A Survey of Knowledge Graph RAG: From Connected Facts to Supported Answers

Abstract

Retrieval-augmented generation (RAG) lets a language model answer from an external document collection. Ranking passages independently can miss facts connected across documents and give a narrow view of a whole collection. Knowledge graph RAG uses connections among entities, passages or summaries to find and combine evidence. This survey follows that evidence from graph construction through retrieval to generation, comparing what each stage preserves and what it loses. A running example of reports about research labs shows how an extracted fact can lose a qualification, how a search can follow an irrelevant connection and how source text can recover missing detail. Controlled comparisons find gains on multi-hop questions and on facts unseen during training, but also losses on single-hop questions and section summaries. Equal token budgets remove some apparent gains, and graph construction can cost far more than text indexing. A valid graph path, a claim supported by its sources and a correct answer require separate checks. The resulting research priorities are to predict when graph retrieval helps, measure losses by their effect on answers, preserve attribution through synthesis, and maintain facts and permissions as sources change.

1 Introduction

Large language models answer questions using knowledge learned during training, which can be outdated or omit private information. *Retrieval-augmented generation* (RAG) gives a model access to an external document collection: a retriever finds material relevant to a question, and a generator answers from that material [1]. Because the collection can be updated or replaced, the same model can answer from current or private sources without retraining. In a standard passage-based pipeline, documents are split into passages, each passage is ranked by its similarity to the question, and the top results are passed to the generator. The retrieved wording gives the model facts it may never have learned and gives the reader sources against which to check the answer. This pipeline works well when one passage both resembles the question and contains the needed fact.

Questions that connect documents or ask about a whole collection require more than a matching passage. Consider a hypothetical collection of reports about research labs whose facts no language model saw during training. Asked "Where is Cedar Lab located?", a retriever finds the passage that places Cedar Lab in Oslo. Asked "In which city is the lab that Mira founded?", it instead ranks a report about Mira and Cedar Lab highly, while the passage giving the lab's city never mentions Mira and can rank far lower. Both questions ask for the same city, but the second requires identifying the lab first. Independent passage ranking can miss this bridging fact [2, 3]. Asked "What do the labs in this collection study?", the retriever faces a different problem: the nearest passages describe only a few labs, so the answer leaves much of the collection out [4].

A graph makes the connections needed by these questions explicit. Following relations from a founder to a lab and from the lab to a city retrieves the location even when its passage never names the founder [3, 5, 6]. Grouping connected entities and summarizing each group in advance lets a generator cover a collection one group at a time [4]. We call this family *knowledge graph RAG* (KG RAG): systems that retrieve graph facts or use connections among entities, passages

or summaries to select evidence for a language model's response. KG RAG is a branch of RAG that changes how evidence is organized and found. It overlaps with knowledge graph question answering, where systems answer from curated graphs such as Freebase and Wikidata; methods that retrieve their paths or subgraphs for a language model belong to both fields [6–8]. Graphs built from documents also support longer outputs, including summaries and dialogue [4, 9]. Training graph facts into model parameters is a sibling approach; retrieval keeps those facts available externally [10].

KG RAG inherits two difficulties from passage-based RAG and adds two of its own. (i) *Finding related material can miss support.* Graph traversal reaches passages that similarity misses, but also reaches irrelevant neighbors; another lab in Oslo contributes nothing to Cedar Lab's location [11, 12]. (ii) *Retrieved evidence can go unused.* A generator can overlook an answer present in its input, and graph structure makes the presentation of that evidence especially consequential [13, 14]. The graph adds two further difficulties. (iii) *Extraction can change the facts.* A *triple* stores a head entity, relation and tail entity. Turning "Mira founded Cedar Lab to study forests" into (Mira, founded, Cedar Lab) drops the purpose; splitting a fact across triples can lose a shared condition [15]. (iv) *Connections are costly to maintain.* Graph construction adds work beyond text indexing [16], and changing a source can require revising both extracted facts and summaries derived from them [17, 18].

We organize the survey around three stages where evidence can be lost: construction determines what survives extraction (§3); retrieval determines which connections bring useful facts into view (§4); integration determines what the generator can read and cite (§5). Evaluation must separate gains from extra context and computation (§6), while reliability depends on preserving support as sources change or are attacked (§7). Earlier surveys organize the field by pipeline stage [19], graph domain [20], or the graph's role as knowledge or an index [21]. These accounts predate systematic graph–text comparisons [16, 22, 23] and therefore leave the costs and benefits of their design choices untested. Our contributions are:

- A formulation that separates the document collection, the graph built from it, the graph elements retrieved for a question and the evidence the generator reads (§2). It distinguishes three properties that can fail independently: a valid path through the graph, support from the source text and a correct answer.
- A comparison of construction, retrieval and integration methods by what each keeps or loses of the source documents, traced through one running example (§3–§5).
- A synthesis of controlled comparisons that locates when graphs help and at what cost (§6), together with the failures that incomplete, changing and adversarial collections cause and the open problems they leave (§7–§8).

2 Foundations

RAG retrieves evidence and generates a response from it. KG RAG changes this pipeline by making connections available to retrieval, then converting the selected graph elements into evidence the generator can read (§2.1). What those steps must preserve depends on whether the question asks for a single fact, connected facts or a view of the collection (§2.2).

2.1 Knowledge graph RAG

Passage-based RAG, called *text RAG* here, answers from external text in two steps. Given a question q and a collection of passages $\mathcal{D}$, a retriever scores each passage d with a matching function s and keeps the k highest-scoring passages as candidates C (1a). A generator with

parameters θ produces the response y from the question and those candidates (1b):

$$C = \underset{d \in \mathcal{D}}{\text{top-}k}\, s(q, d), \tag{1a}$$

$$y \sim p_\theta(y \mid q, C). \tag{1b}$$

Each passage is ranked independently. A passage stating Cedar Lab's city can therefore fall below passages about Mira even though it supplies the answer.

KG RAG adds a graph and separates what retrieval selects from what the generator reads. For a document collection, construction builds a graph G from $\mathcal{D}$ (2a); for an existing knowledge graph, G is supplied directly. Nodes can hold entities, passages, summaries or *propositions*, short statements that stand on their own. Edges record relations, mentions or links between summaries and their underlying material. Retrieval selects graph elements S, such as triples, paths or subgraphs, by matching the question and following connections (2b). Integration ϕ converts that selection into readable evidence E (2c), and the generator answers from the question and this evidence (2d):

$$G = \text{build}(\mathcal{D}), \tag{2a}$$

$$S = \text{retrieve}(q, G), \tag{2b}$$

$$E = \phi(q, S), \tag{2c}$$

$$y \sim p_\theta(y \mid q, E). \tag{2d}$$

Compared with Eq. (1), retrieval uses connections in G, and the generator reads E rather than the candidates as found. Retrieval can repeat, choosing each search from earlier results. A node or edge g can retain a *source link*, $\text{src}(g)$, to the text it came from; $\text{src}(S)$ denotes the sources linked to the selection. Those links let integration recover wording absent from the selected graph elements.

The integration step ϕ separates two roles a graph can play [21]. When the graph is a *knowledge carrier*, ϕ writes the selected triples or paths as text, and the relations themselves are the evidence. When it is an *index*, ϕ returns the source text $\text{src}(S)$ of the selected elements, and the edges serve only to find that text. Such edges may link passages to the pages and tables they come from (KGP [5]) or place passages under recursively written summaries (RAPTOR [24]). The two roles make different claims. An extracted relation asserts that a source states a fact, while an edge between two passages with similar wording asserts only that they are related. Many systems play both roles, using triples to find passages and giving the generator both [25, 26].

Systems also differ in when they build the graph. A *persistent* graph is built once for the whole collection before any question arrives, so its initial build cost is shared across questions [4, 27]. A *query-local* graph is built over the candidates C retrieved for one question and used to decide which of them to keep (PAGE-RAG [11]). A query-local graph can connect only what the first retrieval step found, whereas a persistent graph fixes in advance what any question can reach.

The hypothetical lab collection assumes facts unseen during training; a model that already knew the lab's identity could search for its location directly. Let s_1 be "Mira founded Cedar Lab to study forests" and s_2 be "Cedar Lab is located in Oslo." Construction extracts the founding and location triples and links each to its sentence. Suppose text retrieval for "In which city is the lab that Mira founded?" returns s_1 and other passages about Mira, but misses s_2. Graph retrieval instead starts at Mira, follows the founding edge to Cedar Lab and the location edge to Oslo, and returns this path as S. Written as text, it supplies "Oslo". Changing the question to "Why did Mira found Cedar Lab?" changes what counts as enough evidence. The founding triple omits the purpose, so integration must follow its source link and return s_1. Table 1 compares the two ways the graph helps: connecting the facts needed for the city and recovering the source wording needed for the purpose.

Table 1: A graph path supplies the lab's city; its source sentence also supplies the purpose of its founding.

Question q	Selection	Evidence E	Result
City of the lab Mira founded? (text RAG)	s_1 and other passages about Mira	No passage gives the city	City unsupported
City of the lab Mira founded? (KG RAG)	Path Mira → Cedar Lab → Oslo	The path written as text	"Oslo"
Why did Mira found Cedar Lab? (KG RAG)	The founding edge	The triple, without the purpose	Needs s_1 through the source link

The field arrived at this design space through three lines of work (Figure 2). Early systems retrieved triples or followed paths in curated knowledge graphs and gave the results to a generator as text [6, 28, 29]. Construction then expanded the available sources: passage graphs connected documents, summary trees organized long collections, and extracted entity graphs supported summaries of connected groups [4, 5, 24]. Retrieval added ways to rank those connections, from spreading relevance through a graph to trained scorers and learned search policies [3, 8, 30]. Integration changed how generators receive the result, using rewritten text, learned vectors or spans linked to sources [14, 31, 32].

The three lines of development yield choices that can be combined. A document graph can use propagation to retrieve source passages (HippoRAG 2 [33]), while a curated graph can use a trained triple scorer and a generator that needs no task-specific training (SubgraphRAG [8]). Figure 1 maps these choices onto the survey. We compare construction by what it preserves (§3), retrieval by how it selects evidence (§4), and integration by what reaches the generator (§5). Tables 2 and 3 locate systems along these three dimensions.

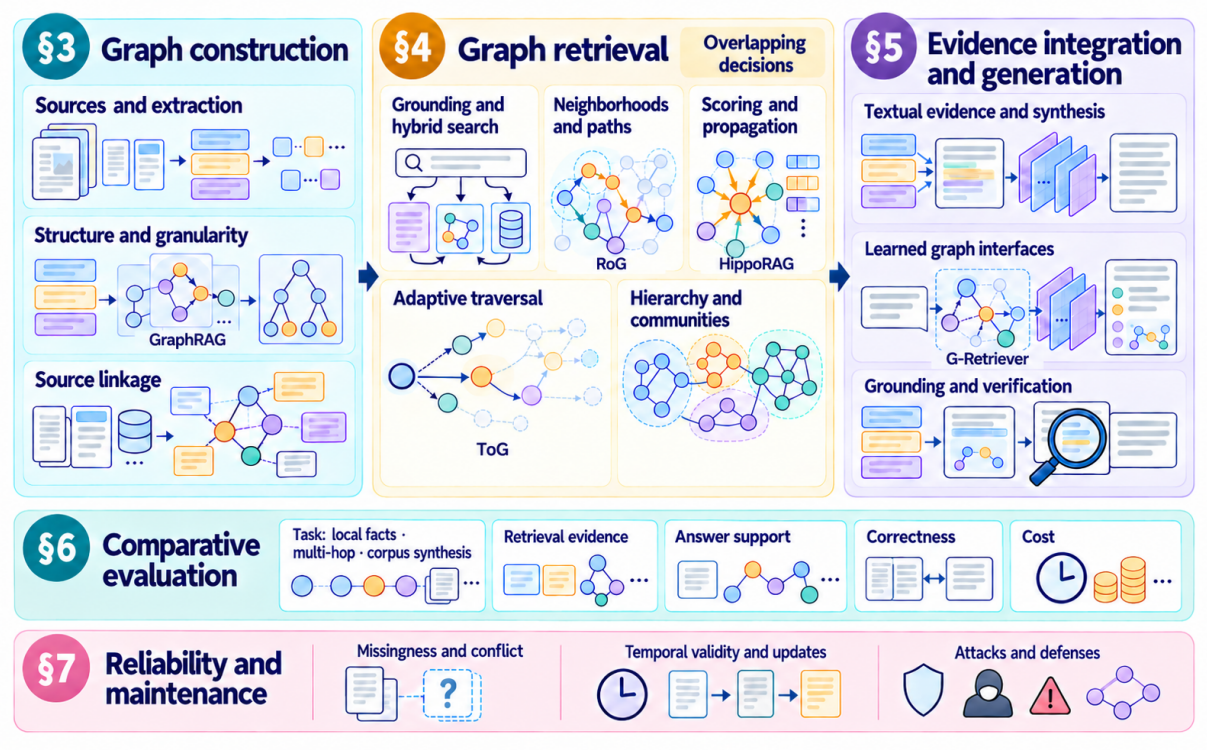


Figure 1: Evidence passes through construction (§3), retrieval (§4) and integration (§5); evaluation (§6) and reliability (§7) test what survives and at what cost [3, 4, 6, 29, 32].

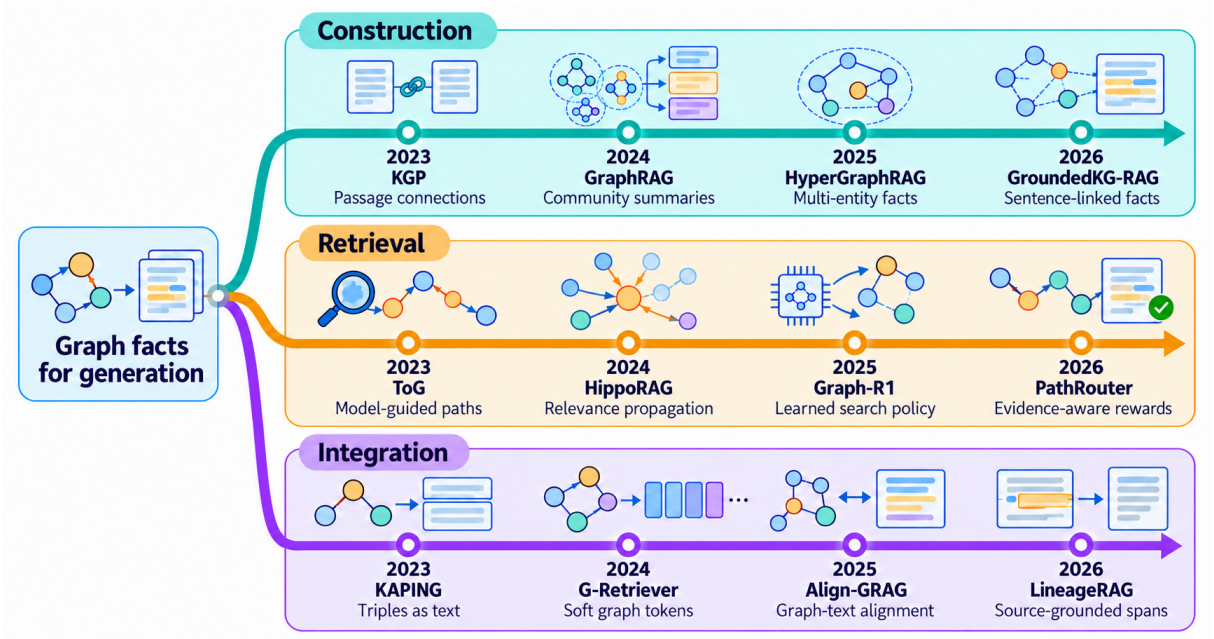


Figure 2: KG RAG evolved along three lines: construction expanded from passage connections to richer, source-linked facts (top, §3); retrieval moved from model-guided paths to propagation and learned control (middle, §4); integration added learned representations and source-grounded spans to textual facts (bottom, §5) [3–5, 14, 15, 28–30, 32, 34–36].

2.2 Evidence requirements

Each kind of question can go wrong in its own way. A single-fact answer fails when the system picks the wrong entity, a chain breaks when one link is missing, and a broad answer falls short when it leaves out much of the collection. Even evidence that passes these tests can mislead. A path can exist in the graph without any source that states it, and a well-sourced fact can be out of date.

Local factual grounding A single-fact question such as Cedar Lab's location can be answered from one triple or sentence, so retrieval mainly needs to locate the right entity. If the collection also mentions Cedar Labs, a software company, linking the question to the wrong node produces a confident wrong answer. Over a curated graph, the question's entities and relations have to be linked to stored ones (§4.1). Over documents, one passage usually suffices, and text retrieval answers such questions as well as graph retrieval or better [16].

Connected multi-hop evidence A *multi-hop question* requires combining facts, often from different documents [2]. Finding several passages about Mira still leaves the city unanswered if none supplies Cedar Lab's location. Each required fact can arrive as a compact triple or as a sentence or passage that keeps its context. A system can begin with triples and add source text when a triple leaves a gap [37]. The evidence must cover every required step rather than repeat the question's topic.

Corpus-wide synthesis A question about the whole collection, such as "What do the labs in this collection study?", has no single chain of facts behind it. Its answer has to draw on many documents, so it is judged by how much of the relevant material it covers as well as by whether each statement is accurate [4]. Coverage matters within a single long document as well, where an answer can depend on material spread across its sections [38]. Summaries prepared in advance

cannot anticipate every question, but written at several levels of abstraction they let a system answer such questions without reading every document [4, 24].

Validity, support and correctness An answer can follow the graph and still misrepresent a source. A path has *path validity* when every step is an edge of G. Suppose s_1 instead said that Mira visited Cedar Lab, but extraction recorded a founding. The path from Mira to Oslo remains valid, although the source never states the founding. Checking *source support* asks whether the linked source text, $\mathrm{src}(S)$, entails the response's claims. Even a supported answer can be outdated: if Cedar Lab moved after s_2 was written, "Oslo" faithfully repeats an old source but gives the wrong current city. *Answer correctness* asks whether the response agrees with the accepted answer. Constrained decoding enforces path validity [39]; evidence checks target support [11]; answering correctly also requires sources suited to the question's time and circumstances. Evaluation must distinguish these three properties (§6).

Table 2: Different graph contents G, retrieval selections S and evidence forms E support different tasks.

System	Graph G	Selection S	Evidence E and training	Task
GraphRAG[4]	Extracted entities and relations; nested communities	Community summaries at a chosen level; scored partial answers	Summary text; prompted API model	Global synthesis
RAPTOR[24]	Passages under recursive cluster summaries	Similarity search across levels, or tree traversal	Node text; API or open generator	Long-document QA
LightRAG[27]	Extracted entities and relations with text profiles	Local and global keywords; vector match; one-hop expansion	Profiles and source excerpts; prompted API model	Global synthesis
LinearRAG[40]	Entities, sentences and passages; mention links	Semantic bridging; PPR over passages and entities	Passage text; prompted API model	Multi-hop QA
RoG[6]	Freebase entities and typed relations	LLM relation plans; paths found by constrained BFS	Paths as text; fine-tuned planner and generator	Multi-hop KGQA
GNN-RAG[7]	Freebase entities and typed relations	GNN-scored candidates; shortest paths	Paths as text; fine-tuned open LLM	Multi-hop KGQA
SubgraphRAG [8]	Freebase entities and typed relations	MLP-scored triples with directional distances	Triples as text; untuned open or API generator	Multi-hop KGQA

QA: question answering; KGQA: knowledge graph question answering; LLM: large language model; GNN: graph neural network, which combines information over neighboring nodes; MLP: multilayer perceptron, a feed-forward neural scorer; PPR: personalized PageRank; BFS: breadth-first search, which explores one hop at a time. Vector matching compares learned numerical representations of text. API models are accessed through an application programming interface.

3 Graph construction

Construction determines which information from the collection $\mathcal{D}$ is available through the graph G. Extracted relations make facts searchable, while linked text preserves qualifications and summaries make groups readable at once. We separate three choices: where the graph's content comes from (§3.1), which units and connections it stores (§3.2), and how source links and quality checks make the result verifiable (§3.3).

3.1 Graph sources and extraction

A graph's facts are only as reliable as the step that produced them. Some sources already state their relations, and parsing can read them off directly. Prose needs a language model to choose which relations to keep and which names refer to the same entity. Each of these choices can drop a fact or invent one, so systems often keep the text searchable beside the graph.

Table 3: Hybrid and learned systems combine graph facts, source text and trained components across the same three stages.

System	Graph G	Selection S	Evidence E and training	Task
HippoRAG 2 [33]	Extracted phrases, synonym links and passage nodes	Vector triple matching; LLM filter; PPR over passages	Passage text; optimized filter prompt	Factual and multi-hop QA
ToG-2[41]	Wikidata relations; Wikipedia contexts	LLM relation pruning; dense context scores; iteration	Paths and contexts; untuned open or API LLM	KG and document QA
HyperGraph-RAG [15]	Entities and hyperedges for facts that join several entities	Vector search over entities and hyperedges; expansion; passages	Hyperedge facts and passages; prompted API model	Domain QA
G-Retriever[32]	Textual knowledge, scene and commonsense graphs	Connected subgraph balancing relevance prizes and edge costs	Graph text plus learned vector; trained GNN and projection; frozen generator	Graph QA
KAG[25]	Typed and extracted KGs linked to passages	Logical forms; graph and text retrieval; reflection	Facts, text and memory; general or fine-tuned LLM	Professional QA
SURGE[9]	Entities and relations linked to the dialogue	Context–triple similarity; sampled subgraphs	GNN-adjusted token vectors; generator trained with retriever	Grounded dialogue
KG2RAG[26]	Extracted triples linked to passages	Dense seed passages; BFS; spanning trees; reranking	Ordered source passages; prompted open LLM	Multi-hop QA

Abbreviations follow Table 2. A frozen generator's weights stay fixed; a projection maps graph vectors into its input space. Dense retrieval ranks by vector similarity.

Existing graphs and source-defined structure Curated graphs provide stored facts [42]; support tickets supply nested fields and ticket references [43]; code supplies inheritance and dependency relations recoverable by parsing [44]. Using this existing structure avoids asking a language model to infer it. On a Java codebase, language-model extraction indexed only 64% of the passages held by a text index, compared with 90% for deterministic parsing [44]. Parsing is therefore preferable where the required relations are explicit; prose such as the lab reports still requires extraction.

Schema guidance and open extraction Extraction from prose needs a rule for which entities and relations to keep. A *schema* specifies their allowed types. An extractor can follow a predefined schema and discard triples whose endpoints have the wrong types (OMD-GraphRAG [45]). For the lab collection, allowing (Person, founded, Organization) and (Organization, located in, City) captures the founding and location but leaves no place for the purpose. Open extraction instead keeps relations before assigning them to a schema. Written relation definitions can guide that later assignment (EDC [46]), while checks against an *ontology*, a specification of types and their constraints, can repair only entries that violate it (OAK [47]). This flexibility can preserve (Cedar Lab, studies, forests), at the cost of reconciling names such as "studies" and "researches". Both approaches still need source checks: (Mira, founded, Cedar Lab) fits the types even if the report says only that Mira visited. Schema compliance checks the form of a fact; source linkage makes its content checkable (§3.3).

Entity resolution Extracting separate passages can give one entity several nodes. *Entity resolution* merges those mentions so that facts about the same entity connect. Until "Cedar Lab" and "the Cedar Laboratory" share a node, the founding and location facts can remain disconnected. Similar-name clusters can be checked by a language model for duplicates (KGGen [48]); larger semantic units describing events and relations can also be merged across the corpus (GOSU [49]). The merge must preserve identity, not merely similarity. Combining Cedar Lab with Cedar Labs, the software company, joins facts that no source connects. Resolution therefore has to repair fragmented identities without creating false ones.

Graph–text construction Keeping a text index beside the graph gives retrieval another route to omitted facts. The purpose of Cedar Lab remains in its founding passage even if extraction drops it. Separate graph and vector searches can recover complementary evidence from the same documents [50]. Linking each extracted fact to its passage in both directions goes further: a planner can move directly from a graph lookup to the text that explains it (KAG [25]). The construction choice is whether graph and text remain independent search results or become connected evidence (§3.3).

3.2 Graph structure and granularity

Each kind of node and edge keeps part of a source and loses the rest. A triple is cheap to read but drops conditions such as a purpose or a date. A passage keeps those conditions but costs more to read, and a summary covers many passages at once but can leave details out. No single unit suits every question, so many graphs connect small units to the larger ones that contain them.

Entity graphs and community summaries A graph of extracted entities can support both local lookup and summaries of connected groups. For example, GraphRAG extracts entities, relations and claims, merges repeated names, then uses the Leiden clustering algorithm to form nested groups of connected entities [4]. A language model writes a *community summary* describing each group; summaries of smaller groups supply context when all their descriptions would not fit. The hierarchy lets a broad answer cover the collection without reading every passage. Its coverage still depends on extraction and merging: exact-name matching leaves "Cedar Lab" and "the Cedar Laboratory" apart, and a summary can omit a relation such as a shared funder. Community summaries suit recurring broad questions; linked passages retain the details needed for questions the summaries did not anticipate.

Passage and entity associations A graph can link passages through mentions without extracting the relations between their entities. Entity recognition identifies names in sentences and passages (LinearRAG [40]), while document structure links passages to containing pages and tables (KGP [5]). Similarity groups can also connect passages through shared concepts (ContextRAG [51]). Mira and Cedar Lab then connect because they occur together, not because an edge asserts a founding. The generator must read s_1 to distinguish founding from visiting. Relation-free graphs avoid relation-extraction errors by deferring that interpretation to the generator.

Detailed relations and broad coverage can share an index when extraction is selective. A graph-centrality ranking can identify passages for expensive relation extraction, while keyword links connect the remaining passages (KET-RAG [52]). The central skeleton supports explicit relation search, and the cheaper links keep the rest reachable. This design spends extraction effort where connectivity suggests it may help most.

Recursive summaries Grouping passages by similarity creates summaries without first extracting entities. For example, RAPTOR clusters passage vectors, lets a passage join several clusters, summarizes each cluster, and repeats the process on those summaries [24]. The resulting tree retains the original passages as leaves. Its parent groups similar text, whereas an entity-community summary groups entities joined by extracted relations. Neither grouping link asserts a source fact. For "How did Cedar Lab's research change over its first decade?", a passage summary can combine successive reports into a narrative while its leaves retain the original details.

Propositions and linked granularities Linking fine and coarse units lets retrieval choose how much context to return. A graph can place propositions beside the entities and passages they mention [53], link triples to their sentences and passages [37], or distinguish event descriptions and attributes as additional node types [54]. The proposition "Cedar Lab was founded to study forests" preserves the purpose that a founding triple omits. Its source passage can in turn preserve who made that claim and in what circumstances. Finer units reduce reading cost; links to coarser ones recover qualifications when the question needs them.

Higher-order and multimodal evidence Some facts need several participants to stay connected. "Mira directed Cedar Lab from 2015 to 2020" ties a person, a lab and a period together. Separate binary relations can leave the period's role ambiguous. A *hyperedge* connects all participants in one fact; storing its description as a separate node linked to each participant keeps the directorship and its period together (HyperGraphRAG [15]). The same design can keep interacting medical variables together [55]. Grouping passages by theme adds another kind of shared context (Cog-RAG [56]), while linking concepts to supporting text, table or image regions preserves evidence across modalities (HVM-GraphRAG [57]). These structures preserve different relationships, so the unit should match what the question must combine: participants in a fact, passages on a theme, or a concept and its visual evidence.

3.3 Source linkage and graph quality

Every extracted element claims that some source states it. A link back to that source lets a system or a reader check the claim. Such a check catches extraction errors but not a source that is itself wrong, so the graph also needs to record where sources disagree and how far each can be trusted. Graph quality depends on these records and on how many facts survive extraction, not on graph size.

Links back to the source A source link lets later stages recover wording that a graph element omits. Sentence anchors return the exact local context (GroundedKG-RAG [34]); page and section anchors return a larger region in which the generator must find the relevant sentence (PolyUQuest [58]). For the founding edge, that difference determines whether the generator receives the purpose clause directly or has to search a report page. Links also enable checks during construction: proposed entries can be verified against their cited passages before insertion (RAGA [59]). Such checks catch a misread founding but faithfully retain a misreported city, so source accuracy remains a separate problem (§7.1). A stricter design uses the graph only to navigate and answers from the linked text [60]. Source links then determine the evidence returned, rather than merely documenting extraction.

Quality beyond graph size Graph quality depends on which facts remain recoverable, not simply on the number of nodes and edges. A fact-retention test gives extractors generated articles with verified facts, retrieves a neighborhood for each fact, and asks a language model whether the subgraph entails it (KGGen's MINE-1 [48]). Stage-level diagnostics additionally check confidence, source coverage, redundancy and consistency [61]. Neither substitutes for recording disagreements or source reliability. A graph can attach reliability estimates informed by human feedback [62] and explicit support or contradiction links between claims [63]. For Cedar Lab's location, these records answer separate questions: which report asserted Oslo, which report disagrees, and how far each is trusted. Together they make extraction quality, disagreement and source reliability separately inspectable.

4 Graph retrieval

Retrieval selects graph elements S that supply the facts needed by the question. Connections can reveal a missing location passage, but also lead to unrelated labs through a shared city. We compare entry points and hybrid search (§4.1), shapes and pruning of the selection (§4.2), candidate ranking (§4.3), adaptive control (§4.4), and the choice of summary level (§4.5). These are decisions within a search, so one system can combine several of them.

4.1 Query grounding and hybrid search

Before it can follow any connection, retrieval has to decide where in the graph to start. Sometimes the question's wording matches nothing the graph stores. Many systems therefore run a text search alongside the graph. It adds candidates that graph search misses, and it gives the system a fallback when the question cannot be tied to the graph at all.

Entity and passage seeds The elements a search starts from are called *seeds*, and choosing the wrong ones can keep useful evidence out of reach. The two-hop question names Mira, so its natural seed is Mira's node. Seeds can be entities, relations or passages. One system extracts specific keywords from the question to match entity descriptions and broader keywords to match relationship descriptions, then gathers the matched elements' neighbors (LightRAG [27]). Another retrieves source passages by similarity first and enters the graph through the entities they contain, which helps when the question's wording resembles a passage more than any entity name (KG2RAG [26]). Over a curated graph, a language model can propose candidate entities, paths and graph queries, and tools keep only the proposals that resolve against the stored graph (ByoKG-RAG [42]). Whatever their source, seeds help only once they resolve to elements the graph actually stores.

Parallel retrieval and fusion Running graph and text retrieval side by side recovers passages missed by either route [50]. Keyword search can add another route, while typed relation sequences, called *meta-paths*, guide traversal through text-bearing nodes [64]. Combining the rankings requires comparable scores. One approach maps graph and similarity scores to percentile ranks; once normalized, different combination rules perform similarly on the tested retrieval benchmarks (PhaseGraph [65]). Combination can also happen while the graph expands, using semantic similarity to add nodes and neighboring scores to rerank them [66]. In conversation, entity matches and nearby turns supply candidates, semantic similarity ranks them, and dense retrieval fills the remaining slots [67]. Graph connections and text similarity can complement each other at candidate selection or ranking; normalization prevents an arbitrary score scale from deciding which route dominates.

Routing and query expansion Search can adapt its entry point before it begins traversing. Retrieved entity and relation terms can expand the query [68], while a router can choose a textual or relational retriever and revise that choice using feedback [69]. If a graph query fails, source passages can supply relations for a query-local evidence graph [70]. In a comparison on cyber-threat intelligence, repairing failed graph queries and combining graph with text retrieval were the most reliable architectures [71]. Text search therefore supplies both additional candidates and a recovery route when a question cannot be grounded in the stored graph.

4.2 Neighborhoods and relational paths

After choosing seeds, retrieval can retain a *neighborhood*, the elements within a bounded number of edges, or a *path*, an ordered chain of relations. A *subgraph* retains selected nodes and edges,

including paths that branch or meet. The shape determines what must be pruned: nearby elements in expansion, relation sequences in path search, or connections joining several facts.

Bounded expansion Expansion adds connected elements to the seeds, then filters them for the question. Shared extracted facts can connect initially retrieved passages to further passages; organizing the result as branching trees gives the generator a connected text sequence (KG2RAG [26]). Metadata links can instead supply candidate neighbors for combined semantic and keyword ranking (SAGE [72]). In SAGE's tests, skipping expansion helps when a structured query already specifies the edges to follow. A fixed hop limit controls distance, while the filter determines whether the added material supplies useful evidence. Expansion is most useful when the initial search leaves a connection unresolved.

Relation plans and instantiated paths Planning relations before retrieving entities separates a proposed search from stored evidence. For example, RoG learns relation sequences from paths connecting training questions to their answers, then uses breadth-first search restricted to each sequence to retrieve matching entity paths [6]. "Founded, then located in" yields Mira, Cedar Lab, Oslo only where the graph contains those edges. Planning can also operate over propositions without a language-model call at query time [73]. Linking a retrieved triple chain back to passages additionally recovers the wording it omits [74]. An uninstantiated plan supplies no evidence; matching it against stored edges makes the resulting chain checkable.

Scoring and pruning connections As paths branch, retrieval must discard connections that consume context without helping the answer. Distance-decayed flow [75], shortest-path context [76], and combined structural and semantic scores [77, 78] offer different ways to rank them. A structural score alone can favor a hub such as Oslo, through which many unrelated organizations connect. A query-local graph can instead score each connection by its sources, the specificity of its shared entity and how coherently it joins the passages, then retain connections within the generator's budget (PAGE-RAG [11]). The useful connection contributes a missing fact or its context; merely reaching another passage through the same city is not enough.

4.3 Neural scoring and graph propagation

Scoring many candidates together avoids a language-model call for every hop. A trained scorer learns ranking parameters ψ from supervision, giving a selection $S = \text{retrieve}_{\psi}(q, G)$. A fixed propagation rule instead spreads the question's relevance through existing connections. Both rank graph evidence, but only the scorer learns how to assign the scores.

Trained candidate and triple scoring Supervision can rank answer entities, triples or passages, and that choice determines what the generator receives. For example, GNN-RAG trains a graph neural network on question–answer pairs to rank candidate answer entities, then returns shortest paths to the highest-ranked candidates as text [7]. Adding paths proposed by a language model further improves answers, combining structural ranking with semantic planning. Scoring triples independently simplifies the ranking model and lets the generator read as many facts as its context permits (SubgraphRAG [8]). Passage scorers instead propagate question relevance across linked passages, sometimes across successive searches [79, 80]. These are trained retrieval methods; unlike the interfaces of Section 5.2, they leave the generator's text input unchanged.

A retriever can generate identifiers for its selection instead of scoring candidates separately. Encoding each relation as one token lets a small language model generate a subgraph's relation sequence [81]. In that study, the specialized retriever outperforms a much larger model, and single-token relations speed retrieval. Reading graph structure and text together also improves transfer

to unseen domains relative to graph-neural-network retrievers in another study [82]. Specialized representations can therefore reduce retrieval work without requiring a larger general-purpose model.

Nonparametric propagation A fixed graph walk can spread relevance without generating intermediate queries. Personalized PageRank repeatedly follows edges and restarts at the question's seeds; passages rank by the activation they receive (HippoRAG [3]). Cedar Lab can therefore carry relevance from Mira to the location passage even though it never mentions Mira. Adding passage nodes, filtering matched triples with a language model and retaining similarity-based passage weights improves the initial seeds (HippoRAG 2 [33]). The walk also applies to relation-free graphs [40], including designs that run without a graphics processor [83]. Hubs can divert activation to irrelevant neighbors, so query-aware variants weight edges by their endpoints' relevance and favor passages containing key facts [84, 85]. A fixed walk saves repeated reasoning calls; question-dependent seeds and weights keep its ranking focused.

4.4 Adaptive graph traversal

Some questions reveal what they lack only after a first search. Adaptive retrieval uses what the system has found so far to decide where to go next, whether to change direction and when to stop, and its controller can call any of the searches and scorers above.

Follow-up queries A first retrieval can expose a fact missing from the original query. Once s_1 identifies Cedar Lab, the controller can ask directly for its city. Combining such residual queries with the original question's propagation signal preserves what the first search established (EviReform [86]). A tentative answer supplies a less reliable basis: guessing Trondheim and searching only for confirmation narrows the search around an unsupported premise. Follow-ups should target a gap revealed by retrieved evidence, while treating generated answers as hypotheses.

Prompted exploration and correction A prompted language model can alternate reasoning with graph queries [87] and consult text or tables to revise its rationale [88]. Keeping several candidate paths, then pruning relations and neighboring entities at each step, makes this a *beam search* (Think-on-Graph [29]). Pruning saves later exploration but can discard the correct branch early. If the retained entity is Cedar Labs, the software company, later hops remain in the wrong neighborhood. Linked source text gives the pruning decision more evidence [41]; recording unfinished subgoals and explored branches lets the controller backtrack (Plan-on-Graph [89]). Text helps prevent a wrong turn, and an explicit search record lets the controller recover from one.

Graph algorithms as tools Some questions require computing over all relevant edges. "How many labs in the collection are in Oslo?" calls for counting matching location edges, rather than returning one path. A controller can call graph algorithms for these operations. In a supply-chain case study, algorithm access enabled aggregation and comparison questions that text retrieval and traversal failed to answer [90]. For such questions, retrieval must return a computed result over the graph rather than leave the generator to infer a count from a few examples.

Stopping and evidence records A controller can stop once its record of evidence covers the question's requirements. One design checks sufficiency before escalating to targeted graph retrieval or the source text behind graph signals (A2RAG [91]). Evidence records also guide navigation through document hierarchies [92] and graphs of dependent subquestions [93]. The

lab's identity may be established while its location remains open; recording those separately directs the next search to the missing location. Stopping and follow-up search then depend on unresolved facts rather than on a fixed number of hops.

Reinforcement-learned control Training can teach a controller when to retrieve and answer, rather than prescribing those choices in a prompt. Multi-turn policies learn graph operations from answer rewards [30], while cost-aware rewards favor correct runs that spend less retrieval time [94]. Final-answer rewards alone can credit a search that found nothing useful if the model answered from memory. Adding overlap with an annotated evidence path rewards retrieval itself, and a teacher with access to that evidence can guide unsuccessful runs (PathRouter [35]). Answer correctness and evidence overlap then improve together. Rewards that score both establish a stronger training target than correctness alone.

A systematic study of iterative retrieval in KG RAG shows where adaptation pays [12]. Iteration helps most on multi-hop bridge questions, by promoting the documents that hold the bridging entity into the leading ranks; on single-hop and simple comparison questions it brings little or nothing, and widening the candidate pool adds noise. Coverage was rarely the problem, since the relevant documents usually appeared somewhere in the top hundred. The problem was ranking them among the top few that the generator reads. An adaptive controller therefore has to decide which evidence reaches the top ranks, not only whether to search again.

4.5 Hierarchical and community retrieval

When construction has built several levels of summaries, retrieval also chooses the level. "What do the labs in this collection study?" needs the breadth of the top levels, "Why did Mira found Cedar Lab?" needs the detail of the leaves, and many questions need some of both.

Coverage and selective community search Reading every community summary at one level gives retrieval coverage across a collection (GraphRAG [4]). Coarser levels offer fewer, broader summaries; finer levels offer more detail. Selective search reduces the reading by choosing only relevant communities, using either a hierarchy of attributed groups [95] or topic-specific subgraphs [96]. For "Which labs study forests?", it can skip unrelated groups but also miss a forest lab summarized under another theme. A third choice samples representative edges within a token budget, using a hierarchy formed by repeatedly removing the least-connected nodes, or k-core decomposition [97]. Reading all groups prioritizes collection coverage; selecting or sampling groups spends less context on material that appears irrelevant. The breadth required by the question determines which saving is acceptable.

Searching across levels Ranking several levels together lets a question select both a summary and its underlying detail. In RAPTOR's comparison on a subset of the scientific-paper question-answering benchmark QASPER, a *collapsed tree*, which ranks passages and summaries together, consistently outperformed traversal down from the root [24]. A question could then reach a fine-grained match even when its parent summary was less similar. Other methods explicitly move from entities up to connecting summaries [98] or from global summaries down to detail [99]. The same choice appears in book hierarchies [100], cross-document summary trees [101], and indexes combining propositions, topics or hyperedges [102, 103]. Summary-level matches locate broad context, while linked leaves supply facts the summary omits. Keeping both levels available avoids making the initial choice of granularity irreversible.

5 Evidence integration and generation

Integration converts the retrieved selection S into the evidence $E = \phi(q, S)$ the generator reads. Text interfaces present triples, source passages or synthesized summaries (§5.1); learned interfaces encode structure in vectors or special tokens (§5.2). The choice changes which details remain visible, as Figure 3 illustrates. We separate these presentation choices from the checks applied to the resulting answer: valid graph paths, source support and attribution (§5.3).

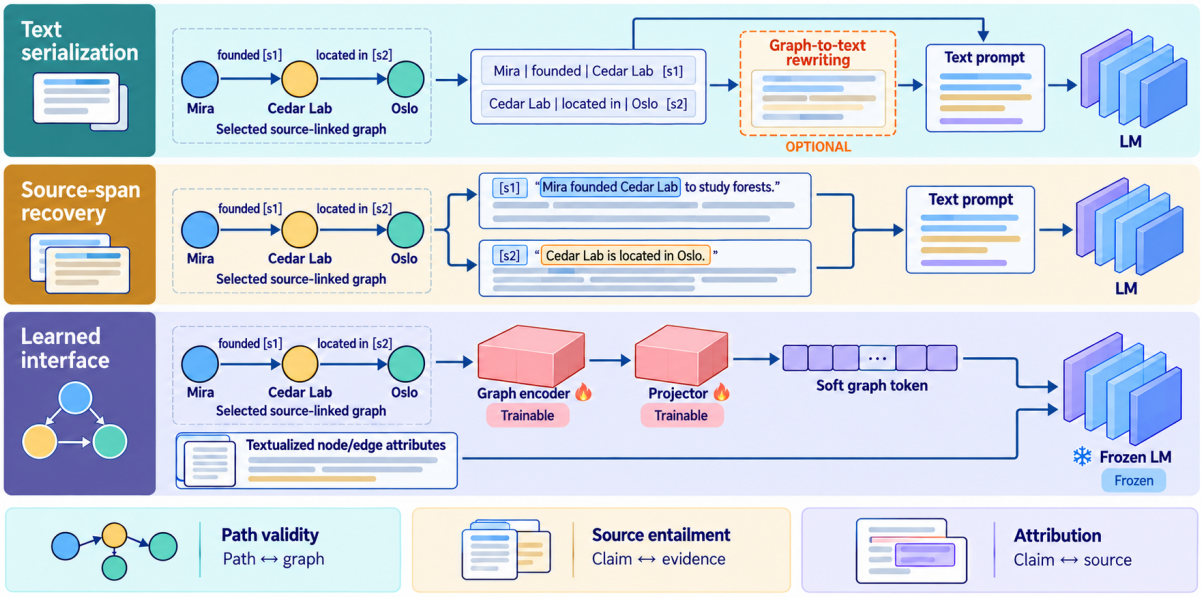


Figure 3: The generator can receive relation text or linked source wording (top two rows, §5.1) or a learned graph vector (third row, §5.2). The source wording restores the founding purpose. Path validity, source support and attribution require separate checks (bottom, §5.3) [14, 28, 31, 32, 39].

5.1 Textual evidence and answer synthesis

Writing the selected graph elements as text lets any generator read them through an ordinary prompt. Shorter text takes less room in the prompt but leaves out more. Triples are the shortest, and rewriting makes them easier to read, but neither brings back details that extraction dropped. Linked source text does, at a higher cost. Evidence too large for one prompt has to be condensed in stages, and each stage can lose facts.

Triples and paths When the graph carries facts (§2.1), integration can write selected triples as short statements without retraining the model (KAPING [28]). The running example becomes "Mira | founded | Cedar Lab; Cedar Lab | located in | Oslo" (Figure 3, top). Paths additionally order the facts, and arranging several paths changes where the strongest evidence appears. One method puts more reliable paths last to reduce their exposure to the poorly attended middle of a long prompt (PathRAG [75]). Templates can similarly expose control and data flow in retrieved code [104]. Serialization preserves the selected relations and their order; the qualifications available to the generator remain those preserved during extraction.

Graph-to-text rewriting Rewriting reduces the repetition and scattered facts in a list of triples. A trained rewriter can combine shared entities into sentences, using graph–text pairs selected for helping the generator answer (Retrieve-Rewrite-Answer [31]). A summarizer can additionally learn preferences for text that helps answer without adding unsupported facts

(EFSum [105]). These objectives address different risks: readability helps the model use the facts, while fidelity constrains what the rewriter may assert. Neither recovers the founding purpose from a triple that never stored it. Rewriting suits evidence whose needed facts are already present; omitted context requires the source text.

Source text Returning linked source text restores details omitted by extraction. Whole passages can carry the context behind ranked graph elements [26, 33]; selected spans can identify the wording that answers each part of the question (LineageRAG [14]). For the founding edge, returning s_1 restores "to study forests" (Figure 3, middle). This choice affects evidence use even when retrieval is unchanged: adding source spans to the same passages improves answer F1 (§6.2). Dialogue evaluations likewise find that graph and text presentations favor different properties, with text improving factual consistency [106]. Source text spends more context than triples but preserves qualifications and makes the answer checkable against the original wording.

Staged synthesis Answering from more evidence than fits in one prompt requires intermediate answers or summaries. For example, GraphRAG generates a partial answer from each community summary, rates its helpfulness, discards zero-rated answers and combines the remaining answers within the final context budget [4]. Each forest-research community can then contribute to the collection-wide answer. Finer, query-focused summaries reduce irrelevant material before synthesis [107]. A planned tree of retrievals instead lets the intended answer structure determine which branches are gathered and combined [108], while an outline can organize scientific synthesis into chains of cited contributions [109]. All intermediate summaries can omit facts, so broad coverage depends on what survives each stage. Staged synthesis is useful when the gain from covering more sources outweighs those additional opportunities for loss.

5.2 Learned graph interfaces

Learned interfaces require a generator that accepts graph vectors or has been trained to read structural tokens. This differs from training the retriever (§4.3), which changes candidate ranking while leaving the generator's text interface intact.

Graph encoders and soft prompts A learned interface can compress a selected subgraph into a vector the generator reads like a prompt token. For example, G-Retriever assigns relevance prizes to nodes and edges and selects a connected subgraph that balances those prizes against edge costs [32]. A graph neural network encodes it, and a trained projection maps the encoding into the generator's embedding space as a *soft graph token*. The frozen generator reads that vector alongside the question and the subgraph's textual attributes (Figure 3, third row). Training adjusts the encoder and projection to improve generated answers. Related approaches align graph and text representations [36, 104], or let a smaller model identify entities and relations before a larger model writes the answer [110]. Soft graph tokens suit generators whose embedding layer is accessible and for which a compatible projection can be trained.

Entity representations and graph memory Structure can enter at the level of individual entities rather than one pooled vector. A relation-aware network can modify retrieved entities' token embeddings and train retrieval and generation together (SURGE [9]). Its contrastive training brings graph and text representations closer when they describe matching facts; altering the supplied facts tests whether responses follow that evidence rather than memory. Other designs combine commonsense subgraphs [111], evidence gathered during conversation [112], or hierarchical graph memory with the model's token representations [113]. The granularity

changes from entities to subgraphs or memory slots, but each design needs access to embeddings or attention. A prompt-only generator therefore calls for the textual interfaces of Section 5.1.

Learned linearization A token sequence can preserve graph structure when the generator is trained to read an explicit notation. Special tokens distinguish heads, relations, reverse relations and tails, while a graph-constrained retriever generates the sequence (DialogGSR [114]). Training the response model to reconstruct the graph as well as answer teaches it how those markers connect. The resulting text keeps the relations explicit while remaining a token sequence, provided the response model has been trained to read the markers.

Whether the generator uses the graph Evaluating a learned interface requires testing whether the response depends on the supplied structure. A diagnostic asking for the hop count between two nodes finds zero exact match for graph-augmented baselines even when their input contains the necessary evidence [13]. Query-based sampling, readable endpoint roles and tokens that preserve adjacency raise exact match to 37–77% across four domains. On this task, selecting and presenting the right structure matters more than merely adding a graph representation to the input.

5.3 Grounding and verification

Once evidence has been presented, checks can enforce that a generated path exists, test whether the source text supports a claim, and locate the cited wording (Figure 3, bottom). We take these checks in that order because each establishes something the previous one leaves open. Answer correctness requires the task-specific evaluation of Section 6, including whether the sources answer the intended question.

Constrained paths Constrained decoding can enforce path validity while a model writes. A *prefix tree*, or trie, stores allowed graph paths and restricts each next token to one that continues a stored path [39]. The same idea can constrain subgraph generation [114]. It blocks "Mira, founded, Birch Lab" when that edge is absent. It still permits a founding edge extracted from a source that said "visited", so validity checking needs the separate source checks below.

Entailment and consistency checks To check source support, a verifier asks whether the evidence entails each generated claim. It can match response sentences to supporting evidence sentences [115], separate retrieval, legal assessment and conclusion writing among agents [116], or check medical summary sentences against retrieved evidence chains [117]. Verification can also precede generation: support and contradiction among atomic claims determine which claims enter the answer (ArbGraph [63]). Similar wording is a weaker test. A graph-based consistency score tracks hallucinations in one model family but reverses direction in several others on RAGTruth, a dataset with hallucination annotations [118]. "Cedar Lab studies forests in Oslo and Bergen" reuses the example's wording while adding an unsupported city. The check must test the claim's meaning against the evidence, rather than their structural or lexical resemblance.

Attribution and evidence lineage *Attribution* connects each answer claim to a source a person can inspect. A verifier can check that a quoted span occurs in a selected passage for the relevant information need (LineageRAG [14]). Other checks locate cited web blocks [58], trace financial claims through recorded source chains [119], or confirm that cited code files and line ranges exist [120]. Locating a quotation establishes its existence; the entailment checks above establish whether it supports the claim. Citing s_1 beside "Mira founded Cedar Lab in 2015" leaves the year unsupported even though the quotation is genuine. Citations also omit some

information that affects the answer: agentic graph retrieval experiments find dependence on visited but uncited entities [121]. A useful evidence record therefore distinguishes the source offered to support a claim from the observations that influenced its generation.

6 Comparative evaluation

Evaluation has to say whether a graph helps, and Section 2.2 showed that helping can mean several things: a correct answer, an answer its sources support, or better evidence, each at some cost. Benchmarks differ in which of these they can observe (§6.1). Comparisons that hold the task, the generator and the budget fixed show when graphs help and when they do not (§6.2), and cost accounting separates the one-time work of building a graph from the work each question adds (§6.3).

6.1 Tasks and benchmarks

Benchmarks give a system either documents to search, a graph to query, or both, and ask it to recover facts or generate an answer. Document question answering ranges from single facts (Natural Questions [122]) to facts connected across Wikipedia articles (HotpotQA and MuSiQue [123, 124]) or news articles (MultiHop-RAG [2]). Curated-graph question answering supplies stored relations instead of requiring their extraction, as in WebQSP over Freebase [125]. These tasks test the direct and connected questions in the lab example. Success requires the right answer and, where annotated, recovery of the supporting evidence.

Broader generation tasks ask for themes, summaries or explanations rather than one fact. They include questions over medical guidelines and novels (GraphRAG-Bench, Xiang et al. [23]), section summaries grounded in web sources (WildGraphBench [126]), and textbook reasoning with reference rationales (GraphRAG-Bench, Xiao et al. [127]). Success requires coverage of the requested material as well as correct individual claims. Further families test entity ranking without generation (STaRK [128]), answers over textual or scene graphs (GraphQA [32]), and evidence crossing text, tables and knowledge graphs or images (mmRAG and MKG-RAG-Bench [129, 130]). Table 4 compares these settings and their controls.

Answer and evidence metrics *Accuracy* averages whether answers are correct under a benchmark's scoring rule. *Exact match* (EM) gives one for a prediction matching the reference string and zero otherwise, usually after normalization. *Judged accuracy* is the fraction of answers a human or model judge accepts as correct. Some benchmarks award partial credit for partially correct selections.[1] These rules compare answers differently and must be held fixed within a comparison.

Overlap metrics also give partial credit. *Precision* P is the fraction of predicted items matching the reference. *Recall* R is the fraction of reference items recovered. *F1* combines them as $F_1 = 2PR/(P+R)$, with zero assigned when there is no overlap. Answer F1 compares answer tokens. Statement F1 compares the claims in a generated summary with reference statements [126]. Triple recall counts relevant graph triples recovered [8]; passage recall counts annotated passages; citation F1 compares cited source identifiers with gold identifiers [119]. These variants apply the precision, recall and F1 calculations to their named units.

Ranking and generation metrics Ranking scores reward finding relevant items near the top. *Hit@k* usually counts the fraction of questions with a relevant item in the first k results. *Mean*

[1] GraphRAG-Bench (Xiao et al.) gives half credit for a correct subset in multi-select questions and uses model-judged scores for open-ended and fill-in-blank answers [127].

reciprocal rank (MRR) averages the reciprocal rank of the first relevant result, assigning zero when none is found. *Mean average precision* (MAP) averages over questions the sum of precision at relevant ranks divided by the number of annotated relevant items. A cutoff limits which retrieved ranks contribute. *Normalized discounted cumulative gain* (NDCG) sums relevance gains discounted by the logarithm of one plus rank, then divides by the score of the ideal ordering. The counted unit still matters: MultiHop-RAG calls evidence recall Hit@k [2], GraphQA's Hit@1 counts whether the top generated answer matches an accepted answer, and STaRK's scores entity ranking [32, 128].

Longer outputs need both support and coverage checks. A *win rate* is the fraction of paired answers a judge prefers on a stated criterion, such as comprehensiveness or diversity [4]. *Faithfulness* estimates the fraction of answer claims supported by retrieved context [23]. Text-overlap scores compare the response with a reference: *ROUGE-L* uses the longest common token subsequence to calculate precision, recall and F1; *BLEU-1* multiplies token precision by a brevity penalty, counting each predicted token no more often than it appears in the reference. *Contains@1* is the fraction of generated answers containing the reference answer [130]. Structural checks compare generated graph elements with the stored graph, while a *rationale score* averages a judge's assessment of agreement between generated and reference explanations [32, 127]. These checks distinguish matching the reference, using the supplied sources and producing valid structure.

What the measurements reveal Recording intermediate evidence helps locate failures. Supplying MultiHop-RAG's gold evidence exposes errors that remain after retrieval has succeeded [2]. Reporting retrieved relevance and recall beside answer quality separates retrieval losses from generation losses [23]. Even then, a metric must match the task: entity-level F1 underscored correct structural answers in a supply-chain case study [90], and a graph-based evaluator's path overlap tracked correctness more closely than faithfulness [131]. Stage records explain where a system loses evidence; task-specific answer and support scores establish whether that loss matters.

6.2 Comparative findings

Controlled comparisons ask whether changing retrieval or evidence presentation improves answers on the same questions. Success means a gain under a stated generator, evidence budget and scoring rule, with build and query costs accounted for separately. Table 5 uses the answer, evidence and ranking metrics defined in Section 6.1. Each block is a separate study; the interpretation below compares alternatives within blocks rather than pooling their scores.

Task and budget dependence The same retrieval method can help one task and hurt another. With one generator and ten retrieved units, a propagation-based graph method trails text RAG on single-hop Natural Questions but leads it on multi-hop HotpotQA by roughly three to four answer-F1 points in each direction [16]. The evidence budget changes the comparison too. Local community retrieval leads text RAG on MultiHop-RAG at ten units (69.0% against 67.0%), but its units carry more text. At equal input tokens, text RAG matches it overall (69.3% against 69.0%). The community method remains stronger on temporal questions, while text retrieval is stronger on inference questions. Combining their evidence or routing questions to the better retriever beats either alone. A separate comparison finds that adding entities and relations to communities and passages raises multi-hop accuracy from 55.2% to 59.7% [22]. Graph connections and source text are complementary, and equal token budgets are necessary to distinguish that complementarity from simply reading more.

Broad generation exposes a second task distinction. On WildGraphBench, global community

Table 4: KG RAG benchmarks test different sources, outputs and stages, with different controls and cost measurements.

Benchmark	Sources	Task / output	Tested controls	Metrics	Cost evidence
GraphRAG-Bench (Xiang et al.)[23]	Medical guidelines; novels	Fact and reasoning QA; summaries; creative answers	Flat RAG with and without reranking; several generators	R: relevance, recall; G: accuracy, ROUGE-L, faithfulness	Build time and tokens; prompt tokens
GraphRAG-Bench (Xiao et al.)[127]	20 CS textbooks	Five answer formats; rationales	GPT-4o-mini; $k = 5$; 1,200-token passages	R: not reported; G: accuracy, rationale score	Build, index and retrieval time; tokens
WildGraphBench [126]	Web pages cited by Wikipedia	Single- and multi-fact QA; section summaries	GPT-4o-mini; $k = 5$ or 10; common passage splitting	R: not reported; G: accuracy, statement P/R/F1	Not reported
MultiHop-RAG[2]	609 news articles	Inference, comparison, temporal and null questions	Retrieved or gold evidence; six generators	R: MAP, MRR, Hit; G: accuracy	Not reported
STaRK[128]	Amazon; MAG; PrimeKG	Ranked entities; retrieval only	Synthetic and human queries; rerankers	R: Hit, recall, MRR; G: none	Retrieval latency
GraphQA[32]	ExplaGraphs; scene graphs; Freebase	Commonsense, scene and KG answers	Llama2-7B; prompting, tuning, LoRA	R: not separate; G: accuracy, answer Hit@1; structural checks	Tokens and nodes; training time
mmRAG[129]	Six QA datasets; text, tables, KGs	Answers; passage retrieval; routing	Two generators; top 3; oracle retrieval	R: NDCG, MAP, Hits; G: EM, F1	Not reported
MKG-RAG-Bench[130]	MarKG; MedMKG; text and images	Masked relation or tail answers	Shared CLIP; modality subsets; GPT-5; $k = 5$	R: NDCG, P/R; G: EM, F1, Contains@1, BLEU-1	Throughput with cached embeddings

R/G: retrieval and generation; P/R: precision and recall. Metrics are defined above. LoRA: low-rank adaptation, which trains small weight updates; CLIP: an encoder that maps images and text into a shared vector space. MAG is a scholarly publication graph; PrimeKG is a biomedical graph; ExplaGraphs contains commonsense explanation graphs. Retrieval metrics count the evidence units or entities specified by each benchmark.

retrieval raises multi-fact accuracy from 35.1% to 47.6% but lowers section-summary statement F1 from 15.8% to 13.8% [126]. Finding the few facts needed to identify forest labs differs from covering the statements in an entire report section. Summary-based retrieval can help the former while dropping details needed by the latter.

How synthesis is judged changes the apparent benefit too. Some studies compare reference overlap, using *ROUGE-2* to apply precision, recall and F1 to adjacent token pairs. *BERTScore* instead matches each answer token to its most similar reference-token vector, and vice versa, then combines the two average similarities as F1 [16]. Others judge paired answers for comprehensiveness and diversity [22], while WildGraphBench scores statement coverage [126]. A varied thematic answer can win a preference judgment while missing reference facts. Pairwise judging adds further variation: reversing answer order can shift win rates by more than thirty points, and answer length can influence preference [132]. Under bias controls and questions generated from indexed text, one reported 72% to 28% graph win becomes a slight loss. Comparing broad answers therefore requires matched coverage criteria and judge controls, not a preference score alone.

Evidence coverage and evidence use Retrieving more supporting material helps only if the generator uses it. With fixed embeddings and retrieval budget, a conversation graph raises evidence recall from 79.7% to 84.5%, a statistically supported gain, while answer F1 changes little [67]. Conversely, adding source spans tied to the question's information needs to the same five passages leaves passage recall unchanged but raises MuSiQue answer F1 from 50.1% to

Table 5: Graph retrieval gains depend on the task, evidence budget and property scored. Each block compares conditions within one study.

Task	Control → alternative	Metric	Control	Alternative	Budget or condition
Han et al. [16]; Llama-3.1-8B-Instruct generator					
Natural Questions	RAG → HippoRAG 2	Answer F1	64.78	61.03	Top 10 units
HotpotQA	RAG → HippoRAG 2	Answer F1	60.04	63.01	Top 10 units
MultiHop-RAG	RAG → community (local)	Accuracy	67.02	69.01	Top 10 units; unequal tokens
MultiHop-RAG	Token-matched RAG → community (local)	Accuracy	69.33	69.01	Equal input tokens
MultiHop-RAG	RAG → community graph	Build time (s)	135	5,560	Graph built with GPT-4o-mini
WildGraphBench [126]; GPT-4o-mini generator, GPT-5-mini judge					
Multi-fact	Text RAG → GraphRAG (global)	Accuracy	35.08	47.64	Top 5 units
Summary	Text RAG → GraphRAG (global)	Statement F1	15.84	13.78	Top 10 units
Entity–memory graph [67]; GPT-3.5-turbo generator					
LoCoMo dialogue	Dense → entity–memory graph	Evidence recall	79.7	84.5	Same vectors; top 25
LoCoMo dialogue	Dense → entity–memory graph	Answer F1	42.1	42.6	Same generator protocol
SubgraphRAG [8]; retrieval only, Freebase					
WebQSP	GNN-RAG → SubgraphRAG	Triple recall	40.5	86.5	GPT-4o-labeled triples
WebQSP	GNN-RAG → SubgraphRAG	Runtime (s)	68	6	Excludes KG queries
Zhou et al. [22]; Llama-3-8B generator					
MultihopQA	Communities and passages → plus entities and relations	Accuracy	55.2	59.7	Same retrieval setup
LineageRAG [14]; GPT-4o-mini generator					
MuSiQue	Without → with source spans	Passage R@5	76.53	76.53	Same five passages
MuSiQue	Without → with source spans	Answer F1	50.11	55.76	Spans tied to information needs
KDAF [119]; Qwen3 generator, FinanceBench reports					
145 questions	BM25 → ontology graph	Citation F1	46.3	51.5	Pool of 189 records
145 questions	BM25 → ontology graph	Correctness	11.7	11.0	Exact or numeric scoring

Quality metrics are percentages. Passage R@5 applies recall to the first five retrieved passages; evidence recall counts annotated conversation turns. Correctness uses exact or numeric agreement with the reference answer. Top-k counts evidence units, which differ by method. SubgraphRAG returns 100 triples, while GNN-RAG returns a variable number of paths. The entity–memory recall gain and KDAF's citation gain hold under the studies' resampling tests; the corresponding answer-F1 and correctness differences do not.

55.8% [14]. Supplying gold evidence exposes the remaining generation limit: on MultiHop-RAG, one generator's accuracy rises from 56% to 89%, while two others reach only 32% and 36% [2]. Evidence coverage and answer quality must be scored separately because either can improve while the other stays fixed.

On a curated graph, simpler retrieval can also be better retrieval. SubgraphRAG's evaluation on WebQSP compares its triple scorer, a small network that rates each triple independently, with GNN-RAG's graph neural network [8]. The two return different amounts of evidence, a fixed 100 triples against a variable set of paths, and the timing excludes graph queries. Under these conditions, the triple scorer more than doubles the recall of judged-relevant triples (86.5% against 40.5%) and cuts retrieval time from 68 to 6 seconds, so on this graph the simpler scorer is both more accurate and faster.

Correctness and attribution Better citations can coexist with unchanged answer correctness. Over 145 financial questions, an ontology-driven graph raises citation F1 from 46.3% to 51.5% relative to BM25, a standard lexical ranking function [119]. Paired resampling supports that citation gain, while correctness remains near 11% across retrieval conditions. The graph improves where the answer points without fixing what it says, so attribution and correctness require separate scores.

Domain and model conditions Existing domain structure can make graph construction useful without inferring every relation from prose. A support-ticket graph preserves fields and ticket links, improving retrieval rankings and reducing deployed issue-resolution time [43]. Parsed code dependencies support repository-level generation [133], while linking private medical records to literature and controlled vocabularies improves clinician-judged source use [134]. These applications share useful connections already present in their sources. Their gains motivate retaining such structure before paying to extract additional relations.

Whether the generator already knows the facts changes how much retrieval adds. On a public biomedical graph, triple retrieval and agent-written graph queries change medical-exam accuracy by at most 3.4 points across the tested models; on novel and counterfactual facts, the same pipeline lifts accuracy from chance to nearly 100% [135]. A battery-materials study likewise finds that a model without graph access recovers much of the graph's content, while random graph subsets reproduce much of the effect of top-ranked facts [136]. These findings make private or new facts, such as Cedar Lab's location, especially useful tests of evidence use. Comparing familiar and unseen facts separates information supplied by retrieval from knowledge the generator already has.

6.3 Computational cost

Cost evaluation gives a system a collection, a workload of questions and an update schedule. Success means retaining the required answer quality with affordable construction, query and maintenance work. Building the graph G is paid before questions and again where updates require it; retrieval and generation are paid per question. Figure 4 separates these stages.

Cost measures *Latency* is elapsed time for the operation being measured. *Token counts* measure model input and output separately; *storage* measures the bytes retained in indexes and summaries. *Monetary cost* applies the stated prices of model calls, tools and source access to the workload. *Throughput* counts completed operations per unit time. Reporting these separately distinguishes a one-time indexing expense from a fast query path.

In one controlled comparison, community-graph construction takes about forty times as long as text indexing (5,560 against 135 seconds), while retrieval over the study's questions takes less time (1,249 against 1,724 seconds) [16]. Storage also grows (165 against 127 MB). The graph's cheaper retrieval does not immediately repay its construction cost; how many questions it serves and how often it changes determine whether it does.

Where savings arise A saving is useful only after the work moved to other stages has been counted. (i) *Construction.* Extracting a central skeleton, simplifying the graph, or replacing relation extraction with parsing reduces indexing work [52, 137, 138]. (ii) *Retrieval.* Reusing previous traversals [139] or escalating only when evidence is missing reduces repeated search (§4.4). (iii) *Generation.* A router can select a cheaper graph method and generator using a learned policy [140] or the distribution of retrieval scores [141]. Total accounting must include the router, any training, and the answer quality retained, so a local saving is credited only after its additional work is charged.

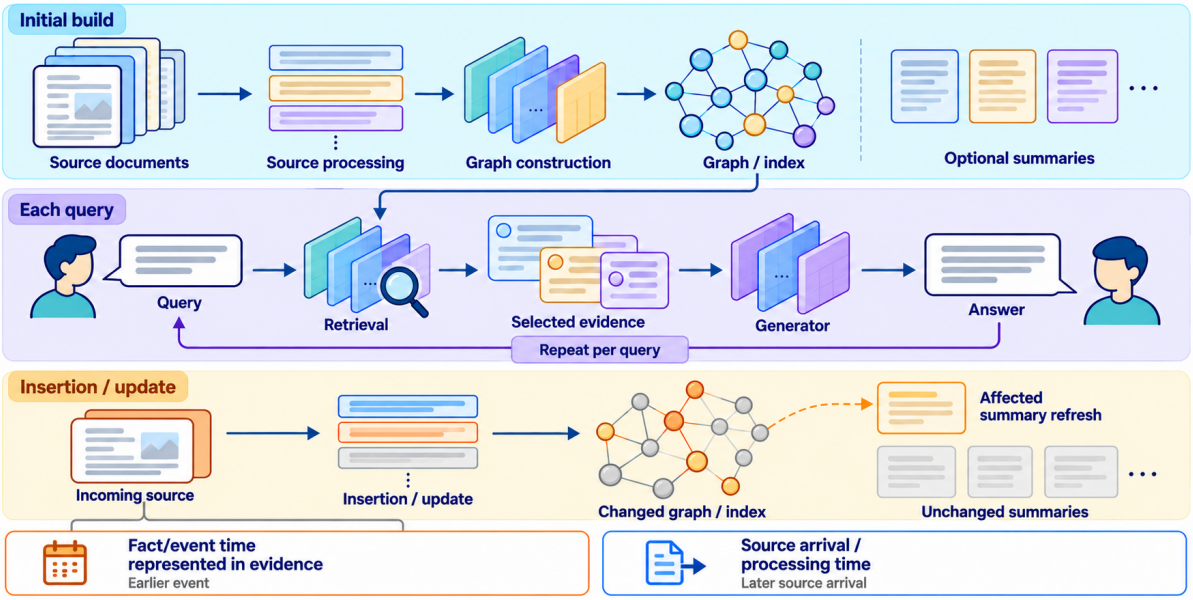


Figure 4: Construction incurs an initial cost, queries add retrieval and generation costs (top two rows, §6.3), and localized updates refresh affected graph elements and summaries (third row, §7.2). A fact's valid time can differ from its source's arrival time (bottom) [16–18, 27].

Whether a graph pays for itself depends on how often it is queried and how often it changes. A lab collection that answers thousands of questions a day spreads its build cost thin; one that receives new reports every day pays for updates again and again. Benchmarks report different parts of this account. GraphQA reports training time per pass through the training data [32], STaRK reports retrieval latency [128], and MKG-RAG-Bench reports throughput with cached embeddings and warm retrievers, which leaves out building the index and generating the answer [130]. Adaptive systems add variation, because their work depends on each question and on when they stop. A fair comparison reports build, update and per-question costs separately, in their original units.

Scale and refresh Corpus growth can rule out a design before answer quality is measured. Across nested collections spanning about a 450-fold size range, BM25 overtakes an agent that explores files above roughly ten million tokens, while the tested graph methods reach construction limits earlier [142]. Their scalable variants remain below BM25 wherever both run. Query-time alignment with an external graph avoids offline triple extraction and has been used over millions of documents, but can drift to a related topic [143]. Linking Cedar Lab to Cedar Labs, the software company, would create that error during a query rather than construction. Updates add recurring work because changed facts may affect several summaries (§7.2). The scalable choice must therefore account for both construction capacity and repeated alignment or refresh work.

7 Reliability and maintenance

A graph carries source errors into every path and summary that depends on them. Reliability therefore requires recovering missing support and exposing conflicts (§7.1), preserving the time of facts while updating their derivatives (§7.2), and resisting both poisoned sources and attempts to extract protected knowledge (§7.3). These failures share a dependency problem: changing one source fact can affect many later retrievals and answers.

7.1 Incomplete and conflicting evidence

A graph can lack an edge the answer needs, or hold claims that cannot both be true. In both cases the answer should rest on a source. A missing fact can sometimes be reached through other edges or found in the source text, but a fact the model supplies from memory has no source behind it. Conflicting claims can be settled only once the system knows why they differ, for instance because one is out of date.

Missing support and alternative paths Losing an answer-bearing edge hurts more than losing unrelated context. One path-based method loses about five accuracy points on WebQSP after random deletion of a fifth of its triples, but about eleven after deletion along reasoning paths [144]. A separate analysis finds the largest damage next to the answer, with some context-edge deletions harmless or helpful [145]. Robustness tests must also distinguish recoverable omissions from questions left without any supporting path. Removing direct facts only where a logical rule guarantees a detour preserves answerability (BRINK [146]). If Cedar Lab's city edge is missing, edges placing the lab at the University of Oslo and that university in Oslo might supply a detour. BRINK finds that systems often miss such alternatives and recover better with recognizable entity names than with hidden names. Testing both graph detours and hidden identities separates recovery from evidence from recovery through model memory.

Textual recovery and generated completion A missing graph edge can be recovered from source text or proposed from model knowledge. A controller can check each hop's requirements and return to raw passages when graph evidence cannot establish the next entity (CS-RAG [147]). Fuzzy text matching provides another fallback when decomposition or verification fails [148]. Generating and verifying missing triples instead draws on what the model already knows (Generate-on-Graph [149]). This improves answers on the incomplete graphs tested, but supplies no dependable source for an unseen private fact such as Cedar Lab's city. Returning to text recovers evidence the graph omitted; generated completion should remain a hypothesis until evidence confirms it.

Conflicting assertions Detecting a conflict becomes harder when its two claims connect only through several relations [150]. A verifier can compare retrieved claims for support and contradiction before generation (§5.3). A different conflict arises when sources disagree with model memory: one method finds paths that most change the uncertainty of the model's answer distribution [151]. Testing answers under counterfactual unreliable paths can also improve *calibration*, agreement between stated confidence and observed accuracy [152]. Resolving the answer still requires knowing why the claims differ. Reports from different periods may both be right, while two current reports may disagree about Cedar Lab's city. Recording source, time and contradiction lets the system give a dated answer in the first case and disclose disagreement in the second.

7.2 Temporal validity and updates

Facts change, but an old fact stays true for the period it describes. A graph that dates its facts, rather than overwriting them, can answer questions about the past as well as the present. The cost falls on maintenance. Each new fact also changes the summaries built on it, and rebuilding them all for every update is expensive.

Time-constrained evidence Selecting evidence by the period it describes prevents old and current facts from being mixed. Suppose Cedar Lab moves from Oslo to Bergen in 2024.

"Where was it in 2022?" and "Where is it now?" then need different location edges. One system decomposes multiple time constraints, retrieves time-specific subgraphs and then gathers finer evidence and source text (T-GRAG [153]). On annual-report questions with several time constraints, its reported accuracy rises from 36% to 75% with one generator and from 46% to 70% with another. Other methods narrow retrieval through time-aligned rules [154] or learn a temporal retriever with explicit retrieval and reasoning steps [155]. The representation must also suit the temporal question. Intervals accommodate uncertain boundaries [156]; event timelines and linked entity–event graphs preserve ordering and change [157, 158]; dated episodic memory retains successive observations [159]. Choosing the right period before generation reduces the conflicting evidence the generator must resolve.

Localized maintenance Adding new facts can be local even when their summaries form a hierarchy. After inserting extracted nodes and edges [27], a system can repartition and summarize affected text groups, then propagate the change to their ancestors (EraRAG [17]). Its experiments test repeated insertions. A time-organized hierarchy instead adds summaries for new time nodes and their ancestors while retaining earlier relations (TG-RAG [18]). Its answers about past earnings calls remain stable as new transcripts arrive. Applied to the lab collection, the dated 2024 move adds Bergen while preserving Oslo for questions about 2022. Restricting refresh to affected groups and ancestors reduces repeated summarization while retaining historical evidence.

7.3 Attacks and defenses

Attackers either put false facts into a KG RAG system or get protected facts out of it. Both attacks exploit the graph's connections. A planted fact reaches every question whose search passes through it. Carefully chosen queries can piece together relations the system should keep private, even when no answer quotes a source. Defenses therefore have to act on sources and connections, not only on wording.

Poisoning assertions and topology An attacker who changes source text can redirect the graph's connections. Injected triples can complete a misleading chain [160], target a relation shared by many questions [161], or make a false fact resemble a legitimate update [162]. A fabricated report that Cedar Lab moved to Stockholm can then enter retrieval as a newer location claim. Small edits to existing documents can be harder to notice: changing less than 0.05% of corpus words cut one system's accuracy from 95% to 50% without detection by the tested defenses [163]. Substituting entities of the same type likewise preserves fluent wording while redirecting multi-hop connections [164]. Graph poisoning exploits not only false content but also the routes through which later questions find it.

Defenses that inspect surface wording do little to repair poisoned connections. Query paraphrasing barely reduces success against injected relations or entity substitutions [161, 164], and filters based on text predictability poorly separate poisoned from clean passages [161]. Source trust labels have a larger effect when they are supplied with knowledge of which sources were poisoned: attack success on MuSiQue falls from 89.2% to 45.7% [161]. The remaining problem is obtaining those judgments without the attack labels. Source links and reliability estimates provide the records on which such defenses must act (§3.3).

Extracting hidden graph knowledge Query access can expose relations even when answers reveal little source text. Targeted prompts recover entities and relations, with leakage growing as retrieval windows and query counts increase [165]. Adaptive attacks can reconstruct topology through traversal [166] or choose queries for the new structure they reveal [167]. Restricting similarity thresholds reduces leakage at a substantial answer-quality cost. Summarizing evidence

helps against untargeted extraction but can worsen leakage on targeted facts [165]. A community summary naming Cedar Lab's confidential funder reveals the relation without quoting the funding report. Protecting documents therefore requires controlling which facts and connections each user can recover, including through summaries.

8 Open problems

The remaining problems concern preserving answer-relevant evidence and predicting when its connections help. We group them into measuring representation loss, selecting useful retrieval operations, retaining attribution under conflict, and maintaining support and permissions as sources change.

Representation errors and evidence loss Construction quality needs to be measured by the questions its omissions change. A dropped qualification, a false entity merge or a fragmented multi-part fact can leave a connected graph that misstates its source (§3). Fact-retention tests [48] and stage diagnostics [61] identify losses without establishing their answer-level consequences. Dropping "to study forests" is harmless for Cedar Lab's city but decisive for its founding purpose; a linked passage can also make a lost graph relation recoverable. Generation adds another failure point because the model may overlook evidence that is present [13]. Measuring recoverability together with evidence use would distinguish when to repair extraction, retain more text or change the generator's input.

Transferable graph benefit and retrieval control Choosing graph retrieval requires predicting a benefit beyond the collection on which it was measured. Existing comparisons show sensitivity to token budgets, output criteria and judge biases [16, 126, 132], as well as to whether the model already knows the decisive facts [135]. A useful predictor would use observable properties of a question, its sources and the available budget to select graph or text retrieval. Adaptive control needs the same prediction for each next operation. Another traversal may promote a missing bridge into the few positions the generator reads, or merely add distraction [12]. The controller needs a record of unresolved support so it can distinguish a useful next search from repeated evidence and stop when the required facts are covered.

Source-grounded attribution under conflict Claim-level support needs to survive retrieval, compression and synthesis. Citations locate the support offered for an answer, while traversal records capture other observations that influenced it [14, 121]. Connecting those records would show which retrieved statements justify each generated claim. Conflicts require an additional judgment: weighing support and contradiction makes disagreement explicit [63], but the answer must also distinguish a correctly quoted source from a reliable source and a historical claim from a current one. Cedar Lab's move calls for a dated answer; incompatible current reports call for qualified alternatives or abstention. Evaluations should score whether answers preserve those distinctions, rather than force every conflict into a single unqualified reference answer.

Efficient maintenance and trustworthy deployment Insertion methods show how summary hierarchies can grow without rebuilding everything [17, 18]. Correction and withdrawal require tracking dependencies in the other direction. A retracted report can affect extracted facts, entity merges, summaries and answers, so maintenance must find those derivatives and revise them while leaving unaffected content intact. Access control poses the same dependency problem. Graph systems can leak relations through summaries even while protecting source text [165]. If access to Cedar Lab's funding report is revoked, a readable community summary must

not continue revealing its confidential funder. Deployment therefore needs dependency records that carry both support and permissions through extraction, aggregation and traversal, and tests that verify their revision when a source changes.

9 Conclusion

Knowledge graph RAG makes connections available to retrieval so that a generator can combine facts across documents or draw on summaries of a collection. Its usefulness depends on what survives each stage. Construction can retain a founding relation while dropping its purpose; retrieval can reach both the lab's city and irrelevant neighbors; integration can deliver a path or the source wording that qualifies it. Linking graph elements to source text keeps those omitted details recoverable and gives readers evidence they can inspect.

Controlled comparisons favor graphs on some connected questions and unseen facts, while finding losses on simpler questions or detailed summaries. Equal token budgets remove some apparent advantages, and construction costs must be repaid across queries and updates. Valid paths, source-supported claims and correct answers require separate measurements. Progress therefore depends on predicting when connections help, measuring which losses change answers, and preserving support and permissions as evidence is combined, corrected or withdrawn.

References

[1] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems*, 2020.

[2] Y Tang and Y Yang. MultiHop-RAG: Benchmarking Retrieval-Augmented Generation for Multi-Hop Queries. arXiv preprint arXiv:2401.15391, 2024.

[3] Bernal Jimenez Gutierrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. HippoRAG: Neurobiologically Inspired Long-Term Memory for Large Language Models. Neural Information Processing Systems, 2024.

[4] Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. From Local to Global: A Graph RAG Approach to Query-Focused Summarization. arXiv preprint arXiv:2404.16130, 2024.

[5] Yu Wang, Nedim Lipka, Ryan A. Rossi, Alexa Siu, Ruiyi Zhang, and Tyler Derr. Knowledge Graph Prompting for Multi-Document Question Answering. arXiv preprint arXiv:2308.11730, 2023.

[6] Linhao Luo, Yuan-Fang Li, Gholamreza Haffari, and Shirui Pan. Reasoning on Graphs: Faithful and Interpretable Large Language Model Reasoning. International Conference on Learning Representations, 2023.

[7] C Mavromatis and G Karypis. Gnn-rag: Graph neural retrieval for large language model reasoning. arXiv preprint arXiv:2405.20139, 2024.

[8] Mufei Li, Siqi Miao, and Pan Li. Simple Is Effective: The Roles of Graphs and Large Language Models in Knowledge-Graph-Based Retrieval-Augmented Generation. arXiv preprint arXiv:2410.20724, 2024.

[9] Minki Kang, Jin Myung Kwak, Jinheon Baek, and Sung Ju Hwang. Knowledge Graph-Augmented Language Models for Knowledge-Grounded Dialogue Generation. arXiv preprint arXiv:2305.18846, 2023.

[10] Jaafer Klila, Sondes Bannour Souihi, Rahma Boujelben, Nasredine Semmar, and Lamia Hadrich Belguith. Injecting Structured Biomedical Knowledge into Language Models: Continual Pretraining vs. GraphRAG. arXiv preprint arXiv:2604.16422, 2026.

[11] Haokun Deng, Xunkai Li, Hongchao Qin, and Rong-Hua Li. PAGE-RAG: Provenance-Aware Graph

Evidence Promotion for Fixed-Budget Multi-hop Retrieval-Augmented Generation. arXiv preprint arXiv:2608.29753, 2026.

[12] Kai Guo, Xinnan Dai, Shenglai Zeng, Harry Shomer, Haoyu Han, Yu Wang, and Jiliang Tang. Beyond Static Retrieval: Opportunities and Pitfalls of Iterative Retrieval in GraphRAG. arXiv preprint arXiv:2509.25530, 2025.

[13] Xiaoyu Guo, Pengcheng Chen, Jiong Yu, Yi Lu, Yaohua Wang, and Ziyang Li. Graph Evidence Is Not Enough: Diagnosing Native Decoder Use in Graph-Augmented LLMs. arXiv preprint arXiv:2608.30437, 2026.

[14] Linyao Zheng, Xuhang Shi, Zhifang Mao, Sai Zhou, Shuaixian An, Xiuquan Hou, and Jinze Li. LineageRAG: Harnessing GraphRAG by Constructing Evidence Lineages with Source Grounding. arXiv preprint arXiv:2608.16004, 2026.

[15] Haoran Luo, Haihong E, Guanting Chen, et al. Hypergraphrag: Retrieval-augmented generation via hypergraph-structured knowledge representation. arXiv preprint arXiv:2503.21322, 2025.

[16] Haoyu Han, Li Ma, Yu Wang, et al. RAG vs. GraphRAG: A Systematic Evaluation and Key Insights. arXiv preprint arXiv:2502.11371, 2025.

[17] Fangyuan Zhang, Zhengjun Huang, Yingli Zhou, Qintian Guo, Zhixun Li, Wensheng Luo, Di Jiang, Yixiang Fang, and Xiaofang Zhou. EraRAG: Efficient and Incremental Retrieval Augmented Generation for Growing Corpora. arXiv preprint arXiv:2506.20963, 2025.

[18] Jiale Han, Austin Cheung, Yubai Wei, Zheng Yu, Xusheng Wang, Bing Zhu, and Yi Yang. RAG Meets Temporal Graphs: Time-Sensitive Modeling and Retrieval for Evolving Knowledge. arXiv preprint arXiv:2510.13590, 2025.

[19] Boci Peng, Yun Zhu, Yongchao Liu, Xiaohe Bo, Haizhou Shi, Chuntao Hong, Yan Zhang, and Siliang Tang. Graph Retrieval-Augmented Generation: A Survey. arXiv preprint arXiv:2408.08921, 2024.

[20] Haoyu Han, Yu Wang, Harry Shomer, et al. Retrieval-Augmented Generation with Graphs (GraphRAG). arXiv preprint arXiv:2501.00309, 2024.

[21] Qinggang Zhang, Shengyuan Chen, Yuanchen Bei, et al. A Survey of Graph Retrieval-Augmented Generation for Customized Large Language Models. arXiv preprint arXiv:2501.13958, 2025.

[22] Yingli Zhou, Yaodong Su, Youran Sun, et al. In-depth Analysis of Graph-based RAG in a Unified Framework. arXiv preprint arXiv:2503.04338, 2025.

[23] Zhishang Xiang, Chuanjie Wu, Qinggang Zhang, Shengyuan Chen, Zijin Hong, Xiao Huang, and Jinsong Su. When to use Graphs in RAG: A Comprehensive Analysis for Graph Retrieval-Augmented Generation. arXiv preprint arXiv:2506.05690, 2025.

[24] Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D. Manning. RAPTOR: Recursive Abstractive Processing for Tree-Organized Retrieval. arXiv preprint arXiv:2401.18059, 2024.

[25] Lei Liang, Mengshu Sun, Zhengke Gui, et al. KAG: Boosting LLMs in Professional Domains via Knowledge Augmented Generation. arXiv preprint arXiv:2409.13731, 2024.

[26] Xiangrong Zhu, Yuexiang Xie, Yi Liu, Yaliang Li, and Wei Hu. Knowledge Graph-Guided Retrieval Augmented Generation. arXiv preprint arXiv:2502.06864, 2025.

[27] Zirui Guo, Lianghao Xia, Yanhua Yu, Tu Ao, and Chao Huang. LightRAG: Simple and Fast Retrieval-Augmented Generation. arXiv preprint arXiv:2410.05779, 2024.

[28] Jinheon Baek, Alham Fikri Aji, and Amir Saffari. Knowledge-Augmented Language Model Prompting for Zero-Shot Knowledge Graph Question Answering. arXiv preprint arXiv:2306.04136, 2023.

[29] Jiashuo Sun, Chengjin Xu, Lumingyuan Tang, Sai Wang, Chen Lin, Yeyun Gong, Lionel M. Ni, H. Shum, and Jian Guo. Think-on-Graph: Deep and Responsible Reasoning of Large Language Model on Knowledge Graph. International Conference on Learning Representations, 2023.

[30] Haoran Luo, E. Haihong, Guanting Chen, et al. Graph-R1: Towards Agentic GraphRAG Framework via End-to-end Reinforcement Learning. arXiv preprint arXiv:2507.21892, 2025.

[31] Yike Wu, Nan Hu, Sheng Bi, Guilin Qi, Jie Ren, Anhuan Xie, and Wei Song. Retrieve-Rewrite-Answer: A KG-to-Text Enhanced LLMs Framework for Knowledge Graph Question Answering. arXiv preprint arXiv:2309.11206, 2023.

[32] Xiaoxin He, Yijun Tian, Yifei Sun, Nitesh V. Chawla, Thomas Laurent, Yann LeCun, Xavier Bresson,

and Bryan Hooi. G-Retriever: Retrieval-Augmented Generation for Textual Graph Understanding and Question Answering. arXiv preprint arXiv:2402.07630, 2024.

[33] Bernal Jiménez Gutiérrez, Yiheng Shu, Weijian Qi, Sizhe Zhou, and Yu Su. From RAG to Memory: Non-Parametric Continual Learning for Large Language Models. International Conference on Machine Learning, 2025.

[34] Tianyi Zhang and Andreas Marfurt. GROUNDEDKG-RAG: Grounded Knowledge Graph Index for Long-document Question Answering. arXiv preprint arXiv:2604.04359, 2026.

[35] Bo Wang, Heyan Huang, Yaolin Li, Wei Tang, Yuan Zhang, Wenbo Li, Mingze Gao, Ge Shi, and Chong Feng. PathRouter: Aligning Rewards with Retrieval Quality in Agentic Graph Retrieval-Augmented Generation. arXiv preprint arXiv:2606.16409, 2026.

[36] Derong Xu, Pengyue Jia, Xiaopeng Li, et al. Align-GRAG: Anchor and Rationale Guided Dual Alignment for Graph Retrieval-Augmented Generation. arXiv preprint arXiv:2505.16237, 2025.

[37] Weidong Bao, Yingying Sun, Jun Yang, Yilin Wang, Zili Wei, Yubin Bao, Fangling Leng, Minghe Yu, Tiancheng Zhang, and Ge Yu. MEGRAG: Multi-Granular Evidence Graphs for Answer-Aware Multi-Hop RAG. arXiv preprint arXiv:2608.02195, 2026.

[38] Pouria Mortezaagha and Arya Rahgozar. Graph-Aware Late Chunking for Retrieval-Augmented Generation in Biomedical Literature. arXiv preprint arXiv:2603.22633, 2026.

[39] L Luo, Z Zhao, C Gong, G Haffari, and S Pan. Graph-constrained Reasoning: Faithful Reasoning on Knowledge Graphs with Large Language Models. arXiv preprint arXiv:2410.13080, 2024.

[40] Luyao Zhuang, Shengyuan Chen, Yilin Xiao, Huachi Zhou, Yujing Zhang, Hao Chen, Qinggang Zhang, and Xiao Huang. LinearRAG: Linear Graph Retrieval Augmented Generation on Large-scale Corpora. arXiv preprint arXiv:2510.10114, 2025.

[41] Shengjie Ma, Chengjin Xu, Xuhui Jiang, Muzhi Li, Huaren Qu, Cehao Yang, Jiaxin Mao, and Jian Guo. Think-on-Graph 2.0: Deep and Faithful Large Language Model Reasoning with Knowledge-guided Retrieval Augmented Generation. arXiv preprint arXiv:2407.10805, 2024.

[42] C Mavromatis, S Adeshina, VN Ioannidis, Z Han, Q Zhu, et al. BYOKG-RAG: Multi-Strategy Graph Retrieval for Knowledge Graph Question Answering. arXiv preprint arXiv:2507.04127, 2025.

[43] Z Xu, MJ Cruz, M Guevara, T Wang, M Deshpande, et al. Retrieval-Augmented Generation with Knowledge Graphs for Customer Service Question Answering. arXiv preprint arXiv:2404.17723, 2024.

[44] Manideep Reddy Chinthareddy. Reliable Graph-RAG for Codebases: AST-Derived Graphs vs LLM-Extracted Knowledge Graphs. arXiv preprint arXiv:2601.08773, 2026.

[45] Jie Wang, Honghua Huang, Xi Ge, Jianhui Su, Wen Liu, and Shiguo Lian. OMD-GraphRAG: Enhancing GraphRAG with Ontology-Guided Extraction, Multi-Dimensional Clustering and Dual-Channel Fusion. arXiv preprint arXiv:2603.25152, 2026.

[46] Bowen Zhang and Harold Soh. Extract, Define, Canonicalize: An LLM-based Framework for Knowledge Graph Construction. arXiv preprint arXiv:2404.03868, 2024.

[47] Lorenzo Loconte, Timothy Hospedales, and Cristina Cornelio. Better Later Than Sooner: Neuro-Symbolic Knowledge Graph Construction via Ontology-grounded Post-extraction Correction. arXiv preprint arXiv:2605.29168, 2026.

[48] B Mo, K Yu, J Kazdan, P Mpala, L Yu, C Cundy, et al. KGGen: Extracting Knowledge Graphs from Plain Text with Language Models. arXiv preprint arXiv:2502.09956, 2025.

[49] Xuecheng Zou, Ke Liu, Bingbing Wang, Huafei Deng, Li Zhang, and Yu Tang. GOSU: Retrieval-Augmented Generation with Global-Level Optimized Semantic Unit-Centric Framework. arXiv preprint arXiv:2509.00449, 2025.

[50] Bhaskarjit Sarmah, Benika Hall, Rohan Rao, Sunil Patel, Stefano Pasquali, and Dhagash Mehta. HybridRAG: Integrating Knowledge Graphs and Vector Retrieval Augmented Generation for Efficient Information Extraction. arXiv preprint arXiv:2408.04948, 2024.

[51] Roman Prosvirnin, Sergei Kuznetsov, and Seungmin Jin. ContextRAG: Extraction-Free Hierarchical Graph Construction for Retrieval-Augmented Generation. arXiv preprint arXiv:2605.19735, 2026.

[52] Yiqian Huang, Shiqi Zhang, and Xiaokui Xiao. KET-RAG: A Cost-Efficient Multi-Granular Indexing Framework for Graph-RAG. arXiv preprint arXiv:2502.09304, 2025.

[53] Maxime Delmas, Lei Xu, and André Freitas. A Navigational Approach for Comprehensive RAG via

Traversal over Proposition Graphs. arXiv preprint arXiv:2601.04859, 2026.

[54] Tianyang Xu, Haojie Zheng, Chengze Li, Haoxiang Chen, Yixin Liu, Ruoxi Chen, and Lichao Sun. NodeRAG: Structuring Graph-based RAG with Heterogeneous Nodes. arXiv preprint arXiv:2504.11544, 2025.

[55] Chengfeng Dou, Ying Zhang, Zhi Jin, Wenpin Jiao, Haiyan Zhao, Yongqiang Zhao, and Zhengwei Tao. Enhancing LLM Generation with Knowledge Hypergraph for Evidence-Based Medicine. arXiv preprint arXiv:2503.16530, 2025.

[56] Hao Hu, Yifan Feng, Ruoxue Li, Rundong Xue, Xingliang Hou, Zhiqiang Tian, Yue Gao, and Shaoyi Du. Cog-RAG: Cognitive-Inspired Dual-Hypergraph with Theme Alignment Retrieval-Augmented Generation. arXiv preprint arXiv:2511.13201, 2025.

[57] Xin He, Yili Wang, Wenqi Fan, Qing Li, Qinggang Zhang, Yi Chang, and Xin Wang. HVM-GraphRAG: Holistic-View Multimodal Graph Retrieval-Augmented Generation on Complex Document. arXiv preprint arXiv:2607.24861, 2026.

[58] Ying Liu, Yi Ye, Quanyu Feng, Mingxi Ye, Mingtao Zhang, Haoyang Li, Chen Jason Zhang, and Qing Li. PolyUQuest: Verifiable Structure-Aware Web RAG over Heterogeneous Graphs. arXiv preprint arXiv:2607.08269, 2026.

[59] Chengrui Han and Zesheng Cheng. RAGA: Reading-And-Graph-building-Agent for Autonomous Knowledge Graph Construction and Retrieval-Augmented Generation. arXiv preprint arXiv:2605.17072, 2026.

[60] Xingyu Chen, Junxiu An, Jun Guo, and Li Wang. PAGE-RAG: Evidence-Grounded Adaptive Graph Retrieval for Long-Document Question Answering. arXiv preprint arXiv:2607.19301, 2026.

[61] Axel TahmasebiMoradi, Lucas Schott, and Martin Royer. TRIAGE: Trustworthy Retrieval Instrumentation And Graph Evaluation. arXiv preprint arXiv:2607.03447, 2026.

[62] Zhuoheng Li and Ying Chen. Feedback-Assisted Trust Propagation over Document Relation Graphs for Retrieval-Augmented Generation. arXiv preprint arXiv:2609.00543, 2026.

[63] Qingying Niu, Yuhao Wang, Ruiyang Ren, Bohui Fang, and Wayne Xin Zhao. ArbGraph: Conflict-Aware Evidence Arbitration for Reliable Long-Form Retrieval-Augmented Generation. arXiv preprint arXiv:2604.18362, 2026.

[64] Y Yang, H Wu, T Wang, J Yang, H Ma, and G Luo. Pseudo-Knowledge Graph: Meta-Path Guided Retrieval and In-Graph Text for RAG-Equipped LLM. arXiv preprint arXiv:2503.00309, 2025.

[65] Andre Bacellar. Calibrated Fusion for Heterogeneous Graph-Vector Retrieval in Multi-Hop QA. arXiv preprint arXiv:2603.28886, 2026.

[66] Seonho An, Chaejeong Hyun, and Min-Soo Kim. GraFine: Retrieval-Time Refinement for Efficient Graph RAG over Corpus Graphs. arXiv preprint arXiv:2601.18579, 2026.

[67] Shumao Sun. Entity-Memory Graph Retrieval Improves Evidence Coverage in Long-Conversation Question Answering. arXiv preprint arXiv:2608.27925, 2026.

[68] Yu Xia, Junda Wu, Sungchul Kim, Tong Yu, Ryan A. Rossi, Haoliang Wang, and Julian McAuley. Knowledge-Aware Query Expansion with Large Language Models for Textual and Relational Retrieval. arXiv preprint arXiv:2410.13765, 2024.

[69] Meng-Chieh Lee, Qi Zhu, Costas Mavromatis, Zhen Han, Soji Adeshina, Vassilis N. Ioannidis, Huzefa Rangwala, and Christos Faloutsos. HybGRAG: Hybrid Retrieval-Augmented Generation on Textual and Relational Knowledge Bases. arXiv preprint arXiv:2412.16311, 2024.

[70] Manzong Huang, Chenyang Bu, Yi He, Xingrui Zhuo, and Xindong Wu. Relink: Constructing Query-Driven Evidence Graph On-the-Fly for GraphRAG. arXiv preprint arXiv:2601.07192, 2026.

[71] Dzenan Hamzic, Florian Skopik, Max Landauer, Markus Wurzenberger, and Andreas Rauber. Beyond RAG for Cyber Threat Intelligence: A Systematic Evaluation of Graph-Based and Agentic Retrieval. arXiv preprint arXiv:2604.11419, 2026.

[72] Prasham Titiya, Rohit Khoja, Tomer Wolfson, Vivek Gupta, and Dan Roth. SAGE: Structure Aware Graph Expansion for Retrieval of Heterogeneous Data. arXiv preprint arXiv:2602.16964, 2026.

[73] Jingjin Wang and Jiawei Han. PropRAG: Guiding Retrieval with Beam Search over Proposition Paths. arXiv preprint arXiv:2504.18070, 2025.

[74] Yuqi Huang, Ning Liao, Kai Yang, Anning Hu, Shengchao Hu, Xiaoxing Wang, and Junchi Yan.

HELP: HyperNode Expansion and Logical Path-Guided Evidence Localization for Accurate and Efficient GraphRAG. arXiv preprint arXiv:2602.20926, 2026.

[75] Boyu Chen, Zirui Guo, Zidan Yang, Yuluo Chen, Junze Chen, Zhenghao Liu, Chuan Shi, and Cheng Yang. PathRAG: Pruning Graph-based Retrieval Augmented Generation with Relational Paths. arXiv preprint arXiv:2502.14902, 2025.

[76] Hairu Wang, Yuan Feng, Xike Xie, and S Kevin Zhou. Path Pooling: Training-Free Structure Enhancement for Efficient Knowledge Graph Retrieval-Augmented Generation. arXiv preprint arXiv:2503.05203, 2025.

[77] Xingyu Tan, Xiaoyang Wang, Qing Liu, Xiwei Xu, Xin Yuan, and Wenjie Zhang. Paths-over-Graph: Knowledge Graph Empowered Large Language Model Reasoning. arXiv preprint arXiv:2410.14211, 2024.

[78] Rong Fu, Yemin Wang, Tianxiang Xu, Yongtai Liu, Weizhi Tang, Wangyu Wu, Xiaowen Ma, and Simon Fong. S-Path-RAG: Semantic-Aware Shortest-Path Retrieval Augmented Generation for Multi-Hop Knowledge Graph Question Answering. arXiv preprint arXiv:2603.23512, 2026.

[79] Zijian Li, Qingyan Guo, Jiawei Shao, Lei Song, Jiang Bian, Jun Zhang, and Rui Wang. Graph Neural Network Enhanced Retrieval for Question Answering of LLMs. arXiv preprint arXiv:2406.06572, 2024.

[80] Vibhor Agrawal, Fay Wang, and Rishi Puri. Query-Aware Graph Neural Networks for Enhanced Retrieval-Augmented Generation. arXiv preprint arXiv:2508.05647, 2025.

[81] Wenyu Huang, Guancheng Zhou, Hongru Wang, Pavlos Vougiouklis, Mirella Lapata, and Jeff Z. Pan. Less is More: Making Smaller Language Models Competent Subgraph Retrievers for Multi-hop KGQA. arXiv preprint arXiv:2410.06121, 2024.

[82] Maya Arseven, Anette Frank, Beni Egressy, Johann Higl, and Moritz Plenz. GLM-RAG: Graph Language Models for Graph-Based Retrieval-Augmented Generation. arXiv preprint arXiv:2607.28397, 2026.

[83] Qizhi Wang. Democratizing GraphRAG: Linear, CPU-Only Graph Retrieval for Multi-Hop QA. arXiv preprint arXiv:2602.23372, 2025.

[84] Zhuoping Zhou, Davoud Ataee Tarzanagh, Sima Didari, Wenjun Hu, Baruch Gutow, Oxana Verkholyak, Masoud Faraki, Heng Hao, Hankyu Moon, and Seungjai Min. Query-Aware Flow Diffusion for Graph-Based RAG with Retrieval Guarantees. arXiv preprint arXiv:2605.18775, 2026.

[85] Kwun Hang Lau, Fangyuan Zhang, Boyu Ruan, Yingli Zhou, Qintian Guo, Ruiyuan Zhang, and Xiaofang Zhou. Breaking the Static Graph: Context-Aware Traversal for Robust Retrieval-Augmented Generation. arXiv preprint arXiv:2602.01965, 2026.

[86] Xinlong Xu and Yoshua Y. Li. EviReform: Evidence-Guided Query Reformulation for Multi-Hop Graph Retrieval. arXiv preprint arXiv:2608.13006, 2026.

[87] Bowen Jin, Chulin Xie, Jiawei Zhang, Kashob Kumar Roy, Yu Zhang, Suhang Wang, Yu Meng, and Jiawei Han. Graph Chain-of-Thought: Augmenting Large Language Models by Reasoning on Graphs. Annual Meeting of the Association for Computational Linguistics, 2024.

[88] Xingxuan Li, Ruochen Zhao, Yew Ken Chia, Bosheng Ding, Shafiq Joty, Soujanya Poria, and Lidong Bing. Chain-of-Knowledge: Grounding Large Language Models via Dynamic Knowledge Adapting over Heterogeneous Sources. arXiv preprint arXiv:2305.13269, 2023.

[89] L Chen, P Tong, Z Jin, Y Sun, J Ye, and H Xiong. Plan-on-Graph: Self-Correcting Adaptive Planning of Large Language Model on Knowledge Graphs. arXiv preprint arXiv:2410.23875, 2024.

[90] Grama Chethan. Beyond Vector Similarity: A Structural Analysis of Graph-Augmented Retrieval for Industrial Knowledge Graphs. arXiv preprint arXiv:2606.06003, 2026.

[91] Jiate Liu, Zebin Chen, Shaobo Qiao, et al. A2RAG: Adaptive Agentic Graph Retrieval for Cost-Aware and Reliable Reasoning. arXiv preprint arXiv:2601.21162, 2026.

[92] Dongyang Xie, Yao Tian, Hao Zhang, Yifei Yuan, Tieyun Qian, Ming Zhong, Jiawei Jiang, and Yuanyuan Zhu. DocNavRAG: Document-Structured Graph RAG with Stateful Evidence Construction for Complex Document Question Answering. arXiv preprint arXiv:2608.01565, 2026.

[93] Xiangjun Zai, Xingyu Tan, Xiaoyang Wang, Qing Liu, Xiwei Xu, and Wenjie Zhang. PRoH: Dynamic Planning and Reasoning over Knowledge Hypergraphs for Retrieval-Augmented Generation. arXiv preprint arXiv:2510.12434, 2025.

[94] Yucan Guo, Miao Su, Saiping Guan, Zihao Sun, Xiaolong Jin, Jiafeng Guo, and Xueqi Cheng. RouteRAG: Efficient Retrieval-Augmented Generation from Text and Graph via Reinforcement Learning. arXiv preprint arXiv:2512.09487, 2025.

[95] Shu Wang, Yixiang Fang, Yingli Zhou, Xilin Liu, and Yuchi Ma. Archrag: Attributed community-based hierarchical retrieval-augmented generation. arXiv preprint arXiv:2502.09891, 2025.

[96] Boyoung Kim, Dosung Lee, Sumin An, Jinseong Jeong, and Paul Hongsuck Seo. ReTAG: Retrieval-Enhanced, Topic-Augmented Graph-Based Global Sensemaking. arXiv preprint arXiv:2509.25814, 2025.

[97] Jakir Hossain and Ahmet Erdem Sarıyüce. Core-based Hierarchies for Efficient GraphRAG. arXiv preprint arXiv:2603.05207, 2026.

[98] Yaoze Zhang, Rong Wu, Pinlong Cai, Xiaoman Wang, Guohang Yan, Song Mao, Ding Wang, and Botian Shi. LeanRAG: Knowledge-Graph-Based Generation with Semantic Aggregation and Hierarchical Retrieval. arXiv preprint arXiv:2508.10391, 2025.

[99] Yuejie Li, Ke Yang, Tao Wang, Bolin Chen, Bowen Li, and Chengjun Mao. Deep GraphRAG: A Balanced Approach to Hierarchical Retrieval and Adaptive Integration. arXiv preprint arXiv:2601.11144, 2026.

[100] Shu Wang, Yingli Zhou, and Yixiang Fang. BookRAG: A Hierarchical Structure-aware Index-based Approach for Retrieval-Augmented Generation on Complex Documents. arXiv preprint arXiv:2512.03413, 2025.

[101] Ziwen Zhao and Menglin Yang. Hierarchical Abstract Tree for Cross-Document Retrieval-Augmented Generation. arXiv preprint arXiv:2605.00529, 2026.

[102] Abdellah Ghassel, Ian Robinson, Gabriel Tanase, Hal Cooper, Bryan Thompson, Zhen Han, Vassilis N. Ioannidis, Soji Adeshina, and Huzefa Rangwala. Hierarchical Lexical Graph for Enhanced Multi-Hop Retrieval. arXiv preprint arXiv:2506.08074, 2025.

[103] Mengzhu Chen, Haodong Yang, Jia Cai, and Xiaolin Huang. FlexStructRAG: Flexible Structure-Aware Multi-Granular Relational Retrieval for RAG. arXiv preprint arXiv:2604.16312, 2026.

[104] Kounianhua Du, Jizheng Chen, Renting Rui, Huacan Chai, Lingyue Fu, Wei Xia, Yasheng Wang, Ruiming Tang, Yong Yu, and Weinan Zhang. CodeGRAG: Bridging the Gap between Natural Language and Programming Language via Graphical Retrieval Augmented Generation. arXiv preprint arXiv:2405.02355, 2024.

[105] Sungho Ko, Hyunjin Cho, Hyungjoo Chae, Jinyoung Yeo, and Dongha Lee. Evidence-Focused Fact Summarization for Knowledge-Augmented Zero-Shot Question Answering. arXiv preprint arXiv:2403.02966, 2024.

[106] Yizhe Yang, Heyan Huang, Yihang Liu, and Yang Gao. Graph vs. Sequence: An Empirical Study on Knowledge Forms for Knowledge-Grounded Dialogue. arXiv preprint arXiv:2312.07868, 2023.

[107] Yubin Hong, Chaofan Li, Jingyi Zhang, and Yingxia Shao. FG-RAG: Enhancing Query-Focused Summarization with Context-Aware Fine-Grained Graph RAG. arXiv preprint arXiv:2504.07103, 2025.

[108] KK Roy, PS Akash, KCC Chang, and L Popa. ConTReGen: Context-driven Tree-structured Retrieval for Open-domain Long-form Text Generation. arXiv preprint arXiv:2410.15511, 2024.

[109] Hang Ding, Yilun Zhao, Tiansheng Hu, Manasi Patwardhan, and Arman Cohan. SciRAG: Adaptive, Citation-Aware, and Outline-Guided Retrieval and Synthesis for Scientific Literature. arXiv preprint arXiv:2511.14362, 2025.

[110] Shuai Wang, Xixi Wang, and Yinan Yu. Topology-Aware Reasoning over Incomplete Knowledge Graph with Graph-Based Soft Prompting. arXiv preprint arXiv:2604.12503, 2026.

[111] Hongbo Zhang, Chen Tang, Tyler Loakman, Bohao Yang, Stefan Goetze, and Chenghua Lin. CADGE: Context-Aware Dialogue Generation Enhanced with Graph-Structured Knowledge Aggregation. arXiv preprint arXiv:2305.06294, 2023.

[112] Parag Jain and Mirella Lapata. Integrating Large Language Models with Graph-based Reasoning for Conversational Question Answering. arXiv preprint arXiv:2407.09506, 2024.

[113] Zafar Ali, Asad Khan, Aalia Malik, and Pavlos Kefalas. GraphLoom: Reliability-Calibrated Graph Evidence Routing for Multimodal KG-RAG. arXiv preprint arXiv:2608.15056, 2026.

[114] Jinyoung Park, Minseok Joo, Joo-Kyung Kim, and Hyunwoo J. Kim. Generative Subgraph Retrieval

for Knowledge Graph-Grounded Dialog Generation. arXiv preprint arXiv:2410.09350, 2024.

[115] Longchao Da, Parth Mitesh Shah, Kuan-Ru Liou, Jiaxing Zhang, and Hua Wei. GE-Chat: A Graph Enhanced RAG Framework for Evidential Response Generation of LLMs. arXiv preprint arXiv:2505.10143, 2025.

[116] Zerui Chen, Qinggang Zhang, Zhishang Xiang, Zhimin Wei, Linfeng Gao, Xiao Huang, Zhihong Zhang, and Jinsong Su. LegalGraphRAG: Multi-Agent Graph Retrieval-Augmented Generation for Reliable Legal Reasoning. arXiv preprint arXiv:2605.28120, 2026.

[117] Severin Ye, Xiao Kong, Xiaopeng He, Guangsu Yan, and Dongsuk Oh. CuraView: A Multi-Agent Framework for Medical Hallucination Detection with GraphRAG-Enhanced Knowledge Verification. arXiv preprint arXiv:2605.03476, 2026.

[118] Jianru Shen. Evidence Graph Consistency in Retrieval-Augmented Generation: A Model-Dependent Analysis of Hallucination Detection. arXiv preprint arXiv:2606.06748, 2026.

[119] Sergiy Lunyakin. Auditable by Construction: An Ontology-Driven Framework for Trustworthy LLM Analytics in Enterprise Finance. arXiv preprint arXiv:2608.20661, 2026.

[120] Jahidul Arafat. Citation-Grounded Code Comprehension: Preventing LLM Hallucination Through Hybrid Retrieval and Graph-Augmented Context. arXiv preprint arXiv:2512.12117, 2025.

[121] Riccardo Terrenzi, Maximilian von Zastrow, and Serkan Ayvaz. Why Neighborhoods Matter: Traversal Context and Provenance in Agentic GraphRAG. arXiv preprint arXiv:2605.15109, 2026.

[122] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural Questions: A Benchmark for Question Answering Research. *Transactions of the Association for Computational Linguistics*, 2019.

[123] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2018.

[124] Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. MuSiQue: Multihop Questions via Single-hop Question Composition. *Transactions of the Association for Computational Linguistics*, 2022.

[125] Wen tau Yih, Matthew Richardson, Christopher Meek, Ming-Wei Chang, and Jina Suh. The Value of Semantic Parse Labeling for Knowledge Base Question Answering. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2016.

[126] Pengyu Wang, Benfeng Xu, Licheng Zhang, Shaohan Wang, Mingxuan Du, Chiwei Zhu, and Zhendong Mao. WildGraphBench: Benchmarking GraphRAG with Wild-Source Corpora. arXiv preprint arXiv:2602.02053, 2026.

[127] Yilin Xiao, Junnan Dong, Chuang Zhou, Su Dong, Qian wen Zhang, Di Yin, Xing Sun, and Xiao Huang. GraphRAG-Bench: Challenging Domain-Specific Reasoning for Evaluating Graph Retrieval-Augmented Generation. arXiv preprint arXiv:2506.02404, 2025.

[128] S Wu, S Zhao, M Yasunaga, K Huang, K Cao, Q Huang, et al. STaRK: Benchmarking LLM Retrieval on Textual and Relational Knowledge Bases. arXiv preprint arXiv:2404.13207, 2024.

[129] C Xu, Q Chen, Y Feng, and G Cheng. mmRAG: A Modular Benchmark for Retrieval-Augmented Generation over Text, Tables, and Knowledge Graphs. arXiv preprint arXiv:2505.11180, 2025.

[130] Xiaochen Wang, Bao Hoang, Han Liu, Ting Wang, and Fenglong Ma. MKG-RAG-Bench: Benchmarking Retrieval in Multimodal Knowledge Graph-Augmented Generation. arXiv preprint arXiv:2606.26458, 2026.

[131] Sicheng Dong, Vahid Zolfaghari, Nenad Petrovic, and Alois Knoll. Knowledge-Graph Based RAG System Evaluation Framework. arXiv preprint arXiv:2510.02549, 2025.

[132] Qiming Zeng, Xiao Yan, Hao Luo, Yuhao Lin, Yuxiang Wang, Fangcheng Fu, Bo Du, Quanqing Xu, and Jiawei Jiang. How Significant Are the Real Performance Gains? An Unbiased Evaluation Framework for GraphRAG. arXiv preprint arXiv:2506.06331, 2025.

[133] Mihir Athale and Vishal Vaddina. Knowledge Graph Based Repository-Level Code Generation. arXiv preprint arXiv:2505.14394, 2025.

[134] Junde Wu, Jiayuan Zhu, Yunli Qi, Jingkun Chen, Min Xu, Filippo Menolascina, and Vicente Grau. Medical graphrag: Towards safe medical large language model via graph retrieval-augmented generation. arXiv preprint arXiv:2408.04187, 2024.

[135] Madhulatha Mandarapu and Sandeep Kunkunuru. Knowledge-Graph Grounding Helps LLMs Only for Out-of-Training Knowledge: A Controlled Study on Clinical Question Answering. arXiv preprint arXiv:2606.22419, 2026.

[136] Shashwat Sourav, Viktoriia Baibakova, Sanjay Das, Ran Elgedawy, Maria Mahbub, Emily Herron, and Tirthankar Ghosal. The Compressive Knowledge Graph Hypothesis: Which Graph Facts Matter for Scientific Hypothesis Generation? arXiv preprint arXiv:2605.27176, 2026.

[137] Qiao Xiao, Hong Ting Tsang, and Jiaxin Bai. TERAG: Token-Efficient Graph-Based Retrieval-Augmented Generation. arXiv preprint arXiv:2509.18667, 2025.

[138] Congmin Min, Sahil Bansal, Joyce Pan, Abbas Keshavarzi, Rhea Mathew, and Amar Viswanathan Kannan. Towards Practical GraphRAG: Efficient Knowledge Graph Construction and Hybrid Retrieval at Scale. arXiv preprint arXiv:2507.03226, 2025.

[139] Yikuan Hu, Jifeng Zhu, Lanrui Tang, and Chen Huang. ReMindRAG: Low-Cost LLM-Guided Knowledge Graph Traversal for Efficient RAG. arXiv preprint arXiv:2510.13193, 2025.

[140] Dongzhe Fan, Chuanhao Ji, Zimu Wang, Tong Chen, and Qiaoyu Tan. GraphRAG-Router: Learning Cost-Efficient Routing over GraphRAGs and LLMs with Reinforcement Learning. arXiv preprint arXiv:2604.16401, 2026.

[141] Hairu Wang, Yuan Feng, Yukun Cao, Xike Xie, and S Kevin Zhou. SkewRoute: Training-Free LLM Routing for Knowledge Graph Retrieval-Augmented Generation via Score Skewness of Retrieved Context. arXiv preprint arXiv:2505.23841, 2025.

[142] Pengyu Wang, Benfeng Xu, Shaohan Wang, Mingxuan Du, Xin Zeng, Huarui Wu, Lei Zhang, and Licheng Zhang. BM25 Wins at Scale: A Scaling Study of Retrieval-Augmented Generation Paradigms. arXiv preprint arXiv:2607.26497, 2026.

[143] Zhili Shen, Chenxin Diao, Pascual Merita, Pavlos Vougiouklis, and Jeff Z. Pan. Millions of GeAR-s: Extending GraphRAG to Millions of Documents. arXiv preprint arXiv:2507.17399, 2025.

[144] Dongzhuoran Zhou, Yuqicheng Zhu, Xiaxia Wang, Hongkuan Zhou, Yuan He, Jiaoyan Chen, Steffen Staab, and Evgeny Kharlamov. Evaluating Knowledge Graph Based Retrieval Augmented Generation Methods under Knowledge Incompleteness. arXiv preprint arXiv:2504.05163, 2025.

[145] Hang Wang, Hang Dong, Lu Liu, and Chuanru Ren. MissDiag: Diagnostic Evaluation of Incomplete-Knowledge Robustness in KGQA and KG-RAG. arXiv preprint arXiv:2608.18489, 2026.

[146] Dongzhuoran Zhou, Yuqicheng Zhu, Xiaxia Wang, Hongkuan Zhou, Yuan He, Jiaoyan Chen, Steffen Staab, and Evgeny Kharlamov. What Breaks Knowledge Graph based RAG? Benchmarking and Empirical Insights into Reasoning under Incomplete Knowledge. arXiv preprint arXiv:2508.08344, 2025.

[147] Yizhuo Ma, Jinchuan Xu, Tao Wen, Qizhi Chen, Jiakai Li, Rongzheng Wang, Muquan Li, Shuang Liang, and Ke Qin. Toward Robust GraphRAG: Mitigating Retrieval Drift and Hallucination from Imperfect Knowledge Graphs. arXiv preprint arXiv:2603.14828, 2026.

[148] Zhefan Wang, Huanjun Kong, Jie Ying, Wanli Ouyang, and Nanqing Dong. ROGRAG: A Robustly Optimized GraphRAG Framework. arXiv preprint arXiv:2503.06474, 2025.

[149] Yao Xu, Shizhu He, Jiabei Chen, Zihao Wang, Yangqiu Song, Hanghang Tong, Guang Liu, Kang Liu, and Jun Zhao. Generate-on-Graph: Treat LLM as both Agent and KG in Incomplete Knowledge Graph Question Answering. arXiv preprint arXiv:2404.14741, 2024.

[150] Jungyeon Lee, Kangmin Lee, and Taeuk Kim. MAGIC: A Multi-Hop and Graph-Based Benchmark for Inter-Context Conflicts in Retrieval-Augmented Generation. arXiv preprint arXiv:2507.21544, 2025.

[151] Shuyi Liu, Yuming Shang, and Xi Zhang. TruthfulRAG: Resolving Factual-level Conflicts in Retrieval-Augmented Generation with Knowledge Graphs. arXiv preprint arXiv:2511.10375, 2025.

[152] Jing Ren, Bowen Li, Ziqi Xu, Xikun Zhang, Haytham Fayek, and Xiaodong Li. When to Trust: A Causality-Aware Calibration Framework for Accurate Knowledge Graph Retrieval-Augmented Generation. arXiv preprint arXiv:2601.09241, 2026.

[153] Dong Li, Yichen Niu, Ying Ai, Xiang Zou, Biqing Qi, and Jianxing Liu. T-GRAG: A Dynamic

GraphRAG Framework for Resolving Temporal Conflicts and Redundancy in Knowledge Retrieval. arXiv preprint arXiv:2508.01680, 2025.

[154] Zulun Zhu, Haoyu Liu, Mengke He, and Siqiang Luo. Right Answer at the Right Time - Temporal Retrieval-Augmented Generation via Graph Summarization. arXiv preprint arXiv:2510.16715, 2025.

[155] X Qian, Y Zhang, Y Zhao, B Zhou, X Sui, and X Yuan. Plan of Knowledge: Retrieval-Augmented Large Language Models for Temporal Knowledge Graph Question Answering. arXiv preprint arXiv:2511.04072, 2025.

[156] Xiaoman Wang, Yaoze Zhang, Wenzhuo Fan, Hongwei Zhang, Ding Wang, Guohang Yan, Song Mao, Botian Shi, Yunshi Lan, and Pinlong Cai. IA-RAG: Interval-Algebra-Driven Temporal Reasoning for Dynamic Knowledge Retrieval. arXiv preprint arXiv:2606.06044, 2026.

[157] Qingyun Sun, Jiaqi Yuan, Shan He, Xiao Guan, Haonan Yuan, Xingcheng Fu, Jianxin Li, and Philip S. Yu. DyG-RAG: Dynamic Graph Retrieval-Augmented Generation with Event-Centric Reasoning. arXiv preprint arXiv:2507.13396, 2025.

[158] ZY Zhang, Z Li, Y Li, B Ding, and BKH Low. Respecting Temporal-Causal Consistency: Entity-Event Knowledge Graphs for Retrieval-Augmented Generation. arXiv preprint arXiv:2506.05939, 2025.

[159] Ali Sarabadani and Mahtab Tajvidiyan. DYNA : Dynamic Episodic Memory Networks for Augmenting Large Language Models with Temporal Knowledge Graphs in Continuous Learning. arXiv preprint arXiv:2606.15778, 2026.

[160] Tianzhe Zhao, Jiaoyan Chen, Yanchi Ru, Haiping Zhu, Nan Hu, Jun Liu, and Qika Lin. RAG Safety: Exploring Knowledge Poisoning Attacks to Retrieval-Augmented Generation. arXiv preprint arXiv:2507.08862, 2025.

[161] Jiacheng Liang, Yuhui Wang, Changjiang Li, Rongyi Zhu, Tanqiu Jiang, Neil Gong, and Ting Wang. GraphRAG under Fire. arXiv preprint arXiv:2501.14050, 2025.

[162] Qizhi Chen, Chao Qi, Yihong Huang, Muquan Li, Rongzheng Wang, Dongyang Zhang, Ke Qin, and Shuang Liang. KEPo: Knowledge Evolution Poison on Graph-based Retrieval-Augmented Generation. arXiv preprint arXiv:2603.11501, 2026.

[163] Jiayi Wen, Tianxin Chen, Zhirun Zheng, and Cheng Huang. A Few Words Can Distort Graphs: Knowledge Poisoning Attacks on Graph-based Retrieval-Augmented Generation of Large Language Models. arXiv preprint arXiv:2508.04276, 2025.

[164] Yilin Xiao, Jin Chen, Qinggang Zhang, Yujing Zhang, Chuang Zhou, Longhao Yang, Lingfei Ren, Xin Yang, and Xiao Huang. LogicPoison: Logical Attacks on Graph Retrieval-Augmented Generation. arXiv preprint arXiv:2604.02954, 2026.

[165] Jiale Liu, Jiahao Zhang, and Suhang Wang. Exposing Privacy Risks in Graph Retrieval-Augmented Generation. arXiv preprint arXiv:2508.17222, 2025.

[166] Jinze Gu, Qinghua Mao, Xi Lin, and Jun Wu. GraphSteal: Structural Knowledge Stealing from Graph RAG via Traversal Reconstruction. arXiv preprint arXiv:2605.28645, 2026.

[167] Shuhua Yang, Jiahao Zhang, Yilong Wang, Dongwon Lee, and Suhang Wang. Query-Efficient Agentic Graph Extraction Attacks on GraphRAG Systems. arXiv preprint arXiv:2601.14662, 2026.