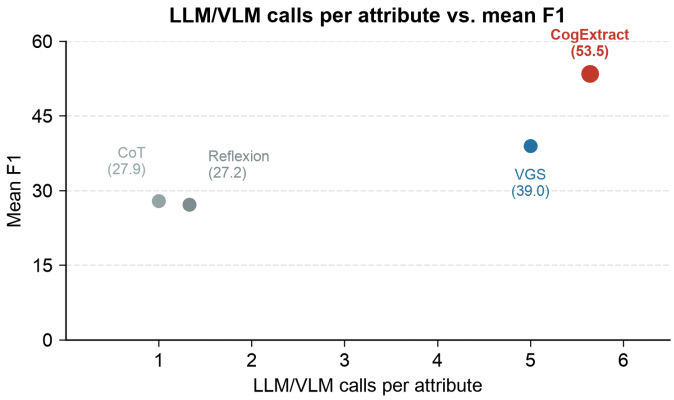
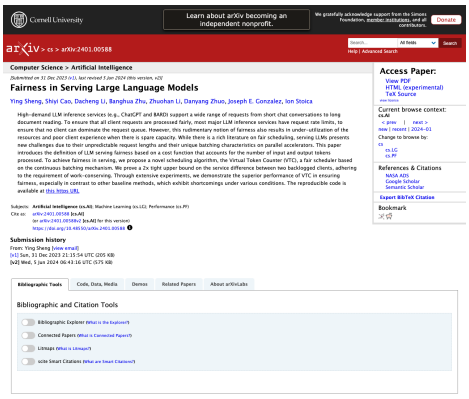


Figure 5: Average LLM/VLM calls per attribute versus average F1 across 11 backbones. CogExtract substantially improves extraction performance over VGS with a modest increase in model calls.

D Error Case Gallery

This appendix presents representative cases of each branch (a1, c1–c3) of the three root-cause categories in §3.6, together with a benchmark ambiguity case, each with the page screenshot, extraction query, and details.

c1: XPath element mislocalization — author @ arxiv.org



Query: `Please provide each author's name and the link to their detailed profile`

URL: `arxiv.org/abs/2401.00588`

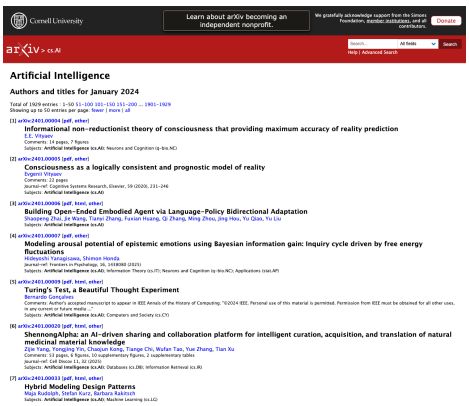
XPath: `//div[@id='abs']//div[contains(@class,'authors')]//a`

Prediction: `search/cs?searchtype=author&query=Sheng,+Y`

Ground truth: `Ying Sheng`

The XPath `//a` extracted the `href` attribute (a search URL) inside the authors div instead of the author text.

c2: list boundary and alignment error — subject @ arxiv.org



Query: `I'd like to know the subject of all the papers`

URL: `arxiv.org/list/cs.AI/2024-01`

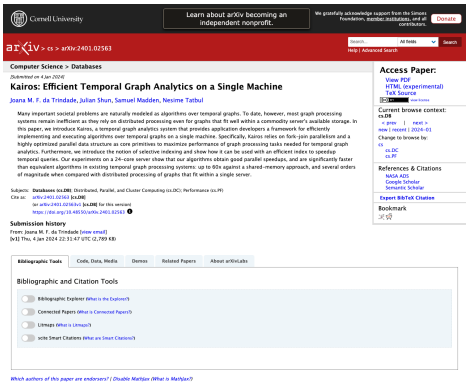
XPath: `//div[contains(@class,'meta')]//span[contains(text(),'Subjects:')]/following-sibling::`

Prediction: `; Databases (cs.DB); Distributed, Parallel…` (4 sub-disciplines)

Ground truth: `Data Structures and Algorithms`

The XPath captured all subject sibling nodes (4 sub-disciplines) rather than the primary subject listed first.

c3: text contamination and formatting — title @ arxiv.org



Query: `Extract the title of the paper`

URL: `arxiv.org/abs/2401.02563`

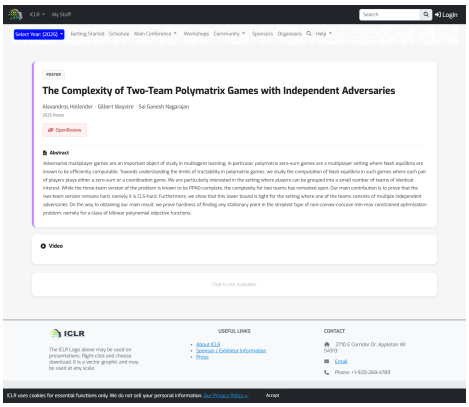
XPath: `//h1[contains(@class,'title') or contains(@class,'heading')]`

Prediction: `Title:Kairos: Efficient Temporal Graph Analytics on a Single Machine`

Ground truth: `Kairos: Efficient Temporal Graph Analytics on a Single Machine`

A CSS `::before` pseudo-element injects the label text `"Title:"` into the DOM. The XPath extracted the contaminated string; the correct title is buried inside it but cannot be cleanly separated.

Benchmark annotation ambiguity (semantic-level URL mismatch) — abstract_link @ iclr.cc



Query: `Present the paper's abstract link`

URL: `iclr.cc/virtual/2025/poster/30687`

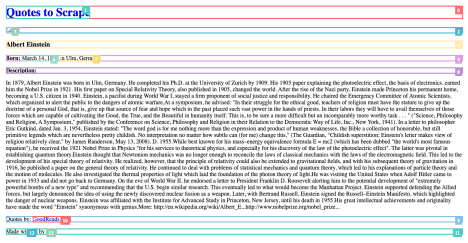
XPath: `//span[contains(@class,'nowrap')]/a[contains(@class,'href_URL')]`

Prediction: `https://openreview.net/forum?id=9VGTk2NYjF`

Ground truth: `#abstract_details`

The XPath extracted the actual OpenReview URL (a valid external link), while the ground truth records an in-page anchor. Both are valid—this is a semantic target mismatch, not a formatting error.

a1: visual mislocalization — author @ quotes.toscrape.com



Query: `Fetch the quote author's name`

URL: `quotes.toscrape.com/author/Albert-Einstein`

XPath: `//div[contains(@class,'col-md-8')]//h1`

Prediction: `Quotes to Scrape`

Ground truth: `Albert Einstein`

The page has two `<h1>` elements: (1) the site title at the top, (3) the author name below. The XPath targeted (1) instead of (3)—the VLM visually grounded on the larger, more prominent heading.

E Additional Experimental Details

This appendix provides additional details of the experimental setup, including dataset composition, evaluation protocol, model configurations, and scoring procedures.

E.1 Dataset Composition

We evaluate on LiveWeb-IE, a benchmark designed for real-world web information extraction under structural variation. The benchmark originally contains 15 websites. We exclude one website (dp.art) from evaluation because it deploys access restrictions and anti-scraping mechanisms. We do not bypass such restrictions; therefore, all reported results are obtained on the remaining publicly accessible websites.

The final evaluation set contains 14 websites, 312 page groups, and 50,524 URLs. The evaluated websites cover diverse domains, including academic repositories, e-commerce platforms, governmental portals, and structured reference databases. Table 4 summarizes the complete website composition.

Table 4: Website composition of the LiveWeb-IE evaluation subset.

Category	Websites
Academic repositories	arXiv, NeurIPS, ICML, ICLR, HuggingFace
E-commerce	BooksToScrape, SandboxOxyLabs
Government data	FuelEconomy
Structured reference databases	MarineSpecies, TheSportsDB, TheMealDB, TheCocktailDB, QuotesToScrape, ScrapeThisSite

Each URL paired with its target attributes forms one extraction instance. Therefore, the benchmark contains 50,524 extraction instances.

Some analyses report attribute-level statistics. These numbers are obtained by expanding each extraction instance into individual target attributes. For example, 67,216 attribute evaluations correspond to the total number of (URL, attribute) pairs rather than additional webpages or instances.

E.2 Evaluation Protocol

For each page group, CogExtract follows a transductive test-time evaluation protocol.

One page is randomly selected as the seed page for candidate rule generation, and one sibling page is selected as the verification page for Execution-Grounded Verification (EGV). Both pages are sampled from the evaluated URLs within the same page group.

The verification page provides only unlabeled execution evidence. It is not used for training, parameter optimization, or supervision. After candidate selection and possible failure-conditioned repair, the final XPath rule is evaluated over all pages belonging to the corresponding page group.

This protocol is designed to measure cross-page robustness under realistic test-time availability of related pages, rather than inductive generalization to completely unseen websites.

E.3 Model Configurations

We evaluate a diverse collection of vision-language models (VLMs) covering both proprietary and open-source families.

The closed-source models include:

- GPT-5 (OpenAI, 2025)
- GPT-4o (OpenAI, 2024)
- Qwen3.7-Plus
- Qwen3.7-Max (Qwen Team, 2026b)
- Kimi-K2.5 (Kimi Team, 2026)
- MiniMax-M2.5 (MiniMax, 2026)

The open-source models include:

- Qwen3.5-35B-A3B (MoE) (Qwen Team, 2026a)

- Qwen3-VL-32B
- Qwen3-VL-30B
- Qwen3-VL-8B (Qwen Team, 2025b)
- Gemma3-12B (Gemma Team, 2025)

For Qwen3-VL models, both thinking and instruction variants are evaluated in the corresponding analysis, following the hybrid thinking design of the Qwen3 family (Qwen Team, 2025a). The main comparison reports 11 model variants, including six closed-source models and five open-source non-thinking models. Additional thinking variants are analyzed separately.

All models receive the same visual anchoring procedure, candidate generation prompt, and XPath execution environment. The only difference is the backbone VLM.

E.4 Baseline Configuration

We compare against three representative baselines:

VGS. VGS is the perception–action baseline. It directly generates a single XPath from visual grounding without cross-page execution verification.

Reflexion. Reflexion extends single-path generation with iterative self-reflection based on model-generated feedback.

Chain-of-Thought (CoT). CoT prompts the model to explicitly reason about webpage structure before generating the extraction rule.

All methods are evaluated on the same benchmark instances and use the same XPath execution environment. VGS and CogExtract share the visual-grounding pipeline, whereas CoT and Reflexion follow their respective text-based prompting protocols described in Appendix F.

E.5 Evaluation Metrics

For each page-attribute pair, the predicted extraction result is compared with the annotated ground-truth value set.

Precision, recall, and F1 are computed as:

$$P = \frac{|Y_p \cap Y_g|}{|Y_p|}, \tag{2}$$

$$R = \frac{|Y_p \cap Y_g|}{|Y_g|}, \tag{3}$$

$$F1 = \frac{2PR}{P + R}, \tag{4}$$

where Y_p and Y_g denote predicted and ground-truth value sets.

For multi-valued attributes, predictions are evaluated as unordered sets. For single-valued attributes, both prediction and ground truth are treated as singleton sets.

Before comparison, extracted text values are normalized using:

- case folding,
- Unicode NFKC normalization,
- confusable-character mapping.

For remaining surface variation, fuzzy matching is applied with a token overlap threshold of 0.8.

Final results are reported as macro averages over all page-attribute instances. Missing predictions and unmatched ground-truth attributes receive zero contribution.

The human baseline uses the sampled human annotations released with the original LiveWeb-IE benchmark. It is reported only as a reference upper bound; we do not perform additional full-scale manual re-annotation.

F Prompt Templates

Below are the complete verbatim prompt templates used by the four methods: CoT, Reflexion, VGS, and CogExtract.

F.1 CoT / Reflexion

CoT and Reflexion share the same Top-Down prompt. Reflexion additionally uses a Self-Reflection prompt for iterative revision.

Top-Down Generation (TOP_DOWN_PROMPT)

```
You are a web parser that is good at reading and understanding the HTML code and can give clear
  ↪ executable code on the browser.

Please read the following HTML code, and then return XPaths grouped by field names that directly match
  ↪ the elements in the page, satisfying the instructions below.

Instruction: {instruction}

Rules:
- Field Determination:
  - Use explicitly listed field names from the instruction
  - Otherwise, infer minimal relevant fields present in the DOM without inventing unsupported fields
- XPath Construction:
  - Avoid embedding exact literal values or visible text from the HTML content
  - Create structurally robust selectors using attribute and structural patterns
  - Prefer stable element attributes and hierarchical relationships over brittle identifiers
- Field Organization:
  - Generate separate XPaths for each target node when multiple nodes exist for a field
  - Maintain distinct XPaths that satisfy the instruction requirements for each field
- Data Structure Requirements:
  - Ensure xpath and value dictionaries have identical key sets
  - Align the XPath and value lists so each position corresponds to the same target node
  - Extract text from single nodes without concatenating content from multiple elements
- Missing Information Handling:
  - Return empty lists for explicitly enumerated fields when HTML lacks suitable content
  - Return empty objects for both xpath and value when no instruction fields are specified and no
  ↪ suitable content is found

Please output in the following JSON format:
{{
  "thought": "",
  "value": {{}},
  "xpath": {{}}
}}

Here's the HTML code:
```
{html}
```
```

Self-Reflection (SELF_REFLECTION_PROMPT, Reflexion only)

```
Please read the following HTML code, and then return all possible XPaths that can recognize the elements
  ↪  in the HTML matching the instructions below.

Instruction: {instruction}

Rules:
- History Analysis:
  - Evaluate consistency between extraction results and expected values
```

```
  - Identify irrelevant elements that were incorrectly captured
  - Check for empty results indicating failed extraction
  - Accept raw values with redundant separators as consistent, since post-processing will handle them
- Value Assessment:
  - Re-examine expected values in the context of the HTML structure
  - Determine optimal XPath strategies for locating target content
  - Consider structural patterns and element relationships for reliable targeting
- XPath Refinement:
  - Generate new XPaths when current ones fail to meet requirements
  - Retain existing XPaths when they demonstrate accurate extraction
  - Base decisions on analysis findings and extraction quality assessment
- XPath Construction Constraints:
  - Avoid embedding exact literal values or specific HTML element content
  - Prevent overly broad selectors that match multiple nodes with different meanings
  - Use specific class attributes and positional indicators to differentiate target nodes
  - Maintain precision through structural anchors and attribute-based targeting
- Missing Content Handling:
  - Return empty outputs when HTML lacks information matching the instruction
  - Acknowledge extraction limitations rather than forcing inappropriate matches

Please output in the following JSON format:
{{
  "thought": "",
  "consistent": "",
  "value": {{}},
  "xpath": {{}}
}}

And here's the history about the thoughts, XPaths, and results extracted by the crawler:
{history}

Here's the HTML code:
```
{html}
```
```

F.2 VGS (4-Stage Visual Grounding)

The VGS baseline comprises 4 stages and 5 prompts: attribute identification, visual grounding, element scanning, element selection, and XPath synthesis.

Stage 1: Attribute Identification (ATTRIBUTE_IDENTIFICATION_PROMPT)

```
You are an Attribute Extractor that converts user extraction requests into structured JSON payloads for
  ↪ web scraping systems.
Task: Transform a user's data extraction request into a compact JSON specification that defines what
  ↪ attributes to extract.
Instructions:
- Analyze the user's request to identify specific data points they want extracted
- Focus on the minimal set of attributes needed to fulfill the user's request
Rules:
- JSON Structure: Return a JSON object with exactly one key: "attributes"
- The "attributes" key must contain a list of attribute names
- Each attribute name should be a concise, descriptive label for the data point
- Do not include any additional keys or nested structures
Please output in the following JSON format:
{{
    "attributes": ["attribute_1", "attribute_2", ...]
}}

User request: {query}
```

Stage 2: Visual Grounding (VISUAL_GROUNDING_PROMPT)

```
You are a visual grounding assistant that identifies which webpage region contains a specific UI element
  ↪  from a set of viewport screenshots.
Task: Given a set of viewport screenshots representing different regions of a webpage and a requested
  ↪ attribute, determine which specific region (if any) contains the target attribute clearly visible
  ↪ within its boundaries.
Instructions:
- You will receive multiple viewport screenshots representing different regions of a webpage
- Each screenshot will be labeled with a region identifier
- Analyze each region independently to determine if it contains the target attribute
Rules:
```

```
- Evaluate each region screenshot individually for the presence of the target attribute
- Do not assume elements exist based on typical webpage patterns or templates
- Focus only on elements that are clearly within each region's viewport boundaries
- Return only the single best-matching region
Please output in the following JSON format:
{{
    "matching_region": ""
}}

Target attribute: {attribute}
```

Stage 3a: Element Scanning (ELEMENT_SCANNING_PROMPT)

```
You are a precise visual extractor that analyzes a web UI screenshot to extract specific attribute
  ↪ content based on modality classification.
Task: Given a viewport screenshot and an attribute name, determine the correct modality from the
  ↪ attribute name and perform the appropriate extraction task.
Instructions:
Classify the attribute into exactly one modality based on naming patterns:
1. TEXT Modality - For textual attributes
- Extract all visible text strings exactly as they appear
- Preserve natural reading order (top-to-bottom, left-to-right)
- No text normalization or paraphrasing
2. IMAGE Modality - For image-related attributes (containing keywords like: image, photo, picture,
  ↪ thumbnail, logo, icon, banner, poster, fanart, artwork, cover, avatar, badge, flag, screenshot,
  ↪ gallery)
- Count all visible images in the region
- Report total count only
3. LINK/HYPERLINK Modality - For link-related attributes (containing keywords like: link, url, href,
  ↪ hyperlink, redirect, profile link, homepage, website)
- Count all visible hyperlinks/clickable elements in the region
- Report total count only
Please output in the following JSON format:
{{
    "modality": "",
    "items": []
}}

Target attribute: {attribute}
```

Stage 3b: Element Selection (ELEMENT_SELECTION_PROMPT)

```
You are a precise visual element selector that analyzes an annotated region screenshot to identify
  ↪ specific UI elements matching a target attribute.
Task: Given an annotated region screenshot where potential candidate elements have colored bounding
  ↪ boxes with unique numerical labels, select the exact bounding box IDs that contain values for the
  ↪ specified target attribute.
Instructions:
- Each element is outlined with a colored box containing a small ID label in the top-right corner
- The label shows an integer ID with white text on a colored background matching the box color
- Focus on elements that directly contain or represent the values of the target attribute
Rules:
- Relevance Only: Return IDs only for bounding boxes that directly contain values of the target
  ↪ attribute
- Completeness: Include all matching elements, not just the first occurrence
- No Guessing: If no bounding box matches, return an empty list
- ID Accuracy: Use exact integer IDs as shown in labels
- The selected_ids list must NOT be empty if matching elements exist on the screenshot
Please output in the following JSON format:
{{
    "selected_ids": [2, 5, 8]
}}

Target attribute: {attribute}
```

Stage 4: XPath Synthesis (XPATH_SYNTHESIS_PROMPT)

```
You are an expert XPath generator that creates robust, generalizable selectors for web scraping based on
  ↪  HTML segments, marked region, and target attribute.
Task: Generate one reliable XPath selector from the provided HTML segments, marked region screenshot,
  ↪ and target attribute that will work across structurally similar pages.
Instructions:
- Analyze HTML segments and marked regions to identify structural patterns and stable anchoring elements
```

```
- Use the marked region screenshot as visual context to understand the target element's location and
  ↪ appearance
Rules:
- Generalizability: Create XPaths that work across pages with similar structure, not just the current
  ↪ page
- Stability: Prefer class names, semantic tags, and structural patterns over dynamic IDs or positional
  ↪ indices
- Specificity: Balance between being too broad (matching unwanted elements) and too narrow (breaking on
  ↪ similar pages)
- Avoid embedding exact literal values or visible text from the HTML content
- Avoid using dynamic attributes like data-reactid, data-v-*, or session-specific identifiers
Please output in the following JSON format:
{{
    "xpath": "",
    "explanation": ""
}}

Target attribute: {attribute}
HTML segments:
{html_segments}
```

F.3 CogExtract

CogExtract reuses VGS's Stages 1–3b (attribute identification, visual grounding, element scanning, element selection), replaces the single-candidate XPath synthesis of Stage 4 with multi-candidate synthesis (three variants split by modality—text/link/image—sharing the same structure but differing only in the extraction target rules), and adds Failure-Conditioned Reflection (FCR).

Stage 4+: Multi-Candidate Synthesis (MULTI_CANDIDATE_SYNTHESIS_PROMPT)

```
You are an expert XPath engineer. Your task: extract the VALUE of "{attribute}" from web pages.

You are given a screenshot showing the page and HTML segments around candidate elements. Generate THREE
  ↪ different XPath expressions that could correctly extract the target attribute's VALUE.

=== Structural Context (from DOM analysis) ===
{structural_context}

=== CRITICAL: What to Extract ===
- You must extract the ACTUAL VALUE of "{attribute}", NOT labels/headers/descriptors
- For example, if attribute is "title", extract the actual title TEXT, not a label that says "Title:"
- If attribute is "price", extract "$29.99", not a label that says "Price:"
- Look at the SCREENSHOT to identify where the actual value appears visually

=== Diversity Requirement ===
Generate 3 XPaths using DIFFERENT structural paths to reach the same target value:
- They should all aim to extract the SAME content (the actual value)
- But use different DOM navigation strategies to get there
- This ensures at least one will generalize to other pages from the same site

=== Rules ===
1. Target the element containing the ACTUAL VALUE, not labels/descriptors
2. Prefer SHORT paths (2-3 steps with //) over deeply nested ones
3. NEVER use positional indices [N]
4. NEVER embed literal text values or page-specific content as predicates
5. Use contains(@class, 'token') for compound class values
6. For repeating content (lists), match ALL items
7. Each candidate must be independently valid
8. Prefer selectors within the MAIN CONTENT area -- avoid elements inside navigation bars, sidebars,
  ↪ footers, or other UI chrome
9. If the attribute appears to be a list of items, ensure at least one candidate targets the repeating
  ↪ container structure

Output JSON:
{{
    "xpath_1": "first XPath to extract {attribute} value",
    "xpath_2": "second alternative XPath",
    "xpath_3": "third alternative XPath",
    "target_description": "brief description of what the actual value looks like"
}}

Target attribute: {attribute}
HTML segments:
{html_segments}

---
```

```
Modality Variants (same structure, different extraction target):
- Link: extract <a> elements via @href (NEVER use text() or contains(text(), ...))
- Image: extract <img> elements via @src (NEVER use contains(@alt, ...) or contains(@src, ...))
---
```

Stage 6: Failure-Conditioned Reflection (UNIFIED_REFLECTION_PROMPT)

```
You are an XPath generation expert. Your task is to extract the value of "{attribute}" from web pages.
  ↪ Previous attempts have failed. You must analyze WHY they failed and generate new XPath candidates.

Target attribute: {attribute}
Original query: {query}

=== Previous Attempts ===
{failed_candidates}

=== Extraction Results ===
Page A (seed): {values_a_summary}
Page B (validation): {values_b_summary}

=== Structural Context ===
- Common classes between pages: {common_classes}
- Page A DOM depth: {depth_a}  |  Page B DOM depth: {depth_b}

=== Page HTML ===
Page A:
{html_a_section}

Page B:
{html_b_section}

=== Analysis Required ===
Before generating new XPaths, diagnose the failure:
- If BOTH pages extracted 0 values: the class names or structure you assumed do not exist. Look at the
  ↪ ACTUAL HTML above.
- If Page A extracted values but Page B did not: the XPath is too page-specific. Find patterns SHARED
  ↪ across both pages.
- If extracted values look like navigation/UI elements instead of content: target the main content area
  ↪ instead.
- If extracted too many values: the selector is too broad, add specificity.

=== Generation Rules ===
1. Use ONLY class names, tags, and attributes that you can SEE in the HTML above
2. The XPath MUST extract values on BOTH pages
3. Target CONTENT elements, not navigation bars, sidebars, footers, or UI chrome
4. Prefer SHORT paths (2-3 steps with //)
5. NEVER use positional indices [N]
6. NEVER embed literal text values as predicates
7. Generate 3 candidates using fundamentally DIFFERENT strategies from all previous attempts

Output JSON:
{{
    "diagnosis": "what specifically went wrong and why",
    "strategy": "your new approach, different from all previous attempts",
    "xpath_1": "first new XPath candidate",
    "xpath_2": "second new XPath candidate",
    "xpath_3": "third new XPath candidate"
}}
```

G Additional Analysis of Hierarchical Resolution and FCR

G.1 Resolution Across the Two-Layer Hierarchy

We first provide a more detailed analysis of how failures are resolved across the two-layer hierarchy. Among 326 audited initial failure cases, 136 (41.7%) are resolved at Layer 1 through candidate substitution and execution-grounded selection, without invoking FCR. The remaining cases are escalated to Layer 2 because none of the current candidates satisfies the cross-page eligibility criterion.

The Layer-1 resolution rate varies moderately across failure categories. Among the 67 cases associated with cross-page structural variation (category b), 33 (49.3%) are resolved through candidate selection. For the 259 cases belonging to categories a/c, 103 (39.8%) are resolved at Layer 1. The difference is not statistically significant under a two-sided Fisher's exact test ($p = 0.167$). These results suggest that structural variation can often be handled by selecting an alternative candidate already present in the search space, although candidate substitution does not eliminate all such failures.

This analysis clarifies the role of the hierarchy. EGV does not merely identify failures that are immediately sent to reflection. Instead, execution evidence first acts as a selection signal over the existing candidate space. FCR is invoked only when this first layer fails to produce an eligible cross-page candidate, at which point the same execution evidence is repurposed for diagnosis and targeted candidate regeneration.

G.2 FCR Triggering and Recovery Behavior

We next examine the behavior of FCR after escalation to Layer 2. On Qwen3.7-Plus, FCR is triggered for 22.5% of evaluated attributes. Among triggered cases, 94.2% generate at least one new candidate that subsequently passes EGV, and 32.3% switch to a different anchoring strategy after reflection. Most successful recoveries occur in the first reflection round, motivating the single diagnose-and-regenerate cycle used in the default configuration.

These statistics characterize the behavior of the repair process rather than its ground-truth extraction accuracy. In particular, passing EGV means that a regenerated candidate satisfies the execution-grounded eligibility criteria; it does not guarantee that the extracted value matches the ground truth. Nevertheless, the high post-regeneration pass rate indicates that failure-conditioned feedback frequently moves the search toward candidates that are structurally more consistent across the seed and verification pages.

To isolate the contribution of the diagnosis step itself, we run a paired no-diagnosis control on the same Qwen3.7-Plus trigger set used for the funnel statistics above. Both arms observe the same failed candidates and execution evidence and share the same candidate budget, regeneration protocol, and EGV selection rule; the control arm removes only the failure-mode diagnosis and its mapped repair strategy, replacing them with a neutral regeneration instruction. Since both arms are re-executed under this fixed protocol, the diagnosis-conditioned recovery rate in Table 5 (92.3%) differs slightly from the 94.2% observed in the original main run.

Table 5: Paired no-diagnosis control on the Qwen3.7-Plus trigger set. Recovery rate: fraction of triggered cases in which at least one regenerated candidate passes EGV. Correct-repair rate: fraction of triggered cases in which a regenerated candidate matches the ground truth.

Repair variant	Recovery rate	Correct-repair rate
Diagnosis-conditioned (FCR)	92.3%	19.5%
No-diagnosis control	78.8%	9.1%

Diagnosis-conditioned repair improves both margins: it recovers 92.3% of triggered cases versus 78.8% for the control, and more than doubles the correct-repair rate (19.5% vs. 9.1%). The gain of FCR therefore cannot be attributed to regeneration alone; conditioning repair on the diagnosed failure mode improves both the quantity and the correctness of recovered candidates.

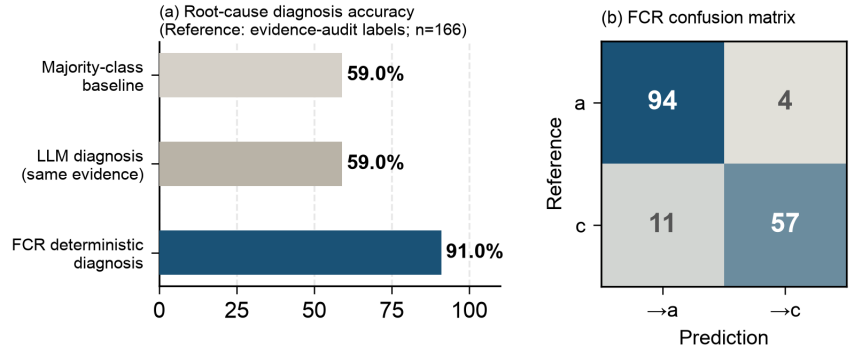


Figure 6: Failure-diagnosis analysis on 166 audited reflection-triggered cases. Left: diagnostic accuracy of the structured execution-grounded procedure, direct LLM judgment, and the majority-class baseline. Right: confusion matrix of the structured diagnosis procedure. The current audited subset contains categories a and c only.

G.3 Diagnostic Value of Execution-Grounded Evidence

We further examine whether the execution traces used by FCR contain repair-relevant information about the underlying failure cause. Manual root-cause annotations are available for 166 reflection-triggered cases with complete diagnostic evidence. The currently audited subset covers categories a and c, but does not contain category-b cases; the following results should therefore be interpreted within this restricted diagnostic scope.

Figure 6 compares the structured execution-grounded diagnosis procedure with direct LLM judgment from the same evidence. The structured procedure achieves 91.0% diagnostic accuracy, compared with 59.0% for direct LLM judgment. As shown by the confusion matrix, direct LLM judgment tends to collapse toward the majority failure category, whereas the structured procedure more reliably separates the audited failure causes.

The comparison is intended to test whether structured execution evidence provides a more reliable basis for repair-oriented diagnosis than free-form model judgment. It should not be interpreted as a complete three-way evaluation of all FCR failure categories. In particular, category-b failures must be explicitly included in the diagnostic audit before drawing conclusions about full a/b/c diagnostic accuracy. Within the currently audited a/c subset, however, the results support the use of structured execution signals for conditioning subsequent candidate regeneration.

H End-to-End Case: FCR-Driven Cross-Page Repair

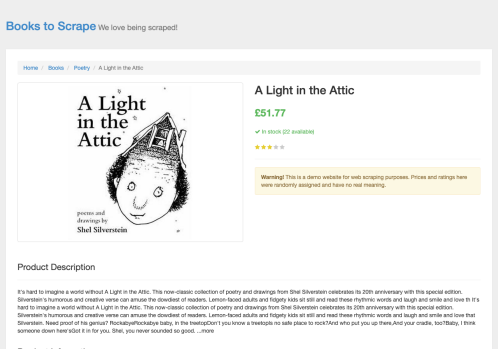
We use books.toscrape.com to illustrate the complete CogExtract workflow.

Task Configuration

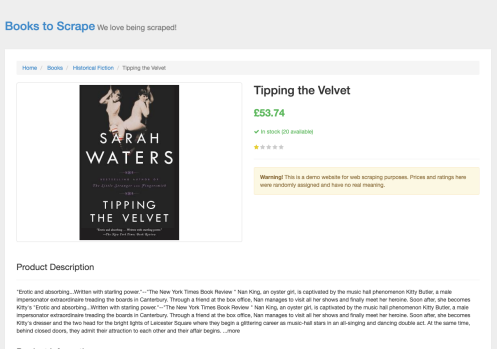
Website	books.toscrape.com
Query	Scrape the book stock status and associated image
Target	`image_url` (image modality)
Ground Truth	`['../../media/cache/fe/72/fe72f0...jpg']`
Group Size	1,000 pages (different books on the same site)

EGV Cross-Page Verification: Seed Page vs. Validation Page

After the XPath is generated on the seed page (Page A), it is executed on the validation page (Page B) of the same group, comparing extraction-count consistency between the two pages.



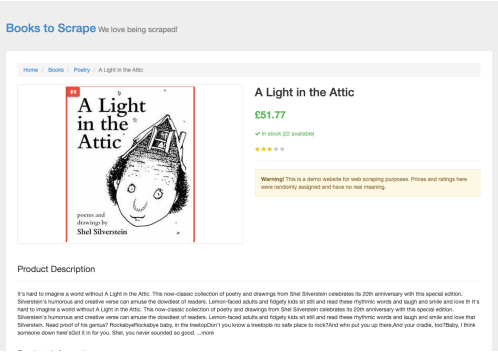
Page A (Seed): A Light in the Attic



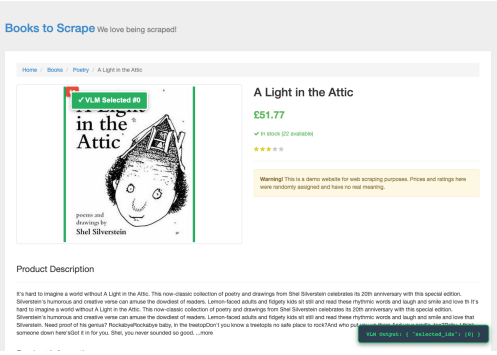
Page B (Validation): Tipping the Velvet

Stage 3b: Element Selection (SoM)

For the image modality, the VLM receives an SoM-annotated screenshot (colored boxes marking `<img>` elements) and selects #0 (the main product cover image) as the candidate.



SoM-annotated screenshot—colored boxes mark candidate `<img>` elements



VLM selection result: selected_ids = [0]

Stage 6: FCR Repair

When cross-page verification fails, the diagnosis, grounded in deterministic execution evidence, localizes the root cause and triggers targeted rule repair. In this case, FCR finds that the XPath extracts an inconsistent number of elements on the validation page, diagnoses category c (rule formulation error), corrects the anchoring strategy of the XPath and regenerates it, and re-verification passes.

I Additional Ablation Materials

Table 6: Diversity-controlled comparison (75 stratified-sampled groups, paired comparison)

Condition	Strategy entropy	Trigger rate	Group-mean F1	Diff. vs. (a)
(a) Diversity instruction + temp 0 (default)	0.941	21.8%	60.55	—
(b) No instruction + temp 1.0	0.879	21.4%	59.16	−1.39
(c) No instruction + temp 0	0.861	20.6%	58.65	−1.90
(d) Forced single strategy (class-based)	0.101	31.3%	48.92	−11.63

We further examine how candidate coverage changes with the candidate-set size K. As shown in Figure 7, coverage increases rapidly for the first few candidates: the first three candidates already capture 92.5% of the distinct structural variants present in the full $K = 8$ pool. The marginal gain then diminishes, with the curve largely plateauing after $K = 5$. This trend indicates that a small candidate set captures most of the available structural diversity, supporting our default choice of $K = 3$ as a practical trade-off between search coverage and generation cost.

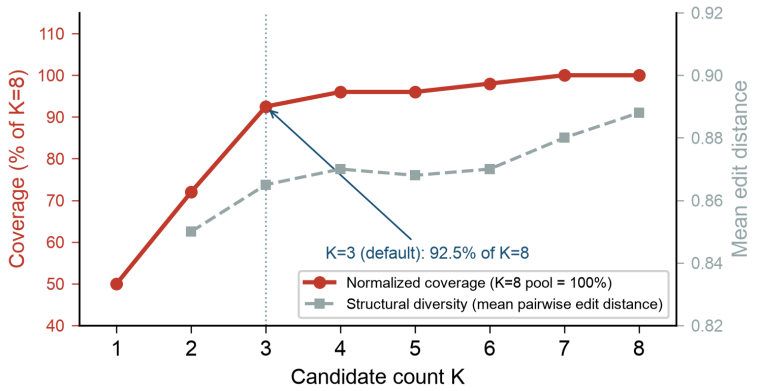


Figure 7: K saturation curve (DCG-Exp-1, qwen3.7-plus, prefix subsets of a single K=8 generation over 131 attributes). Coverage is the fraction of distinct structural variants in the K=8 candidate pool captured by the first k candidates; K=3 already reaches 92.5% of K=8, and the curve plateaus after K=5.

J Dataset Composition Details

J.1 Breakdown by Website

Table 7: LiveWeb-IE dataset breakdown by website. dp.art is excluded due to anti-scraping mechanisms and is not counted in the totals.

Website	Domain	Groups	URLs	Attributes
arxiv.org	Academic	12	471	title, author, subject, pdf_link, author_profile_link
iclr.cc	Academic	11	1,285	title, author, abstract_link, author_profile_link
neurips.cc	Academic	12	1,652	title, author, abstract_link, author_profile_link
icml.cc	Academic	13	1,391	title, author, abstract_link, author_profile_link
books.toscrape.com	E-commerce	32	16,800	title, price, stock_status, category, book_image, …(8)
sandbox.oxylabs.io	E-commerce	24	13,278	title, price, genre, platform, developer, image, …(8)
fueleconomy.gov	Government	28	701	vehicle, vehicle_type, tank_size, driver_mpg, …(7)
quotes.toscrape.com	Content	10	180	quote, author, birth, author_profile_link, tag_link
marinespecies.org	Natural science	43	952	species, species_image, species_link, author, …(13)
thesportsdb.com	Sports	62	3,209	team, team_link, member_image, poster, banner, …(19)
themealdb.com	Food	16	3,627	meal, meal_image, meal_link, ingredient, …(7)
thecocktaildb.com	Food	15	5,263	cocktail, cocktail_image, cocktail_link, …(6)
scrapethissite.com	Practice	21	197	country, capital, population, team, turtle_image, …(14)
dp.art	*Academic (excluded, anti-scraping)*	30	2,008	*title, collection, source_image, article_link, …(12)*
huggingface.co	AI	13	1,518	title, author, paper_link, model_citation_count, …(8)
Total	**14 sites**	**312**	**50,524**	**113 distinct attributes**

J.2 Task Type Distribution

Table 8: LiveWeb-IE task type distribution

Type	Description	Groups	Share
type_1	Single-attribute extraction	65	20.8%
type_2	Multi-attribute extraction	66	21.2%
type_3	List extraction	97	31.1%
type_4	Paired extraction	84	26.9%
Total		312	100%

J.3 Attribute Modality Distribution

Automatically classified by attribute-name keywords: containing `image/photo/poster/banner/logo/icon` → image; containing `link/url/href` → link; otherwise → text. 20 attributes (3.1%) could not be classified by the keyword rules and are not counted.

Table 9: LiveWeb-IE attribute modality distribution

Modality	Count	Share	Notes
Text	329	50.9%	Numbered HTML list
Link	195	30.2%	Triggers the SoM annotation flow
Image	102	15.8%	Triggers the SoM annotation flow
Total	626	96.9%	

The Image and Link modalities trigger VGS's SoM annotation flow (Stage 3b); the Text modality uses numbered HTML lists. In total there are 646 attribute instances (329+195+102+20).

K Thinking vs. Non-Thinking Models

Table 10 reports the complete comparison between instruct and thinking variants of Qwen3-VL-8B, Qwen3-VL-30B-A3B, and Qwen3-VL-32B across CogExtract, VGS, CoT, and Reflexion.

Table 10: Thinking vs. non-thinking comparison (3 model scales, F1 / P / R)

	CogExtract			VGS			CoT			Reflexion		
Model	F1	P	R	F1	P	R	F1	P	R	F1	P	R
qwen3-vl-32b (instruct)	37.1	30.2	48.1	17.1	12.6	26.5	24.0	22.7	25.5	22.3	18.9	27.1
qwen3-vl-32b (thinking)	69.0	66.7	75.5	25.0	24.1	28.0	16.5	16.0	18.6	17.7	16.9	20.8
qwen3-vl-30b (instruct)	30.5	25.5	37.9	22.6	18.6	28.8	13.9	12.8	15.2	12.8	10.2	17.2
qwen3-vl-30b (thinking)	37.1	36.0	41.1	26.6	24.5	35.0	29.6	28.5	33.8	34.5	33.9	36.0
qwen3-vl-8b (instruct)	21.4	15.4	35.1	15.9	13.2	25.8	10.7	8.9	13.4	9.6	7.4	13.9
qwen3-vl-8b (thinking)	36.2	35.4	39.2	28.0	26.7	33.9	17.4	16.8	19.4	19.7	19.1	21.8

L Execution-Grounded Verification Details

This appendix provides the implementation details of the Execution-Grounded Verification (EGV) score introduced in Section 2.4. EGV evaluates XPath candidates using only cross-page execution behavior and DOM structures, without access to ground-truth extraction labels or additional LLM/VLM calls.

Given a candidate XPath rule r, a seed page p_{seed}, and a verification page p_{val}, the candidate is executed on both pages:

$$M_{\text{seed}} = r(p_{\text{seed}}), \quad M_{\text{val}} = r(p_{\text{val}}), \tag{5}$$

where M_{seed} and M_{val} denote the matched elements and extracted values returned by the XPath execution.

A candidate is eligible only if it produces non-empty results on both pages, i.e., $|M_{\text{seed}}| > 0$ and $|M_{\text{val}}| > 0$. This eligibility check is a hard gate rather than a scored signal: it ensures execution feasibility but is not treated as a correctness indicator, since a candidate may consistently extract irrelevant content. Ineligible candidates are excluded from ranking; if no candidate is eligible, EGV returns the execution evidence to FCR (Section 2.5).

The EGV score of an eligible candidate is computed as:

$$S(r) = \sum_{i=1}^{5} w_i s_i(r), \tag{6}$$

where $s_i(r)$ represents an individual execution-grounded signal and w_i represents its fixed weight.

The default weight configuration is:

$$(w_1, w_2, w_3, w_4, w_5) = (0.20, 0.10, 0.10, 0.15, 0.15)/0.70 \approx (0.286, 0.143, 0.143, 0.214, 0.214). \tag{7}$$

The weights are fixed before evaluation and are not optimized on the test benchmark. They are designed to balance structural consistency, extraction plausibility, and over-selection control. Since the weights are only used to rank eligible candidates, the normalization constant does not affect selection. Section 5.3 further examines the robustness of EGV to alternative weighting schemes.

L.1 Count Stability

The count stability signal evaluates whether the number of extracted elements remains consistent across pages.

Let:

$$n_{\text{seed}} = |M_{\text{seed}}|, \quad n_{\text{val}} = |M_{\text{val}}|. \tag{8}$$

The score is defined as:

$$s_{\text{count}}(r) = 1 - \frac{|n_{\text{seed}} - n_{\text{val}}|}{\max(n_{\text{seed}}, n_{\text{val}}) + \epsilon}, \tag{9}$$

where ϵ is a small constant.

A candidate that produces substantially different numbers of matched elements across sibling pages receives a lower score, as such behavior often indicates unstable or overly broad XPath expressions.

L.2 Depth Consistency

The depth consistency signal measures whether the matched elements occupy similar structural positions in the DOM hierarchy.

For a matched node x, let $\text{depth}(x)$ denote its DOM depth. The average depth on each page is:

$$d_{\text{seed}} = \frac{1}{|M_{\text{seed}}|} \sum_{x \in M_{\text{seed}}} \text{depth}(x), \tag{10}$$

$$d_{\text{val}} = \frac{1}{|M_{\text{val}}|} \sum_{x \in M_{\text{val}}} \text{depth}(x). \tag{11}$$

The consistency score is:

$$s_{\text{depth}}(r) = \exp\left(-\frac{|d_{\text{seed}} - d_{\text{val}}|}{d_{\max}}\right), \tag{12}$$

where $d_{\max}$ is the maximum DOM depth observed across the seed and verification pages.

L.3 Value-form Coherence

The value-form coherence signal captures whether the extracted values have similar surface characteristics across pages.

For each extracted value x, we compute its normalized length:

$$l(x) = |x|. \tag{13}$$

The extracted lengths from the two pages are converted into discrete distributions:

$$P_{\text{seed}}(l), \quad P_{\text{val}}(l). \tag{14}$$

The Jensen–Shannon divergence is used to measure distribution difference:

$$D_{\text{JS}} = \frac{1}{2} D_{\text{KL}}(P_{\text{seed}} \| M) + \frac{1}{2} D_{\text{KL}}(P_{\text{val}} \| M), \tag{15}$$

where:

$$M = \frac{P_{\text{seed}} + P_{\text{val}}}{2}. \tag{16}$$

The coherence score is:

$$s_{\text{coh}}(r) = 1 - D_{\text{JS}}. \tag{17}$$

This signal serves as a lightweight deterministic proxy for extracted-value consistency rather than semantic understanding.

L.4 Attribute Plausibility

The plausibility signal evaluates whether extracted values satisfy attribute-specific deterministic constraints.

For each extracted value x, a validity function is defined:

$$v(x) \in \{0, 1\}. \tag{18}$$

The validity criteria depend on the target attribute type. Examples include:

- Text attributes: non-empty textual content.
- Image attributes: valid image source references.
- Numeric attributes: values matching numeric patterns.

The plausibility score is:

$$s_{\text{plaus}}(r) = \frac{\sum_{x \in M_{\text{seed}}} v(x) + \sum_{x \in M_{\text{val}}} v(x)}{|M_{\text{seed}}| + |M_{\text{val}}|}. \tag{19}$$

L.5 Specificity

The specificity signal controls the tendency of XPath candidates to match excessively broad regions.

Let n denote the number of extracted elements and τ denote the expected matching cardinality according to page type.

The specificity score is:

$$s_{\text{spec}}(r) = \exp\left(-\alpha \frac{|n - \tau|}{\tau + \epsilon}\right), \tag{20}$$

where α controls the penalty strength.

For detail pages, τ is close to one, encouraging precise extraction. For listing pages, larger matching cardinalities are allowed.

L.6 Weight Robustness

To examine whether EGV performance depends on the manually specified weights, we compare the default configuration with alternative weighting strategies.

In particular, we evaluate:

- uniform weighting, where all five signals receive equal weights;
- perturbed weighting, where individual weights are randomly varied while maintaining normalization.

The results show that the candidate discrimination ability remains stable under alternative weighting schemes, suggesting that the effectiveness of EGV comes primarily from combining complementary execution-grounded signals rather than from precise weight tuning.