

---

# FAILURE IS FEEDBACK: History-Aware Backtracking for Agentic Traversal in Multimodal Graphs

---

Anonymous Authors<sup>1</sup>

## Abstract

Open-domain multimodal document retrieval aims to retrieve specific components (paragraphs, tables, or images) from large and interconnected document corpora. Existing graph-based retrieval approaches typically rely on a uniform similarity metric that overlooks hop-specific semantics, and their rigid pre-defined plans hinder dynamic error correction. These limitations suggest that a retriever should adapt its reasoning to the evolving context and recover intelligently from dead ends. To address these needs, we propose FAILURE IS FEEDBACK (FiF), which casts subgraph retrieval as a *sequential decision process* and introduces two key innovations. (i) We introduce a *history-aware backtracking mechanism*; unlike standard backtracking that simply reverts the state, our approach piggybacks on the context of failed traversals, leveraging insights from previous failures. (ii) We implement an *economically-rational agentic workflow*. Unlike conventional agents with static strategies, our orchestrator employs a cost-aware traversal method to dynamically manage the trade-off between retrieval accuracy and inference costs, escalating to intensive LLM-based reasoning only when the prior failure justifies the additional computational investment. Extensive experiments show that FiF achieves state-of-the-art retrieval on the benchmarks of MULTIMODALQA, MMCoQA and WEBQA.

## 1. Introduction

Searching the web has become a part of everyday life. This routine increasingly underpins multimodal retrieval-augmented generation (RAG), where a model answers

<sup>1</sup>Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

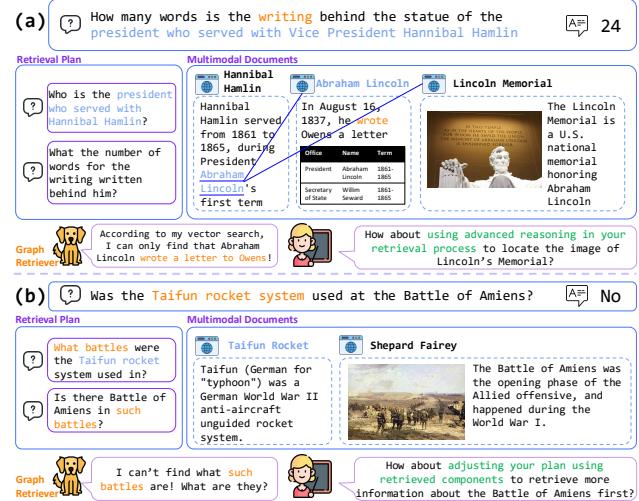


Figure 1. Motivating examples of multihop retrieval failures in existing graph retrieval approaches. (a) Vector-similarity-driven traversal follows a spurious cue. (b) Fixed retrieval plan produces an underspecified hop and fails to recover from a dead end.

a user query by grounding its output in retrieved evidence (Faysse et al., 2024). In practice, much of this evidence lives in webpages or PDFs—multimodal documents with three salient characteristics: (i) each document is composed of multimodal *components* (paragraphs, tables, and images); (ii) the meaning of a component is often shaped by local document context (e.g., captions and surrounding components); and (iii) components are connected through explicit signals (hyperlinks, cross-references) as well as implicit signals (e.g., same-section adjacency). Moreover, documents themselves are linked via hyperlinks and citations, forming a large graph that users implicitly navigate while browsing. We refer to the resulting setting as *open-domain multimodal document retrieval* (OMDR): given a query, the system must return a small ranked set of relevant components from this large, noisy, and interlinked graph, often requiring multihop and multimodal exploration (Yun et al., 2025; Trivedi et al., 2023; Faysse et al., 2024).

Given these intricate characteristics of OMDR, representing a document collection as a graph has emerged as a powerful paradigm for capturing the multi-granularity and interconnectedness of multimodal evidence (Yun et al., 2025). It shows the pros of preserving the structural depen-

dencies and navigational scaffolds inherent in webpages, which is required when navigating heterogeneous components. The most recent work introduces the *layered component graph*, which organizes components together with their constituent subcomponents (sentences, table rows, and image objects) (Yun et al., 2025). In this formulation, *navigational edges* encode relations among components (e.g., hyperlinks, same-section adjacency), while *hierarchical edges* connect each component to its subcomponents. By jointly modeling these edge types across layers, a retriever can traverse component-to-component paths for multihop exploration and move up/down the hierarchy to operate at the appropriate granularity.

While graph-based structures provide a rich representation of multimodal evidence, existing retrieval algorithms often struggle to fully exploit this potential due to operational rigidity. In particular, (a) traversal is typically driven by a single, hop-agnostic embedding-based scoring rule and (b) executed with a largely pre-specified procedure, limiting dynamic error correction. Figure 1 highlights these failures. In Figure 1(a), it follows a superficially related textual cue and retrieves an irrelevant snippet, failing to ground on the crucial visual evidence. In Figure 1(b), it issues an under-specified follow-up (“such battles”) and gets stuck in a dead end, rather than adapting its trajectory after the failure. As a result, once the retriever follows a spurious edge or reaches a dead end, errors propagate across hops and degrade final retrieval quality (Trivedi et al., 2023; Asai et al., 2024). Moreover, these methods lack a principled mechanism for deciding *when* expensive reasoning is warranted, leading to under-reasoning on ambiguous hops or over-spending computation across a trajectory.

To overcome these limitations, we argue that a retriever must evolve from a static path-follower into an adaptive decision-maker that navigates the graph through a sequential reasoning process. Concretely, OMDR is naturally stateful: as evidence accumulates, the information need shifts, and failures reveal which interpretations, routes, or strategies are unproductive. This suggests casting traversal as a *sequential decision process* over an evolving information state, where each step chooses (i) what to ask next (subquery), (ii) how to retrieve (tool/strategy), and (iii) where to move (edge type and granularity) conditioned on the current evidence. Achieving this requires addressing three coupled challenges. First, edge-following is not merely similarity matching; it often requires high-level reasoning to judge whether a candidate node will lead to the final answer under the current context. Second, the retriever must adapt to evolving context by refining hypotheses and subqueries, and by recovering from dead ends using failure signals rather than adhering to a fixed, pre-defined plan. Third, it must balance accuracy and efficiency: while LLM-based reasoning can improve retrieval precision, it intro-

duces substantial overhead, so the system must decide economically when to escalate from lightweight matching to intensive reasoning.

To address these needs, we propose **FAILURE IS FEEDBACK** (FiF). We formalize OMDR as a finite-horizon *information-state MDP*, where the state is a structured memory that records accumulated evidence together with the history of attempted subqueries, strategies, and explicit success/failure outcomes. This formulation turns graph traversal into an *economically-rational* agentic workflow: at each hop, an orchestrator dynamically decides *what to ask*, *how to retrieve*, and *where to move* given the current information state, rather than executing a rigid traversal recipe. To realize cost-sensitive control, FiF maintains a portfolio of strategies across an accuracy–efficiency spectrum, starting from low-cost vector matching and escalating to higher-cost LLM reasoning only when a hop is ambiguous or an attempt fails. Finally, to make multihop navigation resilient in noisy open-domain graphs, FiF introduces *history-aware backtracking*: unlike standard backtracking that simply reverts the state, our approach piggybacks on failure traces to re-anchor the search to a more promising prior context, revise subsequent subqueries, and avoid repeating previously failed routing patterns.

In summary, we make three primary contributions:

1. We formulate the OMDR problem as a sequential decision process with economic rationality. We redefine OMDR as an information-state MDP, operationalizing it through an LLM-enabled agentic workflow that treats retrieval strategy as a dynamic choice.
2. We propose dynamic cost-aware strategy escalation. We introduce a novel mechanism that maintains a portfolio of strategies across an accuracy-efficiency spectrum. Our orchestrator avoids over-reasoning by starting with low-cost vector matching and only escalating to high-cost LLM reasoning when a hop is identified as ambiguous or follows a recorded failure.
3. We propose history-aware backtracking for resilient navigation, which converts failed traversals into constructive feedback. By piggybacking on failure traces, the orchestrator re-anchors its search to prior contexts while revising its subqueries and escalating its strategy, enhancing both robustness and efficiency.

## 2. Related Work

### 2.1. Multimodal Retrieval Methods

Early multimodal retrievers were largely *TextRAG*-style: they transformed multimodal components into textual surrogates via OCR, captioning, or serialization, enabling mature text retrieval pipelines but inevitably discarding vision-specific cues (Yang et al., 2023; Yu et al., 2023; Luo et al., 2023a; Park et al., 2025). More recently, *VisRAG*-style

110 pipelines unify modalities by rasterizing documents into  
 111 page- or region-level screenshots and embedding all content in a single *visual* space (Yu et al., 2024; Faysse et al.,  
 112 2024; Cho et al., 2025). However, they suffer from two  
 113 key limitations: (i) *fixed granularity*, where large screenshots dilute query-relevant signals with irrelevant context,  
 114 and (ii) *limited multihop reasoning*, treating pages independently without exploiting structural links (Chen et al.,  
 115 2024; Zhong et al., 2025).

116 Closest to our work, LILAC represents a multimodal document corpus as a layered graph structure and performs  
 117 structure-aware retrieval through vector-embedding-based  
 118 graph traversal (Yun et al., 2025). It builds a component  
 119 graph linking coarse nodes (paragraphs, tables, images)  
 120 and fine-grained subcomponents, using edges to represent  
 121 both hierarchical containment and navigational relations.  
 122 At query time, LILAC performs an edge-wise beam search  
 123 driven by late-interaction scores between subqueries and  
 124 nodes. While effective, this pipeline operates under a rigid,  
 125 pre-defined plan: it relies on a uniform similarity metric for  
 126 traversal, overlooking hop-specific semantics, and executes  
 127 a linear expansion strategy.

## 2.2. Graph Retrieval Methods

128 Graph-based retrieval has been extensively studied in  
 129 knowledge graph QA, where systems traverse graphs  
 130 curated with *typed* and *semantically meaningful* relations  
 131 (Asai et al., 2019; Fang et al., 2020; Sun et al., 2023;  
 132 Luo et al., 2023b; Xu et al., 2024). However, multimodal  
 133 document graphs differ fundamentally: edges are primarily  
 134 *navigational* (e.g., hyperlinks) rather than semantic predicates.  
 135 Consequently, a retriever cannot treat traversal as  
 136 simple path-finding over valid facts; it must perform online  
 137 interpretation to resolve what a navigational link implies  
 138 for the current query context (Yun et al., 2025). Thus, KG  
 139 methods struggle with the adaptive capabilities to navigate  
 140 large-scale, non-edge-labeled graphs where semantic reso-  
 141 lution is needed (Yang et al., 2023; Hu et al., 2025).  
 142

## 2.3. Agentic Retrieval Methods

143 A growing line of work treats retrieval as an iterative de-  
 144 cision process interleaved with reasoning, rather than a  
 145 single-shot nearest-neighbor lookup. IRCOT shows that  
 146 multi-step questions benefit from repeatedly generating in-  
 147 termediate sub-questions and retrieving evidence for each  
 148 step (Trivedi et al., 2023). REACT formalizes a general  
 149 reasoning-and-acting loop, motivating RAG controllers  
 150 that plan retrieval actions based on intermediate obser-  
 151 vations (Yao et al., 2022).

152 Despite this, most operate on *flat* indices, lacking the struc-  
 153 tural awareness to navigate inter-document links. Further-  
 154 more, their error correction is typically limited to query  
 155 rewriting rather than history-aware backtracking. While  
 156

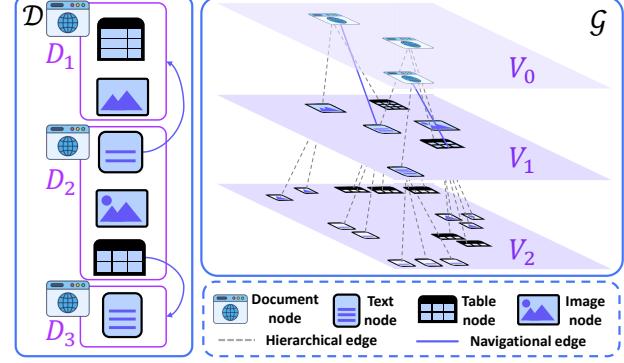


Figure 2. Visualization of an example corpus  $\mathcal{D}$  and its corresponding layered component graph  $\mathcal{G}$ .

systems like DOC-REACT and MARA apply agents to multimodal documents, they target single-document or small-scale contexts (Wu et al., 2025b;a). They do not address the open-domain challenge of traversing vast, interconnected graphs where utilizing failure feedback is critical for routing optimization.

## 3. Multimodal Document Retrieval

We study *open-domain multimodal document retrieval* to find relevant components from a large multimodal corpus on a natural language query. In this study, we follow the setup of graph-based retrieval approaches (Yun et al., 2025), which have shown promising performances over existing naive approaches.

### 3.1. Problem Setting

**Corpus and components.** For the source of retrieval, a corpus  $\mathcal{D} = \{D_1, \dots, D_{k_{doc}}\}$  is a set of documents. Each document  $D_j$  includes an ordered list of multi-modal components  $D_j = [C_{j,1}, \dots, C_{j,k_j}]$ , where the global component pool is defined as  $\mathcal{C} = \bigcup_{j=1}^{k_{doc}} \{C_{j,1}, \dots, C_{j,k_j}\}$ . The modality of each component  $C \in \mathcal{C}$  can be a paragraph  $P$ , a table  $T$ , or an image  $I$  as shown in Figure 2.

**Navigational links.** We assume a link signal  $\mathcal{L}$  capturing navigational associations (e.g., hyperlinks and cross-document pointers), modeled as  $\mathcal{L} : \mathcal{C} \rightarrow \mathcal{D}$ .

**Multimodal retrieval task.** Given  $Q$ ,  $\mathcal{D}$ , and  $\mathcal{L}$ , the retriever ranks components in  $\mathcal{C}$  and returns  $\mathcal{C}_R = [C_{R_1}, \dots, C_{R_n}]$ . Let  $\mathcal{C}_{gt}(Q) = \{C_{gt_1}, \dots, C_{gt_r}\}$  be the ground-truth relevant set; the goal is to rank its elements in  $\mathcal{R}$ , ideally near the top.

### 3.2. Layered Component Graph

We incorporate a layered component graph (Yun et al., 2025) to support an effective *coarse-to-fine* retrieval across documents and their components. To efficiently retrieve relevant documents and evidence for a given query, open-domain retrieval requires to plan both which documents

165 to visit and which components to read. Thus, we adopt  
 166 the three-layered component graph to effectively represent  
 167 documents, components, subcomponents, and their com-  
 168 plex relations. Figure 2 shows an example graph  $\mathcal{G}$ .  
 169

170 **Nodes and Layers.** Let  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$  denote the graph. Us-  
 171 ing the definitions from Section 3, we construct a three-  
 172 layered hierarchy  $\mathcal{V} = V_0 \cup V_1 \cup V_2$ :

- 173 • *Layer 0 (Documents)*  $V_0$
- 174 • *Layer 1 (Components)*  $V_1$ : paragraphs/tables/images
- 175 • *Layer 2 (Subcomponents)*  $V_2$ : sentences, table rows, or  
 176 visual objects

177 For nodes in  $V_1 \cup V_2$ , we save the raw multimodal content.  
 178 For nodes in  $V_0$ , we save a short textual summary of each  
 179 document to avoid long inputs while preserving high-level  
 180 semantics for global routing.

181 Different from LILAC (Yun et al., 2025), which uses two  
 182 layers only with components and subcomponents, we add  
 183 an explicit *Document Layer* to store a concise *textual sum-  
 184 mary* per document node. Adding document layer enables  
 185 early pruning before descending to fine-grained evidence  
 186 and can significantly improve the retrieval performance.  
 187

188 **Hierarchical Edges.** These edges represent the “contains”  
 189 relationship, allowing the agent to drill down from coarse  
 190 to fine granularity.

- 191 • Edges  $(D_j, C_{j,i})$  for all components  $C_{j,i} \in D_j$ .
- 192 • Edges linking a component to its extracted subcom-  
 193 ponents.

194 **Navigational Edges.** These edges capture explicit naviga-  
 195 tional paths across the corpus, allowing the retriever to trans-  
 196 sition between different document contexts based on the  
 197 link signal  $\mathcal{L}$ . Using the link signal  $\mathcal{L}$ , we generate an edge  
 198  $(C, D_k)$  if  $\mathcal{L}(C) = D_k$ .

199 Figure 2 illustrates the two edge types: dotted lines ex-  
 200 press hierarchical edges, and blue lines express navi-  
 201 gational edges.  
 202

## 203 4. FAILURE IS FEEDBACK

205 We propose FAILURE IS FEEDBACK (FiF), an LLM-driven  
 206 agentic retriever that traverses the layered component graph  
 207  $\mathcal{G}$ . Instead of executing a static, pre-defined traversal plan,  
 208 FiF formulates retrieval as a *sequential decision process*:  
 209 an Orchestrator iteratively chooses to (i) TRAVERSE from  
 210 selected anchors under a strategy that explicitly trades off  
 211 accuracy and cost, (ii) PLAN by revising or expanding  
 212 subqueries as the information need evolves, or (iii) STOP  
 213 and invoke a final RERANKER over all accumulated candi-  
 214 dates. A structured traversal memory records retrieved evi-  
 215 dence together with explicit success/failure outcomes, op-  
 216 erationalizing our core principle that *failure is feedback*: the  
 217 Orchestrator escalates to stronger (but costlier) reasoning  
 218 for traversal when lightweight hops are ambiguous or fail,  
 219

and performs history-aware backtracking by re-anchoring to more promising prior contexts while avoiding previously failed routing patterns. We start by formalizing the sequential decision process, then explain the details for each agent that comprise the process.

### 4.1. Retrieval as a Sequential Decision Process

We formulate open-domain multimodal retrieval as a *sequential decision process* over the layered component graph  $\mathcal{G}$ . Given a query  $Q$ , the agent iteratively traverses  $\mathcal{G}$  to output an ordered list of relevant components. We formalize the process as  $\langle \mathcal{S}, \mathcal{A}, \mathcal{T} \rangle$ , where the state is the agent’s information state  $s_t$ ; executing  $a_t$  yields observation  $o_t$ , which is integrated into  $s_{t+1}$  by the transition.

**State ( $\mathcal{S}$ ).** We represent the information state as a structured *memory*  $s_t = M_t$ .  $M_t$  is basically a trajectory log capturing decisions and outcomes:  $M_t = (Q, \mathcal{Q}_t, H_t)$ .

- *Original Query (Q):* The user’s initial input.
- *Subquery List ( $\mathcal{Q}_t$ ):* a list of decomposed query serving as a retrieval plan.
- *Action History ( $H_t$ ):* an ordered sequence  $[h_1, \dots, h_t]$ . Each record  $h_k$  serves as a log of each action.

**Action ( $\mathcal{A}$ ).** An action  $a_t \in \mathcal{A}$  is a structured tool call selected by the orchestrator given  $s_t$ :

- TRAVERSE( $\mathcal{D}_{anc}, q_t, \tau_t$ ) executes one retrieval hop for subquery  $q_t$  with strategy  $\tau_t$  from an anchor document set  $\mathcal{D}_{anc}$ .
- PLAN( $M_t$ ) generates updated subqueries needed to solve  $Q$ , consulting both  $\mathcal{Q}_t$  and  $H_t$ .
- STOP terminates the overall process. It applies a final RERANK module to all components stored in memory  $M_t$  and returns a ranked list  $\mathcal{C}_R$  based on their relevance to the original query  $Q$ .

**Observation ( $o_t$ ).**  $o_t$  contains the *new* outputs produced by the action at step  $t$ . For TRAVERSE, the observation is the traversed documents/components and if the traversal was successful or not. For PLAN, the observation is the updated list of subqueries.

**Transition ( $\mathcal{T}$ ).** The transition process updates the state  $s_t$  using the observation  $o_t$  to generate the next state  $s_{t+1}$ . Specifically, it appends the observation to the action history  $H_t$ . It appends the newly generated subqueries to  $\mathcal{Q}_t$ .

### 4.2. Action Design

In this subsection, we detail the LLM-powered agents that implement each action in our sequential decision process defined in § 4.1. Crucially, FiF’s actions are designed to make two principles *operational*: (i) *economically-rational control* via a portfolio of traversal strategies that can be escalated on demand, and (ii) *failure-as-feedback* via explicit success/failure signals and history-aware re-anchoring for

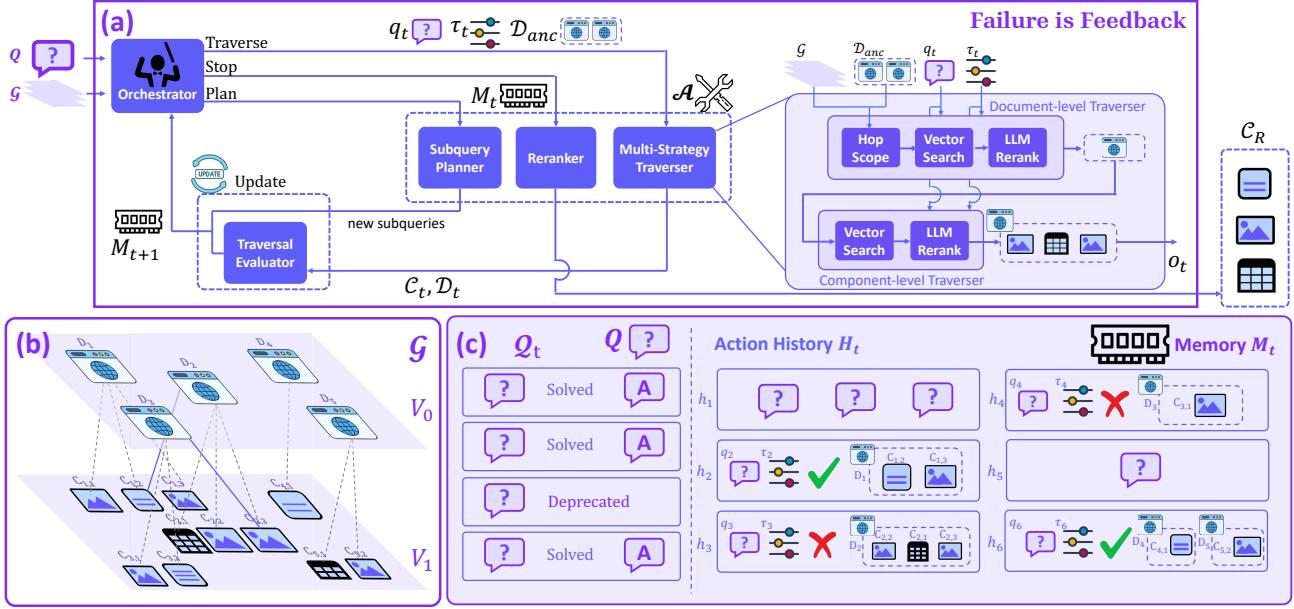


Figure 3. Overview of FAILURE IS FEEDBACK. (a) High-level orchestration loop and action interactions, labeled using the notation from the problem formulation (§ 4.1). (b) An example layered component graph  $\mathcal{G}$ . Note that only the document and component nodes are shown for brevity. (c) An example memory  $M_t$  of a traversal over  $\mathcal{G}$ .

robust multihop navigation in noisy document graphs.

**(1) Multi-strategy Traverser.** It executes the  $\text{TRAVERSE}(\mathcal{D}_{anc}, q_t, \tau_t)$  action on the layered graph  $\mathcal{G}$ , aiming to find subquery  $q_t$ -relevant components and documents, anchoring from the documents  $\mathcal{D}_{anc}$ . It consists of two stages: (i) a *document-level traverser* that selects a small set of candidate documents using document-node summaries (Layer  $V_0$ ), and (ii) a *component-level traverser* that identifies  $q_t$ -relevant components among the candidates by searching their children nodes.

Crucially, the Traverser is a *configurable strategy engine*: the strategy tuple  $\tau_t$  specifies retrieval behaviors that trade off accuracy and efficiency. **Strategy tuple.**  $\tau_t$  controls traversal along three dimensions:

- **Hop Scope (Global vs. Local).** *Local Hop* restricts candidate documents to the direct neighbors of the anchor document set  $\mathcal{D}_{anc}$  via the children’s navigation edges; *Global Hop* considers the full document corpus to escape local neighborhoods when evidence is dispersed. It performs a dense retrieval over layer  $V_0$ ’s document summaries to search the most query-relevant document set.
- **Vector Scoring Granularity.** It configures the layer at which vector similarity is computed. Intuitively, coarse-grained subqueries are often best matched at the document/component level, while fine-grained subqueries benefit from scoring at the subcomponent level. The document-level traverser uses the granularities of  $g \in \{0, 1, 2\}$  and the component-level uses  $g \in \{1, 2\}$ . Let

$\text{Desc}_g(v)$  be descendants of  $v$  at Layer  $V_g$  (including  $v$  if already in  $V_g$ ). We score a node  $v$  by

$$\text{SCORE}_{\text{vec}}(q_t, v; g) = \max_{u \in \text{Desc}_g(v)} \text{sim}(q_t, x(u))$$

- **LLM-Reasoning.** This option specifies whether to use LLM-based reasoning to accurately rerank components with  $q_t$  beyond vector scores, supporting modes of  $\{\text{None}, \text{Component-only}, \text{Both}\}$ . To control LLM inference cost, we pre-filter top- $k$  candidates by  $\text{SCORE}_{\text{vec}}$  and then pass their contents to the LLM for reranking.

The traversal outputs a document set  $\mathcal{D}_t$  and a component set  $\mathcal{C}_t$ , which are recorded in the action history  $H_t$ .

**(2) Subquery Planner.** The Planner implements  $\text{PLAN}(M_t)$  and generates new subqueries to address the remaining information needs for answering  $Q$ , conditioning on the current subquery statuses  $\mathcal{Q}_t$  and the retrieved evidence stored in  $H_t$ . Invoking  $\text{PLAN}$  allows the agent to revise its plan using the newly acquired evidence in the cases where initial subquery list is suboptimal. Newly generated subqueries are appended to the subquery list  $\mathcal{Q}$  in memory  $M_t$ ; importantly, we keep earlier subqueries rather than overwriting them, so the system retains a trace of what was tried and where the plan went off track.

**(3) Traversal Evaluator.** The Evaluator produces part of the observation  $o_t$  of  $\text{TRAVERSE}$  by assessing whether the traversal outcome  $\mathcal{C}_t$  is useful for the target subquery  $q_t$ . It provides two outputs. First, it judges whether the retrieved components can answer  $q_t$ ; this success/failure signal is

275 stored in the action history  $H_t$  together with the traversal  
 276 results. Second, it checks whether the retrieved content  
 277 can resolve any remaining items in the subquery list  
 278  $\mathcal{Q}_t$ , and if so, extracts tentative answers and updates  $\mathcal{Q}_t$  ac-  
 279 cordingly. These logged outcomes later guide backtracking  
 280 decisions, helping the Orchestrator distinguish promising  
 281 contexts from unproductive ones.

282 **(4) Reranker.** The Reranker is invoked by STOP to pro-  
 283 duce the final ranked list of components  $\mathcal{R}$  from the accu-  
 284 mulated memory  $M_t$ . Concretely, it aggregates all candi-  
 285 date components stored in memory and assigns each a final  
 286 relevance score with respect to the *original* query  $Q$ , then  
 287 returns  $\mathcal{C}_R$  by selecting the top- $k$  components.  
 288

### 289 4.3. Orchestrator

290 The Orchestrator is the central LLM-driven controller that  
 291 implements the policy over actions, selecting  $a_t$  given the  
 292 current information state  $M_t$ . At each iteration, it chooses  
 293 among TRAVERSE, PLAN, and STOP to maximize retrieval  
 294 accuracy while remaining efficient, treating *failure as feedback*.  
 295

296 Beyond selecting actions, the FiF Orchestrator actively  
 297 *manages plan errors*. Rather than using  $q_t$  as a direct copy  
 298 from the subquery list, it treats the list as a scaffold and  
 299 synthesizes a task-specific target by rewriting, refining, or  
 300 composing subqueries based on the current evidence and  
 301 unresolved constraints. In particular, it can use newly re-  
 302 trieval components to resolve missing information from  
 303 earlier subqueries, or fuse multiple planned subqueries into  
 304 a single sharper retrieval objective when they are interde-  
 305 pendent. When the subquery list becomes clearly unreliable  
 306 (e.g., overly underspecified, drifting, or repeatedly unpro-  
 307 ductive), the Orchestrator can invoke PLAN at any time to  
 308 regenerate a better set of subqueries conditioned on  $M_t$ ,  
 309 while preserving prior subqueries as a trace of what was  
 310 attempted.  
 311

312 The Orchestrator also enables *history-aware backtracking*  
 313 via *re-anchoring*. Instead of always hopping from the most  
 314 recent set of documents, it chooses an anchor document set  
 315  $\mathcal{D}_{anc}$  that best matches the newly formed  $q_t$ , potentially  
 316 restarting from an earlier successful context. To do so, it  
 317 consults the action history  $H_t$ , leveraging (i) summaries  
 318 of previously retrieved documents and (ii) the Traversal  
 319 Evaluator’s success/failure outcomes to identify the most  
 320 promising region to resume from and to avoid repeating  
 321 failed routing patterns. This re-anchoring makes multihop  
 322 navigation more resilient to dead ends and spurious cues.  
 323 In addition, the Orchestrator explicitly reasons about effi-  
 324 ciency through the strategy tuple: for each new  $q_t$ , it pre-  
 325 dictes an economical traversal configuration  $\tau_t$  to try first.  
 326 If such a hop fails, it can retry from the same anchor (or a  
 327 better re-anchored set) with an escalated, more accurate  
 328

329  $\tau_t$  (e.g., finer granularity, leveraging LLM reranking, or us-  
 330 ing global scope). This closed-loop control reduces failures  
 331 caused by insufficient reasoning while keeping computa-  
 332 tion focused where it is most needed.

## 5. Experiments

### 5.1. Experimental Setups

**Datasets and evaluation metrics.** We evaluate open-domain multimodal *component* retrieval and downstream QA on three benchmarks: MULTIMODALQA (Talmor et al., 2021), MMCoQA (Li et al., 2022), and WEBQA (Chang et al., 2022). Following LILAC (Yun et al., 2025), we use the URL-annotated setting to reconstruct realistic webpage-style corpora, parsing each page into multimodal components (paragraphs, tables, images). This yields MULTIMODALQA<sup>Doc</sup> (3,235 pages, avg.  $\sim$ 37 components), MMCoQA<sup>Doc</sup> (453 pages, avg.  $\sim$ 32 components), and WEBQA<sup>Doc</sup> (7,662 pages, avg.  $\sim$ 13 components). Consistent with prior work (Yun et al., 2025), we report retrieval Recall@ $k$  ( $R@k$ ,  $k \in \{1, 2, 5, 10\}$ ) and MRR@10:  $R@k$  checks whether at least one ground-truth component appears in the top- $k$  list, and MRR@10 captures the rank of the first relevant component. For end-to-end QA, we feed the top-10 retrieved components into the same multimodal LLM and report Exact Match (EM) and token-level F1.

**Compared methods.** We compare FiF with strong baselines spanning graph traversal, agentic retrieval, and single-shot indexing. We include LILAC (Yun et al., 2025), a layered-graph retriever designed for multi-hop scenarios, and IRCOT (Trivedi et al., 2023), an agentic retriever that interleaves retrieval with chain-of-thought reasoning. Since IRCOT was originally proposed for text-only corpora, we adapt it to our multimodal setting by (i) replacing its retriever with the same multimodal embedder used throughout our experiments and (ii) using the same multimodal LLM for reasoning and generation over multimodal components. For VisRAG approaches, we employ VISRAG-RET (Yu et al., 2024), which directly encodes document images via VLMs, and COLPALI (Faysse et al., 2024), which uses late-interaction multi-vector embeddings from document images. We also compare with NV-EMBED-v2 (Lee et al., 2024), a TextRAG baseline that embeds textualized components.

**Model configurations.** To ensure fair comparison, we standardize backbone models across all methods whenever applicable. We use MM-EMBED (Lin et al., 2024) as the unified multimodal embedder, and the OPENAI API (GPT-5) (Singh et al., 2025) with `reasoning_effort = low` as the multimodal LLM for all planning, reasoning, reranking, and generation steps.

330  
331 *Table 1.* Retrieval accuracy (Recall and MRR) of FiF and its competitors on three benchmarks. R@k, M@k indicates recall at  $k$  and  
MRR at  $k$ , respectively. The best score in each column is in **bold**.

Algorithm	MULTIMODALQA <sup>Doc</sup>					MMCoQA <sup>Doc</sup>					WEBQA <sup>Doc</sup>				
	R@1	R@2	R@5	R@10	M@10	R@1	R@2	R@5	R@10	M@10	R@1	R@2	R@5	R@10	M@10
NV-EMBED-V2	26.42	37.63	52.08	61.45	68.13	18.22	28.46	40.22	48.19	41.97	22.18	31.38	39.80	51.19	55.81
VisRAG	34.19	42.12	53.38	56.91	57.88	19.57	27.53	33.59	37.31	30.01	25.58	41.46	46.10	48.75	50.60
COLPALI	38.38	52.61	61.73	63.95	67.65	31.11	41.97	46.89	51.27	43.13	35.21	46.90	53.37	57.78	56.90
LILAC	33.59	50.59	65.13	72.23	79.12	25.25	38.13	51.02	60.86	53.36	32.67	45.35	57.77	64.23	77.87
IRCoT	39.56	55.09	69.87	74.97	82.24	43.03	53.69	62.70	64.62	62.84	40.85	56.73	68.78	73.72	83.15
FiF	<b>42.17</b>	<b>57.58</b>	<b>74.88</b>	<b>85.82</b>	<b>86.82</b>	<b>46.31</b>	<b>58.40</b>	<b>69.47</b>	<b>75.17</b>	<b>74.88</b>	<b>43.91</b>	<b>61.78</b>	<b>74.83</b>	<b>80.79</b>	<b>87.77</b>

339 *Table 2.* End-to-end QA accuracy (EM and F1) of FiF and its  
340 competitors for the three benchmarks. The best score in each column  
341 is in **bold**.

Algorithm	MULTIMODALQA <sup>Doc</sup>		MMCoQA <sup>Doc</sup>		WEBQA <sup>Doc</sup>	
	EM	F1	EM	F1	EM	F1
NVEMBED-V2	53.80	61.15	39.20	47.73	51.82	59.88
VisRAG	51.40	61.72	35.20	42.66	46.53	55.19
COLPALI	52.82	63.14	40.73	47.46	49.87	57.78
LILAC	57.78	63.98	42.57	51.14	53.38	60.71
IRCoT	60.32	68.09	47.21	57.71	57.03	65.88
FiF	<b>65.15</b>	<b>70.47</b>	<b>51.11</b>	<b>62.42</b>	<b>62.63</b>	<b>72.47</b>

## 5.2. Retrieval Accuracy Comparison

We evaluate open-domain multimodal *component* retrieval on MULTIMODALQA<sup>Doc</sup>, MMCoQA<sup>Doc</sup>, and WEBQA<sup>Doc</sup> using Recall@ $k$  ( $k \in \{1, 2, 5, 10\}$ ) and MRR@10, with results reported in Table 1. FiF achieves the best performance across all three benchmarks and all reported cutoffs, indicating its overall effectiveness. Averaged across datasets, FiF reaches average Recall@10 of 80.26 and average MRR@10 of 83.16, improving over LILAC +22.03% and +18.60%, respectively. Compared with the strongest agentic baseline IRCoT, FiF gains 12.88% Recall@10 and 9.31% MRR@10. The gap is even larger against single-shot embedding-based retrievers.

We analyze two interesting points. One is that the largest dataset-specific margin over LILAC appears on WEBQA<sup>Doc</sup> at higher cutoffs, suggesting that WEBQA<sup>Doc</sup> more frequently requires *escaping* local neighborhoods to reach dispersed evidence. This aligns with WEBQA<sup>Doc</sup>'s construction where ground-truth components are not necessarily adjacent or tightly coupled, making global navigation critical. In contrast, the performance gap over IRCoT is most pronounced on the more explicitly multi-hop benchmarks MULTIMODALQA<sup>Doc</sup> and MMCoQA<sup>Doc</sup>, where effectively leveraging the underlying link/structure signal (rather than pure global searching) is essential. Second, improvements are particularly strong at low- $k$ , indicating that FiF not only increases *coverage* of relevant components but also ranks them substantially earlier.

## 5.3. End-to-end QA Accuracy Comparison

We measure end-to-end QA performance by feeding the top-10 retrieved components into the same multimodal LLM generator for every method, and report EM and token-level F1 in Table 2. FiF is consistently the best-

performing method on all three datasets, achieving average EM/F1 of 59.63/68.45. Relative to LILAC, FiF improves EM by +16.37% and F1 by +16.79% on average, confirming that more reliable evidence discovery yields better grounded generation. Compared to IRCoT, FiF still provides a clear advantage of +8.71% EM and +7.14% F1, despite both methods being agentic. Dataset-wise, the gains are especially visible on MMCoQA<sup>Doc</sup> and WEBQA<sup>Doc</sup>, consistent with Table 1 where FiF yields substantially higher top- $k$  retrieval accuracy.

## 5.4. Algorithm Efficiency

Table 3 reports wall-clock retrieval time (with breakdown into LLM, vector search, and embedding), the number of LLM calls, token usage, and estimated API cost. As expected, single-shot retrievers (VisRAG, COLPALI, NV-EMBED-V2) are the fastest and incur no LLM API cost during retrieval. Among agentic methods, FiF shows a slightly lower average runtime than IRCoT (116,231 vs. 122,458 ms), which is the strongest agentic baseline: while FiF executes more reasoning steps (7.31–8.61 vs. 5.44–6.94 LLM calls), it uses fewer input tokens and achieves comparable or lower latency overall. Notably, FiF is substantially faster on WEBQA<sup>Doc</sup> (128,748 vs. 173,815 ms; –25.93%), suggesting that structure-aware navigation together with failure-aware re-anchoring reduces unproductive reasoning on large and sparsely connected corpora. The efficiency comes with a moderate increase in API usage compared to IRCoT (\$0.10–\$0.13 vs. \$0.075–\$0.108 per query), consistent with our higher retrieval/QA accuracy. Relative to LILAC, FiF is 5.20–6.71× slower in wall-clock time and incurs 9.64–16.60× higher API cost, quantifying the additional budget required by adaptive agentic control.

## 5.5. Ablation Study

Table 4 isolates the contribution of each major design component in FiF on MULTIMODALQA<sup>Doc</sup> and WEBQA<sup>Doc</sup>, reporting retrieval accuracy (R@10, MRR@10) alongside efficiency. We omit MMCoQA<sup>Doc</sup> from this analysis: because it is conversational, accumulated dialogue history can introduce confounding factors (e.g., varying context length and carryover information) that may blur the impact of individual retrieval modules. We run all ablations on a representative 10% subset of each dataset due to OpenAI

Table 3. Efficiency (Time, # LLM Calls and API Usage) comparison of FiF and its competitors for the three benchmarks.

Dataset	Algorithm	Time (ms)				# LLM Calls	API Usage		
		Total	LLM	Vector Search	Embedding		# Input Toks	# Output Toks	\$
MULTIMODALQA <sup>Doc</sup>	VISRAG	371	0	218	153	0.00	0	0	0.0000
	COLPALI	9,849	0	9,210	639	0.00	0	0	0.0000
	LILAC	19,528	15,943	3,153	432	1.00	1,165	1,119	0.0115
	IRCOT	95,744	95,140	119	484	5.44	41,943	3,296	0.0752
	FiF	114,591	113,642	538	411	7.92	48,863	4,785	0.1109
MMCoQA <sup>Doc</sup>	VISRAG	362	0	215	147	0.00	0	0	0.0000
	COLPALI	1,836	0	1,173	663	0.00	0	0	0.0000
	LILAC	20,244	16,879	2,977	388	1.00	1,160	1,041	0.0107
	IRCOT	97,816	97,220	116	479	5.61	52,286	3,360	0.0753
	FiF	105,355	104,630	366	359	7.31	53,149	4,151	0.1037
WEBQA <sup>Doc</sup>	VISRAG	386	0	225	161	0.00	0	0	0.0000
	COLPALI	7,919	0	7,298	621	0.00	0	0	0.0000
	LILAC	19,187	14,822	3,782	583	1.00	1,162	737	0.0077
	IRCOT	173,815	173,011	152	651	6.94	65,167	3,913	0.1082
	FiF	128,748	127,664	581	505	8.61	64,683	5,159	0.1278

Table 4. Ablation study analyzing retrieval accuracy and efficiency of different FiF variants.

Dataset	Variation	Retrieval Accuracy			Efficiency			
		R@10	MRR@10	Time (ms)	# LLM Calls	# Input Toks	# Output Toks	\$
MULTIMODALQA <sup>Doc</sup>	FAILURE IS FEEDBACK	85.82	86.82	117,651	7.97	49,153	4,812	0.1111
	w/o Backtracking Orchestration	79.26	84.31	211,951	12.53	78,864	6,113	0.1502
	w/o LLM Reasoning in traversal agent	78.04	83.97	168,780	7.64	45,079	4,203	0.0887
	w/o Global Hop	75.79	82.56	85,417	7.02	51,839	4,137	0.1039
	w/o Vector Granularity	83.16	85.43	116,918	8.28	55,281	5,179	0.1158
WEBQA <sup>Doc</sup>	w/o Subquery Planner	76.2	83.77	66,974	5.31	35,984	3,335	0.0765
	FAILURE IS FEEDBACK	81.91	89.24	126,567	8.55	63,971	5,103	0.1270
	w/o Backtracking Orchestration	77.45	87.62	228,934	13.77	89,244	7,214	0.1715
	w/o LLM Reasoning in traversal agent	78.92	88.82	135,274	7.57	50,501	4,275	0.0960
	w/o Global Hop	73.30	86.83	93,115	9.05	68,483	4,082	0.1247
	w/o Vector Granularity	79.21	87.77	132,972	8.72	66,201	5,319	0.1290
	w/o Subquery Planner	76.38	89.95	85,583	5.40	40,721	3,358	0.0820

API costs. (i) *History-aware backtracking orchestration*. Removing backtracking consistently hurts both *effectiveness* and *efficiency*: R@10 drops by 4.46–6.56, while runtime increases by ∼80% and LLM calls rise by 57–61%. This highlights that backtracking prevents wasted hops by adapting effort only when needed. It also improves accuracy by returning to more promising anchors and narrowing candidates to the right neighborhood, rather than repeatedly exploring uninformative branches. (ii) *LLM reasoning inside the traversal agent*. Disabling LLM reasoning during traversal substantially degrades retrieval and can even slow down the search: on MULTIMODALQA<sup>Doc</sup>, R@10 drops by 7.78 and runtime increases by 43%, despite lower per-step cost. We reason that additional replanning and extra iterations were triggered, as traversal is more likely to take ambiguous or unproductive hops without LLM reasoning. (iii) *Global hop*. Global hops are crucial for escaping local neighborhoods. When disabled, R@10 suffers the largest drop (10.03 on MULTIMODALQA<sup>Doc</sup>; 8.61 on WEBQA<sup>Doc</sup>), even though the variant becomes faster. This indicates that neighbor-based traversal alone is insufficient: some questions require jumping across distant regions of the corpus, including multi-hop paths and cases where relevant evidence is not directly linked. (iv) *Adaptive vector-search granularity*. Removing granularity adaptation yields consistent but smaller drops (about 2.66–2.70 R@10) with minimal efficiency change. (v) *Subquery plan-*

ner. removing replanning reduces LLM calls and cost substantially (e.g., −33% to −37% calls), but causes large recall drops (9.62 on MULTIMODALQA<sup>Doc</sup>; 5.53 on WEBQA<sup>Doc</sup>). Without the option to revise the plan, the Orchestrator is more likely to terminate early once progress stalls, which simultaneously lowers cost and recall. This underscores the importance of correcting early planning mistakes through evidence-conditioned replanning.

## 6. Conclusion

We propose FAILURE IS FEEDBACK (FiF), an agentic multimodal retriever that models graph traversal as a *sequential decision process* over an explicit information state. FiF maintains a structured memory of subqueries and action history, transforming failures into actionable signals rather than dead ends. Building on this, the Orchestrator makes retrieval *economically rational*: it dynamically chooses when to traverse, replan, or stop, and performs cost-aware strategy escalation—starting from low-cost matching and selectively pursuing accuracy over efficiency when ambiguity or failure justifies the added computation. Finally, FiF introduces history-aware backtracking via re-anchoring, using history to resume from more promising prior contexts. Extensive experiments on MULTIMODALQA, MMCoQA, and WEBQA demonstrate state-of-the-art accuracy on retrieval and downstream QA, validating the effectiveness of failure-aware traversal.

**References**

- 440 Asai, A., Hashimoto, K., Hajishirzi, H., Socher, R., and  
 441 Xiong, C. Learning to retrieve reasoning paths over  
 442 wikipedia graph for question answering. *arXiv preprint arXiv:1911.10470*, 2019.  
 443
- 444 Asai, A., Wu, Z., Wang, Y., Sil, A., and Hajishirzi, H. Self-  
 445 rag: Learning to retrieve, generate, and critique through  
 446 self-reflection. 2024.
- 447 Chang, Y., Narang, M., Suzuki, H., Cao, G., Gao, J., and  
 448 Bisk, Y. Webqa: Multihop and multimodal qa. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 16495–16504, 2022.
- 449 Chen, T., Wang, H., Chen, S., Yu, W., Ma, K., Zhao, X.,  
 450 Zhang, H., and Yu, D. Dense x retrieval: What retrieval  
 451 granularity should we use? In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 15159–15177, 2024.
- 452 Cho, J., Mahata, D., Irsoy, O., He, Y., and Bansal, M.  
 453 M3docrag: Multi-modal retrieval is what you need  
 454 for multi-page multi-document understanding. *arXiv preprint arXiv:2411.04952*, 2024.
- 455 Cho, J., Mahata, D., Irsoy, O., He, Y., and Bansal, M.  
 456 M3docvqa: Multi-modal multi-page multi-document un-  
 457 derstanding. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 6178–6188,  
 458 2025.
- 459 Fang, Y., Sun, S., Gan, Z., Pillai, R., Wang, S., and Liu,  
 460 J. Hierarchical graph network for multi-hop question  
 461 answering. In *Proceedings of the 2020 conference on empirical methods in natural language processing (EMNLP)*, pp. 8823–8838, 2020.
- 462 Faysse, M., Sibille, H., Wu, T., Omrani, B., Viaud, G.,  
 463 Hudelot, C., and Colombo, P. Colpali: Efficient doc-  
 464 ument retrieval with vision language models. *arXiv preprint arXiv:2407.01449*, 2024.
- 465 Hu, Y., Wang, X., Liu, Q., Xu, X., Fu, Q., Zhang, W., and  
 466 Zhu, L. Mmapg: A training-free framework for multi-  
 467 modal multi-hop question answering via adaptive plan-  
 468 ning graphs. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 14195–14211, 2025.
- 469 Lee, C., Roy, R., Xu, M., Raiman, J., Shoeybi, M., Catanzaro, B., and Ping, W. Nv-embed: Improved techniques  
 470 for training llms as generalist embedding models. *arXiv preprint arXiv:2405.17428*, 2024.
- 471 Li, Y., Li, W., and Nie, L. Mmcoqa: Conversational ques-  
 472 tion answering over text, tables, and images. In *Proceedings of the 60th Annual Meeting of the Association  
 473 for Computational Linguistics (Volume 1: Long Papers)*, pp. 4220–4231, 2022.
- 474 Lin, S.-C., Lee, C., Shoeybi, M., Lin, J., Catanzaro, B., and  
 475 Ping, W. Mm-embed: Universal multimodal retrieval  
 476 with multimodal llms. *arXiv preprint arXiv:2411.02571*, 2024.
- 477 Luo, H., Shen, Y., and Deng, Y. Unifying text, tables, and  
 478 images for multimodal question answering. *Association for Computational Linguistics*, 2023a.
- 479 Luo, L., Li, Y.-F., Haffari, G., and Pan, S. Reasoning on  
 480 graphs: Faithful and interpretable large language model  
 481 reasoning. *arXiv preprint arXiv:2310.01061*, 2023b.
- 482 Park, S., Yun, J., Lee, J., and Han, W.-S. Helios: Harmoniz-  
 483 ing early fusion, late fusion, and llm reasoning for multi-  
 484 granular table-text retrieval. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 32424–32444,  
 485 2025.
- 486 Singh, A. et al. Openai gpt-5 system card, December 2025.  
 487 Released alongside GPT-5 (Aug 2025). Accessed 2026-  
 488 01-27.
- 489 Sun, J., Xu, C., Tang, L., Wang, S., Lin, C., Gong, Y.,  
 490 Ni, L. M., Shum, H.-Y., and Guo, J. Think-on-graph:  
 491 Deep and responsible reasoning of large language model  
 492 on knowledge graph. *arXiv preprint arXiv:2307.07697*, 2023.
- 493 Talmor, A., Yoran, O., Catav, A., Lahav, D., Wang, Y., Asai,  
 494 A., Ilharco, G., Hajishirzi, H., and Berant, J. Multi-  
 495 modalqa: Complex question answering over text, tables  
 496 and images. *arXiv preprint arXiv:2104.06039*, 2021.
- 497 Trivedi, H., Balasubramanian, N., Khot, T., and Sabharwal,  
 498 A. Interleaving retrieval with chain-of-thought reasoning  
 499 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.
- 500 Wu, H., Zhai, H., Li, Y., Cai, H., Zhang, P., Zhang, Y.,  
 501 Wang, L., Wang, C., Hou, Y., Wang, S., et al. Mara:  
 502 A multimodal adaptive retrieval-augmented framework  
 503 for document question answering. In *Proceedings of the 33rd ACM International Conference on Multimedia*, pp.  
 504 4329–4338, 2025a.
- 505 Wu, J., Xia, Y., Yu, T., Chen, X., Harsha, S. S., Maharaj,  
 506 A. V., Zhang, R., Bursztn, V., Kim, S., Rossi, R. A.,  
 507 et al. Doc-react: Multi-page heterogeneous document  
 508 question-answering. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 67–78, 2025b.

- 495 Xu, Y., He, S., Chen, J., Wang, Z., Song, Y., Tong, H., Liu,  
496 G., Liu, K., and Zhao, J. Generate-on-graph: Treat llm as  
497 both agent and kg in incomplete knowledge graph ques-  
498 tion answering. *arXiv preprint arXiv:2404.14741*, 2024.  
499
- 500 Yang, Q., Chen, Q., Wang, W., Hu, B., and Zhang, M. En-  
501 hancing multi-modal multi-hop question answering via  
502 structured knowledge and unified retrieval-generation.  
503 In *Proceedings of the 31st ACM International Confer-  
504 ence on Multimedia*, pp. 5223–5234, 2023.
- 505 Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan,  
506 K. R., and Cao, Y. React: Synergizing reasoning and  
507 acting in language models. In *The eleventh international  
508 conference on learning representations*, 2022.
- 509 Yu, B., Fu, C., Yu, H., Huang, F., and Li, Y. Unified lan-  
510 guage representation for question answering over text,  
511 tables, and images. *arXiv preprint arXiv:2306.16762*,  
512 2023.
- 513
- 514 Yu, S., Tang, C., Xu, B., Cui, J., Ran, J., Yan, Y., Liu, Z.,  
515 Wang, S., Han, X., Liu, Z., et al. Visrag: Vision-based  
516 retrieval-augmented generation on multi-modality docu-  
517 ments. *arXiv preprint arXiv:2410.10594*, 2024.
- 518
- 519 Yun, J., Lee, D., and Han, W.-S. Lilac: Late interacting in  
520 layered component graph for open-domain multimodal  
521 multihop retrieval. Association for Computational Lin-  
522 guistics, 2025.
- 523
- 524 Zhong, Z., Liu, H., Cui, X., Zhang, X., and Qin, Z. Mix-  
525 of-granularity: Optimize the chunking granularity for  
526 retrieval-augmented generation. In *Proceedings of the  
527 31st International Conference on Computational Lin-  
528 guistics*, pp. 5756–5774, 2025.
- 529
- 530
- 531
- 532
- 533
- 534
- 535
- 536
- 537
- 538
- 539
- 540
- 541
- 542
- 543
- 544
- 545
- 546
- 547
- 548
- 549

**A. Limitations**

550  
 551 FiF incurs higher wall-clock latency and API cost than lightweight or fixed-call traversal methods such as LILAC. Al-  
 552 though our cost-aware strategy escalation mitigates unnecessary computation, multi-step orchestration and occasional  
 553 LLM-based reranking remain as bottlenecks. Our method assumes that each webpage/document is already parsed into a  
 554 clean set of multimodal components and that navigational signals are reliably extracted. While we focus on URL-annotated  
 555 corpora with structured navigation and component graphs, truly open-web deployment may involve noisier pages, weaker  
 556 link signals, dynamic content, and heterogeneous layouts. An important direction for future work is to develop retrieval  
 557 pipelines that are robust to diverse webpage structures and can jointly learn or adapt the parsing, graph construction, and  
 558 retrieval policy so that the approach generalizes more seamlessly to arbitrary web content.  
 559

**B. Model Details****(Multimodal) Large language models:**

- Open-AI ChatGPT5

**Text embedders**

- NV-Embed-v2: 7.85B parameters

**Cross-modal embedders:**

- ColPali: 3B parameters
- VisRAG: 3.43B parameters

**Multimodal embedders:**

- MM-Embed: 8.18B parameters

**C. Experimental Details****C.1. Hardware and Software Settings**

581 All experiments were conducted on a Linux server equipped with an Intel Xeon Gold 6230 CPU @ 2.10 GHz, 1 TB of  
 582 RAM, and four NVIDIA RTX A6000 GPUs, running Ubuntu 22.04.3 LTS.

**C.2. Implementation Details**

583 Our main hyperparameter is the vector-search shortlist size  $k$  used by the Multi-strategy Traverser at both the document-  
 584 level and component-level stages. Unless otherwise noted, we set  $k=30$  for all experiments.

**C.3. Benchmark Details**

590 MULTIMODALQA<sup>Doc</sup>: We use the extended version of MultimodalQA, following the augmentation procedure intro-  
 591 duced in M3DocRAG (Cho et al., 2024). It spans diverse document modalities (text, images, and tables) and is designed  
 592 to stress multi-hop reasoning over multiple documents. The evaluation split contains 2,441 questions grounded in 3235  
 593 webpages.

594 MMCQA<sup>Doc</sup>: We use an extended variant of MMCQA that moves beyond the original distractor-only setting to evaluate  
 595 conversational, multi-turn multimodal QA. The benchmark consists of coherent dialogue sessions in which later questions  
 596 depend on earlier context and require aggregating evidence across text, images, and tables. It includes 5,753 questions  
 597 grouped into 1,179 conversations, with a corpus of 218,285 text passages, 10,042 tables, and 57,058 images.

598 WEBQA<sup>Doc</sup>: WEBQA<sup>Doc</sup> is a Wikipedia-based multimodal QA benchmark with 4,966 questions over 7,662 documents.  
 599 Because the original answers are often verbose, we rewrite them into concise references using the ChatGPT-5 OpenAI API  
 600 with the prompt below.

605 Answer Concisification Prompt for WEBQA<sup>Doc</sup>  
 606  
 607 You are an assistant that extracts concise answers from an Original Answer.  
 608  
 609 Task:  
 610 Given a Question, its Question Category (Qcate), and an Original Answer, extract a  
 611 concise version of the answer.  
 612  
 613 Category hints:  
 614 - YesNo: respond only "Yes" or "No" matching the polarity of the Original Answer.  
 615 - text: return the minimal noun phrase/name that answers the question.  
 616 - choose: return only the chosen option or label.  
 617 - number: return the numeric value (and unit if present) without extra words.  
 618 - color: return the color term(s) only.  
 619 - shape: return the shape descriptor only.  
 620 - Others: follow the general concise rules below.  
 621  
 622 Rules:  
 623 - Output ONLY the concise answer text (no extra words, no labels, no punctuation-only  
 624 output).  
 625 - Keep the minimum span that directly answers the Question.  
 626 - Prefer a single word when possible.  
 627 - If the question asks what object/thing, output the object noun phrase only (e.g.,  
 628 fountain).  
 629 - If the question asks for a choice/comparison attribute (e.g., taller or shorter,  
 630 happy or upset, or similar), output only the chosen option word from the answer (e.  
 631 g., "taller", "upset").  
 632 - If the Original Answer is verbose by repeating or paraphrasing words/phrases already  
 633 present in the Question, do NOT copy those repeated Question words into the concise  
 634 answer; extract only the new, directly-question-answering information (If those  
 635 repeated words are necessary for answering the question, then you may include them)  
 .  
 636 - Preserve the original casing/pluralization as used in the Original Answer (e.g., "  
 637 Circles").  
 638 - Do not include locations, explanations, or surrounding context unless they are  
 639 required to uniquely answer the question.  
 640  
 641 Examples:  
 642 # Example 1  
 643 Question Category: YesNo  
 644 Question: Does a Minnetonka Rhododendron flower have petals in a cup shape?  
 645 Original Answer: No, a Minnetonka Rhododendron flower does not have petals in a cup  
 646 shape.  
 647 Concise Answer: No  
 648  
 649 # Example 2  
 650 Question Category: Others  
 651 Question: What water-related object is sitting in front of the Torre del Reloj?  
 652 Original Answer: A fountain is sitting in front of the Torre del Reloj.  
 653 Concise Answer: fountain  
 654  
 655 # Example 3  
 656 Question Category: choose  
 657 Question: Is the fence in front of The Glass House in Fulham taller or shorter than a  
 658 bicycle?  
 659 Original Answer: The fence in front of the building is taller than a typical bicycle.  
 660 Concise Answer: taller  
 661  
 662 # Example 4  
 663 Question Category: shape  
 664 Question: What shape is found 3 times on the front of the Archway in King Charles  
 665 Street?  
 666 Original Answer: Circles may be spotted three times on the face of the Archway on King

```

660     Charles Street.
661 Concise Answer: Circles
662
663 # Example 5
664 Question Category: choose
665 Question: Does the character in the work \"Beslotentuinfeest\" look happy or upset?
666 Original Answer: The character in the work \"Beslotentuinfeest\" looks upset.
667 Concise Answer: upset
668
669 # Example 6
670 Question Category: number
671 Question: How many more skis were used by Anders S\u00f6dergren during the 2010
672    Olympics than were used by Martin Rulsch during the 2020 Winter Youth Olympics?
673 Original Answer: Anders S\u00f6dergren used two more skis during the 2010 Olympics.
674 Concise Answer: two
675
676 Inputs:
677 Question Category (Qcate): {qcate}
678 Question: {question}
679 Original Answer: {answer}
680 Concise Answer:
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714

```

## D. Prompts Templates of FAILURE IS FEEDBACK

Orchestrator

```

684 You are a retrieval action-decider assistant.
685
686 Task:
687 Given a user query, a retrieval plan, serialized retrieval memory from prior steps, and
688 the titles of neighbor documents from the most recent step, decide the next
689 retrieval action.
690
691 You must output EXACTLY ONE action describing:
692 - what action to perform
693     - stop retrieving because there is plenty of information
694     - search for a new piece of information
695     - replan the subtasks because the plan is stale, misaligned, or incomplete
696
697 # Decision requirements:
698 1) Use ONLY provided context
699     - You MUST use ONLY information directly inferable from:
700         (a) the user query,
701         (b) the initial plan,
702         (c) the serialized retrieval memory,
703         (d) the neighbor docs list.
704     - Do NOT add facts, assumptions, background knowledge, or outside context.
705
706 2) No clarifying questions
707     - You are not allowed to ask the user for clarification.
708     - Make the best decision using only the given inputs.
709
710 3) Internal reasoning only (if needed)
711     - Perform your analysis internally (do NOT output reasoning), and perform it only in
712       cases where analysis is necessary.
713     - Output JSON only no explanations or extra text.
714
715 # Rule Regarding Next Retrieval Subtask:
716     - If the latest retrieval was marked "answerable" OR the previous action used "llm"

```

```

715 reasoning" for BOTH 'document_search_mode' and 'component_search_mode', you should
716 advance to the next unresolved subtask when selecting 'next_retrieval_subtask'.
717 - Otherwise, keep targeting the current subtask (or the earliest unresolved subtask
718 if none is explicitly active).
719 - When advancing, prefer the earliest unresolved subtask in the list; if none remain
720 unresolved, continue with the last subtask that still needs information or reuse
721 the most recent unresolved one.
722
723 # Rule Regarding Dynamic Cost-Aware Strategy Escalation:
724 - Treat retrieval configuration as a cost ladder:
725   neighbors < vector search < llm reasoning
726   and granularity: document < component < subcomponent.
727 - Default (cheapest) for a new/clean subtask:
728   document_search_mode="neighbors" (if Neighbor Docs likely contain missing info),
729   else document_search_mode="vector search";
730   component_search_mode="vector search";
731   vector_granularity="document";
732   anchor=null unless a clearly relevant prior candidate set exists.
733 - Escalate ONLY when justified by evidence in Serialized Retrieval Memory:
734   * current subtask's latest attempt is marked Failure/unanswerable, OR
735   * repeated near-misses (retrieved content is close but misses a constraint), OR
736   * the hop is ambiguous/underspecified per memory (multiple plausible targets / unclear referent), OR
737   * the prior attempt already used low-cost modes and did not progress.
738 - Escalation policy (monotonic within the same anchor unless you backtrack or replan):
739   :
740   1) neighbors + vector search + document granularity
741   2) vector search (global over docs) + vector search components + document
742   granularity
743   3) vector search with vector_granularity="component"
744   4) component_search_mode="llm reasoning" (keep doc mode as-is)
745   5) document_search_mode="llm reasoning" AND component_search_mode="llm reasoning" (highest cost)
746 - Scope widening rule (local -> global):
747   If Neighbor Docs are irrelevant OR the same neighborhood fails twice, switch
748   document_search_mode
749   from "neighbors" to "vector search" (or "llm reasoning" if ambiguity persists).
750
751 # Rule Regarding History-Aware Backtracking (Failure-is-Feedback):
752 - Use failure traces in Serialized Retrieval Memory as first-class signals (do not
753 ignore them).
754 - Define a "failed routing pattern" as repeating essentially the same route:
755   same subtask intent + same (or null) anchor + same document_search_mode/
756   component_search_mode/granularity,
757   where the memory marks Failure/unanswerable or shows no new evidence gained.
758 - If a failed routing pattern exists for the current subtask, you MUST change at
759   least one of:
760   (i) anchor, (ii) document_search_mode (scope), (iii) component_search_mode, (iv)
761   vector_granularity,
762   or (v) rewrite next_retrieval_subtask to add missing constraints / choose a
763   different target entity.
764 - Backtracking triggers:
765   * >=2 consecutive failures on the current subtask, OR
766   * the previous attempt already used ("llm reasoning", "llm reasoning") and still
767   failed, OR
768   * Neighbor Docs list is exhausted/irrelevant for the missing information.
769 - Backtracking procedure (re-anchoring):
770   1) Prefer re-anchoring to the most recent Success (or best partial/near-success)
771   step in memory:
772     set anchor to that step index so downstream retrieval starts from its candidate
773   documents.
774   2) If multiple candidate anchors exist, prefer the one whose retrieved evidence

```

```

770 best matches
771     the unresolved constraint(s) of the current subtask (as described in memory).
772 3) After re-anchoring, prefer document_search_mode="neighbors" if the missing info
773 is likely
774     adjacent to that anchor context; otherwise use "vector search"/"llm reasoning" to
775 escape the neighborhood.
776 - Backtracking MUST revise the next_retrieval_subtask to incorporate lessons from
777 failures:
778     explicitly negate dead ends, add missing constraints, or select the next-most-
779 likely candidate entity.
780 - Replan is reserved for plan-level problems:
781     choose "replan" only if subtasks are stale/misaligned, or if backtracking across
782 >=2 distinct anchors
783     still fails to make progress.
784
785 # Action Fields
786
787 (1) "stop"
788     - Meaning: Retrieval memory already contains sufficient components to answer the
789 original query.
790
791 Return schema:
792 {{{
793     "action": {{
794         "next_action": "stop",
795         "next_retrieval_subtask": null,
796         "document_search_mode": null,
797         "component_search_mode": null
798     }}
799 }}}
800
801 (2) "search"
802     - Meaning: More retrieval is needed to answer the original query.
803     - Fields to specify:
804
805     a) next_retrieval_subtask (string)
806         - Meaning: The next concrete retrieval task to run.
807         - Must be a short, actionable retrieval prompt (imperative verb + object).
808         - Must be consistent with the user query and the initial plan.
809         - Must be chosen to address what is still missing or failed, as indicated by the
810 serialized retrieval memory.
811         - De-contextualize: avoid pronouns; restate the entity/target explicitly.
812         - Generate a retrieval task that tries to solve either the first 'Target Subtask
813 to Solve', or combinations of them. Note that we are retrieving one of paragraph/
814 table/image components - generate a retrieval task that most suits this granularity
815 of retrieval.
816         - [IMPORTANT] If the 'User Query' is about finding a single entity that meets a
817 certain condition, and 'Target Subtask to Solve' indicates retrieving multiple
818 entities then choosing one that meets a certain condition,
819             try using common sense to pick the most likely entity that meets the condition
820             and generate a retrieval task for that entity only.
821             * If the memory suggests that finding one entity failed, then indicate it as
822             that such entity does not exist, and then try to pick the second most likely entity
823             that meets the condition.
824
825     b) document_search_mode (one of: "neighbors" | "vector search" | "llm reasoning")
826         - Meaning: How to select which documents to look at next (before selecting
827 components inside them).
828         - "neighbors":
829             Use when the neighbor documents list (given as '# Neighbor Docs') is likely
830 sufficient for the next_retrieval_subtask.

```

```

825     Choose this if the subtask is a direct continuation of the last step and the
826     missing info is likely in adjacent/related documents.
827     - "vector search":
828         Use when neighbor docs (given as '# Neighbor Docs') are likely insufficient
829         and you need to search across the entire corpus.
830         This performs a vector search over all documents (fast but potentially less
831         accurate).
832         Choose this when the subtask is relatively specific/unambiguous and broad
833         recall is needed.
834         - "llm reasoning":
835             Use when neighbor docs are likely insufficient AND the next_retrieval_subtask
836             is ambiguous, underspecified, or requires careful disambiguation.
837             This performs an initial vector search to shortlist documents, then uses LLM
838             reasoning to choose the best document(s) (slower but more accurate).
839
840             c) component_search_mode (one of: "vector search" | "llm reasoning")
841                 - Meaning: How to select components (paragraphs, tables, images) within the
842                 chosen documents.
843                 - "vector search":
844                     Use when the subtask target is fairly specific and likely to match component
845                     embeddings directly (fast but less accurate).
846                     - "llm reasoning":
847                         Use when the subtask is ambiguous, requires multi-constraint matching, or
848                         prior attempts show that pure vector search returns near-misses.
849                         This filters components using vector search first, then uses LLM reasoning to
850                         pick the most relevant components (slower but more accurate).
851
852             d) anchor (integer or null)
853                 - Meaning: If provided, reuse candidate documents from the memory step at this 0-
854                 based index.
855                 - How to use:
856                     * anchor refers to the memory's retrieval steps list index.
857                     * If anchor is provided, downstream retrieval will start from the candidate
858                     documents of that step's last attempt.
859                     - When to set:
860                         * Use when a previous step already surfaced a good candidate set that should be
861                         reused/refined.
862                         * Otherwise set to null.
863
864             e) vector_granularity (one of: "document" | "component" | "subcomponent")
865                 - Meaning: The granularity at which vector search should be applied for the next
866                 retrieval step.
867                 - How to choose:
868                     * "document": do vector search over documents, then pick components within them
869                     (default).
870                     * "component": do vector search directly over components.
871                     * "subcomponent": do vector search at a finer granularity (e.g., sentences/
872                     snippets) when component-level recall is too coarse.
873
874             Return schema:
875             {{ "action": {
876                 "next_action": "search",
877                 "next_retrieval_subtask": str,
878                 "document_search_mode": "neighbors" | "vector search" | "llm reasoning",
879                 "component_search_mode": "vector search" | "llm reasoning",
880                 "anchor": int | null,
881                 "vector_granularity": "document" | "component" | "subcomponent"
882             }
883         }
884
885             (3) "replan"
886                 - Meaning: The current retrieval plan is inadequate or misaligned; produce a
887

```

```

880     refreshed set of subtasks before continuing.
881 - Choose this when: target subtasks are exhausted, clearly off-target, missing
882     necessary steps, or memory shows repeated failures that imply the plan is wrong.
883 - No retrieval is executed in this step; the system will run the replan tool using
884     the existing context.

885     Return schema:
886 {{{
887     "action": {{
888         "next_action": "replan",
889         "next_retrieval_subtask": null,
890         "document_search_mode": null,
891         "component_search_mode": null
892     }}
893 }}}

894     Output format (strict):
895     Return ONLY valid JSON matching exactly one of the following schemas (no markdown, no
896     extra text):
897 {{{
898     "action": {{
899         "next_action": "stop",
900         "next_retrieval_subtask": null,
901         "document_search_mode": null,
902         "component_search_mode": null
903     }}
904 }}}

905     {{{
906         "action": {{
907             "next_action": "search",
908             "next_retrieval_subtask": str,
909             "document_search_mode": "neighbors" | "vector search" | "llm reasoning",
910             "component_search_mode": "vector search" | "llm reasoning",
911             "anchor": int | null
912         }}
913     }}}

914     {{{
915         "action": {{
916             "next_action": "replan",
917             "next_retrieval_subtask": null,
918             "document_search_mode": null,
919             "component_search_mode": null
920         }}
921     }}}

922     Inputs:
923     # User Query
924     {query}

925     # Split Retrieval Subtasks (answerability status and answers also)
926     {serialized_subtasks}

927     # Serialized Retrieval Memory (what has been tried + what succeeded/failed + what is
928     # still missing)
929     {serialized_memory}

930     # Neighbor Docs (titles of documents neighboring the last retrieved documents)
931     {neighbor_docs}

932     Output:
933
934

```

```

935 Document-level Traverser
936
937 You are a retrieval selection assistant.
938
939 Task:
940 Given (1) an original user query, (2) a subtask query, and (3) a list of candidate
941 documents, select which candidate documents should be retrieved next.
942 Vector granularity (document | component | subcomponent) is provided to signal the
943 intended search granularity; prioritize candidates that best match the subtask at
944 that granularity.
945
946 Objective:
947 Select up to {max_results} candidate documents that are most directly useful for
948 fulfilling BOTH:
949 - the Original Query, and
950 - the Subtask Query (the immediate retrieval goal).
951
952 Hard constraints:
953 1) Use ONLY provided context
954   - You MUST use ONLY information directly inferable from:
955     (a) the Original Query,
956     (b) the Subtask Query,
957     (c) the Candidate documents list.
958   - Do NOT add facts, assumptions, background knowledge, or external context.
959   - Do NOT introduce any documents that are not in the candidate list.
960
961 2) No clarifying questions
962   - You are not allowed to ask the user for clarification.
963   - Make the best selection using only the given inputs.
964
965 3) Validity + exact matching
966   - Indices must be valid 0-based indices into the candidate list.
967   - Filenames must exactly match a filename from the candidate list.
968
969 4) Internal reasoning only
970   - Perform analysis internally (do NOT output reasoning).
971   - Output JSON only no explanations or extra text.
972
973 Scoring rules (for "score"):
974   - Assign a relevance score in [0.0, 1.0] for each selected document.
975   - Scores should reflect *direct usefulness* for retrieval to answer BOTH queries:
976     - 1.0: highly likely to contain the needed information/evidence for the subtask while
977       staying aligned with the original query
978     - 0.5: partially relevant (may help, but incomplete or tangential)
979     - 0.0: clearly irrelevant
980   - Prefer documents that strongly match the Subtask Query, but do not pick documents
981     that obviously diverge from the Original Query's scope/constraints.
982
983 Selection & ordering rules:
984   - Output ONLY up to {max_results} selections (or fewer if fewer are relevant).
985   - Do NOT include candidates outside the top selections.
986   - Sort selections by descending relevance score (highest first).
987   - If unsure, pick the most likely candidates based on the queries.
988   - Never return an empty selection unless nothing is relevant at all (i.e., all
989     candidates are clearly irrelevant).
990
991 Output format (strict):
992 Return ONLY valid JSON matching exactly this schema (no markdown, no extra text):
993 {{ "selection": [
994   {
995     "index": int,           // 0-based index matching the candidate list
996     "filename": string,    // exact filename from the candidate list
997   }
998 ]}}

```

```

990         "score": float           // optional relevance score (0.0-1.0)
991     }
992   ]
993 }
994
995 Inputs:
996 # Original Query
997 {original_query}
998
999 # Subtask Query
{subtask_query}
1000
1001 # Vector granularity
{vector_granularity}
1002
1003 # Candidate documents (0-based indices):
1004 {candidates}
1005
1006 # Max results
{max_results}
1007
1008 Output:
1009
1010
1011 Component-level Traverser

```

```

1012 You are a component selection assistant for retrieval.
1013
1014 Task:
1015 Given (1) an original user query, (2) a subtask query, and (3) a list of candidate
1016 components, select which components should be kept for the next step.
1017 Vector granularity (document | component | subcomponent) is provided to signal the
1018 intended search granularity; prioritize components that best match the subtask at
1019 that granularity.
1020
1021 Objective:
1022 Select up to {max_results} candidate components that are most directly useful for
1023 fulfilling BOTH:
1024 - the Original Query, and
1025 - the Subtask Query (the immediate retrieval goal).
1026
1027 Hard constraints:
1028 1) Use ONLY provided context
1029   - You MUST use ONLY information directly inferable from:
1030     (a) the Original Query,
1031     (b) the Subtask Query,
1032     (c) the Candidate components list.
1033   - Do NOT add facts, assumptions, background knowledge, or external context.
1034   - Do NOT invent or introduce any components that are not in the candidate list.
1035
1036 2) No clarifying questions
1037   - You are not allowed to ask the user for clarification.
1038   - Make the best selection using only the given inputs.
1039
1040 3) Validity + exact matching
1041   - Indices must be valid 0-based indices into the candidate list.
1042   - Filenames and component_ids must exactly match a candidate entry.
1043
1044 4) Internal reasoning only
1045   - Perform analysis internally (do NOT output reasoning).
1046   - Output JSON only no explanations or extra text.
1047
1048 Scoring rules (for "score"):
```

```

1045
1046     - Assign a relevance score in [0.0, 1.0] for each selected component.
1047     - Scores should reflect *direct usefulness* for retrieval to answer BOTH queries:
1048         - 1.0: highly likely to contain the needed information/evidence for the subtask while
1049             staying aligned with the original query
1050         - 0.5: partially relevant (may help, but incomplete or tangential)
1051         - 0.0: clearly irrelevant
1052     - Prefer components that strongly match the Subtask Query, but do not pick components
1053         that obviously diverge from the Original Query's scope/constraints.
1054     - If useful information may be distributed across multiple components, include multiple
1055         complementary components when within {max_results}.

1056 Selection & ordering rules:
1057     - Output ONLY up to {max_results} selections (or fewer if fewer are relevant).
1058     - Do NOT include candidates outside the selected set.
1059     - Sort selections by descending relevance score (highest first).
1060     - If unsure, pick the most likely candidates based on the queries.
1061     - Avoid empty outputs unless nothing is relevant at all (i.e., all candidates are
1062         clearly irrelevant).

1063 Output format (strict):
1064     Return ONLY valid JSON matching exactly this schema (no markdown, no extra text):
1065     {{{
1066         "selection": [
1067             {
1068                 "index": int,           // 0-based index matching the candidate list
1069                 "filename": string,    // exact filename from the candidate list
1070                 "component_id": string, // exact component_id from the candidate list
1071                 "score": float          // optional relevance score (0.0-1.0)
1072             }
1073         ]
1074     }}}

1075 Inputs:
1076     # Original Query
1077     {original_query}

1078     # Subtask Query
1079     {subtask_query}

1080     # Vector granularity
1081     {vector_granularity}

1082     # Candidate components (0-based indices):
1083     {candidates}

1084     # Max results
1085     {max_results}

1086 Output:
1087

```

## Subquery Planner

```

1088
1089
1090
1091 You are a retrieval-plan revision assistant.
1092
1093 Task:
1094 Given (1) an original user query and (2) serialized retrieval memory from a failed
1095 attempt, produce 1-2 concrete retrieval tasks (strings) that should be attempted
1096 next to gather the missing information needed to answer the original query.
1097
1098 Rules:
1099 1) Task count

```

```

1100      - If the remaining gap is small and single-scope, output exactly 1 task.
1101      - Otherwise output 2 tasks.
1102
1103 2) Task wording
1104      - Each task must be a short, **actionable retrieval prompt** (imperative verb +
1105        object).
1106      - Do not put reasoning (logical derivation) as a task. Such reasoning should be done
1107        along with corresponding retrieval.
1108      - Keep tasks focused: ideally one clear information need per task.
1109
1110 3) Use ONLY provided inputs
1111      - You MUST use ONLY information directly inferable from:
1112          (a) the Original Query, and
1113          (b) the Retrieval Memory.
1114      - Do NOT add facts, assumptions, background, or likely details not present or
1115        directly implied by the inputs.
1116      - Do NOT rely on outside knowledge.
1117
1118 4) Entity & noun-phrase coverage
1119      - Every named entity and key noun phrase from the Original Query that is still
1120        needed must appear at least once across the tasks (you may distribute them).
1121      - Use the Retrieval Memory to identify which entities/noun phrases are missing,
1122        underspecified, or unresolved.
1123
1124 5) De-contextualize
1125      - Replace pronouns and implicit references so each task is understandable standalone
1126        .
1127      - Avoid it/they/this; restate the referenced entity.
1128      - If the Retrieval Memory introduces placeholders or intermediate identifiers,
1129        restate them explicitly.
1130
1131 6) Constraint distribution
1132      - Spread constraints logically across tasks instead of cramming everything into one
1133        task.
1134      - Prefer separating: identification/disambiguation vs. evidence gathering vs.
1135        attribute extraction.
1136
1137 7) Dependency ordering
1138      - If one task's output is needed for another, put the prerequisite first.
1139      - Express the dependency explicitly in the task description using (i), (ii), (iii),
1140        etc.
1141      - When a later task depends on earlier results, explicitly reference them by
1142        replacing (i) with the specific entity name once known (i.e., keep the "(i)"  

1143        placeholder until resolved).
1144
1145
1146 Inputs:
1147 # Original Query
1148 {original_query}
1149
1150 # Retrieval Memory
1151 {memory}
1152
1153 Output:
1154

```

1155 Traversal Evaluator

1156

1157 You are a retrieval evaluation assistant.

1158

1159 Task:

1160 Given an Original Query, a target Subtask Query, a list of all Subtasks (with their  
1161 current statuses, if any), and the Retrieved components, decide whether retrieval  
1162 succeeded for the target Subtask Query and which subtasks are answerable using the  
1163 current components.

1164

1165 Objective:

1166 Evaluate whether the retrieved components contain enough clearly relevant evidence to  
1167 answer the target Subtask Query (or to progress directly to answering it), while  
1168 remaining consistent with the Original Query. Also identify any other subtasks that  
1169 are answerable from the same retrieved components.

1170 Hard constraints:

1171 1) Use ONLY provided context

1172 - You MUST use ONLY information directly inferable from:  
1173 (a) the Original Query,  
1174 (b) the target Subtask Query,  
1175 (c) the Subtasks list, and  
1176 (d) the Retrieved components list.

1177 - Do NOT add facts, assumptions, background knowledge, or external context.  
1178 - Do NOT assume missing details (including what unseen documents might contain).

1179

1180 2) No clarifying questions

1181 - You are not allowed to ask the user for clarification.  
1182 - Make the best determination using only the given inputs.

1183

1184 3) Internal reasoning only

1185 - Perform analysis internally (do NOT output reasoning).  
1186 - Output JSON only no explanations or extra text.

1187

1188 # Decision rules for 'status':

1189 - Output "answerable" if at least one retrieved component appears:  
1190 (i) clearly relevant to the given '# Subtask Query', OR  
1191 (ii) directly answers the '# Original query', if combined with the '# Subtask status  
1192 ', OR  
1193 (iii) directly answers any of the '# Subtask query' that are marked as "not  
1194 answerable" or "unknown" in the '# Subtask status' list, , if combined with the '#  
1195 Subtask status'.  
1196 - Otherwise output "not answerable".

1197

1198 # Updated subtasks (independent of target status):

1199 Using the '#Retrieved components' and '# Subtask status', scan the provided Subtask  
1200 status list and identify which subtasks are answerable now.

1201 - "updated\_subtasks" MUST include ONLY the unanswerable subtasks from the '# Subtask  
1202 status' that have become answerable using the current Retrieved components +  
1203 subtask status.

1204 - For each included subtask that are not answerable:  
1205 - If it is answerable using 'Retrieved components + subtask status', then  
1206 - Provide its 1-based index from the provided Subtask status list.  
1207 - Set status = "answerable".  
1208 - Provide a concise answer drawn ONLY from the Retrieved components + subtask  
1209 status.  
1210 - Else skip it.  
1211 - If no subtasks are answerable, output an empty list [].

1212

1213 Output format (strict):

1214 Return ONLY valid JSON matching exactly this schema (no markdown, no extra text):

1215

1216

```

1210
1211 {{ "status": "answerable" | "not answerable",
1212 "updated_subtasks": [
1213   {
1214     "index": int,           // 1-based index of the subtask from the provided
1215     "list"
1216     "status": "answerable",
1217     "answer": string       // required; answer derived only from Retrieved
1218     components
1219   }
1220 ]
1221 }
1222
1223 Inputs:
1224 # Original Query
1225 {original_query}
1226
1227 # Subtask Query
1228 {subtask_query}
1229
1230 # Retrieved components (0-based indices):
1231 {candidates}
1232
1233 # Subtask status (1-based index; include current status if any)
1234 {subtasks}
1235
1236 Output:

```

## Reranker

```

1237 You are a reranking assistant for retrieval.
1238
1239 Task:
1240 Given a user query and a list of candidate components, select and rank the TOP-{top_k}
1241 candidates by how directly useful they are for answering the user query.
1242
1243 Hard constraints:
1244 1) Use only provided context
1245   - You MUST use ONLY information directly inferable from:
1246     (a) the user query, and
1247     (b) the candidate components list.
1248   - Do NOT add facts, assumptions, background knowledge, or external context.
1249   - Do NOT complete missing details with parametric knowledge.
1250
1251 2) No clarifying questions
1252   - You are not allowed to ask the user for clarification.
1253   - Make the best ranking using only the given inputs.
1254
1255 3) Internal reasoning only
1256   - Perform an internal step-by-step analysis before finalizing scores (do NOT output
1257     the reasoning).
1258   - During reasoning, consider semantic overlap, specificity to the query, and
1259     usefulness for answering.
1260
1261 Scoring rules:
1262 - Assign a relevance score in [0.0, 1.0] to selected candidates.
1263 - Scores should reflect *direct* usefulness for answering the query:
1264   - 1.0: highly likely to contain the needed answer or the most relevant evidence
1265   - 0.5: partially relevant or tangentially useful
1266   - 0.0: clearly irrelevant
1267 - Relevant information may be distributed across multiple components, such as
1268   components that are needed to resolve a query

```

```

1265 - Including identifying, disambiguating, or mapping implicit entities (e.g., aliases,
1266 acronyms, related identifiers) that are not explicitly stated in the query but are
1267 required to answer it.
1268 - Give appropriate credit to partially query-relevant candidates that could
1269 contribute necessary pieces of the answer, even if they are not sufficient on their
1270 own.
1271 - If multiple candidates seem similarly relevant, prefer the ones that more directly
1272 match the query's key entities/constraints.
1273
1274 Selection & ordering rules:
1275 - Output ONLY the TOP-{top_k} candidates by relevance (or fewer if fewer than {top_k}
1276 candidates are provided).
1277 - Do NOT include candidates outside the top-{top_k}, even if they are mildly relevant.
1278 - Sort the output by descending score (highest first).
1279 - Indices are 0-based and must match the candidate list.
1280
1281 Output format (strict):
1282 Return ONLY valid JSON matching exactly this schema (no markdown, no extra text):
1283 {{{
1284   "ranking": [
1285     {{
1286       "index": int,           // 0-based index matching the candidate list
1287       "filename": string,    // exact filename from the candidate object
1288       "component_id": string, // exact component_id from the candidate object
1289       "score": float         // relevance score (0.0-1.0)
1290     }}
1291   ]
1292 }}}
1293
1294 Example:
1295
1296 Input:
1297 User Query: "How does the payment processing component handle errors?"
1298 Top-K: 2
1299 Candidates:
1300 0: {"filename": "billing.json", "component_id": "po_0"}
1301 1: {"filename": "auth.json", "component_id": "t_1"}
1302 2: {"filename": "billing.json", "component_id": "i_10"}
1303
1304 Output:
1305 {{{
1306   "ranking": [
1307     {"index": 0, "filename": "billing.json", "component_id": "po_0", "score": 0.93},
1308     {"index": 2, "filename": "billing.json", "component_id": "i_10", "score": 0.71}
1309   ]
1310 }}}
1311
1312 REMINDER:
1313 Your objective is to read the user query, reason internally about each candidate's
1314 relevance using ONLY the provided inputs, select the TOP-{top_k} candidates, assign
1315 scores, sort them by descending score, and output only the defined JSON.
1316
1317 Inputs:
1318 # User Query
1319 {query}
1320
1321 # Candidate Components (indices are 0-based; refer to them as "index"):
1322 {candidates}
1323
1324 # Top-K
1325 {top_k}
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339

```

1320  
1321     Output:  
1322  
1323  
1324  
1325  
1326  
1327  
1328  
1329  
1330  
1331  
1332  
1333  
1334  
1335  
1336  
1337  
1338  
1339  
1340  
1341  
1342  
1343  
1344  
1345  
1346  
1347  
1348  
1349  
1350  
1351  
1352  
1353  
1354  
1355  
1356  
1357  
1358  
1359  
1360  
1361  
1362  
1363  
1364  
1365  
1366  
1367  
1368  
1369  
1370  
1371  
1372  
1373  
1374