RAISE: LOSS-AWARE, DIRECTION-FREE RETRIEVAL OVER HIERARCHICAL DOCUMENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

When querying over thousands of multimodal documents, the evidence a query needs often sits in a table, a chart, or even within the page layout. Recent retrievers therefore build each document into a tree whose internal nodes carry short summaries of their children, and an agent descends it like a table of contents, opening only the promising branch. A summary, however, is written before any query is known and under a length budget, so it cannot keep every detail of its children. We call the resulting limitation the *abstraction bottleneck*: when a summary omits the detail a query asks about, the agent can no longer tell the right branch from its siblings, and must either guess or open them all. We present RETRIEVAL OVER ABSTRACTION-INDUCED SUMMARY ERASURE (RAISE), which records the omission caused by summarization and routes on it. Offline, RAISE contrasts every summary against the children it was written from and records what the summary omitted as *loss witnesses*, each attached to the parent-to-child edge it came from, which together form the LOSS LEDGER. Online, an agent traverses the documents by extending a *map* of its traversal so far, and the witnesses serve it in two ways: (i) a witness is itself retrievable, so a query can match it and start its traversal from the edge the witness is attached to, and (ii) the witnesses are drawn on the map, so the agent can judge a branch before opening it. The agentic traversal is also *direction-free*: it can start traversing at any level of a tree without losing the hierarchy, and extend downward or upward at every step, so no direction has to be fixed before retrieving. Extensive experimental results show that RAISE achieves the best average accuracy on eight multimodal document QA benchmarks in the open-corpus setting, outperforming the strongest of seven competitors by 5.8 points of exact match on average.

1 INTRODUCTION

Real-world queries are often asked over collections of multimodal documents, such as annual reports, scientific papers, product manuals, slide decks, and web pages, where the evidence may sit in a table, a chart, or the page layout. Answering them means searching a corpus of such documents, a task we call *multimodal document RAG* (MDRAG): given a query and a corpus but no chosen document, retrieve a small set of source-grounded *components* (e.g., paragraphs, tables, and images) that support an answer, then generate from them (Lewis et al., 2020). A document, unlike a passage, has three properties that shape what such a retriever must do. (i) Its components differ in modality, so matching a query against them has to work across text, tables, and images (Ma et al., 2024; Suri et al., 2025; Dong et al., 2025a; Shen et al., 2025). (ii) A component's meaning is often determined by the components around it: its caption, the column header above a cell, the heading of its section. (iii) The surrounding context is organized as a hierarchy (a title over sections, sections over pages and components), so the evidence for a query may be a small component deep inside a long document.

MDRAG methods have advanced by honoring more of these properties in turn. *Vision-native* retrieval embeds the rendered page of each document (Faysse et al., 2025; Yu et al., 2025; Cho et al., 2024), and *component-wise* retrieval indexes each parsed component separately (Yun et al., 2025; Chen et al., 2024; Zhong et al., 2025; Mei et al., 2025); both address modality, and both still judge a component on its own. Recent work therefore makes the document's own organization the search space: each document is expressed as a tree that follows its sections and subsections, and every internal node carries a short summary of the content beneath it. A large language model (LLM) then descends such

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.

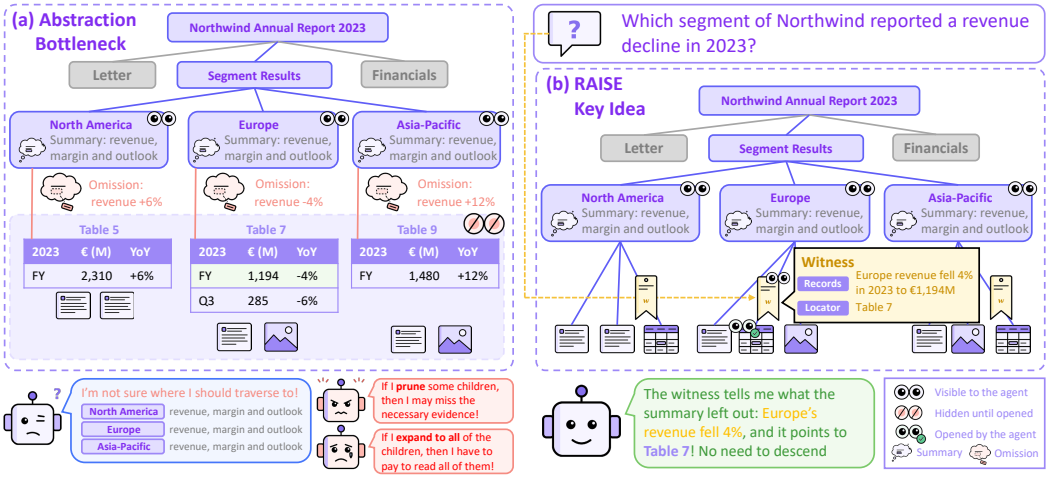


Figure 1: An illustrative annual report. **(a)** The abstraction bottleneck: `Table 7` shows that the revenue of `Europe` fell 4% in 2023, but no summary above it retains the sign of the change, so an agent that sees only the summaries must either prune and risk a miss or expand every candidate. **(b)** RAISE: a loss witness records the omitted fact and the table's locator on the edge into `Table 7`, the query matches the witness directly, and the agent opens the table without descending from the root.

a tree in the way a reader uses a table of contents, comparing the summaries of a node's children with the query at each level and opening only the branch that appears promising. Such a descent restores what the earlier methods lack, since the ancestors along the chosen path supply the context that a lone component does not carry. It is also efficient, because the branches left unopened are never read, and such pruning is precisely what makes a hierarchy worth building.

Despite the benefits, pruning during a traversal is safe only if each summary retains whatever distinguishes its node from its siblings. A summary, however, is by nature an overview that may omit the very cues a query needs. Figure 1(a) makes this concrete: a query asks which segment of `Northwind` reported a revenue decline in 2023, and `Table 7` in the `Europe` subsection of `Segment Results` contains the answer. The table shows the revenue of `Europe` falling 4% to €1,194M, but the ancestor summaries omit the sign of the change, so an agent that sees only the summaries of `North America`, `Europe`, and `Asia-Pacific` must either prune and risk missing the table, or expand every candidate at additional cost. We call the resulting limitation the *abstraction bottleneck*, and it is an information constraint rather than a reasoning-quality one: a stronger LLM cannot reason from what no observation contains. Existing hierarchical retrievers inherit this phenomenon: their agent descends top-down over summaries (Chen et al., 2023; Vectify AI, 2025), or searches every level at once but returns a flat list and traverses no further (Sarthi et al., 2024), and an upward or structure-aware pass (Tao et al., 2025; Xu et al., 2025; Gupta et al., 2025; Shin et al., 2026) only relaxes the traversal order while routing on the same lossy summaries.

We address this problem by recording what each abstraction step omits and letting the retriever route on what was recorded, which takes two coupled ideas. (i) We make the omission observable: each omitted item is written down as a *loss witness*, attached to the edge between a parent and the child, and the witnesses together form the LOSS LEDGER. A witness reaches the agent in two ways. It is a retrieval target: in Figure 1(b), witness w on the edge from the `Europe` subsection to `Table 7` records "Europe revenue fell 4% in 2023 to €1,194M", and a query about the decline matches it and adds `Table 7` to the agent's map although no summary above it mentions a decline. It is also displayed on that edge of the map, so the agent can judge a branch by what its summary omitted before opening it. (ii) Retrieval is agentic and direction-free: the agent draws its own *map* of the corpus and expands it one node at a time, entering a tree wherever a search matches and growing the map from any node on it, with the ancestors of every hit drawn in, so no direction is fixed in advance. Agentic retrieval itself is standard (Yao et al., 2023; Trivedi et al., 2023; Jeong et al., 2024; Wu et al., 2025b; Sun et al., 2025; Wu et al., 2025a;c; Yun et al., 2026); what is new is a map that restores the structure of the hits and a search that reaches the summary, leaf, and loss channels in one call.

We instantiate these ideas as RETRIEVAL OVER ABSTRACTION-INDUCED SUMMARY ERASURE (RAISE). Offline, it builds an index of one tree per document, whose internal nodes carry summaries

and whose edges carry a bounded number of source-linked loss witnesses, together with vector stores that make the trees searchable. Online, an orchestrator LLM traverses such an index by searching summaries, leaves, and witnesses at once, adding each hit to a *map* with its ancestors, and reading or expanding nodes of the map until every obligation it declared is certified from a read node.

In summary, we make four contributions:

1. **The abstraction bottleneck.** We characterize the abstraction bottleneck: when the content that discriminates a query lies in what the summaries omitted, an agent that routes on summaries alone is limited by the omission, regardless of how well it reasons (§2).
2. **The LOSS LEDGER.** We propose a novel data structure that records what each summary omitted as a bounded set of source-linked loss witnesses on the edges of the tree, so that the agent can both search them and read them on its map (§4).
3. **Direction-free agentic retrieval.** We design an agentic traversal that fixes no direction in advance: one search reaches summaries, leaves, and witnesses at once, and the map keeps the ancestors of every hit, so the traversal can start at any level of a tree without losing the hierarchy (§4, §D.6).
4. **Accuracy gains.** Through extensive experiments on eight corpora of web pages and PDFs against seven competitors, we show that RAISE achieves the best average accuracy, outperforming the strongest competitor by 5.8 points of exact match with a local backbone, and we analyze where the ledger, the traversal, and the backbone contribute (§5, §D).

2 PRELIMINARIES AND PROBLEM STATEMENT

Corpus and components. Let a corpus $\mathcal{C} = \{D_1, \ldots, D_N\}$ contain heterogeneous documents such as web pages and PDF reports. Each document decomposes into an ordered list of source-grounded *components* (text, a list, a table, or an image), each carrying a *locator* into the original artifact, such as a page and a position on it.

Retrieval task. Given a query $\mathcal{Q}$, the system must return an answer A together with a source-grounded evidence set $R_{\mathcal{Q}}$ that suffices to answer $\mathcal{Q}$, drawn from original content rather than from a generated summary, under an explicit budget on retrieval and LLM calls.

Hierarchical indexing. A retriever may build an index I over $\mathcal{C}$, and we build on the one recent hierarchical retrievers adopt, the document's own organization (Sarthi et al., 2024; Chen et al., 2023; Vectify AI, 2025). Each document D is mapped to a rooted ordered tree $\mathcal{T}_D = (V_D, E_D)$, so that $I = \{\mathcal{T}_{D_1}, \ldots, \mathcal{T}_{D_N}\}$: its leaves hold the components of D, and its internal nodes hold natural language *abstractions* of their descendants at the granularity of a page, a subsection, or a section. Every node v carries a *label* $l(v)$, the text a retriever matches against: a leaf's label is its own content, and an internal node's label is a summary written from the labels of its children under a finite token budget b. Because b is finite, $l(v)$ is generally lossy, and we write the *omission* carried by edge (v, c) as

$$\Delta(v, c) \;=\; \big\{\, \text{retrieval-relevant content recoverable from } l(c) \text{ but not from } l(v) \,\big\}.$$

$\Delta(v, c)$ is defined without reference to any particular query, as what some query could obtain from $l(c)$ but not from $l(v)$, and §4.1 approximates it offline by a bounded set of witnesses. None of the hierarchical retrievers surveyed in §3 represents $\Delta(v, c)$.

The abstraction bottleneck. We now state the limitation that this paper addresses, which we call the *abstraction bottleneck*. An agentic retriever searches such an index with an *agent* that traverses it and decides which children to expand at each node, over an index that was built lossily before the agent ever runs (§2). A *pure top-down* agent starts at the root and observes only the summaries of the nodes it visits, so it meets the bottleneck: summaries are the only evidence it can weigh when choosing among children, and they may have omitted exactly what distinguishes them. Suppose the answer lies under a target child $c^\star$, and the content distinguishing $c^\star$ from $m - 1$ of its siblings lies entirely in the omissions on their edges. All m children are then identical to the agent before expansion, so under a uniform target prior, expanding $k < m$ of them before any further observation reaches $c^\star$ with probability at most k/m, and pruning nothing is the only way to guarantee recall (§A). What bounds the agent is what it can observe rather than how well it reasons, so escaping the bound needs an observation the summaries do not carry: the omission itself, whose worth §5.4 measures, or the leaf content beneath them.

3 Related Work

Hierarchical document retrieval. Recent retrievers organize a corpus as trees of summaries rather than retrieving flat chunks. RAPTOR recursively clusters children into parent summaries and retrieves over the tree or a collapsed pool of its nodes, which searches every level at once (Sarthi et al., 2024), and MemWalker and PageIndex navigate by LLM reasoning over the summaries themselves, returning to a parent when a branch fails (Chen et al., 2023; Vectify AI, 2025). More recent work traverses the tree dynamically: TreeRAG combines a leaf-to-root pass with the root-to-leaf one (Tao et al., 2025), and HiKEY casts multimodal open-domain document QA as hierarchical routing (Shin et al., 2026). None of them addresses the abstraction bottleneck (§2: reading a tree leaf-to-root as well as root-to-leaf changes which summaries are seen first, not what those summaries say.

Summary information loss. A summary compressed under a budget trades recall for precision against full-text indexing (Sakai & Jones, 2001), and QA-based faithfulness estimators quantify how much of a source a summary conveys (Durmus et al., 2020; Deutsch et al., 2021; Scialom et al., 2021). Their output is a number about a summary, which never says what is missing or where.

Multimodal document retrieval. Vision-native retrievers embed the rendered page directly, so figures, layout, and typography survive the match (Faysse et al., 2025; Yu et al., 2025; Wang et al., 2026; Cho et al., 2024), and later systems combine several signals over the page (Jain et al., 2025; Suri et al., 2025). Most recent work models the search as an agentic process that interleaves reasoning with retrieval (Trivedi et al., 2023; Yao et al., 2023) and coordinates tools or agents over a document or a visual index (Zhu et al., 2026; Wang et al., 2025). Closest to our work, FIF casts multimodal graph traversal as a sequential decision process and turns a failed hop into feedback that re-anchors the traversal (Yun et al., 2026). Such systems differ in how the next node to expand is chosen, not in what they can observe: summaries and original content, never the omission that separates the two.

4 RAISE

We propose Retrieval over Abstraction-Induced Summary Erasure (RAISE), an agentic retriever over the hierarchical index of §2 that records the omission $\Delta(v, c)$ and uses the record during retrieval, instead of attempting to summarize it away (Figure 2). Offline (§4.1), RAISE builds the index: one tree per document, with summaries on its internal nodes and *loss witnesses* on its edges, and vector stores over the trees. Each witness records one item that the summary above its edge omitted, and the witnesses of all edges form the Loss Ledger. Online (§4.2), an orchestrator sets sub-queries and obligations, traverses the index while recording the traversal on a *map*, and certifies what it reads as evidence until the obligations are closed. The ledger supports the traversal through two mechanisms. In *loss search*, witnesses are retrieval targets, so a branch can be reached even when no summary above it matches the query. In *map observations*, witnesses are displayed on the edges of the map, so a branch can be assessed before it is opened. Independently of the ledger, a search can start at any level of a tree and the map can be extended from any node on it, so the agent never commits to a traversal direction.

4.1 Offline: Building the Index

The offline stage builds the index of RAISE, which has two parts: the trees $I = \{\mathcal{T}_D\}_{D \in \mathcal{C}}$ of §2, whose internal nodes are labelled with summaries $l(v)$ and whose edges are labelled with loss witnesses $\mathcal{W}(v, c)$, and a set of vector stores $\mathcal{V}$ that makes them searchable. Steps (1) and (2) build the trees and their node labels as prior hierarchical indexes do, step (3) adds the edge labels, whose collection over the corpus is the Loss Ledger $\mathcal{L}$, and step (4) builds $\mathcal{V}$.

(1) Component tree. The first step builds one tree $\mathcal{T}_D$ per document D, the structure of §2 on which every later step operates. Its leaf nodes hold the components of D, and its internal nodes are initially filled with the section and subsection titles of the document. The edges among internal nodes express the document's own hierarchy: a subsection that sits lower in the hierarchy is placed in the tree as a child of its parent section. Since the documents normally do not come with such a tree, we recover it from the source format. For the Wikipedia-based HTML corpora, we parse the markup: the nesting of the heading tags gives the section hierarchy, and the paragraph, table, list, and image elements under each heading become its components. For the PDF corpora, we use MinerU (Wang et al., 2024), which returns typed components with their positions on the page; it returns section headers as a flat list, so one LLM call per document re-nests them, and a page node is placed between each leaf section and its components (§C.1). Every component keeps a locator into the original artifact, and images are verbalized once so that they are searchable while their rendered region stays readable (§C).

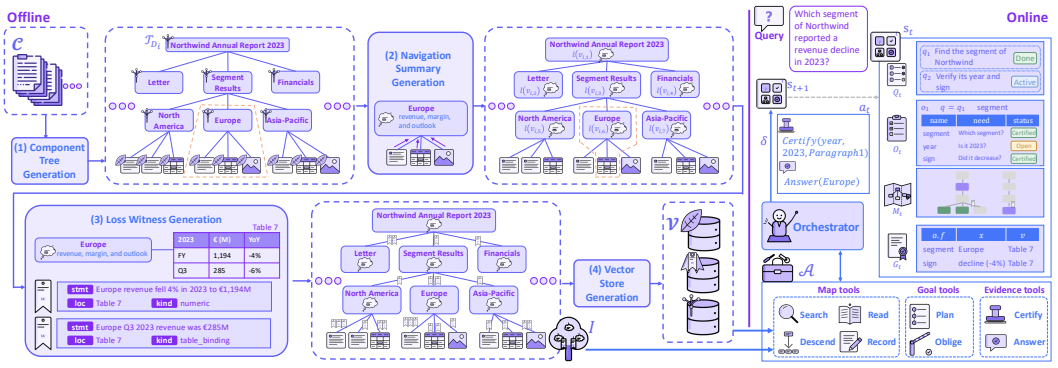


Figure 2: Overview of RAISE on a corpus of annual reports. **Offline (left):** the four steps of §4.1 build the index I of one document, a component tree with a navigation summary $l(v)$ on every internal node and loss witnesses $w = \langle \mathrm{stmt}, \mathrm{loc}, \mathrm{kind} \rangle$ on its edges, and the three vector stores $\mathcal{V}$. **Online (right):** one step of §4.2: the orchestrator observes $s_t = (Q_t, O_t, M_t, G_t)$, emits the batch a_t, and the transition δ yields s_{t+1}; the tools $\mathcal{A}$ are grouped by the part of the state they change. The obligation o_1 still has its `year` field open, so the batch certifies it before answering. The example and the episode state are illustrative.

(2) Navigation summaries. The second step fills every internal node of $\mathcal{T}_D$ with a *navigation summary* $l(v)$, a short text that tells an agent standing at v what lies in the subtree below it. The summaries are generated bottom-up, since $l(v)$ summarizes the labels $l(c)$ of the children c of v (§2), so every child is complete before its parent. Each summary is generated by one call to `Qwen3.6-35B-A3B`, with the rendered page attached when v is a page node, and the call is instructed to be decisive rather than complete (§F.2): $l(v)$ commits to what separates v from its siblings and may omit the rest, so a table is reduced to its subject and dimensions and a chart to what it plots.

(3) The LOSS LEDGER. The third step labels every edge (v, c) of $\mathcal{T}_D$ with a set of *loss witnesses* $\mathcal{W}(v, c)$. Each witness in the set records one item that $l(v)$ omitted about the child c, so an edge carries as many witnesses as its summary omitted, from none up to a cap N. The LOSS LEDGER $\mathcal{L}$ collects the labels of every edge in the corpus; an edge label $\mathcal{W}(v, c)$ holds the witnesses of one edge; and a loss witness w is one written record of one omitted item:

$$\mathcal{L} = \{\, \mathcal{W}(v, c) : (v, c) \in E_D,\ D \in \mathcal{C} \,\},$$
$$\mathcal{W}(v, c) = \{w_1, \ldots, w_n\} \ \text{ with } \ n \le N, \qquad w = \langle\, \mathrm{stmt},\ \mathrm{loc},\ \mathrm{kind} \,\rangle.$$

Here stmt states the omitted content as a self-contained sentence, phrased the way a future query would phrase it and never as a bare value such as "09:00". The locator loc points into the related child node c, so the agent can later open it, and kind is one of seven types of omission: an entity, a numeric value, a table binding, a relation across children, a qualifier, a temporal event, or visual content (Table 6).

RAISE generates the loss witnesses with one LLM call per internal node v. The call receives the summary $l(v)$ of step (2) together with the children it was written from, and returns the items that an agent reading $l(v)$ alone could not find. It discards the items that the summary already conveys and attaches each remaining item to the edge into the child it was read from; an item supported by several children is stored on the node and shown with each of them (§C). Figure 2, step (3), shows the resulting witness $w = \langle$"Europe revenue fell 4% in 2023 to €1,194M", `Table 7`, `numeric`$\rangle$ on the edge from the `Europe` subsection of an annual report to its `Table 7`.

(4) Three vector stores in one embedding space. The last step builds the three vector stores $\mathcal{V}$ that make the trees searchable, one for each kind of object the first three steps placed on a tree: the internal nodes with their summaries, the component nodes at the leaves, and the loss witnesses on the edges. A multimodal embedder encodes all three into one embedding space, so that they can be compared against the same query. Online, the stores are served as the search channels *summary*, *leaf*, and *loss*, and a loss hit resolves to the edge of its witness.

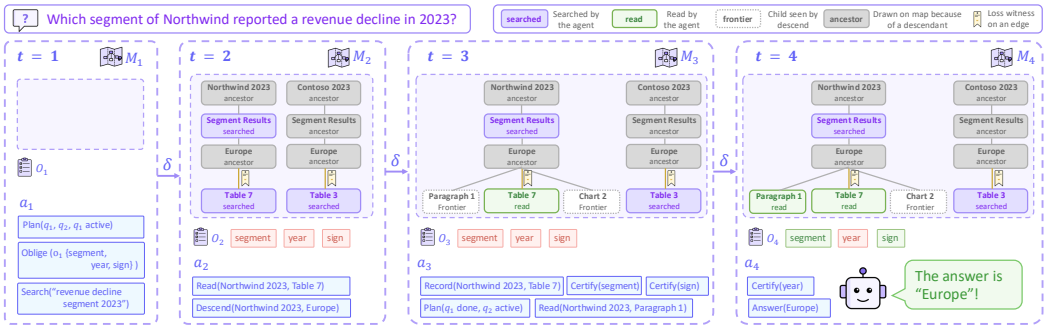


Figure 3: One episode of RAISE on the example of Figure 2. Each frame shows the state the orchestrator observes at step t, its map M_t and obligation fields O_t, and the batch a_t it emits; the transition δ yields the next frame. The state of Figure 2 is the fourth frame.

4.2 Online: Agentic Retrieval on a Map

Online, RAISE answers a query agentically: a single orchestrator LLM explores the index over a bounded number of steps and stops as soon as it can ground its answer. We model an episode as a sequential decision process $\langle \mathcal{S}, \mathcal{A}, \delta \rangle$ with a state space $\mathcal{S}$, a set of tools $\mathcal{A}$, and a transition δ. At step t the orchestrator observes the state $s_t = (Q_t, O_t, M_t, G_t) \in \mathcal{S}$: its sub-queries and obligations, a map of its traversal so far, and its evidence, rendered from scratch at every step under a fixed token budget (§C). Using the observation, the orchestrator emits an action a_t, a *batch* of calls to the tools in $\mathcal{A}$ that the system executes concurrently (§C gives the execution rules), and the transition folds their results into the next state, $s_{t+1} = \delta(s_t, a_t)$ (Figure 3 traces one episode step by step). The tools fall into three groups, one per part of the state:

$$\mathcal{A} = \underbrace{\{\textsc{Plan},\ \textsc{Oblige}\}}_{\text{goal: } Q_t,\, O_t} \cup \underbrace{\{\textsc{Search},\ \textsc{Descend},\ \textsc{Read},\ \textsc{Record}\}}_{\text{map: } M_t} \cup \underbrace{\{\textsc{Certify},\ \textsc{Answer}\}}_{\text{evidence: } G_t}.$$

Sub-queries and obligations. Sub-queries break the query $\mathcal{Q}$ into parts that a single search can satisfy and say what to look for next; obligations say when the episode may stop. The sub-queries are kept in Q_t, a list in which each sub-query has a status and optional dependencies on earlier sub-queries (§C), and at most one of them is active. Plan manages Q_t: it adds sub-queries, drops them, marks them done, and sets the active one.

Obligations make explicit the conditions that $\mathcal{Q}$ places on its answer. Each obligation in O_t belongs to one sub-query and names one value the answer needs. The value comes with the fields F that condition it (the entity, the period, or the unit). A field $f \in F$ is *open* until a fact is certified for it and *contradicted* once a second and different fact is bound to it. An obligation is satisfied when no field is open or contradicted. Oblige(q, F) manages O_t: it adds to sub-query q an obligation with fields F, and obligations are only added, never edited.

Map. M_t is a subgraph of the document forest that records where RAISE has traversed so far. Its nodes carry what the orchestrator has obtained about them, and its edges carry the witnesses of the ledger. A node is in one of three states and is shown accordingly: a *frontier* node (the child of a descended node) by its title only; a *searched* node (returned by a search) by its summary; and a *read* node (whose content has been returned) by the compressed reading that Record stored for it. A node never leaves the read state. Each node also carries a score against the active sub-query that is recomputed at every step. Each edge carries its witnesses $\mathcal{W}(v, c)$. The map is kept connected by *ancestor closure*: after every batch, every ancestor of every node on the map is added as an *ancestor* node shown by its title only, so the orchestrator ascends by reading an ancestor.

Four tools change M_t. (1) Search$(\hat{q}, \sigma, \Gamma)$ issues a search query $\hat{q}$ written for the active sub-query. Its scope σ limits the hits to the whole corpus, to one document D, or to the subtree below a node v already on the map. Its channel set $\Gamma \subseteq \{\text{summary}, \text{leaf}, \text{loss}\}$ defaults to all three, and its hits are added to the map. Independently of the calls, the runtime refreshes the map at every step with the top-12 loss-search hits of each sub-query (§C). (2) Descend(v) adds the children of v as frontier nodes, shown by their titles only, together with the witnesses on the edges into them. (3) Read(v) is the only tool that returns original content: it returns the content of v once and marks v as read. (4) Record(v, x) stores on the node the compressed reading x that the map keeps showing afterwards.

Ancestor closure and the four tools together make a traversal direction unnecessary, with or without the ledger: a search can start an episode at an internal node or at a leaf, and the map can then be extended downward with DESCEND or upward by reading an ancestor (§D.6 measures the resulting trajectories).

The ledger supports the traversal through two mechanisms. In *loss search*, witnesses serve as retrieval targets: the loss channel matches them against the search query, and each hit adds the child node of its edge to M_t, even when no summary on its path scores well. In *map observations*, witnesses serve as annotations: those on an edge are displayed with their statements, locators, and scores, so the orchestrator can assess a branch before opening it, read the locator of a witness directly, or retain a low-scoring branch whose edge carries a witness (§F.5).

Evidence and answer. Evidence serves as the basis on which an answer may be given. Its data structure G_t is a list of *certified* facts, each bound to one obligation field and to the read node it was taken from, and an answer is *grounded* when all of its fields are certified. Two tools manage G_t. (1) CERTIFY$(o.f, x, v)$ adds a fact x for field f of obligation o and rejects any source v that the map does not mark as read; a second, different fact for the same field makes it contradicted. (2) ANSWER(y) terminates the episode and is rejected while any field is open or contradicted; the gate checks the declared fields and the read state of their sources, not whether a fact is entailed by its source. A contradicted field keeps both facts and is never resolved by overwriting, so an episode with one ends only by the budget. If the step budget is exhausted, the runtime reads up to 20 unread nodes of the map once (§C) and a final call produces a forced answer that names its open and contradicted fields.

5 EXPERIMENTS

5.1 EXPERIMENTAL SETUPS

Datasets. We evaluate on eight multimodal document QA benchmarks converted to the open-corpus setting, which span three corpus regimes. *Multi-document wiki and web* collections, MultimodalQA and WebQA, spread evidence over many short hyperlinked pages. *Long single-document PDFs*, DoubleBench, LongDocURL, and MMLongBench-Doc, concentrate evidence inside one lengthy report, and *table- and chart-centric* collections, VisDoMBench, MMDocRAG, and MMDocIR, place answers in visually rich components whose signal a text summary omits most readily.

Evaluation metrics. For end-to-end QA we report exact match (EM) and token-level F1, averaged unweighted over the eight corpora. For efficiency we report LLM calls, input and output tokens, and time per query, where time is the estimated LLM processing time of one query in isolation (§D.3). EM and F1 follow the DROP scorer (§B.4).

Compared methods. We compare against seven systems in three groups (§3): (i) *vision-native* retrievers that match a query against rendered pages, PIXELRAG, SIMPLEDOC, and VISDOMRAG; (ii) *agentic* retrievers that interleave reasoning with retrieval, VIDORAG, DOCLENS, and FIF; and (iii) HIKEY, the *hierarchical* retriever closest to ours in kind. SIMPLEDOC and DOCLENS assume a known document, so each receives the top-1 document of a shared router, whose accuracy of 79.4% bounds their EM (Table 5); VISDOMRAG assumes a closed candidate set, so it receives the eight candidates of a BM25 router (all three marked †). The router is one adaptation of such methods to an open corpus, and its errors are part of their end-to-end scores. The remaining experimental settings, including every model and the configuration of RAISE, are given in §B.

5.2 ACCURACY COMPARISON

We evaluate end-to-end QA on the eight corpora with EM and F1, with results reported in Table 1. RAISE achieves the best average accuracy, at 43.1 EM and 56.2 F1, and is the best method on six of the eight corpora. It improves over VIDORAG, the strongest competitor, by 5.8 EM and 6.0 F1 on average, and by more than twice as much over FIF and HIKEY, the closest agentic and hierarchical competitors. The margin also holds against the three systems that receive their documents from a router.

We first compare RAISE with VIDORAG, which leads on two corpora by small margins in EM (1.5 and 1.2 points). VIDORAG tiles up to twenty candidate pages into one image and inspects them in a single call, which is well suited to both corpora: 80% of the DoubleBench queries have two or more gold pages, which RAISE has to read sequentially, and a WebQA answer is often a visual attribute of one image. The median WebQA document also has a single internal node (Table 19), so there are

Corpus	PixelRAG		SimpleDoc†		DocLens†		ViDoRAG		VisDoMRAG†		FiF		HiKEY*		RAISE	
	EM	F1	EM	F1	EM	F1	EM	F1	EM	F1	EM	F1	EM	F1	EM	F1
MultimodalQA	28.1	34.5	30.0	35.8	25.6	31.8	42.3	50.1	31.9	37.7	36.5	43.4	32.3	38.2	**55.3**	**63.0**
WebQA	37.0	44.3	26.5	32.0	21.9	28.1	**50.5**	**59.9**	36.0	41.1	47.7	54.5	36.3	42.4	49.3	59.5
DoubleBench	18.4	32.7	22.4	36.1	22.7	38.3	**26.3**	43.9	25.8	**44.3**	15.1	25.8	21.9	37.3	24.8	39.1
LongDocURL	35.7	42.1	56.5	64.2	40.2	46.4	51.7	60.8	52.4	59.7	29.0	34.9	48.0	58.6	**63.2**	**73.0**
VisDoMBench	22.7	37.8	35.1	59.0	22.7	40.5	36.4	58.3	32.6	56.1	24.3	43.7	10.8	26.9	**37.5**	**62.0**
MMDocRAG	7.3	23.9	11.3	29.5	9.1	25.3	12.2	34.0	12.6	35.1	5.3	23.0	9.8	31.5	**12.9**	**35.6**
MMLongBench-Doc	27.2	31.7	37.5	40.6	27.2	32.6	32.1	37.2	32.1	39.0	16.9	21.3	25.5	31.4	**46.7**	**49.4**
MMDocIR	21.2	29.0	42.9	52.9	30.4	39.0	47.0	57.5	49.2	60.4	28.5	36.1	46.5	57.7	**55.4**	**67.6**
Average	24.7	34.5	32.8	43.8	25.0	35.3	37.3	50.2	34.1	46.7	25.4	35.3	28.9	40.5	**43.1**	**56.2**

Table 1: End-to-end QA accuracy (EM and F1, %) of RAISE and its competitors. †Runs on routed documents (the top-1 document for SimpleDoc and DocLens, eight candidates for VisDoMRAG). *Our reimplementation. The best score in each row is in **bold**.

few omissions to record. Elsewhere, the evidence is one component inside a long document or a large collection, and it must be contained in the twenty candidate pages, whereas RAISE extends its map until its obligations are closed.

We next examine on which kinds of corpora RAISE is expected to be most effective, as implied by its design. The ledger records what summaries omit, so it is expected to be most beneficial where many levels of summarization lie between a query and its evidence (§2). In addition, each obligation is closed by a value certified from a node that was read, so RAISE is expected to be most accurate when a query is answered by a specific component. The results of Table 1 are consistent with such an expectation: RAISE attains its largest margins over the best competitor on LongDocURL, MMLongBench-Doc, and MMDocIR, whose median documents have about a hundred internal nodes or more (Table 19), and on MultimodalQA, where a query combines components of two documents, each of which is handled by a separate sub-query. Conversely, the two corpora on which RAISE does not attain the best accuracy are those least suited to its design, having flat documents or many gold pages per query. MMDocRAG and VisDoMBench are the exceptions: their documents have dozens of internal nodes and RAISE is the best method on both, but by margins of at most 1.2 EM. The narrow margins are attributable to the form of the gold answers rather than to retrieval. The median gold answer has ten words on MMDocRAG, and VisDoMBench often expects lists, which lowers EM for every method. The LLM judge, in contrast, scores RAISE above 60 on both corpora (§D.1).

5.3 Efficiency Comparison

We measure per-query cost as LLM calls, input and output tokens, and time, reported in Table 2 and per corpus in §D.3. As expected, the single-call systems, PixelRAG and HiKEY, are the cheapest, at the expense of accuracy (Table 1). Among the others, RAISE takes 23.5 seconds per query, which is 19% less than ViDoRAG on the average over the corpora, and less on four of the eight corpora individually (§D.3), so its accuracy gain comes with a lower estimated LLM processing time.

The cost of each method follows from its design (§D.3): RAISE re-renders the sub-queries, the obligations, the evidence, and the map at every step but writes only a batch of tool calls, whereas each agent of ViDoRAG reasons about every candidate page in writing. Since prefill runs about sixty times faster per token than decode on this backbone, time follows output tokens: ViDoRAG reads a third of what RAISE reads, writes almost twice as much, and is slower. Per corpus, the cost of RAISE follows the number of steps a query needs, lowest on LongDocURL and highest on DoubleBench and MMDocRAG (§D.3).

5.4 Ablation Study

Table 3 isolates the two mechanisms through which the ledger is used online (§4.2) by removing them from the full system, one at a time and then together, and then removes three parts of the orchestrator of §4.2 one at a time, with the per-corpus results in §D.4. The full system is the most accurate and the fastest configuration, every removal costs between 1.7 and 4.0 EM, and the whole ledger is the largest single loss.

Method	# LLM calls	# input tokens	# output tokens	Time (s)
PIXELRAG	1.0	3,665	198	2.1
SIMPLEDOC	2.9	25,113	1,221	13.3
DOCLENS	11.7	105,398	4,440	50.3
VIDORAG	5.5	25,844	3,154	29.0
VISDOMRAG	3.0	22,213	1,947	18.8
FIF†	22.8	49,228	1,828	21.5
HIKEY*	1.0	14,126	21	2.1
RAISE	11.0	72,186	1,693	**23.5**

Table 2: Per-query cost of RAISE and its competitors, averaged over the eight corpora. Time is the estimated isolated LLM processing time, not end-to-end wall-clock latency, and excludes retrieval overhead. †Retrieval stage only. *Our reimplementation.

	Ledger			Accuracy (%)			Efficiency / q						
Configuration	search	map	Agent	EM	F1	ΔEM	steps	calls	reads	input tok	output tok	time (s)	Δtime
RAISE (full)	✓	✓	-	**43.1**	**56.2**	-	6.57	**11.00**	7.69	**72,186**	**1,693**	**23.50**	-
w/o loss search		✓	-	41.4	54.1	−1.7	7.07	12.29	8.36	84,215	1,834	26.32	+12.0%
w/o map observations	✓		-	41.3	54.2	−1.8	6.71	11.83	8.00	75,759	1,803	24.91	+6.0%
w/o ledger			-	39.1	50.9	−4.0	7.31	12.68	8.69	77,593	1,872	25.72	+9.5%
w/o ancestor closure	✓	✓	closure off	40.9	53.6	−2.2	7.42	12.37	8.19	75,797	1,891	25.63	+9.1%
map replaced by a log	✓	✓	map → log	40.3	53.4	−2.8	7.52	13.53	9.63	78,845	2,221	28.71	+22.2%
w/o obligation gate	✓	✓	gate off	41.3	55.0	−1.8	**6.35**	11.30	**7.56**	72,945	1,741	24.03	+2.3%

Table 3: Ablation of the ledger (top), through the two mechanisms by which it is used online, and of the orchestrator (bottom). Δ is against the full system, and the best score in each column is in **bold**.

(i) *Loss search.* Without it, a branch enters the map only when a summary or a leaf matches the query, so the orchestrator descends through the summaries to find the rest, and the displayed witnesses lengthen every prompt: the slowest of the ledger configurations. (ii) *Map observations.* Without them, the same branches enter the map, but the orchestrator chooses among siblings from their summaries and scores alone and reads more of them. (iii) *The whole ledger.* Without it, the two effects accumulate and the loss exceeds the two single losses together: the two mechanisms are not redundant, since loss search decides which branches are on the map and the displayed witnesses decide which of them are read (§D.5). (iv) *The orchestrator.* Without ancestor closure, the context that the ancestors supplied has to be read; a log in place of the map repeats every observation and is the most expensive configuration; without the obligation gate, episodes end earliest. Each part is worth about as much as one mechanism of the ledger, and less than the whole ledger.

5.5 ADDITIONAL EXPERIMENTS

The appendix adds the LLM-judge accuracy of every system (§D.1), retrieval accuracy at each method's own reader budget (§D.2), the per-corpus breakdowns of the efficiency comparison and of the ablation together with the substitution of the witnesses by summaries (§D.3, §D.4, §D.5), the sensitivity of RAISE to the online backbone (§D.7), the offline cost of indexing one document (§D.9), and the behavior of the orchestrator (§D.6).

6 CONCLUSION

We characterized the *abstraction bottleneck* of hierarchical retrieval. Once a summary omits the content that discriminates a query, an agent that routes on the summaries alone and expands k of m siblings reaches the right branch with probability at most k/m under a uniform target, however well it reasons. We then presented RETRIEVAL OVER ABSTRACTION-INDUCED SUMMARY ERASURE (RAISE), whose LOSS LEDGER records what each summary omitted as source-linked loss witnesses on the edges of the tree, and whose agent uses the witnesses both as retrieval targets and as annotations on the map of its traversal, so a branch can be reached, and assessed, through what its summary omitted. The traversal is direction-free: a search can start at any level of a tree, the map keeps the ancestors of every hit, and descent, ascent, and loss-guided jumps arise as trajectories rather than as modes fixed in advance. On eight multimodal document QA benchmarks in the open-corpus setting, RAISE achieves the best average accuracy with a locally served open-weight backbone, 5.8 points of exact match ahead of the strongest of seven competitors.

AI Use Statement

In this work, we used generative AI tools, mainly an LLM-based coding assistant, to implement code (RAISE, the reimplemented competitors, and of the evaluation and analysis scripts) and to draft and revise the text of this paper. To a lesser extent, we used them to summarize analysis outputs and to search related literature. Generative models also took part in building the evaluation set: two LLM judges filter the six converted corpora down to the queries that identify their document in the open-corpus setting (§B.4), and an LLM rewrites the gold answers into a concise form (§F.7). An LLM judge further scores the answers in §D.1; the filtering and rewriting models are not used at retrieval or answer time, whereas the judge shares the online backbone (§B.5). We have reviewed all AI-assisted work: the authors reviewed and tested the code, checked the description of the method against the implementation, checked every reported number against our result records, and verified every reference against its official source. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

Ethics Statement

This work involves no human subjects, no crowdsourced annotation, and no newly collected personal data. All experiments use publicly available benchmarks under their original licenses, and our open-corpus conversion only filters and regroups their existing queries and documents (§B.2, §B.4). RAISE is a retrieval method, so its main risks are those of retrieval-augmented generation in general: it can surface whatever a corpus contains, and its index stores model-written summaries and loss witnesses that may be wrong. We limit the second risk by design, since generated text may direct the search but only read source content may be certified as evidence (§4.2). The offline index costs LLM computation per document, and we report that cost so that it can be weighed against the per-query savings (§D.9).

Reproducibility Statement

We provide the source code of RAISE, of the index-building pipeline, and of our reimplemented competitors as supplementary material. The code is publicly released at `https://anonymous.4open.science/r/iclr2027_raise-C550`. The method is specified in §4, and §C gives the data structures, the per-edge witness selection rule, and the per-step observation. §F reproduces the prompts of every LLM stage, including the summarizer, the witness extractor, and the orchestrator. The benchmarks, their conversion to the open-corpus setting, and the reconstruction of HTML corpora are described in §B.2, §B.4, and §B.3. Models, serving precision, and hardware are listed in §B.5 and §B.1, and backbones and the embedder are standardized across methods wherever they apply (§5). §B.6 documents the competitor we had to reimplement, and §D.1, §D.2, and §D.9 define the additional metrics and the cost accounting. The online agent decodes at temperature zero.

References

Yingshan Chang, Mridu Narang, Hisami Suzuki, Guihong Cao, Jianfeng Gao, and Yonatan Bisk. WebQA: Multihop and multimodal QA. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 16495–16504, 2022.

Howard Chen, Ramakanth Pasunuru, Jason Weston, and Asli Celikyilmaz. Walking down the memory maze: Beyond context limit through interactive reading. *arXiv preprint arXiv:2310.05029*, 2023.

Tong Chen, Hongwei Wang, Sihao Chen, Wenhao Yu, Kaixin Ma, Xinran Zhao, Hongming Zhang, and Dong Yu. Dense X retrieval: What retrieval granularity should we use? In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 15159–15177, 2024.

Jaemin Cho, Debanjan Mahata, Ozan Irsoy, Yujie He, and Mohit Bansal. M3DocRAG: Multi-modal retrieval is what you need for multi-page multi-document understanding. *arXiv preprint arXiv:2411.04952*, 2024.

Chao Deng, Jiale Yuan, Pi Bu, Peijie Wang, Zhong-Zhi Li, Jian Xu, Xiao-Hui Li, Yuan Gao, Jun Song, Bo Zheng, and Cheng-Lin Liu. LongDocURL: a comprehensive multimodal long document benchmark integrating understanding, reasoning, and locating. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 1135–1159, 2025.

Daniel Deutsch, Tania Bedrax-Weiss, and Dan Roth. Towards question-answering as an automatic metric for evaluating the content quality of a summary. *Transactions of the Association for Computational Linguistics*, 9:774–789, 2021.

Kuicai Dong, Yujing Chang, Derrick Goh Xin Deik, Dexun Li, Ruiming Tang, and Yong Liu. MMDocIR: Benchmarking multimodal retrieval for long documents. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 30971–31005, 2025a.

Kuicai Dong, Yujing Chang, Shijie Huang, Yasheng Wang, Ruiming Tang, and Yong Liu. Benchmarking retrieval-augmented multimodal generation for document question answering. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2025b.

Esin Durmus, He He, and Mona Diab. FEQA: A question answering evaluation framework for faithfulness assessment in abstractive summarization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 5055–5070, 2020.

Manuel Faysse, Hugues Sibille, Tony Wu, Bilel Omrani, Gautier Viaud, Céline Hudelot, and Pierre Colombo. ColPali: Efficient document retrieval with vision language models. In *International Conference on Learning Representations (ICLR)*, 2025.

Shubham Gupta, Zichao Li, Tianyi Chen, Cem Subakan, Siva Reddy, Perouz Taslakian, and Valentina Zantedeschi. ReTreever: Tree-based coarse-to-fine representations for retrieval. *arXiv preprint arXiv:2502.07971*, 2025.

Chelsi Jain, Yiran Wu, Yifan Zeng, Jiale Liu, Shengyu Dai, Zhenwen Shao, Qingyun Wu, and Huazheng Wang. SimpleDoc: Multi-modal document understanding with dual-cue page retrieval and iterative refinement. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 28410–28427, 2025.

Soyeong Jeong, Jinheon Baek, Sukmin Cho, Sung Ju Hwang, and Jong Park. Adaptive-RAG: Learning to adapt retrieval-augmented large language models through question complexity. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 7036–7050, 2024.

Patrick S. H. Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems 33 (NeurIPS 2020)*, 2020.

Yubo Ma, Yuhang Zang, Liangyu Chen, Meiqi Chen, Yizhu Jiao, Xinze Li, Xinyuan Lu, Ziyu Liu, Yan Ma, Xiaoyi Dong, Pan Zhang, Liangming Pan, Yu-Gang Jiang, Jiaqi Wang, Yixin Cao, and Aixin Sun. MMLongBench-Doc: Benchmarking long-context document understanding with visualizations. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2024.

Lang Mei, Siyu Mo, Zhihan Yang, and Chong Chen. A survey of multimodal retrieval-augmented generation. *arXiv preprint arXiv:2504.08748*, 2025.

Tetsuya Sakai and Karen Sparck Jones. Generic summaries for indexing in information retrieval. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, pp. 190–198, 2001.

Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D. Manning. RAPTOR: Recursive abstractive processing for tree-organized retrieval. In *International Conference on Learning Representations (ICLR)*, 2024.

Thomas Scialom, Paul-Alexis Dray, Sylvain Lamprier, Benjamin Piwowarski, Jacopo Staiano, Alex Wang, and Patrick Gallinari. QuestEval: Summarization asks for fact-based evaluation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 6594–6604, 2021.

Wenxuan Shen, Mingjia Wang, Yaochen Wang, Dongping Chen, Junjie Yang, Yao Wan, and Weiwei Lin. Are we on the right way for assessing document retrieval-augmented generation? *arXiv preprint arXiv:2508.03644*, 2025.

Joongmin Shin, Gyuho Shim, Jeongbae Park, Jaehyung Seo, and Heuiseok Lim. HiKEY: Hierarchical multimodal retrieval for open-domain document question answering. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 17967–17987, 2026.

Li Sun, Liu He, Shuyue Jia, Yangfan He, and Chenyu You. DocAgent: An agentic framework for multi-modal long-context document understanding. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 17701–17716, 2025.

Manan Suri, Puneet Mathur, Franck Dernoncourt, Kanika Goswami, Ryan A. Rossi, and Dinesh Manocha. VisDoM: Multi-document QA with visually rich elements using multimodal retrieval-augmented generation. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics (NAACL)*, pp. 6088–6109, 2025.

Alon Talmor, Ori Yoran, Amnon Catav, Dan Lahav, Yizhong Wang, Akari Asai, Gabriel Ilharco, Hannaneh Hajishirzi, and Jonathan Berant. MultiModalQA: Complex question answering over text, tables and images. In *International Conference on Learning Representations (ICLR)*, 2021.

Wenyu Tao, Xiaofen Xing, Yirong Chen, Linyi Huang, and Xiangmin Xu. TreeRAG: Unleashing the power of hierarchical storage for enhanced knowledge retrieval in long documents. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 356–371, 2025.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 10014–10037, 2023.

Vectify AI. PageIndex: Document index for vectorless, reasoning-based RAG. https://github.com/VectifyAI/PageIndex, 2025.

Bin Wang, Chao Xu, Xiaomeng Zhao, Linke Ouyang, Fan Wu, Zhiyuan Zhao, Rui Xu, Kaiwen Liu, Yuan Qu, Fukai Shang, Bo Zhang, Liqun Wei, Zhihao Sui, Wei Li, Botian Shi, Yu Qiao, Dahua Lin, and Conghui He. MinerU: An open-source solution for precise document content extraction. *arXiv preprint arXiv:2409.18839*, 2024.

Qiuchen Wang, Ruixue Ding, Zehui Chen, Weiqi Wu, Shihang Wang, Pengjun Xie, and Feng Zhao. ViDoRAG: Visual document retrieval-augmented generation via dynamic iterative reasoning agents. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 9113–9134, 2025.

Yichuan Wang, Zhifei Li, Zirui Wang, Paul Teiletche, Lesheng Jin, Matei Zaharia, Joseph E. Gonzalez, and Sewon Min. PixelRAG: Web screenshots beat text for retrieval-augmented generation. *arXiv preprint arXiv:2606.28344*, 2026.

Hui Wu, Haoquan Zhai, Yuchen Li, Hengyi Cai, Peirong Zhang, Yidan Zhang, Lei Wang, Chunle Wang, Yingyan Hou, Shuaiqiang Wang, and Dawei Yin. MARA: A multimodal adaptive retrieval-augmented framework for document question answering. In *Proceedings of the 33rd ACM International Conference on Multimedia (ACM MM)*, pp. 4329–4338, 2025a.

Junda Wu, Yu Xia, Tong Yu, Xiang Chen, Sai Sree Harsha, Akash V Maharaj, Ruiyi Zhang, Victor Bursztyn, Sungchul Kim, Ryan A. Rossi, Julian McAuley, Yunyao Li, and Ritwik Sinha. DocReact: Multi-page heterogeneous document question-answering. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 67–78, 2025b.

Xixi Wu, Yanchao Tan, Nan Hou, Ruiyang Zhang, and Hong Cheng. MoLoRAG: Bootstrapping document understanding via multi-modal logic-aware retrieval. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 14024–14045, 2025c.

Lingnan Xu, Chong Feng, Kaiyuan Zhang, Zhengyong Liu, Wenqiang Xu, and Fanqing Meng. Equipping retrieval-augmented large language models with document structure awareness. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 24608–24631, 2025.

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

Shi Yu, Chaoyue Tang, Bokai Xu, Junbo Cui, Junhao Ran, Yukun Yan, Zhenghao Liu, Shuo Wang, Xu Han, Zhiyuan Liu, and Maosong Sun. VisRAG: Vision-based retrieval-augmented generation on multi-modality documents. In *International Conference on Learning Representations (ICLR)*, 2025.

Joohyung Yun, Doyup Lee, and Wook-Shin Han. LILaC: Late interacting in layered component graph for open-domain multimodal multihop retrieval. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 20540–20559, 2025.

Joohyung Yun, Doyup Lee, and Wook-Shin Han. Failure is feedback: History-aware backtracking for agentic traversal in multimodal graphs. *arXiv preprint arXiv:2602.03432*, 2026.

Zijie Zhong, Hanwen Liu, Xiaoya Cui, Xiaofan Zhang, and Zengchang Qin. Mix-of-granularity: Optimize the chunking granularity for retrieval-augmented generation. In *Proceedings of the 31st International Conference on Computational Linguistics (COLING)*, pp. 5756–5774, 2025.

Dawei Zhu, Rui Meng, Jiefeng Chen, Sujian Li, Tomas Pfister, and Jinsung Yoon. DocLens: A tool-augmented multi-agent framework for long visual document understanding. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 26804–26829, 2026.

APPENDIX

A ABSTRACTION-INDUCED UNREACHABILITY

This appendix states the counting argument behind §2. It makes the motivation precise rather than standing alone as theory.

Proposition 1 (Abstraction-induced unreachability)**.** *Fix a query $\mathcal{Q}$ and an internal node v, and suppose that the only content distinguishing a target child $c^\star \in \mathrm{children}(v)$ from $m-1$ of its siblings lies in $\bigcup_{c\in\mathrm{children}(v)} \Delta(v,c)$. Then every signal a pure top-down agent π observes before expansion is identical across those m children, so under a uniform target prior and any symmetric selection rule, an agent that expands at most $k < m$ of them reaches $c^\star$ with probability at most k/m.*

Proof sketch. The discriminative content lies in the omissions, so it appears in no observation available before expansion and the m children are decision-equivalent to π; a symmetric rule therefore includes $c^\star$ with probability at most k/m, and guaranteeing inclusion for every target requires expanding all m. □

The bound motivates an additional routing signal and is not a recall guarantee for any method: it holds for the selection made before further observations, and further observations, such as a leaf search or an expansion, change what the agent can distinguish.

B EXPERIMENTAL DETAILS

B.1 HARDWARE AND SOFTWARE SETTINGS

All experiments were conducted on a Linux server with an Intel Xeon Gold 6230 CPU @ 2.10 GHz (80 threads), 1.5 TB of RAM, and four NVIDIA RTX A6000 GPUs, running Ubuntu 22.04 LTS. The embedder and every locally served open-weight model run on this machine; the proprietary models used only to build the corpora (§B.5) are queried through their vendor APIs.

B.2 BENCHMARK DETAILS

Multi-document wiki and web. Following LILAC (Yun et al., 2025), we use the URL-annotated setting of MultimodalQA (Talmor et al., 2021) and WebQA (Chang et al., 2022) to reconstruct realistic page-style corpora and parse each page into multimodal components. MultimodalQA is used in the extended form introduced by M3DocRAG (Cho et al., 2024), and stresses multi-hop reasoning across text, tables, and images. WebQA is Wikipedia-based and its reference answers are verbose, so we follow prior work in scoring against concise rewritten references.

Long single-document PDFs. MMLongBench-Doc (Ma et al., 2024) contains 1,082 expert-annotated queries over 135 PDFs averaging 47.5 pages, of which 223 are deliberately unanswerable, which makes it the sharpest test of whether a retriever will invent evidence. LongDocURL (Deng et al., 2025) contains 2,325 query-answer pairs over 396 PDFs and more than 33,000 pages, averaging 85.6 pages per document, and 37.1% of its items require locating evidence across elements. DoubleBench (Shen et al., 2025) spans 3,276 documents and 72,880 pages in six languages and four document types, PDFs, scanned documents, slides, and HTML pages, drawn from DocVQA, MMLongBench-Doc, arXiv, SlideVQA, CommonCrawl, and Wikipedia, with 5,168 single- and multi-hop queries grounded in exhaustively scanned evidence pages, which is what makes gold-evidence recall meaningful rather than approximate. We keep its English subset, 1,950 queries (1,000 single-hop and 950 multi-hop) over 650 documents of all four types, so that it matches the other seven corpora of our panel, which are all English.

Table- and chart-centric. VisDoMBench (Suri et al., 2025) contributes 2,271 samples over five subsets that differ in document type and evidence modality, spanning table-centric QA over web pages and scientific papers, figure- and table-based scientific QA, academic-graph VQA, and multi-page slide decks. We keep the FetaTab, PaperTab, and SlideVQA subsets, 1,278 queries, and drop SciGraphQA and SPIQA, whose queries presuppose a given figure or paper and cannot be posed to a corpus. MMDocIR (Dong et al., 2025a) provides 1,685 expert-annotated queries over 313 documents averaging 65.1 pages across ten domains, with a modality mix of 60.4% text, 18.8% image, and

16.7% table, and defines both page-level and layout-level retrieval. MMDocRAG (Dong et al., 2025b) provides 4,055 expert-annotated pairs over the same document pool, of which 52% are cross-page, 39.2% multi-image, and 61.7% cross-modal, which is the highest concentration of the kinds of omission the LOSS LEDGER targets; we use its 2,055-query dev split.

From benchmark to corpus. The counts above are the sizes each benchmark ships with, not the sizes we evaluate on. Every benchmark is written for a setting where the document is given, so converting one into an open corpus removes the queries that only make sense with a document in hand, and §B.4 reports that conversion and the resulting query counts. Two of the corpora arrive as HTML rather than as documents, and §B.3 describes how they are rendered into pages.

B.3 CORPUS RECONSTRUCTION FROM HTML

Why the pages are rendered. MULTIMODALQA and WEBQA are distributed as annotated URLs whose content is HTML, which carries a DOM but no pagination, whereas the other six corpora are paginated documents. The vision-native competitors of §5 match a query against rendered pages, and a table or a figure is interpretable only in its rendered form, so we render every page into a paginated document and all eight corpora take one form. RAISE itself parses the section hierarchy and the components of such a corpus from the markup (§4.1), and the rendered pages serve the page-based methods.

Document identity. Each page is stored under a file name derived from its URL slug, truncated to 200 characters. A truncation collision appends the first ten hexadecimal digits of the SHA-1 of the full URL to whichever page arrived later. The naming rule accounts for 3,236 of 3,236 document identifiers in MULTIMODALQA and 7,698 of 7,705 in WEBQA, and every remaining identifier is a collision sibling.

Rendering. The stylesheets a page references are fetched once and inlined, after which rendering is fully offline and therefore reproducible. Each page is then printed by a headless Chromium at Letter size, 612 by 792 points, with the text layer preserved, so parsing reads real text rather than OCR output. Page images are rasterized at 1,224 by 1,584 pixels, exactly twice the print geometry, so a component's box in the parse and its region in the page image name the same location.

B.4 OPEN-CORPUS CONVERSION

Why filtering is required. Many queries in the source benchmarks assume that the document is already in hand, so they cannot be asked of a corpus. A query such as "how many times was the author cited in this document" gives retrieval no anchor to start from, and different documents in the corpus would answer it differently. We therefore keep only the queries specific enough to pin down one document, following the query categorization of VISRAG (Yu et al., 2025). The need is not limited to benchmarks that assume a known document: DoubleBench evaluates retrieval over its whole corpus, but each of its queries is synthesized from the pages of one document, so nothing forces the query to identify that document among the others, and only 604 of its 1,950 English queries pass the criterion below.

The criterion. The decision turns on where the query starts. A query of category #A starts from an external anchor, such as a named entity, a proper noun, a unique title, a product code, or a case name, that identifies one document on its own, so we keep it. A query of category #B starts from a referring expression that presupposes a given document ("the document", "this paper", "the second table"), so we drop it. Two boundary cases decide most of the hard calls. Names that look specific but label something internal to a paper, such as "the proposed vector-res method" or "the optimization problem (6)", are referring expressions rather than anchors, so they fall in #B. Multi-hop queries stay in #A whenever their unresolved references can be pinned down by following named entities through successive retrievals, so the filter does not strip out the multi-hop items that make the task hard.

Procedure. Two LLM judges, `gpt-5.4` and `Claude Sonnet`, independently classify each query of the six filtered corpora against the prompt of §F.6, which states the criterion above and supplies 43 labelled in-context examples drawn from the source benchmarks. The judges see the query alone, so the filter is an eligibility filter on the anchor of a query rather than a check that the anchor matches a single document of the corpus. A query survives only if both judges place it in category #A, so the kept set is the intersection of the two judgments, and the aggressive reduction of some corpora, such

	Corpus		Open-corpus conversion		
Name	**Docs**	**Pages**	**Source**	**Kept**	**Ret. GT**
MultimodalQA	3,254	38,885	not filtered	2,441	✓
WebQA	7,743	66,590	not filtered	4,966	✓
DoubleBench	650	16,683	1,950	604	✓
LongDocURL	396	33,766	2,325	269	✓
VisDoMBench	841	12,223	1,278	445	
MMDocRAG	147	10,256	2,055	451	✓
MMLongBench-Doc	135	4,212	1,091	184	✓
MMDocIR	313	10,412	1,658	368	✓
Total	13,479	∼193k		9,728	

Table 4: The eight corpora and their open-corpus conversion. *Source* is the query count before filtering and *Kept* is the number that survived it, which is the number actually evaluated.

Method	**Venue**	**SOTA on**	**Native setting**
PIXELRAG (Wang et al., 2026)	preprint	-	open-corpus
SIMPLEDOC (Jain et al., 2025)	EMNLP'25	LongDocURL	known-document
DOCLENS (Zhu et al., 2026)	preprint	MMLongBench-Doc	known-document
VIDORAG (Wang et al., 2025)	EMNLP'25	DoubleBench	open-corpus
VISDOMRAG (Suri et al., 2025)	NAACL'25	VisDoMBench	closed-candidate
FIF (Yun et al., 2026)	ICML'26	MultimodalQA, WebQA	open-corpus
HIKEY (Shin et al., 2026)*	ACL'26	MMDocIR	open-corpus
RAISE (this work)	-	-	open-corpus

Table 5: Competitors and the setting each was designed for. *Reimplemented by us (§B.6); no code is released.

as LongDocURL, reflects how many of their queries presuppose a given document rather than identify one. Table 4 reports the counts alongside the corpus sizes: *Source* is the query count we started from and *Kept* is the number of queries actually evaluated. *Source* counts the queries of the release we loaded, which is the full release for LongDocURL, the dev split for MMDocRAG, and the subsets named in §B.2 for DoubleBench and VisDoMBench; for MMLongBench-Doc and MMDocIR the released files hold 1,091 and 1,658 queries, whereas their papers report 1,082 and 1,685. *Ret. GT* marks the corpora whose gold evidence is labelled at the page or component level, which the retrieval accuracy of §D.2 requires; VisDoMBench carries such labels only for its SlideVQA subset, so it is excluded there. MULTIMODALQA and WEBQA are not filtered: their queries were written by annotators against a corpus with distractor sources, so they identify their document by construction, whereas the queries of the other six benchmarks were authored from a given document, by annotators or, in DoubleBench, by a model.

Concise reference answers. Verbose gold answers put exact match structurally out of reach, so we rewrite single-span gold into the shortest span that answers the query. A reader that correctly outputs "Yes" or "Gonda Building" cannot match a gold answer written as a full sentence, which drives exact match toward zero for reasons that have nothing to do with retrieval. Rewriting touches single-span answers only, because a gold list already consists of short spans and the metric scores it as a set. The rewriting prompt (§F.7) preserves any output format the query itself fixes, such as a list, a rounding precision, a unit, or a date format.

Scoring. EM and F1 are the DROP scorer: prediction and gold are normalized (case, punctuation, articles, number words), a list answer is scored as a set of spans, with EM requiring the whole set to match and F1 aligning spans optimally and averaging token F1 over them, a query whose gold is a list of spans is scored against that list as a whole, and the prediction is one string. Every number is one run over the Kept queries of Table 4, with no repetition and no partial credit beyond the token F1.

B.5 Model Details

Backbones and the embedder are standardized across methods wherever they apply, so that a difference between two rows of §5.2 is attributable to retrieval rather than to parsing or generation.

Embedder. `nemotron-vl-1b`, a multimodal embedder with a 2,048-dimensional output, is used for all three vector stores of §4.1 and for every baseline that requires a vector index, except HiKEY, which keeps the ColPali encoder of its own design (§B.6), so no method gains from a stronger representation.

Offline models. Section re-nesting, image verbalization, node summarization, and witness extraction all use `Qwen3.6-35B-A3B`, served locally on the hardware of §B.1 across four replicas, so the index is built by the same open-weight model that answers the queries. Building the index with an open-weight model is deliberate rather than incidental: witness extraction runs over every internal node in the corpus, and demonstrating that it does not require frontier-model access is what keeps the offline cost defensible. Chain-of-thought decoding is disabled for the witness extractor, since it raised per-node latency by more than an order of magnitude without improving the extracted witnesses.

Online models. The orchestrator and the reader use `Qwen3.6-35B-A3B` served locally, with temperature-zero decoding, for all main results. Choosing a local open-weight backbone for the headline numbers is deliberate, because a method that only works behind a frontier API is a weaker claim than one that survives on hardware a lab already owns. The model is a mixture-of-experts checkpoint quantized to FP8 and occupies 34.9 GiB, so it fits on one of the GPUs of §B.1. RAISE keeps at most $N = 8$ witnesses per edge, retrieves the top eight hits per channel in each search, and runs for at most 14 steps per query.

Corpus construction models. The two retrievability judges of §B.4 are queried through their vendor APIs. The gold rewriting of §F.7 uses `GPT-5-mini`, since the task is extraction from a supplied answer rather than reasoning over a corpus. None of these models is used at retrieval or answer time, so no evaluated system shares a backbone with the process that built its evaluation set.

B.6 HiKEY Reimplementation Details

HiKEY (Shin et al., 2026) releases no code, so the row in Table 1 is our reimplementation of the method on this stack rather than a reproduction of the paper's numbers. We state what was substituted, because one component of the original has no counterpart here, and what was kept. Its DHP document hierarchy parser is replaced by the MinerU document tree we already build for every corpus, which supplies the same object the method needs, a document split into titled sections with an ancestry path. Its ColPali visual encoder is kept, with page images embedded by `colpali-v1.2` and scored by MaxSim, so the row reflects the method as published rather than a re-encoded variant. Everything else follows the paper: a Doc_card holding the title and the section paths, a Sec_card per section, Stage-1 routing by $\alpha \cdot \mathrm{BM25} + (1 - \alpha) \cdot \mathrm{dense}$ over documents, Stage-2 MaxSim over the sections of the routed documents, and Stage-3 packing of anchors with their ancestor headers and siblings under a token budget.

Three settings the paper does not fix were chosen on the development split and then frozen. α is 0.3, from a sweep over $\{0.0, 0.3, 0.5, 0.7, 1.0\}$ that reaches document recall 92 at 0.3 against 88 at pure dense and 76 at pure lexical. K_{sec} is 40, from $\{5, 10, 20, 40\}$, where component recall climbs 60, 66, 70, 76 while the packed context grows from 2.7K to 16.4K tokens. B_{tok} is 16,384, the paper's own budget, which is where $K_{\mathrm{sec}} = 40$ saturates. Such a budget is roughly twice what the page-level rows send their readers, and it is the reason HiKEY carries the highest component recall in Table 8 while finishing 14.4 points of exact match behind RAISE.

One corpus deserves a caveat. WebQA averages 1.0 sections per document, so its hierarchy is flat and $K_{\mathrm{sec}} = 40$ retrieves 40 whole documents rather than 40 targeted sections. Its component recall of 93.2 in Table 9 is therefore the highest cell in that table and the least informative one, and we read HiKEY's recall on the six corpora with real section structure instead.

C RAISE Implementation Details

This appendix gives the implementation of RAISE at the level of detail that §4 omits. §C.1 details step (1) of §4.1, the construction of a document tree from parser output, for the PDF corpora. §C.2 details how the objects that §4.1 and §4.2 define are represented: the nodes of $\mathcal{T}_D$, the witnesses of $\mathcal{L}$ and the cap N of step (3), the sub-queries Q_t, and the state s_t as the orchestrator observes it at each step.

C.1 Tree Construction from PDF

Step (1) of §4.1 builds one tree per document. For the six PDF corpora, it starts from MinerU (Wang et al., 2024), which parses a PDF into typed components with their positions on the page. Two further steps turn such output into the tree of §4.1. First, the flat heading list is re-nested into a section hierarchy. Second, a page level is inserted between a section and the components it directly contains, so page nodes appear beneath the leaf sections and beneath no section whose children are only subsections. The HTML corpora take their sections from the heading tags and are treated in §B.3.

Section re-nesting. RAISE recovers the section hierarchy of a PDF from the flat list of its section headings, with one LLM call per document. Such a step is necessary because MinerU returns the headings in reading order but without their levels, so its output does not distinguish a subsection from the section that contains it, and the section layer has to be recovered before any node can be summarized. The call receives the heading list, each heading followed by the number of content blocks directly beneath it in brackets, and returns the parent of every heading, under the prompt of §F.1. The count is what separates a container from a leaf section: a heading with few blocks that is immediately followed by another heading is usually a parent, whereas two consecutive headings with many blocks are usually siblings, and the call may introduce a missing parent heading or remove a spurious one such as a running header. The prompt names Wikipedia grouping patterns as category examples because it was written when it was also used for the HTML corpora; the decision itself depends on reading order and on the block counts, and the prompt is applied unchanged to every PDF corpus. The re-nested sections then become the internal nodes of the tree, with the document as the root and the components of each section beneath it.

Page nodes. The section hierarchy of a PDF and its physical pages are two different partitions of the same content, and the tree has to represent both. Sections are needed because they carry the summaries on which the traversal is decided, and pages are needed because a component's locator, its rendered image, and the gold labels of every PDF benchmark are all expressed in pages. We therefore place page nodes only where a section directly contains components, one per (section, page) pair: the components of such a section are grouped by the page they appear on, and each group becomes one page node whose summary is written with the rendered page attached (§F.2). When a physical page spans two sections, it yields two page nodes, one under each section. Each page node holds only the components of its own section, so no component is duplicated and every page node has a single parent. Some components span pages, such as a table continued on the next page. Such a component is attached to the page on which it begins. The resulting tree of a PDF is document → section (re-nested, §4.1) → page → component, where the page level lies directly above the components, so a section whose children are only subsections has no page nodes, and the page level is contained within the section boundaries rather than crossing them.

C.2 Index and State Representation

Tree node. A tree $\mathcal{T}_D$ of §2 is stored as nodes of three kinds, and every node carries a content id, a parent link, an ordered child list, its kind, and a locator into the original artifact.

- `document` is the root of the tree and carries the title of the document.
- `summary` is an internal node, and it additionally records its origin, either a `section` recovered from document structure or a `page` of the rendered artifact, and a page index where one applies.
- `component` is a leaf, and it records its modality and its verbatim content, with any image verbalization appended to it, so that the text a retriever matches against is the same text from which the summary was written.

Loss witness. The ledger $\mathcal{L}$ of step (3) in §4.1 is stored as one record per internal node v, which holds the witness sets $\mathcal{W}(v, c)$ of the edges below v together with an optional coarse `coverage` value.

- *Fields.* Each stored witness is ⟨`lost`, `where`, `kind`, `confidence`⟩, where `lost` is the self-contained statement that a query is matched against, `where` names the child and the locator, `kind` is one of `entity`, `numeric`, `relation`, `qualifier`, `table_binding`, `visual`, or `temporal`, and `confidence` is `low`, `medium`, or `high`. The first three fields are the $\langle \mathrm{stmt}, \mathrm{loc}, \mathrm{kind} \rangle$ of §4.1, and `confidence` is left out of the main text because it is used only when the per-edge cap is applied (below).
- *Use of* `kind` *and* `confidence`. No witness becomes unreachable because the extractor labelled it poorly, since neither field is used as a filter at query time. The two fields only order the witnesses of an edge when the cap is applied, by `confidence` first and by `kind` second, after which the highest-ranked N are kept. `kind` is also included in the text that is embedded for each witness, so a query that asks for a number is slightly more similar to a `numeric` witness.
- *Witnesses across children.* Some witnesses record a relation across several children, such as a comparison between two of them. Such a witness has no child in `where` and is attached to the node itself.
- *Extraction input.* The extraction call for a node v receives $l(v)$ and one row per child: the content of a leaf child, or the summary of an internal child but not its subtree, so a section call sees its page summaries and a page call sees the components of one page.
- *Admissibility.* A witness can be wrong, and it is never admissible as evidence: a value can be certified only from a node that has been read (§4.2). The gate checks the read state of the cited source and the completion of the declared fields; it does not verify that the value is entailed by the source, so the error of a witness can still propagate if the orchestrator does not check the content it read (§E).

Table 6 describes the seven kinds with illustrative witnesses.

Kind	What the summary omits	Illustrative witness
`entity`	a named member of a group the summary only generalizes	"Anna Berg heads the Europe segment"
`numeric`	a specific value, with its unit	"Europe revenue fell 4% in 2023 to €1,194M"
`table_binding`	a row-column-value binding inside a table	"Europe Q3 2023 revenue was €285M"
`relation`	a comparison or a link across children	"Europe is the only segment whose revenue declined"
`qualifier`	a condition, scope, or basis that restricts a statement	"Segment revenue excludes intra-group sales"
`temporal`	a dated event, or an order of events	"The Europe reorganization completed in Q2 2023"
`visual`	what only a figure shows, such as an extremum or a legend mapping	"The bar chart shows Asia-Pacific growth peaking in Q4"

Table 6: The seven kinds of omission a witness records. The witnesses are illustrative and continue the annual-report example of Figures 1 and 2; the `numeric` and `table_binding` rows are the two witnesses of Figure 2.

Per-edge key selection. The cap N of step (3) in §4.1 is applied to the witnesses of each edge by the following selection. Witnesses below the configured confidence floor are discarded, and the remainder are grouped by the child they name, with cross-child relations forming their own group. The floor is set to `low` in all reported runs, so this step discards nothing and the ledger is compacted by the cap alone. Within each group, repeated statements are dropped and the survivors are ordered by confidence and then by kind, with `table_binding` > `relation` = `visual` = `temporal` > `numeric` > `qualifier` > `entity` > untagged, and the top N are kept, with $N = 8$ throughout. The ordering is deliberate: a relation or a binding is the content that no single leaf embedding carries, so its removal cannot be compensated by the leaf channel, whereas an entity or a value can often be found there. Because the cap applies per group rather than per node, a node with d children can retain up to roughly $N\,(d{+}1)$ witnesses, which is the intended per-edge budget.

Sub-query status. The list Q_t of §4.2 records a status for each sub-query, which is what PLAN edits. Each sub-query has one of five statuses (*pending*, *active*, *done*, *blocked*, or *dropped*), and PLAN refuses to activate a sub-query while one of its dependencies is not done.

Per-step observation. The state s_t of §4.2 is rendered to the orchestrator at every step as the following sequence of text blocks.

1. the task and the query $\mathcal{Q}$;
2. the rules (§F.5);
3. the answer-format block;
4. the tool specifications;
5. the sub-queries Q_t;
6. the open obligations of O_t;
7. the certified evidence G_t;
8. the map M_t;
9. a compacted history of the episode;
10. the output of the last batch of tool calls;
11. the remaining step budget.

The first four blocks are invariant within an episode and are placed first so that the prefix cache of the backbone applies across steps. Blocks 5 to 8 are the four parts of the state, and every block after the fourth is re-rendered at every step, so no message history accumulates.

Batch execution. The calls of a batch (§4.2) are executed against the state in which they were issued: the goal tools PLAN and OBLIGE are applied first, ANSWER last, and the remaining calls concurrently. A call that depends on the result of another call in the same batch is therefore rejected and has to be reissued at the next step, as when a node is read and certified in one batch, since the gate of CERTIFY sees the read state of the step before.

Automatic loss search. Independently of the calls, the runtime searches the loss channel with the embedding of every sub-query at every step and adds the child node of each of its top-12 hits to the map as a searched node, together with its ancestors, so the witnesses the map shows are those hits. Such searches are vector lookups and involve no LLM call, so they do not appear in the call, token, or time columns of Table 2.

End-of-budget sweep. Before the step budget of §4.2 forces an answer, the runtime performs one set of reads that the orchestrator did not decide. Such a sweep is an implementation-level safeguard against a failure that we observed with the local backbone: the orchestrator sometimes leaves a high-scoring node on the map unread, keeps searching or descending instead, and exhausts its step budget with an obligation field still open, although the content it needed was one read away. Prompting the orchestrator to read such nodes did not remove the failure, so the runtime reads them itself. The sweep fires once per episode, at the step before the last one of the step budget (step 13 of the 14 steps used in our experiments), and only if an obligation field is still open at that point; it then reads up to 20 of the unread nodes on the map, highest score first. The read counts of Tables 2 and 3 include such reads.

D ADDITIONAL EXPERIMENTS

This appendix collects the experiments that support §5 without belonging in it.

D.1 LLM-JUDGE QUESTION ANSWERING ACCURACY

Table 7 scores every system of Table 1 with a reference-guided LLM judge, as a check on the string metrics on which every claim of §5 rests. The judge sees the query, the acceptable gold answers, and the predicted answer, and returns a score from 1 to 5 together with an error type, under the prompt of §F.8. Scores are rescaled to $[0, 1]$ and written as percentages, one column per corpus. The three *Average* columns are the corpus means of the judge score and the EM and F1 of Table 1, so that the three can be read against one another. The judge is `Qwen3.6-35B-A3B`, which is also the online backbone of the main results, and as a check we re-scored one run with `GPT-5-mini`. We report the judge here rather than in the main text because some of the benchmarks score answers by exact match

Method	Judge (%) per corpus								Average (%)		
	MMQA	WebQA	DBench	LDocURL	VisDoM	MMDRAG	MMLB-Doc	MMDocIR	Judge	EM	F1
PIXELRAG	50.2	65.3	63.7	71.3	67.1	63.1	53.4	62.0	62.0	24.7	34.5
SIMPLEDOC†	49.9	44.0	64.8	84.0	77.6	71.5	62.2	66.2	65.0	32.8	43.8
DOCLENS†	44.9	45.1	60.5	62.8	62.3	55.1	39.3	55.1	53.1	25.0	35.3
VIDORAG	67.7	75.4	**67.6**	81.0	77.3	55.6	55.7	70.7	68.9	37.3	50.2
VISDOMRAG†	63.2	71.1	67.2	78.3	74.0	62.6	58.7	72.9	68.5	34.1	46.7
FIF	57.2	52.9	42.0	55.9	59.7	45.2	40.8	50.1	50.5	25.4	35.3
HIKEY*	50.8	55.4	55.9	75.1	70.8	55.0	48.4	69.7	60.1	28.9	40.5
RAISE	**69.2**	**77.2**	62.6	**87.0**	**79.6**	**66.6**	**72.1**	**79.4**	**74.2**	**43.1**	**56.2**

Table 7: LLM-judge scores (%) per corpus. †Runs on routed documents (the top-1 document for SIMPLEDOC and DOCLENS, eight candidates for VISDOMRAG). *Our reimplementation. **Bold** is the best among the systems that search the full corpus.

and F1 in their original papers, and scoring all eight corpora the same way keeps the main comparison uniform.

We find that the judge is reliable in two respects. First, it is stable across judges and backbones. The two judges agree within one point on the run we re-scored. Such an agreement is an aggregate over one run, so it does not rule out a bias of the judge toward the phrasing of RAISE through their shared backbone on individual queries; the two judges' averages were similar on the run we checked, and across the four backbones of Table 17 the judge ranks the configurations exactly as EM does. Second, it is consistent with EM and F1 regarding the standing of RAISE: it attains the highest judge average at 74.2, which is 5.3 points above VIDORAG, and among the systems that search the full corpus it is surpassed only on DoubleBench.

The judge and EM nevertheless differ widely in level, by 31.4 points on average across the eight systems of Table 7, because EM scores a correct answer zero whenever its surface form differs from the reference. The gap is of similar size for every system, between 25 and 37 points, so it does not favor one method over another, but it varies with the corpus according to the form of the gold answers. For RAISE it is smallest on MultimodalQA (14 points), whose answers are short entities, and largest on MMDocRAG (54), VisDoMBench (42), and DoubleBench (38), the three corpora whose answers are sentences or lists (§5.2). The leader changes on two corpora. On WebQA, VIDORAG leads by EM and F1 but RAISE leads by the judge, since a WebQA answer is often a short attribute whose wording can differ from the reference without being wrong. On MMDocRAG, RAISE leads by EM and F1 but SIMPLEDOC, which is given its document, leads by the judge, since the judge credits a descriptive answer that covers the reference only in part, which EM does not. Such a gap is why neither metric is reported alone.

D.2 RETRIEVAL ACCURACY

Method	Layer	What one query received					Set recall (%)	
		# docs	unit	# units	text (chars)	images	doc	comp
PIXELRAG	reader (top-5)	2.42	page image	4.97	-	4.97	81.0	52.9
SIMPLEDOC	selected pages	0.98	page image	3.46	-	3.46	81.5	60.3
DOCLENS	navigated pages	1.00	page image	19.80	-	19.80	81.5	35.6
VIDORAG‡	candidates	3.57	page image	11.43	-	11.43	90.4	71.0
VISDOMRAG	text	2.84	text chunk	4.85	13,840	0	59.3	25.4
VISDOMRAG	vision	2.18	page image	5.00	-	5.00	86.0	55.1
FIF	rendered components	4.35	component	19.55	-	-	80.6	49.8
HIKEY*	assembled subgraph	3.72	graph node	174.14	56,504	0	95.1	**78.9**
RAISE	L1 available on the map	7.05	tree node	92.28	27,653	0	98.1	92.9
RAISE	L2 opened by the orchestrator	2.02	tree node	7.54	14,484	11.81	**95.1**	57.4
RAISE	L3 cited as evidence	0.92	evidence	1.11	76	0	80.4	42.1

Table 8: Retrieval accuracy at each method's own reader budget, averaged over seven corpora. ‡Candidates rather than what was read. *Our reimplementation. **Bold** is the best among rows that record what the reader read.

Method	Layer	MMQA	WebQA	DBench	LDocURL	MMDRAG	MMLB-Doc	MMDocIR	Avg
Document set-recall									
PIXELRAG	reader (top-5)	61.4	32.0	98.1	94.5	98.1	95.8	87.3	81.0
SIMPLEDOC	selected pages	63.2	60.9	91.9	92.7	93.5	90.5	78.0	81.5
DOCLENS	navigated pages	63.2	60.9	91.9	92.7	93.5	90.5	78.0	81.5
VIDORAG‡	candidates	90.2	44.2	100.0	100.0	100.0	99.5	99.0	90.4
VISDOMRAG	text	61.1	35.9	89.3	87.9	19.5	60.5	61.0	59.3
VISDOMRAG	vision	78.7	39.9	99.5	99.3	98.4	95.8	90.2	86.0
FIF	rendered components	85.2	90.8	66.7	68.8	86.7	89.0	76.7	80.6
HIKEY*	assembled subgraph	86.1	95.7	95.6	96.7	99.3	95.8	96.4	95.1
RAISE	L1 available on the map	91.3	96.5	99.5	100.0	100.0	100.0	99.5	98.1
RAISE	L2 opened by the orchestrator	84.3	91.5	97.0	98.2	99.1	98.4	97.4	95.1
RAISE	L3 cited as evidence	63.0	65.5	79.1	94.1	89.3	81.1	90.4	80.4
Component set-recall									
PIXELRAG	reader (top-5)	30.4	22.0	77.2	74.3	50.9	55.4	60.2	52.9
SIMPLEDOC	selected pages	37.3	37.5	76.0	77.6	62.1	67.5	64.1	60.3
DOCLENS	navigated pages	22.2	12.4	57.1	36.4	39.5	39.5	41.9	35.6
VIDORAG‡	candidates	63.8	37.6	92.3	87.9	66.7	68.2	80.9	71.1
VISDOMRAG	text	22.2	20.0	47.1	28.7	7.4	19.8	32.8	25.4
VISDOMRAG	vision	40.7	27.1	83.0	74.3	44.2	52.9	63.8	55.1
FIF	rendered components	61.7	77.8	45.0	36.4	38.1	45.9	43.7	49.8
HIKEY*	assembled subgraph	74.3	93.2	84.2	82.0	67.4	70.1	81.4	78.9
RAISE	L1 available on the map	83.9	94.6	95.3	94.9	97.9	88.5	94.8	92.8
RAISE	L2 opened by the orchestrator	50.5	76.3	63.9	55.2	42.8	51.0	62.5	57.5
RAISE	L3 cited as evidence	34.1	48.8	37.8	46.3	30.9	42.7	54.0	42.1

Table 9: Set recall (%) per corpus at the two granularities of Table 8. ‡Candidates rather than what was read. *Our reimplementation.

This section measures retrieval accuracy: what each system had in front of it when it answered, as the recall of the gold documents and components (Table 8, and per corpus in Table 9). Gold documents are those of the benchmarks. Gold components have to be defined over the components of our own parse (§4.1), since the benchmarks label evidence by page or by region rather than by component, and we obtain them in three ways. For MultimodalQA and WebQA, the URL-annotated setting of LILAC (Yun et al., 2025) labels the gold component in the same HTML parse that we use, so we take its labels as they are. For MMDocIR, the benchmark provides the bounding box of each evidence, so the gold component is the MinerU component whose box overlaps it. For DoubleBench, LongDocURL, MMDocRAG, and MMLongBench-Doc, the benchmarks stop at the page, so an LLM selects, on each gold page of the benchmark, the one MinerU component that is essential to the answer, given the query, the gold answer, and the components of that page.

Each row of Table 8 is one *layer* of one method, the set of items that reached its reader. The layer is described by what one query received: the number of documents the set spans, the kind of *unit* it is made of, how many units, and their size in characters of serialized text and in page images. Serialized text and page images are reported separately and never summed, because the token cost of an image depends on the reader's vision tokenizer. *Set recall* is the share of the gold documents (*doc*) or of the gold components (*comp*) that the set contains, averaged over the seven corpora of Table 9; VisDoMBench is left out, since its gold evidence is labelled below the document only for its SlideVQA subset (Table 4). We score the set of components that a method actually put in its reader's prompt, not a fixed rank cut, because a shared @k measures a different quantity for every method: PIXELRAG retrieves twenty pages and passes five of them to its reader, and SIMPLEDOC produces no ranking at all. Each row is therefore scored at its own budget, given as *# units* beside it, and a recall should not be read without it. Document recall must also be read against the number of documents opened: DOCLENS and SIMPLEDOC receive exactly one document per query, so their document recall measures the shared router rather than their own navigation. The two granularities measure different things: document recall asks only whether the right file was reached, and component recall whether the exact component was, so a method can rank first on one and last on the other.

RAISE appears as three nested layers: what is available on its map, what the orchestrator reads, and what the answer cites. Its map covers more gold components than any other row, at 92.9%, and the nodes that the orchestrator reads cover 57.4%, which only HIKEY and SIMPLEDOC exceed among the rows that record what was read.

We analyze four interesting points. First, the three layers of RAISE show where its errors arise: component recall falls from 92.9 on the map to 57.4 for what the orchestrator reads and to 42.1 for what the answer cites, so a large part of the loss arises after the evidence is on the map, in reading and citing it. Second, a single-shot page retriever attains a recall close to that of what the orchestrator reads. The five page images of PIXELRAG cover nearly as many gold components, while the orchestrator maintains a map of about ninety nodes in order to read fewer than eight of them, which is the cost of the agentic design. Third, the two rows above RAISE are not like-for-like: SIMPLEDOC attains its recall on a document given by the shared router rather than found by the method itself, and HIKEY places far more units in its prompt than any other reader (§B.6). Fourth, VIDORAG is the one row that is not a reader budget: its planner selects a subset of the retrieved candidates and the selection is not logged, so its recall measures what the agent could have seen rather than what it read.

The two granularities of Table 8 do not order the methods the same way, and the disagreement is corpus-specific (Table 9). WEBQA is where the ordering inverts most sharply: a WEBQA answer is usually one image inside one wiki page, which a page-level unit cannot point at, so the component recall of what the orchestrator reads is at least twice that of every page-level method. HIKEY is higher still on the same corpus, but the number belongs to the corpus rather than to the router: with one section per document, its 40 retrieved sections are 40 whole documents (§B.6). DOUBLEBENCH is the opposite: its queries ask for several gold pages, and page retrievers such as PIXELRAG and the vision branch of VISDOMRAG cover more of them than what the orchestrator reads. The map of RAISE nevertheless covers more than every page retriever on the same corpus, so the page-level advantage is one of ranking rather than of coverage.

The component recall of FIF on MULTIMODALQA and WEBQA is scored after indexing both corpora from the same source parse that the ground truth uses, because an index built from a page render yields identifiers that match by name and differ by referent.

D.3 EFFICIENCY PER CORPUS

Table 10 breaks the per-query cost of §5.3 down by corpus. Its four blocks are the number of LLM calls, the input and the output tokens summed over those calls, and time. Time is the estimated LLM processing time of one query in isolation, not its end-to-end wall-clock latency, as in Table 2. The LLM calls of a method, which account for the time of a query, cannot be timed by a wall clock, because every method is served by shared vLLM lanes on which many queries run concurrently and a wall clock would measure queueing as much as work. We therefore use the processing time of one LLM query alone, without queueing. Every value is a mean over the queries of one corpus.

It takes more time than SIMPLEDOC and VISDOMRAG, which read one given document, and about as much as FIF, whose row covers its retrieval stage only. PIXELRAG and HIKEY issue one call with a bare-answer prompt, whereas the three-call pipelines of SIMPLEDOC and VISDOMRAG require written reasoning before the answer, which considerably increases their output. DOCLENS passes every page of its document to a navigator and samples several reasoned answers, so it consumes the most input and output tokens, and FIF issues the most calls, each of which returns a short decision. VIDORAG reads one tiled image of its candidate pages, so its input is small, but each of its agents has to reason about every page in writing. FIF logs its counters in the retrieval stage only, so its row covers that stage and is a lower bound on the whole pipeline.

RAISE is never the slowest system on a corpus, and it takes less time than VIDORAG, the strongest competitor, on four of the eight.

We analyze three interesting points. First, the cost of RAISE follows the number of steps a query needs. It is lowest on LONGDOCURL, where the evidence lies within one long document and an episode needs about seven calls, and highest on DOUBLEBENCH and MMDOCRAG, where a query asks for several pages or charts and an episode needs up to twice as many. Second, the ordering against VIDORAG is corpus-specific: RAISE is cheaper on MULTIMODALQA, LONGDOCURL, MMLONGBENCH-DOC, and MMDOCIR, and more expensive on the other four. The average favors RAISE mainly because of MMLONGBENCH-DOC, where VIDORAG writes more than four times as many output tokens as RAISE and takes 75.6 seconds per query. Third, the number of input tokens is not a reliable indicator of cost. RAISE reads the most input tokens on four corpora and is the slowest system on none of them, because prefill is inexpensive and its output remains short.

Metric	Corpus	PIXELRAG	SIMPLEDOC	DOCLENS	VIDORAG	VISDOMRAG	FIF†	HIKEY*	RAISE
# LLM calls	MultimodalQA	1.0	2.8	9.5	4.8	3.0	20.7	1.0	10.0
	WebQA	1.0	2.8	4.7	4.1	3.0	31.5	1.0	11.4
	DoubleBench	1.0	2.9	10.1	4.5	3.0	28.5	1.0	15.0
	LongDocURL	1.0	3.0	15.9	4.3	3.0	23.4	1.0	7.1
	VisDoMBench	1.0	3.0	10.5	4.3	3.0	18.8	1.0	10.2
	MMDocRAG	1.0	3.0	11.2	5.2	3.0	18.9	1.0	12.4
	MMLongBench-Doc	1.0	3.0	14.7	11.8	3.0	22.0	1.0	13.3
	MMDocIR	1.0	3.0	16.8	4.9	3.0	19.1	1.0	8.9
	Average	1.0	2.9	11.7	5.5	3.0	22.8	1.0	11.0
# input tokens	MultimodalQA	2,749	15,847	41,241	21,746	24,146	38,837	12,798	67,863
	WebQA	2,636	13,322	11,748	17,759	23,991	42,379	6,998	65,295
	DoubleBench	4,209	34,777	56,593	22,122	19,997	49,870	13,597	112,642
	LongDocURL	3,823	28,613	149,471	18,024	21,845	35,292	16,876	48,140
	VisDoMBench	3,986	19,579	45,490	17,249	20,410	32,855	12,748	66,257
	MMDocRAG	3,958	35,253	189,459	25,512	20,453	73,092	17,209	83,077
	MMLongBench-Doc	4,010	26,062	148,078	61,520	23,705	65,976	16,511	78,834
	MMDocIR	3,949	27,454	201,103	22,822	23,156	55,525	16,275	55,381
	Average	3,665	25,113	105,398	25,844	22,213	49,228	14,126	72,186
# output tokens	MultimodalQA	199	1,182	4,149	3,171	2,252	1,657	12	1,429
	WebQA	173	1,121	1,201	1,767	2,048	2,277	7	1,522
	DoubleBench	216	1,278	4,354	2,504	1,915	2,178	13	2,057
	LongDocURL	142	1,056	4,967	1,640	1,392	1,703	9	1,056
	VisDoMBench	198	1,156	2,756	1,653	1,737	1,453	18	1,610
	MMDocRAG	271	1,582	5,355	3,363	2,358	1,785	53	2,598
	MMLongBench-Doc	212	1,258	7,415	8,317	2,288	1,966	40	1,979
	MMDocIR	171	1,136	5,322	2,820	1,590	1,606	14	1,293
	Average	198	1,221	4,440	3,154	1,947	1,828	21	1,693
Time (s)	MultimodalQA	2.0	11.7	39.2	28.6	21.5	18.7	1.8	20.8
	WebQA	1.8	10.9	11.3	16.7	19.8	24.2	1.0	21.2
	DoubleBench	2.3	15.1	42.9	23.3	18.2	24.4	2.0	32.0
	LongDocURL	1.7	12.4	60.6	15.7	14.2	18.6	2.4	15.1
	VisDoMBench	2.1	12.0	28.5	15.7	16.8	16.2	1.9	22.1
	MMDocRAG	2.7	17.6	69.1	30.7	21.9	24.4	2.8	32.3
	MMLongBench-Doc	2.3	13.7	80.2	75.6	21.7	24.9	2.6	26.8
	MMDocIR	1.9	12.9	70.5	25.9	16.0	20.6	2.3	18.0
	Average	2.1	13.3	50.3	29.0	18.8	21.5	2.1	23.5

Table 10: Per-query cost per corpus, from which Table 2 is averaged. †Retrieval stage only. *Our reimplementation.

DOCLENS is the opposite case: its page navigation reads whole pages at every step and also writes long outputs, so it is the slowest system on every corpus except WEBQA.

D.4 PER-CORPUS ABLATION

	RAISE (full)		**w/o loss search**		**w/o map obs.**		**w/o ledger**		**w/o anc. closure**		**map → log**		**w/o oblig. gate**	
Corpus	EM	F1	EM	F1	EM	F1	EM	F1	EM	F1	EM	F1	EM	F1
MultimodalQA	**55.3**	**63.0**	52.9	57.4	52.9	58.6	47.5	53.2	50.8	58.3	54.2	60.2	52.8	60.7
WebQA	49.3	59.5	50.0	60.5	51.0	61.7	49.4	58.2	**52.2**	**62.5**	50.8	62.0	52.0	61.2
DoubleBench	24.8	39.1	21.5	38.0	**25.5**	**40.8**	20.9	33.0	20.5	34.2	17.9	33.3	22.2	38.6
LongDocURL	**63.2**	**73.0**	59.9	69.1	60.6	71.2	56.1	64.9	61.0	70.5	59.1	68.6	59.5	69.6
VisDoMBench	**37.5**	**62.0**	32.4	56.7	35.3	56.5	37.1	59.2	34.4	57.3	35.3	59.7	33.7	60.2
MMDocRAG	12.9	35.6	12.6	35.6	9.8	31.5	9.5	32.4	12.0	34.5	11.3	35.4	**13.3**	**36.7**
MMLongBench-Doc	**46.7**	**49.4**	45.1	47.3	42.4	46.3	40.2	42.7	41.9	44.9	42.9	45.5	40.8	43.7
MMDocIR	55.4	67.6	**56.8**	68.4	53.0	67.2	52.2	63.4	54.1	66.5	50.8	62.1	56.0	**69.4**
Average	**43.1**	**56.2**	41.4	54.1	41.3	54.2	39.1	50.9	40.9	53.6	40.3	53.4	41.3	55.0

Table 11: Per-corpus accuracy of the configurations of Table 3. The best score in each row is in **bold**.

Table 11 breaks the ablation of §5.4 down by corpus, with the same seven configurations and the same aggregation, and Table 12 gives their time per query. The columns are EM and F1 of each configuration on each corpus, time is the time per query in seconds, and the average row is the unweighted mean over the eight corpora. Every configuration but the log draws the same map, and every edge on it carries the title of its child. What differs is the text shown beneath an unopened branch: a loss witness in the full system, and nothing without the ledger, so that removing the ledger does not reduce traversal to choosing among unlabelled branches. §D.5 fills that place with the child's own summary instead.

Corpus	RAISE (full)	w/o loss search	w/o map obs.	w/o ledger	w/o anc. closure	map → log	w/o oblig. gate
MultimodalQA	**20.80**	25.17	22.64	23.86	23.63	26.65	22.67
WebQA	21.20	24.30	21.82	22.14	**19.42**	24.80	19.49
DoubleBench	32.00	33.94	33.53	36.07	34.67	35.67	**31.08**
LongDocURL	15.10	15.10	15.96	**14.17**	15.29	19.09	15.15
VisDoMBench	**22.10**	24.69	24.01	23.75	24.78	27.86	22.19
MMDocRAG	**32.30**	34.63	32.31	34.65	37.64	38.90	33.86
MMLongBench-Doc	**26.80**	31.64	28.74	30.96	30.45	34.86	28.73
MMDocIR	**18.00**	21.08	20.28	20.15	19.19	21.90	19.05
Average	**23.50**	26.32	24.91	25.72	25.63	28.72	24.03

Table 12: Per-query time (s) of the configurations of Table 11. The lowest in each row is in **bold**.

Variants. Each configuration removes one part of the full system and leaves the prompt, the tools, and the step budget unchanged. *w/o loss search* drops the loss channel from every search, including the automatic per-step loss search (§C), which is disabled with it, so a witness is never a hit and a branch enters the map only through a summary or a leaf; the witnesses are still drawn on the edges. The orchestrator compensates by descending through the summaries, so steps rise from 6.6 to 7.1 and input tokens from 72k to 84k, the largest increase of the ledger variants, because the drawn witnesses lengthen every prompt without shortening the search. *w/o map observations* keeps the loss channel but draws no witness, so a branch that a witness brought in shows its title and summary alone. Steps barely change, at 6.7, but reads rise from 7.7 to 8.0, since the orchestrator has to open a branch to learn what the witness would have told it. *w/o ledger* removes both, which leaves the retriever over summaries and leaves with the map, tools, and rules of the full system; its episodes are the longest of the ledger variants, at 7.3 steps and 8.7 reads. *w/o ancestor closure* disables the step that, after every batch, adds every ancestor of every node on the map up to its document root (§4.2). A hit then appears without the path above it, so the orchestrator sees neither the document the node belongs to nor the titles that place it, and has to read upward to recover them: reads stay at 8.2 while calls rise to 12.4. Its loss is 2.3 EM against the full system, about the size of one ledger mechanism. *map replaced by a log* keeps every observation but not the map: the state block holds a time-ordered record of every call and its result, oldest first, compacted to the same token budget, with no node states, no merging of a node seen twice, and no recomputed scores, and the rules tell the orchestrator that an entry is a fact about one call rather than a place on a tree. The orchestrator re-derives the structure from the record at every step, so calls rise to 13.5 and output tokens by 31%, and the configuration is the most expensive of all at 22% more time. *w/o obligation gate* keeps sub-queries, obligations, and CERTIFY, but ANSWER is accepted while fields are still open, and the open fields are only reported back. Its episodes are the shortest of all configurations, at 6.4 steps and 7.6 reads, and the 1.9 EM it loses are the answers the gate would have refused, whose open field was the condition that separated the right value from a plausible one.

The full system is the best configuration on average and on four of the eight corpora, removing the whole ledger is the weakest on four, and the log is the weakest on two others.

We analyze three interesting points. First, the ledger is most beneficial where the margin of Table 1 is widest: removing it costs about seven EM on MULTIMODALQA, LONGDOCURL, and MMLONGBENCH-DOC. LONGDOCURL is at the same time the cheapest corpus for every configuration, because its queries sit inside one long document. Second, WEBQA is where the ledger matters least: the four ledger configurations fall within 1.7 EM of one another there and the full system is the lowest of all seven, which is consistent with a corpus whose median document has a single internal node and therefore few omissions to record (Table 19). Third, the ordering of the average is not repeated on every corpus. Removing loss search is best on MMDOCIR, removing map observations on DOUBLEBENCH, removing ancestor closure on WEBQA, and removing the obligation gate on MMDOCRAG, each by at most 2.9 EM. The two ledger cases have a common cause: a witness replaces exploration. A loss hit adds one child to the map and a displayed witness names one node to read, so the orchestrator descends less (Table 16) and reads what the witnesses point to instead of the siblings around it. Such focus is what the ledger is for, but it costs accuracy where the answer needs breadth rather than a pointer: a DoubleBench query has 5.8 gold pages on average, and the orchestrator without map observations reads more sibling pages and covers more of them, while on MMDocIR, whose documents carry the largest ledgers (Table 19), loss search adds branches that a summary search would have ranked lower, and removing it leaves the orchestrator

with fewer but better-ranked candidates. On WebQA, whose documents have a single internal node, there is little to record and every witness only lengthens the prompt, so no ledger configuration is best there. The configuration whose map is replaced by a log is the weakest on DOUBLEBENCH and MMDOCIR, where it loses 7.0 and 4.5 EM. On VISDOMBENCH, removing loss search alone costs more than removing the whole ledger, which suggests that the displayed witnesses can mislead the orchestrator when loss search has not first decided which branches are on the map.

D.5 LEDGER SUBSTITUTE

	Accuracy (%)			**Efficiency / q**						
Text on an unopened branch	**EM**	**F1**	ΔEM	steps	calls	reads	input tok	output tok	time (s)	Δtime
loss witness (RAISE, full)	**43.1**	**56.2**	-	**6.57**	**11.00**	**7.69**	**72,186**	**1,693**	**23.50**	-
nothing (w/o ledger)	39.1	50.9	−4.0	7.31	12.68	8.69	77,593	1,872	25.72	+9.5%
summary, opening	40.4	52.7	−2.7	7.60	13.37	9.34	82,334	1,900	26.59	+13.2%
summary, whole	40.5	52.7	−2.6	7.94	13.82	9.75	81,971	1,938	26.85	+14.3%

Table 13: What an unopened branch shows: the loss witness of the full system, nothing, or the child's own summary. No configuration but the first has loss search. Δ is against the full system, and the best score in each column is in **bold**.

	loss witness (full)		**nothing (w/o ledger)**		**summary, opening**		**summary, whole**	
Corpus	EM	F1	EM	F1	EM	F1	EM	F1
MultimodalQA	**55.3**	**63.0**	47.5	53.2	52.2	57.6	49.7	55.1
WebQA	49.3	59.5	49.4	58.2	50.8	60.6	**54.6**	**64.4**
DoubleBench	**24.8**	**39.1**	20.9	33.0	18.4	32.3	19.0	33.5
LongDocURL	**63.2**	**73.0**	56.1	64.9	59.9	68.7	58.0	68.5
VisDoMBench	37.5	**62.0**	37.1	59.2	32.8	57.4	**39.8**	59.0
MMDocRAG	**12.9**	**35.6**	9.5	32.4	10.2	31.0	10.9	32.2
MMLongBench-Doc	**46.7**	**49.4**	40.2	42.7	46.2	49.1	42.9	46.8
MMDocIR	**55.4**	**67.6**	52.2	63.4	52.7	64.6	49.2	61.9
Average	**43.1**	**56.2**	39.1	50.9	40.4	52.7	40.5	52.7

Table 14: Per-corpus accuracy of the configurations of Table 13. The best score in each row is in **bold**.

Table 13 asks whether the summaries that the index already stores could do the work of the ledger if they were shown where the witnesses stand. In the full system a witness occupies the row of an unopened branch and the frontier nodes of the map, under a cap of twelve per step and a map budget of 1,500 tokens. The two preview configurations put the child's own summary in that place under the same cap and budget, either its opening or the whole of it, and read the summaries and vectors that the index already stores, so nothing is computed offline for them. Neither preview configuration has loss search, so the configuration to compare them with is the one without a ledger, which leaves the place empty, and the three differ only in what an unopened branch shows. Table 14 gives the same four configurations per corpus.

The witness is the most accurate and the cheapest of the three texts, and a summary lies between the witness and nothing in accuracy while being the most expensive. Showing a summary recovers 1.3 to 1.4 of the 4.0 EM that the ledger contributes, and its length does not matter: the opening scores 40.4 and the whole summary 40.5. The configuration of Table 3 that matches the previews is *w/o loss search*, which shows the witnesses on the map and has no loss channel either; it scores 41.4, so the witness shown in the place of a summary is worth 0.9 EM over the whole summary and 2.3 over the empty place, and loss search adds the remaining 1.7 of the ledger's contribution. The cost moves the other way: steps rise from 6.57 with the witness to 7.31 with nothing and to 7.60 and 7.94 with the previews, and reads from 7.69 to 8.69, 9.34, and 9.75. Time follows the same order, at 23.5 seconds per query with the full system against 25.7 with nothing and 26.6 and 26.9 with the previews. With loss search disabled, witness annotations take 26.3 seconds against 25.7 for the empty place, so the lower time of the full system reflects the combined ledger configuration, not the witness annotations alone. A preview therefore invites the orchestrator to open the branch, whereas a witness hands it

the location of what the summary dropped. Such a result is the omission of §2 observed a second way: what a summary leaves out is not recovered by showing more of the summary. Per corpus, the whole-summary preview beats the witness on WEBQA and VISDOMBENCH, the two corpora on which the ledger matters least in Table 11, and loses most on MMDOCIR, DOUBLEBENCH, and MULTIMODALQA. The two exceptions have the same explanation as in §D.4: a WebQA document has a single internal node, so its summary is nearly the whole document and omits little, and a summary shown on the branch is then a better description of it than the few witnesses that can be recorded, while on VisDoMBench the ledger contributes almost nothing (37.5 EM with it against 37.1 without), so any text that describes the branch serves, and the whole summary describes it most fully. Where the ledger contributes most, on the three corpora with the largest losses, the summary shown on the branch is the same summary that omitted the answer, so showing it again recovers nothing.

D.6 AGENT BEHAVIOR

We analyze what the orchestrator does within an episode, between the cost and the answer that §5 reports; Figure 3 traces one episode. Everything here is read off the saved episodes of the full system and of two configurations without a ledger, on the same queries: the *w/o ledger* configuration of §5.4, and a *top-down only* variant of it that searches document roots only and may extend the map only by descending from them. Table 15 counts the calls to each tool of §4.2 per query, rejected calls included, and repeats the rejected calls in a column of their own. For the full system it also gives the share of each tool's calls that fall in the early, middle, and late third of the steps of an episode. Table 16 describes the traversal itself through the nodes the orchestrator reads, counting only the calls the runtime accepted and each node once at its first read. Its descends are therefore fewer than the DESCEND calls of Table 15. The first read in a document is its *entry*, and the *depth* of an entry is the number of tree levels below the document root at which it occurs. Every later read is classified by its relation to the nodes read at earlier steps: a *down* move to a descendant, an *up* move to an ancestor, a *lateral* move to a sibling, or a *jump* to another branch of the same document. A call to DESCEND enumerates the children of a node without reading it, so it is counted on its own. Read calls, nodes read, and the reads of Table 3 are therefore three different counts. A read call can be rejected or repeat a node, so the nodes read are fewer than the read calls. The reads of Table 3 are more, because they include the reads that the runtime issues itself at the end of a step (§C), which are not decisions of the orchestrator and are left out of both tables here.

The full system answers a query in 6.6 steps with 6.5 read calls and 2.2 searches. Its first read in a document lies three levels below the root, reached with 0.8 descends per query, and it reads an ancestor of an earlier read only 0.09 times per query. The configuration confined to the root needs 10.0 steps and 4.8 descends per query to reach a first read that lies 0.7 levels higher.

We analyze two interesting points. First, no order is imposed on the tools, but the episodes acquire one anyway. OBLIGE is called in the first third of an episode 90% of the time and ANSWER in the last third 84% of the time, with SEARCH early, READ in the middle, and RECORD and CERTIFY late in between. The orchestrator itself therefore imposes the sequence of setting the goal, traversing, and grounding that §4.2 leaves free. The orchestrator rarely traverses upward, either: the ancestors of every node it reaches are already drawn on the map with their titles, so the context that a bottom-up pass would recover is in front of it without a read. Second, the search decides how the tree is used, and the corpus decides how deep the search lands. On WebQA, whose median document has a single internal node, the first read lies 1.9 levels down and an episode still descends 1.1 times per query to get there. On LongDocURL, whose median document has 230 internal nodes, the first read lies 3.4 levels down and is reached with 0.3 descends, and such episodes are also the cheapest of all corpora (§D.3). The hierarchy is therefore used exactly where it exists, which is where Table 1 places the widest margins. The cost of forcing every episode to start at the root grows with the same depth: on LongDocURL the top-down configuration descends 4.8 times per query and needs 10.5 steps against 4.8. The variant is an internal diagnostic restriction rather than a reproduction of a prior method. Its accuracy falls with the cost: the top-down configuration answers 4.2 EM below the w/o ledger configuration (34.9 against 39.1, paired $t = -7.94$ over the common queries). The loss is 12.0 on LongDocURL and 13.6 on MMDocIR, while on WebQA, whose trees are flat, the confinement gains 3.9.

Configuration	steps	Tool calls / query								rejected
		PLAN	OBLIGE	SEARCH	DESCEND	READ	RECORD	CERTIFY	ANSWER	
RAISE (full)	6.57	1.66	1.50	2.19	0.83	6.49	2.86	2.42	0.96	0.93
w/o ledger	7.31	2.19	1.51	2.38	1.18	7.52	3.31	2.52	1.01	0.67
w/o ledger, top-down only	10.03	2.17	1.56	3.88	4.85	5.36	2.88	2.02	0.79	0.73
RAISE (full), share of calls by phase										
early third of the steps	-	72%	90%	60%	39%	27%	16%	6%	2%	-
middle third	-	12%	7%	24%	44%	47%	35%	35%	15%	-
late third	-	15%	3%	16%	18%	25%	49%	60%	84%	-

Table 15: Tool calls per query, and when in an episode each tool is called. rejected = calls the runtime refused.

Configuration	nodes read	descends	Entry		Read relative to earlier reads			
			documents	depth	jump	down	up	lateral
RAISE (full)	3.28	0.77	2.00	3.05	0.75	0.18	0.09	0.25
w/o ledger	4.23	1.15	2.45	2.97	0.91	0.45	0.10	0.32
w/o ledger, top-down only	3.44	4.78	1.98	2.33	0.58	0.66	0.05	0.16
RAISE (full), per corpus								
MultimodalQA	3.37	1.22	2.19	2.21	0.68	0.10	0.10	0.30
WebQA	3.19	1.07	2.23	1.93	0.38	0.07	0.11	0.40
DoubleBench	4.46	1.34	2.11	3.04	1.34	0.37	0.16	0.47
LongDocURL	2.18	0.28	1.67	3.43	0.18	0.17	0.04	0.12
VisDoMBench	2.99	0.42	2.00	2.97	0.62	0.14	0.06	0.16
MMDocRAG	3.72	0.69	2.16	3.71	0.93	0.33	0.09	0.21
MMLongBench-Doc	3.78	0.69	1.88	3.64	1.34	0.16	0.13	0.27
MMDocIR	2.56	0.48	1.77	3.51	0.54	0.10	0.05	0.10

Table 16: Where the orchestrator enters each document and how each later read relates to the nodes it read before, per query.

D.7 LLM BACKBONE SENSITIVITY

Table 17 reports the experiment with which we chose the online backbone of the main experiments: four backbones run the same configuration of RAISE on the same index, corpora, and query set, so the rows differ only in the backbone. The configuration is an earlier one than that of Table 1, which uses the witnesses for search only and shows none on the map, so its numbers are not those of §5.2; the backbone it selected is the family whose successor, `Qwen3.6-35B-A3B`, runs all main experiments. *prec.* is the numeric precision at which the backbone is served, and *ans.* is the share of queries that the orchestrator answers before its step budget is exhausted. *calls*, *input*, *output*, and *time* are the per-query cost of Table 2. The time of each row uses the prefill and decode rates measured for that backbone on the same uncontended lane, 8,670/70 tokens per second for `Qwen3.5-4B`, 5,479/40 for 9B, 1,476/24 for 27B, and 7,041/124 for 35B-A3B, so the 27B row is expensive because its dense decode runs five times slower than the mixture-of-experts checkpoint. Accuracy tracks the backbone closely, spanning 16.6 EM across the four checkpoints, so RAISE needs a capable backbone.

We analyze three interesting points. First, accuracy rises steadily with the size of the dense checkpoints, from `Qwen3.5-4B` to `Qwen3.5-9B` and `Qwen3.5-27B`, and the number of calls per query falls in the same order, since a stronger backbone needs fewer steps to close its obligations. Second, time per query also grows with model size, except between the two smallest checkpoints, which take about the same time. The 4B checkpoint reasons poorly over the map, so it traverses inefficiently and answers late: it issues one and a half times as many calls per query as the 9B checkpoint and reads far more input tokens, and it ends fewer than half of its episodes with an answer, against nine in ten on the 27B checkpoint. The unanswered episodes exhaust the step budget rather than certifying the wrong evidence, which is where its low EM comes from, and its cheaper calls are offset by the longer episodes. Third, the mixture-of-experts `Qwen3.5-35B-A3B` offers the best balance between reasoning and efficiency. It holds more parameters than the dense 27B checkpoint but activates about 3B per token, so it is the fastest row, 4.1 times faster than the 27B checkpoint, while remaining within 3.0 EM of it. Such a balance is why the main experiments use the mixture-of-experts family, in its later `Qwen3.6-35B-A3B` checkpoint.

		Accuracy			Efficiency / query			
Backbone	prec.	EM (%)	F1 (%)	ans. (%)	calls	input	output	time (s)
`Qwen3.5-4B`	BF16	28.0	37.9	46.5	19.24	119,064	2,678	51.8
`Qwen3.5-9B`	BF16	34.6	47.1	78.7	12.86	85,871	1,711	58.4
`Qwen3.5-27B`	FP8	**44.6**	**59.0**	**89.7**	**9.18**	**61,536**	**1,481**	104.2
`Qwen3.5-35B-A3B`	FP8	41.6	55.5	85.3	10.82	71,017	1,878	**25.3**

Table 17: Accuracy and per-query cost of the backbone-selection configuration of RAISE (witnesses used for search only) across four online backbones. The best checkpoint is in **bold**.

D.8 A Coding Agent in Place of the Online Loop

		Accuracy				Efficiency / query				
Online loop	**Backbone**	EM (%)	F1 (%)	judge	ans. (%)	calls	input	output	time (s)	API cost ($)
RAISE	`Qwen3.6-35B-A3B`	**43.0**	56.2	78.4	81.1	11.43	**74,915**	**1,764**	24.5	**0**
Claude Code	`claude-sonnet-5`	39.3	**56.3**	**86.4**	**95.9**	**7.05**	237,300	2,740	30.5†	0.232

Table 18: The tools of §4.2 and the index, driven by the online loop of RAISE and by Claude Code, on 800 queries. †Measured API latency, not the time of Table 2. The better value in each column is in **bold**.

Table 18 replaces the online loop of RAISE with a general-purpose coding agent, Claude Code, and holds fixed what the loop operates on: the tools of §4.2, the index, the corpora, and the queries. What changes with the loop is its whole harness, not the backbone alone: the backbone, whether it reasons with extended thinking, how the state reaches it, what it sees of a page, and how an episode is closed, as detailed below. The tools are served to the agent over the Model Context Protocol from the same per-corpus index process that RAISE runs, with the same arguments and the same refusals. The agent therefore cannot read anything that RAISE cannot. Claude Code runs headless on `claude-sonnet-5` inside its own harness, which keeps the conversation history, calls tools natively, and reasons with extended thinking. It receives the rules and the answer format of RAISE as an appended system prompt. The state (Q_t, O_t, M_t, G_t) is appended to the last tool result of every step in place of the view that RAISE re-renders. One assistant message counts as one step under the same budget of 14, and the agent sees a page render directly where RAISE reads the text of its image reader; the end-of-budget sweep of §C is not applied, so an exhausted episode is closed by asking the agent for its answer. Both rows are scored on the same 800 queries, 100 per corpus, so the two rows are paired. The RAISE row differs from Table 1 by at most 0.5 points on any corpus, and *judge* is the judge of §D.1.

The two loops agree on most answers, and the three metrics do not move together. F1 is equal, at 56.3 for Claude Code against 56.2 for RAISE, while EM favours RAISE by 3.7 points and the judge favours Claude Code by 8.0 points. Three quarters of the queries are decided the same way by both under EM, either both right or both wrong, against one half expected if the two were independent. What moves between the rows is mostly how an answer is written and when the agent stops.

We analyze why EM and the judge move in opposite directions. First, Claude Code writes fuller answers: it returns *10 days* where the reference is *10*, *91.77 million* where it is *91.77*, and *a fountain* where it is *fountain*. EM rejects such spans and the judge accepts them. F1, which credits the shared tokens, sits between the two, which is where such a difference puts it. Second, Claude Code answers where RAISE abstains: Claude Code ends 95.9% of its episodes with an answer and RAISE 81.1%, and Claude Code reaches the step budget on 5.9% of the queries and RAISE on 22.2%. MMLongBench-Doc is the one corpus whose references include *Not answerable*. There an exhausted episode of RAISE returns that string and is right on some of the queries, whereas Claude Code returns the nearest value it certified, which EM scores as wrong. The EM gap therefore concentrates on the two long-PDF corpora with long or abstaining references, MMLongBench-Doc (12.0 points) and MMDocIR (14.0 points). Claude Code leads on MultimodalQA (7.0 points) and WebQA (6.0 points), and the judge favours it on all eight corpora. Third, the two cost profiles differ in kind. Claude Code issues fewer calls per query than RAISE, 7.05 against 11.43, because it reads a third as many

Corpus	# documents	# pages			# internal nodes			# leaf nodes		
		med	mean	max	med	mean	max	med	mean	max
MultimodalQA	3,254	12	12	-	11	15	227	25	38	411
WebQA	7,743	8	7.6	-	1	4	228	5	13	281
DoubleBench	650	20	23.2	50	58	70	527	136	169	1,234
LongDocURL	396	76	84.4	147	230	284	1,449	524	657	3,931
VisDoMBench	841	12	14.8	265	41	49	911	76	88	1,132
MMDocRAG	147	28	75	830	92	329	2,736	177	643	5,913
MMLongBench-Doc	135	28	47	441	110	192	2,867	199	387	8,311
MMDocIR	313	27	64.3	830	92	265	2,867	169	514	8,311

Table 19: Shape of one document per corpus after the tree of §2 is built.

Corpus	SimpleDoc		DocLens		ViDoRAG		VisDoMRAG		FiF		HiKEY		RAISE	
	med	mean	med	mean	med	mean	med	mean	med	mean	med	mean	med	mean
MultimodalQA	2.2	2.2	1.5	1.5	4.7	4.7	1.9	1.9	1.5	1.5	1.5	1.5	6.5	8.0
WebQA	1.1	1.1	0.9	0.9	2.8	2.8	1.1	1.1	0.9	0.9	0.9	0.9	2.2	5.2
DoubleBench	11.6	13.5	5.0	5.8	10.4	12.1	5.5	6.4	5.0	5.8	5.0	5.8	32.5	37.1
LongDocURL	9.3	10.3	7.2	7.9	28.0	30.9	9.1	10.1	7.2	7.9	7.2	7.9	71.1	82.5
VisDoMBench	2.8	3.5	1.4	1.7	4.8	5.9	1.7	2.1	1.4	1.7	1.4	1.7	19.7	22.3
MMDocRAG	23.2	62.1	1.3	3.6	8.0	21.4	2.1	5.6	1.3	3.6	1.3	3.6	36.3	90.0
MMLongBench-Doc	4.7	7.9	1.8	3.0	7.1	12.0	2.6	4.3	1.8	3.0	1.8	3.0	42.9	59.6
MMDocIR	8.5	20.2	1.8	4.3	9.2	21.9	2.6	6.1	1.8	4.3	1.8	4.3	32.0	66.6

Table 20: Seconds to add one document (median and mean), parsing included.

nodes (2.4 against 7.1) and rarely has a call refused (0.20 against 1.07), with Claude Code first in each pair. Its harness, however, carries the whole conversation into every call, so it reads 237,300 input tokens per query, 3.2 times the bounded view of RAISE, and writes 2,740 output tokens with thinking included. Such tokens cost 0.232 dollars per query on the API, 185 dollars for the 800 queries, where the local backbone costs nothing beyond the GPU it occupies. The time of the Claude Code row is measured API latency, listed for completeness rather than for comparison with the time of the RAISE row, which is computed as in Table 2.

Two conclusions follow. The tools and the state of RAISE are not tied to its own loop: a general coding agent picks them up unchanged, closes 96% of its episodes with an answer, matches RAISE on F1, and exceeds it on the judge. The action set and the data structures therefore transfer to the harnesses that practitioners already run. The loop of RAISE reaches the same F1 from a locally served mixture-of-experts backbone with 3B active parameters, at zero API cost and with a third of the input tokens, which is the setting the main results are built for.

D.9 Offline Indexing Cost

We measure what it costs to add one document to the index of each method, in time (Table 20) and in storage (Table 21), since such a cost is paid before any query and never appears in the per-query cost of Table 2. Both are governed by the shape of the documents, which Table 19 gives first. The time of a document is obtained per stage and per unit: the cost of one unit (a summary per internal node, a set of witnesses per node, one call per image) is measured on serving lanes dedicated to that stage, with every stage on the local backbone of the accuracy experiments and its thinking mode off, and a stage's cost is used only if it wrote every artifact it attempted.

D.9.1 Corpus and Document Shape

The cost of indexing a document is set by what it contains. Table 19 counts pages, internal nodes, and leaf nodes for the median, the mean, and the largest document of each corpus. The two statistics agree on the web corpora and separate by a factor of three on MMDocRAG, whose median document is 28 pages against a mean of 75, which is why every per-document number in this appendix is reported both ways.

Corpus	Source	SimpleDoc		DocLens		ViDoRAG		VisDoMRAG		FiF*		HiKEY*		RAISE	
		MB	×	MB	×	MB	×	MB	×	MB	×	MB	×	MB	×
MultimodalQA	0.6	2.4	4.3	0.2	0.3	2.3	4.0	3.1	5.3	0.4	0.7	0.3	0.6	1.2	2.1
WebQA	0.5	1.6	3.0	0.1	0.2	1.5	2.9	1.9	3.8	0.1	0.3	0.1	0.2	0.4	0.7
DoubleBench	11.4	4.6	0.4	0.2	0.0	4.5	0.4	5.9	0.5	2.2	0.2	1.4	0.1	6.2	0.5
LongDocURL	7.2	16.9	2.1	0.9	0.1	16.7	2.3	21.8	3.0	18.0	2.5	5.4	0.8	24.3	3.4
VisDoMBench	1.1	3.0	2.7	0.1	0.1	3.0	2.7	3.9	3.6	0.9	0.9	0.7	0.7	4.2	3.9
MMDocRAG	4.8	21.1	3.2	1.0	0.2	14.8	3.1	19.3	4.0	33.2	7.0	5.4	1.1	22.4	4.7
MMLongBench-Doc	4.8	10.7	2.2	0.5	0.1	9.4	2.0	12.3	2.6	18.0	3.8	3.2	0.7	11.9	2.5
MMDocIR	3.9	17.9	3.4	0.8	0.2	12.8	3.2	16.6	4.2	24.8	6.3	4.3	1.1	19.3	4.9

Table 21: Index size per document (MB, mean over the documents of a corpus) and its multiple over the *Source* document. *Our reimplementation.

D.9.2 Per-Document Indexing Time

A corpus is rarely built in one shot: documents arrive over time, and what matters then is the cost of adding one, which Table 20 reports. The cost of a document is set almost entirely by how many internal nodes it has. The per-node cost varies little across corpora, at about one second of compute for the summary and two to four for the witnesses, whereas the median node count ranges from 1 on WebQA to 230 on LongDocURL. Node count in turn tracks length, at three to four internal nodes per page on every corpus, so the tree does not grow pathologically on any particular corpus: long documents have proportionally more of it.

An interesting point is that the ordering against ViDoRAG reverses on the thin HTML corpora. A WebQA document carries a single internal node, so the stages that dominate elsewhere are hardly executed, and RAISE indexes such a document faster than ViDoRAG does. RAISE is expensive exactly where documents are long, and there its per-document cost is between 2.5 and six times that of ViDoRAG.

D.9.3 Per-Document Storage

Storage orders the methods differently from time. We count only the files a method's query path opens to answer a new query, one copy each, and we leave out page images, which can be re-rendered from the source. SimpleDoc and DocLens index only the documents the shared router picked, so their numbers are per indexed document. Under such accounting, RAISE is small on the thin HTML corpora, where an indexed WebQA document takes less space than its source, and it is among the largest on long documents, at 24.3 MB per LongDocURL document against 16.7 for ViDoRAG. The growth comes from the ledger, since every witness is embedded on its own in addition to the nodes of the tree. Time and space are thus consumed by different parts of the index, and a method that is expensive in one need not be expensive in the other.

E Limitations

Scope of the ablation. The summaries are written under a policy that lets them omit detail because the ledger records it (§F.2), so the ablation of §5.4 measures the ledger under that policy, and a ledger capped at N witnesses per edge, extracted by a model, does not preserve every omission. Whether omission-aware extraction outperforms a general fact index built without such a policy is not established here.

Offline cost. Node summarization and witness extraction add one LLM call per internal node beyond a plain summary tree, and witness extraction runs over the same nodes a second time. Such a cost is paid once per document and amortized over queries, and we report indexing cost alongside per-query cost (§D.9). Online cost is likewise reported as time on self-hosted hardware, where a long prompt is cheap to prefill.

Witness noise. Loss witnesses are generated by a model and can be wrong: a witness may state an omission that the child does not contain, or miss one that it does. A witness is only a retrieval target

Stage	Purpose	Model	Prompt
Corpus construction			
Filtering	retrievability, #A vs #B	`gpt-5.4` + `Claude Sonnet`	§F.6
Gold rewrite	verbose to single span	one judge	§F.7
Labeling	page and component ground truth	one judge	released code
Offline indexing			
Nesting	recover the section layer	Qwen local	§F.1
Verbalizing	image to text	Qwen local	§F.3
Summarizing	one summary per internal node	Qwen local	§F.2
Ledger	extract the loss witnesses	Qwen local	§F.4
Online			
Orchestrator	the whole episode	Qwen local	§F.5
Evaluation			
QA judge	grade an answer 1–5	Qwen local	§F.8

Table 22: Where each prompt is used. Model identities are given in §B.5.

and an annotation, and a value can be certified only from a node that has been read (§4.2); the gate checks the source reference, not the value. The orchestrator can therefore certify the statement of a witness while citing the read node it points to, and the error of a false witness can still reach the answer.

Dependence on the summarizer. The ledger records what one summarizer omitted from one tree, so it is tied to that summarizer and to the hierarchy the parser recovered. Changing either invalidates the recorded omissions and requires witness extraction to be run again, and a document whose structure the parser flattens yields shallow trees with few edges on which to record anything. We do not evaluate how the ledger transfers across summarizers or how retrieval degrades with the quality of section re-nesting.

F Prompts

Every prompt below is reproduced from the implementation rather than paraphrased. Table 22 locates each one in the pipeline, and the sections that follow give the text. Where a prompt is too long to reproduce whole, we say what was cut and on what principle, and the full text is in the released code. The only edit applied to any reproduced text is typographic: em dashes are set as hyphens, and a runtime-substituted value appears in braces, as in `{max_words}`.

F.1 Section Re-nesting

This prompt is the re-nesting call of §4.1, described in §C.1. The count in brackets after each heading is the number of content blocks directly beneath it, which is what separates a container from a leaf section.

Section re-nesting (system)

```
You reconstruct the nested SECTION HIERARCHY of a document from a flattened heading
    list.
You are given a document title and an ORDERED list of section headings that a layout
     parser extracted but FLATTENED to a single level (nesting lost). Each heading
    is annotated with [N] = the number of CONTENT blocks (paragraphs, tables, images
    , lists...) directly under it before the next heading.
Return JSON {"sections":[{"section":"<heading text>","parent":"<parent heading text
    or ROOT>"}]}: for every heading, give its IMMEDIATE parent heading text, or "
    ROOT" if top-level. Echo each heading's text EXACTLY as given (WITHOUT the [N]
    annotation).
```

```
HOW TO USE [N]:
- A heading with [0] (or very few) components immediately followed by another
    heading is usually a PARENT/container whose body IS its subsections.
- A heading with MANY components is usually a LEAF content section.
- Two content-heavy headings in a row are usually SIBLINGS at the same level.

Respect READING ORDER (a parent appears before its children). Use general knowledge
    of how Wikipedia articles are structured. You MAY invent a missing parent
    heading and MUST drop noise (below).

AVOID TWO OPPOSITE MISTAKES:
(1) OVER-NESTING: most sections are top-level. Do NOT nest a normal stand-alone
    section under a sibling. A parent must be a BROADER CONTAINER, never a narrow
    sibling at the same level. If a section has no clear broader container before it
    , keep parent=ROOT.
(2) UNDER-NESTING: when SEVERAL consecutive headings are sub-types of ONE broader
    theme, group them under their common parent - inventing it (with the
    conventional name) if absent from the list. A [0]-component heading right before
     such a group is the missing container.

Common Wikipedia grouping patterns - recognize the CATEGORY (do not rely on any
    exact title):
- discography sub-types (studio/live/compilation albums; singles as-lead / as-
    featured; videos) -> a "Discography" / "Albums" / "Singles" container
- filmography sub-types (film, television, telefilms, web) -> a "Filmography"
    container
- award / honour lists -> an "Awards" / "Accolades" container
- biography periods and career phases -> a "History" / "Career" container
- sports-season sub-parts (regular season, standings, schedule, playoffs, roster,
    coaching staff) -> "Season" / "Regular season" / "Playoffs" / "Team" containers
- sub-topics of a place (climate under geography; sub-arts under culture; modes
    under transport) -> their broader topic

NOISE - DROP these (do not output them) and NEVER use them as a parent:
- a heading that repeats the DOCUMENT TITLE,
- an infobox field / caption / photo or image label (e.g. a "... release poster", a
    person/product name that is actually a picture caption).
```

F.2 Node Summarization

This prompt writes the node summaries of §4.1, and two of its properties matter for the argument of this paper. It tells the summarizer that omission is expected rather than a failure, because the Loss Ledger records what is omitted, which is what lets the summary be decisive instead of hedged. It also asks for relations across children, since a comparison or a progression exists in no single child and would otherwise be absent from every vector store. The variant that emits a list of propositions instead of prose shares these rules and is omitted here.

Node summarization (system, prose variant)

```
You write one navigation summary for a single node of a document tree.

A reader uses your summary for exactly one decision: whether what they are looking
for is inside THIS node's subtree, or inside one of its SIBLINGS' subtrees.

Your summary is not evidence and is never quoted in an answer. It is also never read
as an exhaustive list - whatever you leave out is recorded separately and stays
findable by other means. So do not try to be complete, and do not apologise for
    being
incomplete. Be decisive.

You are given:
```

```
- the ancestor path (already known to the reader)
- the content of this node's CHILDREN - what you are compressing
- the titles of this node's SIBLINGS - who you compete with

RULES
1. Length: at most {max_words} words. Fewer is fine. Hard limit, not a target.
2. Open with one short clause naming what this subtree is about, in the words
     someone
   searching by topic would use. Spend everything after that on contrast.
3. Be concrete. Name what is named in the CHILDREN: entities, systems, datasets,
   metrics, figures, conditions, places, time periods, quantities. A named thing
     beats
   a characterisation of it ("TPC-H Q17" beats "a benchmark query").
4. If the children only mean something in relation to each other - a comparison
     across
   them, a progression, a before/after, a shared axis - state that relation. It
     exists
   in no single child and nowhere else in the index.
5. Modality. Tables: name the row and column dimensions and their units, not cell
   values. Charts and figures: name the axes and what is being compared. Time-
     ordered
   children: name the order of events.
6. Do not repeat the ancestor path.
7. Never hedge coverage. Banned: "various", "several topics", "a range of", "among
   other things", "different aspects", "and more".
8. If the children will not fit the budget, do NOT hedge and do NOT enumerate
   everything shallowly. Keep the rare and distinctive over the frequent and typical
     ;
   drop the rest silently. A wholly dropped child is expected and handled elsewhere.
9. No meta-phrasing: not "this section presents X and Y" but "X, Y". Naming children
   is encouraged; narrating that they exist is not.
10. Output ONLY the summary text - no preamble, no title, no quotes, no markdown.
```

F.3 Image Verbalization

This prompt produces the verbalization folded into a `component` node in §C. Each image component is verbalized once, offline, so that the text a retriever matches against is the same text the summarizer saw. Captions and footnotes are injected as plain text when the parser recovered them, which grounds names and numbers that the pixels alone would not fix.

Image verbalization (system)

```
You are an image analyst. Write ONE concise English description (<= 60 words) of the
     given image so it can be retrieved and understood without seeing it.

First decide what the image is, then describe accordingly:
- Natural / photo (person, place, object, animal, artwork, logo, map, screenshot,
    etc.): name the main subject(s) as specifically as possible, plus setting,
    notable attributes, and any visible text/labels.
- Document figure (chart, plot, diagram, flowchart, equation, illustration): state
    the figure type and its key content - for charts the chart type + axes/trend/
    notable values; for diagrams the components and their relation; for equations
    what the formula expresses.

If caption / context text is provided, use it to ground names, entities, and numbers
     - do NOT contradict it, but do NOT merely copy it. Be specific and keyword-rich
    . No preamble, no markdown, no "This image shows". Output the description text
    only.
```

F.4 Loss Witness Extraction

This is the prompt that builds the Loss Ledger of §4.1, and it is the inverse of §F.2: the summarizer decides what to keep, and this pass records what it omitted. The summary contract is what makes the contrast tractable, because the summary is already known to hold the topic, the named highlights, and the table dimensions, so the extractor hunts the complement. Rule 1 is the load-bearing one, because without it the extractor paraphrases the summary and the loss channel degenerates into a second copy of the summary channel, which would leave the ablation of §5 measuring nothing. The budget in rule 8 is the per-edge cap N whose selection rule is given in §C.

Loss witness extraction (system)

```
You audit ONE node's summary for what it LEFT OUT.

A navigator sees ONLY this node's summary and uses it for one decision: whether to
descend into this node's subtree. Anything the summary dropped is invisible to that
navigator. You are given the summary AND all of the node's children - the source it
was compressed from. List what is present in the children but a navigator could NOT
find or route to from the summary alone.

Each item is a LOSS WITNESS: a routing key, not evidence. It must be grounded in a
specific child and phrased so a future query about it would match.

RULES
1. Contrast only - the most important rule. A witness must add what the summary does
   NOT already convey. Before you keep one, check it against THIS NODE'S SUMMARY
     above:
   if the summary already states it, even in other words, DROP it - a paraphrase of
     the
   summary is not a witness. Keep only what the summary omits, blurs, or under-
     specifies.
2. Retrieval-facing. Write each witness as a self-contained statement a query could
   match: resolve pronouns, name the subject/entity, keep units. Prefer
   "Model Z scores 78.4 on scanned invoices" over "78.4".
3. Grounded, no invention. Only what actually appears in a child. Name the child it
   comes from (child_cid). If you cannot point to where it is, do not emit it.
4. Prefer what changes routing - especially signals no single child carries alone:
   a comparison across children, a table row-column binding, a qualifier spread over
   several children, a trend or extremum. These are the whole point of this ledger.
5. Modality - recover exactly what a schema-level summary drops. Tables: the
     specific
   row-column-value BINDINGS (the summary kept only the dimensions). Charts/figures:
   the specific extrema, legend mappings, comparisons. Time-ordered: the specific
   events, not just that an order exists.
6. Dropped children. A child the summary does not mention at all is a prime source
     of
   witnesses - recover its distinctive content.
7. Tag each witness with a kind (entity, numeric, relation, qualifier, table_binding,

   visual, temporal) and a coarse confidence (low, medium, high). No fake numeric
     scores.
8. Budget is PER CHILD: at most {k_loss} witnesses for any one child_cid (a cross-
     child
   relation that belongs to no single child gets child_cid = null, with its own
     budget
   of {k_loss}). A node with several children may therefore have many witnesses
     total.
   Self-enforce this limit - do NOT overflow and rely on being truncated. If a child
      has
   MORE than {k_loss} omissions worth keeping, keep the most routing-critical, and
     where
   several small omissions are closely related, MERGE them into one higher-level
     witness
```

```
   (coarser granularity) so the {k_loss} you keep cover the most ground. Scale the
     count to
   the omission: a dense child the summary barely covered deserves several witnesses
     - do
   not stop at one or two while real routable facts remain uncovered - while a child
     the
   summary already covers well deserves few or none.
9. Also give a coarse coverage in [0,1]: how much of the children's retrieval-
    relevant
   content the summary already conveys.
10. Output ONLY a JSON object, no prose:
{"coverage": <float>, "witnesses": [{"lost": "<statement>", "child_cid": "<cid or
    null>", "kind": "<one kind>", "confidence": "<low|medium|high>"}]}

Example:
{"coverage": 0.5, "witnesses": [{"lost": "Model Z scores 78.4 on the scanned-
    invoices subset", "child_cid": "table_12", "kind": "table_binding", "confidence
    ": "high"}]}
```

F.5 ORCHESTRATOR

This prompt drives the orchestrator over the tools of §4.2. The orchestrator's rule block runs to roughly 290 lines, so we reproduce the clauses that carry the three invariants the system is built on rather than the whole of it. Everything else in the block is guidance on reading the map and the scores, and the full text is in the released code. The first invariant is the grounding boundary of §4, and it is the only one enforced in code rather than by instruction. The second is what makes a batch of calls concurrent, since a call that carries its own path cannot depend on another call's side effect. The third is why a score is never compared across steps. The clauses below are those of the full configuration; the ablated arms differ in the loss-related lines, as noted in §B.

Orchestrator rules (excerpt: the three invariants)

```
## What you are doing

You are building a map of a document hierarchy while answering a question.
Each turn you see the map as it stands, what you still owe evidence for, and what
you have already grounded. You choose where to extend the map next and what to read.
You are not choosing a named search strategy. Top-down, bottom-up and bidirectional
are just descriptions of which parts of the map you happened to extend.

## Where map entries come from, and what the △ lines mean

You are shown only the closest few and there is no count, so a node with no △ under
     it
may still have dropped a great deal. Absence of a quote is not absence of content.

- A quote tells you what a branch dropped. It does not tell you that the branch
  answers your question, and it is not something you may cite - read() the place it
  points at and cite that.
- A node you did not search for is a lead, not an answer.

## Reading the scores

Every node on the map carries the cosine of its stored vector against THIS turn's
sub-query, and so does every witness. Both are recomputed from scratch each turn, so
     a
score answers only the question you are asking right now.

Read them as an ORDER, never as a level. No rule here states a threshold,
    deliberately:
```

```
the numbers are uncalibrated and their range moves with the sub-query, the tier and
    the
corpus, so 0.42 is not "weak" - it is only above or below whatever sits next to it
    in
the same list.

## Evidence rules   [enforced in code - violations are rejected, not warned]

- Only content you have read() can be cited. Summaries, descriptors and WARN lines
    are
  navigation aids. certify() will reject a source_ref whose map state is not READ.
- BEFORE you certify, read the fact and the field's `need` side by side and check
    they
  describe the SAME thing. A fact naming a different year, entity, unit, population
    or
  basis than the need asked for does NOT fill that field, however plainly the source
  states it.
- An obligation with one field asks nothing of you. A question almost always fixes
  conditions on its answer as well as asking for a value - which entity, which
    period,
  which unit, which basis, which population - and each of those is a field, because
    a
  value that satisfies the question's shape but not its conditions is wrong in a way
  a single field can never detect.

## Working several places at once

- Calls in one turn are executed independently. Batch everything that does not
    depend
  on the result of another call in the same turn.
- Different obligations can be pursued from different parts of the map in the same
    turn.
  A downward call and an upward call in one batch is normal and preferred.
```

F.6 Retrievability Filtering

This prompt implements the criterion of §B.4. It carries 43 in-context examples drawn from LongDocURL, DoubleBench, and VisDoMBench and labelled by hand; we reproduce eight of them here and the rest are in the released code. The eight are chosen to cover the cases the criterion turns on rather than to be representative of the distribution. Among the #B examples, one is a reference to document structure, one is a name internal to a paper, and one is a topic description with no anchor. Among the #A examples, one is multi-hop, which is the case a reader is most likely to assume we discarded.

Retrievability classification (system, 8 of 43 examples shown)

```
I have some QA data here, and you can observe that the questions can be divided into
     two categories:

The category #A: When you see this question alone without a given document, you are
    sure to find a unique document in a corpus to provide a unique answer.
Multi-hop questions also belong to #A as long as their unresolved references can be
    progressively pinned down by following named entities through successive
    retrieval.
For example, "In the show where Tommy Sands played Joey Rand, who played the first
    McGarrett?" is #A: the show is uniquely determined by "where Tommy Sands played
    Joey Rand", and the chain resolves to one answer. Likewise, "the latest year in
    which Marcello Aliprandi is listed as director" is a resolvable reference
    because Marcello Aliprandi is a named entity.
```

```
The category #B: When you see this question alone without a given document, you will
     find hard to locate a document to give a deterministic answer for this question
    , because you will find multiple candidate documents in a corpus, which may lead
     to different answers for this question.
References to document structure such as "the second table", "the page above", "the
    figure", "this document", or generic phrases like "the proposed method" without
    a name are not resolvable across a corpus, so such questions are #B.

The decisive criterion is where the query begins.
#A queries start from a specific detail in the query itself - a named entity, proper
     noun, unique title, product code, case name, or technical term that pins down a
     document in the corpus on its own.
#B queries start from a referring expression that assumes a document is already in
    hand - phrases like "the document", "this paper", "as described in the doc", "
    the proposed method", "the figure", "the page above".
Names that look specific but are actually paper-internal labels (e.g., "the proposed
     vector-res method", "the optimization problem (6)") are referring expressions
    too, not external anchors, so they signal #B.
Topic-only descriptions without a named anchor (e.g., "utterance-level intent
    recognition in autonomous vehicles") match too many candidate documents to pin
    one down, so they are also #B.

Here are some examples:
What is the price of the 2017 Cote-Rotie La Barbarine from Domaine Gangloff? #A
How many times was the author 'Nelson et al.' cited in this document? #B
Which section of copyright law was pivotal in Atari Games Corp. v. Nintendo? #A
List names of the figures at the page which contains a table whose name is "Figure
    92 Global Block/Sector Unlock Sequence Diagram (QPI)". #B
In the show where Tommy Sands played Joey Rand, who played the first McGarrett? #A
What entity is defined and solved by the optimization problem (6) that specifies and
     solves the transition matrix Pa? #B
What are the operating lease liabilities listed on SeaChange International, Inc.'s
    consolidated balance sheet as of April 30, 2021, and January 31, 2021? #A
Does the paper present a comparison of multiple approaches for utterance-level
    intent recognition in autonomous vehicle scenarios? #B
```

The user turn supplies the query and asks for a single letter.

Retrievability classification (user)

```
Query: {question}
Determine if the query belongs to Category #A or Category #B.
Output only A or B.
```

F.7 Concise Reference Answers

This prompt performs the gold rewriting of §B.4. The rules below are reproduced in full and the eight worked examples that follow them in the source are omitted. The last rule is the one that keeps this rewrite from corrupting the benchmark, since several corpora state an output format inside the query itself and a rewrite that ignored it would make the gold unmatchable in a new way.

Concise gold rewriting (system, examples omitted)

```
You extract a concise answer from a verbose Original Answer.

Given a Question and its Original Answer, output the shortest text that directly
    answers the Question.

Rules:
```

```
- Output ONLY the concise answer text - no labels, no explanation, no surrounding
    quotes.
- Keep the minimum span that directly answers the Question. Prefer a single word/
    phrase when possible.
- Yes/No questions: output exactly "Yes" or "No" matching the Original Answer's
    polarity.
- "What [object/thing]..." questions: output the object noun phrase only (e.g., "
    fountain").
- Choice/comparison questions ("taller or shorter", "happy or upset"): output only
    the chosen option word.
- Numbers: output the numeric value (and unit if the Original Answer has one), no
    extra words.
- Do NOT copy words from the Question that the Original Answer merely repeats/
    paraphrases; keep only the new, answer-bearing information (unless those words
    are required to answer).
- Preserve the Original Answer's casing/pluralization (e.g., "Circles").
- Do not add locations, explanations, or context unless required to uniquely answer.
- RESPECT any output-format instruction stated in the Question. Examples:
  * "answer in list format, e.g., ["a", "b"]" -> output a list like ["a", "b"] (
    comma + SPACE between items).
  * "round to one decimal" / "to two decimal places" -> output the number at that
    precision.
  * "Answer in million(s)" / "in percentage" -> keep that unit/scale.
  * "answer as YYYY-MM-DD" -> output the date in that format.
  If the answer is genuinely a list of items, keep all items (do not drop any).
```

F.8 Correctness Judge

This prompt produces the judge scores of §D.1. The error type it returns is recorded for analysis only and does not enter the score. It is reproduced in full, and the runtime-substituted values appear in braces as elsewhere in this section.

Reference-guided correctness judge (system and user)

```
You are an expert, impartial evaluator for short-answer question answering.

Question:
{question}

Acceptable gold answers (ANY one of these is fully correct):
{gold_answers}
Answer value type: {answer_type}   # string | integer | float (hint, not a hard rule
    )

Predicted answer:
{predicted}

Grade the predicted answer against the gold answer(s) on a 1-5 scale.
- Accept paraphrases, synonyms, reordering, and equivalent numeric/unit/format
  variations (e.g. "5%" == "0.05", "120,000" == "about 120k"). Do NOT require exact
    match.
- Use the answer value type as a hint for numeric equivalence.

Scoring rubric:
5 = fully correct - semantically equivalent to a gold answer (format/unit/paraphrase
     OK)
4 = essentially correct, minor issue only (slightly imprecise wording, unit/format)
3 = partially correct - captures part of it but misses a required element
2 = related but wrong - right topic/entity, wrong value or specifics
1 = incorrect, contradictory, or not answered
For an unanswerable question, a correct abstention = 5.
```

```
Return ONLY a JSON object:
{
  "score": 1-5,
  "error_type": "none | wrong_entity | wrong_number | incomplete | over_specific |
     contradiction | not_answered",
  "rationale": "one short sentence"
}
```