# OG-CLAIMWIKI: OPTIMIZING KNOWLEDGE ORGANIZATION FOR LLM AGENTS THROUGH EVIDENCE-CHAIN PROBING

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

LLM agents often need to retrieve and connect multiple facts into sufficient evidence chains to answer complex questions. Knowledge organization affects whether agents can find these facts and establish the connections needed to support an answer. However, knowledge bases built using predefined rules or heuristic strategies may retain relevant facts without making these chains easy for agents to assemble. We introduce Offline-Grounded ClaimWiki (OG-ClaimWiki), a framework that uses evidence-chain probing to optimize knowledge organization before real user queries are available. OG-ClaimWiki represents source knowledge as verifiable atomic Claims and connects them through source-supported relations to form evidence chains. These chains guide probe generation and specify the evidence required for each answer. Comparing this evidence with the agent's access trace helps locate defects in knowledge content, search indexes, page links, and context organization. Candidate edits are verified against the sources and evaluated on the original probes, independent questions, and regression questions. Only validated edits are incorporated into the Wiki. Experiments on four document QA datasets and two code benchmarks demonstrate consistent improvements over the evaluated baselines. Offline probing increases document macro-average F1 by 2.6–4.3 points over the initial Wiki across answering models. OG-ClaimWiki also outperforms the strongest baselines in document and code QA accuracy after source changes.

## 1 INTRODUCTION

Knowledge bases provide LLM agents with external knowledge for question answering (Lewis et al., 2020). Complex questions often require agents to combine facts from different documents or code locations and reason over their relations, forming evidence chains that support the answers. Prior work aggregates content and establishes relations to make distributed knowledge easier to retrieve and use (Sarthi et al., 2024; Edge et al., 2024; Ming et al., 2026; Hoang et al., 2026). However, predefined structures may not support the combinations of evidence required by different questions. Even when relevant facts are retained, agents may struggle to retrieve and connect them into sufficient evidence chains.

Research on knowledge-base improvement has used construction checks to correct content and structural errors and task feedback to refine knowledge organization (Ming et al., 2026; Tsang et al., 2026). Feedback from actual queries can reveal problems during use, but it requires those queries to become available (Zhao et al., 2025; Huang et al., 2026). Diagnostic probing has also been used to identify facts omitted during knowledge-base construction (Huerta, 2026). Yet fact retention and answer correctness alone do not establish whether an agent can retrieve and connect the facts needed for an answer or indicate which parts of the knowledge organization need revision. We study how to obtain diagnostic feedback before real user queries are available, using it to identify organizational problems that prevent sufficient evidence acquisition and guide local edits.

We introduce Offline-Grounded ClaimWiki (OG-ClaimWiki), which optimizes knowledge organization through evidence-chain probing. Source-grounded atomic statements, called Claims, serve as the basic units for fact verification and local updates. Relations supported by the sources connect



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

Claims into evidence chains that jointly support an answer. Each chain guides the generation of a probe question and specifies the facts and relations needed to answer it, providing a reference for diagnosing the agent's evidence acquisition.

The agent answers these probes using the current Wiki, and its access traces record the evidence obtained through search and reading. OG-ClaimWiki compares the evidence chains with these traces to identify problems in knowledge content, search indexes, page links, and context organization and propose local edits. Candidate edits are verified against the sources and evaluated on the original probes, independent questions, and regression questions. Validated edits are incorporated into the Wiki for subsequent probing and question answering.

Our contributions are as follows:

1. We introduce OG-ClaimWiki, a pre-deployment framework that uses evidence-chain probing to obtain feedback and improve knowledge organization for sufficient evidence acquisition before real user queries are available.
2. We develop a mechanism for local diagnosis and editing based on source evidence and agent access traces. Source verification, validation on independent questions, and regression checks determine which edits are retained.
3. We demonstrate improvements over the evaluated baselines across answering models and document and code QA tasks, additional gains from offline probing, and effective knowledge updates following source changes.

## 2 Related Work

**Knowledge Organization for QA.** Knowledge organization methods support evidence retrieval through passages, hierarchies, graphs, and pages. Passage retrieval and source-code search match queries to relevant content (Lewis et al., 2020; Wang et al., 2026). RAPTOR builds hierarchies through recursive clustering and summarization, while GraphRAG, LightRAG, and HippoRAG2 organize and retrieve knowledge through entities and their relations (Sarthi et al., 2024; Edge et al., 2024; Guo et al., 2025; Gutiérrez et al., 2025). In code repositories, LocAgent, CodexGraph, and RepoGraph use code symbols and structural relations to locate relevant code (Chen et al., 2025; Liu et al., 2025; Ouyang et al., 2025). Another approach organizes knowledge into linked pages or reusable documentation. Wiki-based methods organize sources into pages and links, while RepoAgent, CodeWiki, and RepoDoc generate repository documentation (Karpathy, 2026; Ming et al., 2026; Luo et al., 2024; Hoang et al., 2026; Xu et al., 2026). AutoGraph-R1 further uses downstream retrieval and reasoning rewards to learn a graph-construction policy (Tsang et al., 2026). These methods primarily organize or construct knowledge for retrieval and reasoning. OG-ClaimWiki focuses on optimizing an existing knowledge organization according to the evidence acquired during answering.

**Automatic Question Generation.** Automatic question generation supports evaluation by broadening question coverage and varying reasoning demands. For example, RAGEval generates scenario-specific RAG evaluation data (Zhu et al., 2025), while AutoEvolve and GRADE use graph structures to evolve queries or generate multi-hop questions (Zhang et al., 2025a; Lee et al., 2025). These studies generate questions primarily for evaluation, while we use questions and their supporting source evidence to diagnose and improve knowledge organization.

**Knowledge Diagnosis and Updating.** Knowledge diagnosis and updating use construction checks or question-answering feedback to identify and repair problems. LLM-Wiki records errors in an Error Book to guide page repairs during construction (Ming et al., 2026). During interaction, TRAIL and DeepRefine use feedback from existing queries to modify knowledge bases. TRAIL verifies and updates graphs during query reasoning, while DeepRefine generates knowledge edits from interaction histories and learns an editing policy (Zhao et al., 2025; Huang et al., 2026). Diagnostic probing provides another source of feedback. WiCER uses diagnostic questions about individual source documents to identify omitted facts and preserve them during recompilation (Huerta, 2026). Its focus is fact retention during compilation, whereas OG-ClaimWiki examines how knowledge organization supports agents in retrieving and connecting sufficient evidence for an answer.

# 3 OG-CLAIMWIKI

## 3.1 OVERVIEW

The source collection $X$ consists of documents, tables, and code repositories. As shown in Figure 1, OG-ClaimWiki builds an initial Wiki from $X$ and refines it through evidence-chain probing before real user queries are available. The system uses the resulting access traces to diagnose failures and propose local edits, which are committed only after validation. The refined Wiki then supports online QA, where an agent searches, reads, and navigates the Wiki to produce answers with source citations.

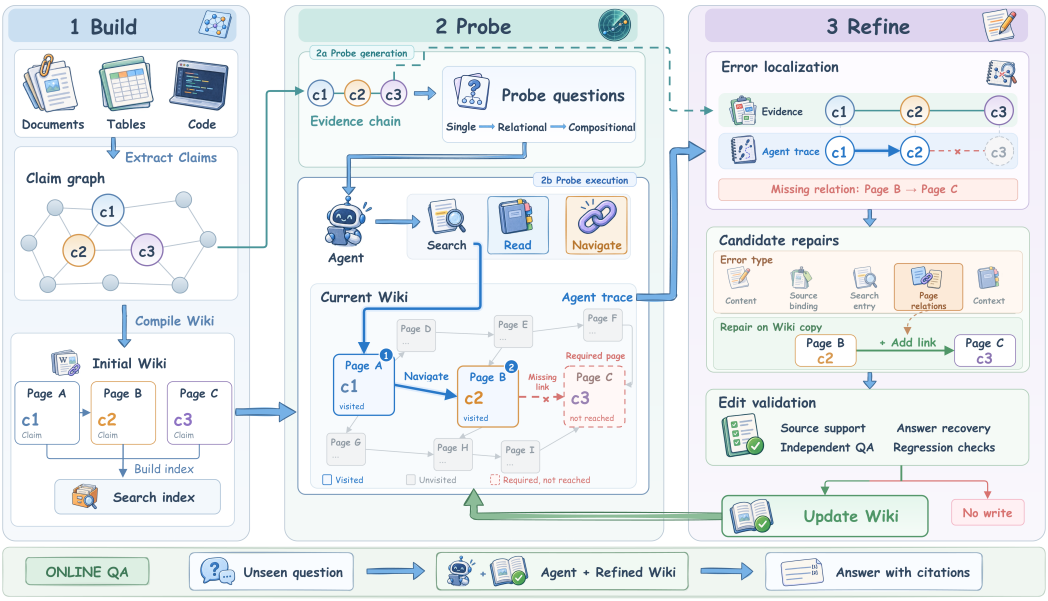


Figure 1: **Overview of OG-ClaimWiki.** (1) **Build.** Extract source-grounded Claims, construct the Claim graph, and compile the Wiki. (2) **Probe.** Generate questions from evidence chains and record the agent's access traces during answering. (3) **Refine.** Compare evidence chains with access traces to locate defects, propose local edits, and commit only validated edits. The refined Wiki supports further probing and online QA.

## 3.2 CLAIMWIKI

### 3.2.1 WIKI-STRUCTURED REPRESENTATION

A source Claim is a verifiable atomic statement linked to its source location and version:

$$c = (s_c, \ell_c, v_c),$$

where $s_c$ is the statement, $\ell_c$ is its location in a document, table, or code repository, and $v_c$ is the source version. After compilation, the system assigns $c$ to a Wiki page or code entry, denoted by $p_c$. A sentence containing multiple facts is split into separate Claims, each retaining its source and version.

The Wiki is a structured retrieval layer between the sources and the answering agent. It contains generated document pages, code entries, source references, verified links, and a search index. The system organizes Claims according to the source type. For documents, it groups Claims about the same entity or topic on one Wiki page, whose title identifies that entity or topic. For code, it groups Claims about the same symbol, configuration, test, or module in one code entry. A Wiki page is a generated container for related Claims, not a page copied from the source document.

These source Claims form the node set $V^X$, and source-supported relations form the edge set $E^X$, giving the Claim graph $G^X = (V^X, E^X)$. The graph supports evidence-chain construction and

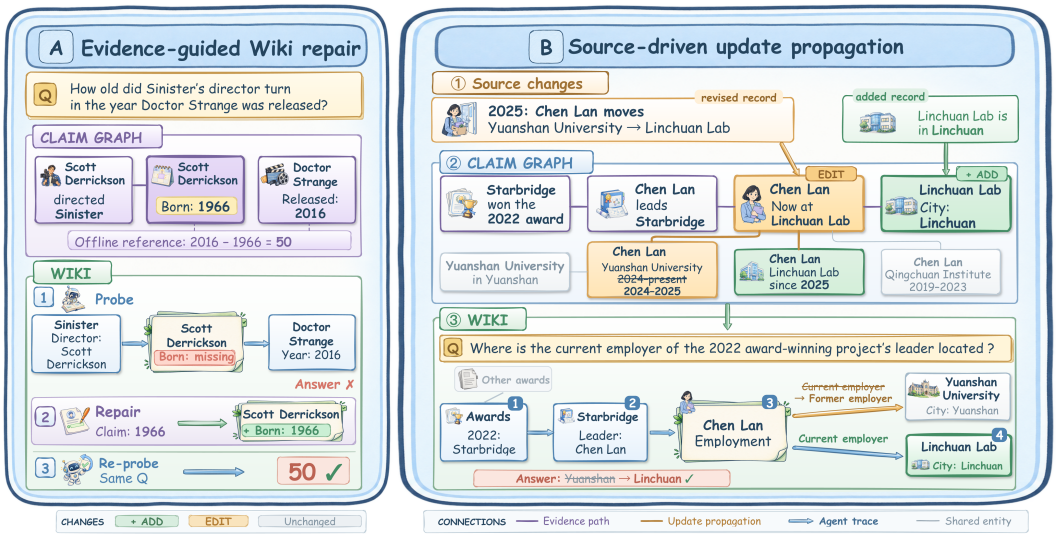


Figure 2: Examples of Wiki repair and update propagation. (A) Probing reveals a missing birth-year Claim. (B) Source changes propagate to employment Claims and Wiki pages.

source verification, while the Wiki provides the pages and entries accessed by the agent. Document edges capture relations involving entities, time, tables, or explicit links. Code edges capture program structure and behavior. Implementation details appear in Appendix A.

### 3.2.2 Evidence-Chain Probing

The system constructs evidence chains by enumerating ordered paths in the Claim graph:

$$\pi_j = \langle c_1, r_1, c_2, \ldots, r_{m-1}, c_m \rangle,$$

where $c_i$ is a Claim and $r_i = (c_i, c_{i+1})$ is a source-supported relation between adjacent Claims. A path starts from a retrievable Claim and ends when its terminal Claim determines a unique answer. Let $p_i = p_{c_i}$ denote the Wiki unit containing $c_i$. Each evidence chain is

$$\kappa_j = (\pi_j, P_{\pi_j}, I_{\pi_j}, y_j),$$

where $P_{\pi_j}$ is the sequence of Wiki units, $I_{\pi_j}$ contains the search terms, and $y_j$ is the expected answer. For each $\kappa_j$, the system generates a probe question $q_j$. Single-Claim, relational, and compositional probes test fact retrieval, cross-unit connections, and multi-Claim reasoning. Evidence chains and expected answers are used only for offline probing and validation.

For multi-hop questions, intermediate entities or code symbols are omitted so that the agent must discover them in the Wiki. A verifier checks the expected answer and supporting evidence against the original sources. To reduce interference from existing model knowledge, each model answers the questions in several independent attempts without the knowledge base or tools. Only questions answered incorrectly in every attempt by every model are retained. Appendix E provides prompt templates, and Appendix F illustrates probing and local repair with document and code examples.

Probing proceeds from single-Claim to relational and then compositional questions. Single-Claim probes test fact retention and retrieval. Relational probes test whether evidence across pages or code entries can be connected through source-supported relations. Compositional probes test whether the agent can acquire and combine at least three Claims. As probing proceeds, validated edits update the Wiki for subsequent probes. Resolving local defects first allows later probes to assess cross-page links and knowledge composition with less repeated testing of the same defect.

### 3.2.3 Diagnosis and Validated Repair

Each probe runs in a fresh session using the same Wiki search, reading, and navigation interfaces as online QA. The agent receives only the question and tool outputs. Its access trace records search and reading actions, retrieved content, and answer citations. The system compares the answer with

the expected answer and checks whether the agent obtained the required Claims. If either check fails, it compares the trace with source-supported evidence chains to identify incomplete or inaccurate Claims and failures in retrieval or cross-page navigation. Based on this diagnosis, the system proposes local edits to Claims, source bindings, search indexes, or page links and tests them on a Wiki copy.

For example, in Figure 2A, the agent reaches Scott Derrickson's page, but the birth year available in the Claim graph is absent from the Wiki. Adding this source-supported Claim restores the correct answer upon re-probing. Even when the agent has obtained and cited all necessary Claims, it may still answer incorrectly. In such cases, an LLM proposes a derived Claim based on these facts and their relations. Candidates that restore a correct answer proceed to validation. When a question involves multiple defects, the system edits a Wiki copy incrementally and retests after each edit. Once the question is answered correctly, the edits are validated together. If the failure remains unresolved or no verifiable edit is available, the Wiki remains unchanged.

Candidate edits are evaluated on a Wiki copy and committed only after passing all four checks:

- **Source verification:** New or updated Claims and relations must be supported by current sources. Relation endpoints must be verifiable, and recorded source locations must point to the corresponding content in the current version. Derived Claims must follow from their recorded premises.
- **Answer recovery:** The agent must consistently answer the original probe correctly in fresh sessions.
- **Validation on additional questions:** Using the edited Wiki, the agent must correctly answer all validation questions concerning the affected knowledge but requiring different Claims or evidence chains from the original probe.
- **Regression check:** Previously correct answers to related and unrelated questions must remain correct.

#### 3.2.4 Dependency-Aware Updating

Even after probing and repair, the Wiki can become outdated as its sources change. Changes to source facts may also require revising derived conclusions and related links. The system therefore updates changed facts and checks dependent Wiki content to keep it consistent with the new sources. As shown in Figure 2B, a change in current employment may also require revising the previous employment period and related Wiki links.

Let $X^t$ denote the $t$th version of the source collection. When the sources change from $X^t$ to $X^{t+1}$, the system identifies changes by matching old and new Claims. Relocations require updating the recorded source locations. Changed content is re-extracted as Claims, new content is added to the Wiki, and Claims corresponding to deleted content are marked invalid. The system then checks whether conclusions and Wiki content that depend on changed Claims remain supported by the new sources and revises content that is no longer valid. All edits undergo the validation procedure above before being committed to the Wiki.

## 4 Experiments

### 4.1 Experimental Setup

**Benchmarks.** For **Document QA**, we evaluate HotpotQA (Yang et al., 2018), MuSiQue (Trivedi et al., 2022), 2WikiMultiHopQA (Ho et al., 2020), and HybridQA (Chen et al., 2020). We select 1,000 test questions from each dataset, retrieve over the full corpus, and report EM and F1. For **Code QA**, SWE-QA (Peng et al., 2026) contains 720 questions from 15 repositories, and SWE-QA-Pro (Cai et al., 2026) contains 260 questions from 26 repositories. We report Overall and Score, respectively.

**Baselines.** Document baselines include BM25 (Robertson & Zaragoza, 2009), Dense RAG (Lewis et al., 2020), RAPTOR (Sarthi et al., 2024), GraphRAG (Edge et al., 2024), LightRAG (Guo et al., 2025), HippoRAG2 (Gutiérrez et al., 2025), and LLM-Wiki (Ming et al., 2026). Code baselines include Agentic Grep (ripgrep contributors, 2026), Function Chunking (Peng et al., 2026), RepoA-

gent (Luo et al., 2024), CodeWiki (Hoang et al., 2026), Raw MCP Graph (DeusData, 2026), and RepoDoc (Xu et al., 2026).

**Models.** Answering models include Qwen3.6-35B-A3B (Qwen Team, 2026), DeepSeek-V4-Pro (DeepSeek-AI, 2026), Gemini 3.5 Flash (Google DeepMind, 2026), GPT-5.6 Sol (OpenAI, 2026), and Claude Opus 5 (Anthropic, 2026a). The main tables report three models, with complete results in Appendix D.1. Claude Sonnet 5 (Anthropic, 2026b) generates probes, and GPT-5.6 Sol constructs and maintains the Wiki. Probing strategy analysis, component ablations, and source-update experiments use GPT-5.6 Sol as the answering model.

**Evaluation protocol.** Wiki construction and optimization do not use test questions. All methods use the same test samples and answering budgets, and the Wiki remains unchanged during evaluation. Data, model configurations, and evaluation details appear in Appendix B.

Table 1: **Document QA results (%).** Macro F1 is the equally weighted mean of F1 across the four datasets. Best and second-best results within each model are bold and underlined, respectively.

| Model | Method | HotpotQA | | MuSiQue | | 2WikiMultiHopQA | | HybridQA | | Macro F1 |
|---|---|---|---|---|---|---|---|---|---|---|
| | | EM | F1 | EM | F1 | EM | F1 | EM | F1 | |
| Qwen3.6 | BM25 | 63.2 | 71.0 | 45.3 | 53.5 | 70.5 | 78.5 | 46.0 | 55.7 | 64.7 |
| | Dense RAG | 65.4 | 75.3 | 49.8 | 57.6 | 73.9 | 82.4 | 51.4 | 60.8 | 69.0 |
| | RAPTOR | 67.3 | 75.9 | 44.1 | 52.2 | 68.2 | 74.0 | 53.1 | 63.4 | 66.4 |
| | GraphRAG | 68.1 | 75.5 | 52.1 | 62.3 | 69.4 | 75.4 | 52.3 | 60.7 | 68.5 |
| | LightRAG | 72.4 | 81.6 | 50.8 | 59.0 | 78.4 | 86.6 | 56.5 | 65.2 | 73.1 |
| | HippoRAG2 | 69.3 | 77.5 | 53.1 | 63.0 | 80.2 | 85.9 | 54.9 | 64.9 | 72.8 |
| | LLM-Wiki | 71.8 | 81.0 | 57.4 | 67.4 | 81.5 | 89.1 | 58.7 | 68.5 | 76.5 |
| | OG-ClaimWiki w/o probing | 70.4 | 80.2 | 62.4 | 71.5 | 82.3 | 90.3 | 58.3 | 68.0 | 77.5 |
| | **OG-ClaimWiki** | **75.4** | **84.1** +3.9 | **66.9** | **76.1** +4.6 | **84.3** | **92.6** +2.3 | **64.4** | **74.4** +6.4 | **81.8** +4.3 |
| GPT-5.6 | BM25 | 66.6 | 75.6 | 48.8 | 58.2 | 73.8 | 82.1 | 50.4 | 60.7 | 69.2 |
| | Dense RAG | 68.8 | 78.4 | 54.1 | 62.9 | 79.9 | 86.3 | 55.6 | 65.2 | 73.2 |
| | RAPTOR | 69.9 | 78.3 | 48.9 | 59.0 | 69.5 | 77.8 | 60.0 | 69.5 | 71.2 |
| | GraphRAG | 72.0 | 79.2 | 58.8 | 67.9 | 73.7 | 79.1 | 56.6 | 64.7 | 72.7 |
| | LightRAG | 73.5 | 82.3 | 56.3 | 65.5 | 86.2 | 92.4 | 57.6 | 67.3 | 76.9 |
| | HippoRAG2 | 74.2 | 81.9 | 60.4 | 68.6 | 81.4 | 88.4 | 66.2 | 74.8 | 78.4 |
| | LLM-Wiki | 79.1 | 86.2 | 63.8 | 73.0 | 86.0 | 92.1 | 63.1 | 71.7 | 80.8 |
| | OG-ClaimWiki w/o probing | 78.8 | 86.4 | 67.9 | 76.3 | 85.6 | 91.9 | 66.1 | 74.6 | 82.3 |
| | **OG-ClaimWiki** | **81.9** | **89.1** +2.7 | **71.4** | **80.0** +3.7 | **86.7** | **93.5** +1.6 | **70.2** | **79.0** +4.4 | **85.4** +3.1 |
| Claude Opus 5 | BM25 | 67.0 | 75.3 | 49.3 | 59.0 | 74.7 | 82.1 | 52.2 | 62.4 | 69.7 |
| | Dense RAG | 71.8 | 80.8 | 55.0 | 64.7 | 78.6 | 86.7 | 60.2 | 68.4 | 75.2 |
| | RAPTOR | 70.1 | 79.1 | 49.0 | 57.0 | 71.2 | 77.6 | 59.9 | 70.2 | 71.0 |
| | GraphRAG | 69.5 | 76.8 | 61.0 | 68.6 | 73.1 | 78.9 | 56.3 | 65.3 | 72.4 |
| | LightRAG | 76.0 | 83.5 | 56.4 | 65.9 | 84.2 | 91.0 | 61.6 | 69.8 | 77.6 |
| | HippoRAG2 | 72.8 | 82.4 | 61.3 | 69.4 | 82.7 | 89.7 | 60.2 | 71.0 | 78.1 |
| | LLM-Wiki | 75.6 | 84.2 | 64.8 | 73.8 | 85.6 | 93.4 | 67.4 | 75.3 | 81.7 |
| | OG-ClaimWiki w/o probing | 78.1 | 86.3 | 67.8 | 77.9 | 86.2 | 93.6 | 66.2 | 75.0 | 83.2 |
| | **OG-ClaimWiki** | **80.4** | **88.6** +2.3 | **71.8** | **81.1** +3.2 | **87.4** | **94.9** +1.3 | **70.5** | **78.6** +3.6 | **85.8** +2.6 |

### 4.2 Main Results

OG-ClaimWiki outperforms the evaluated baselines on document and code QA across all answering models (Tables 1 and 2). The gains from probing vary across document datasets. 2WikiMultiHopQA has the highest F1 before probing and gains the least, while MuSiQue and HybridQA gain more than HotpotQA and 2WikiMultiHopQA. MuSiQue requires combining evidence across reasoning steps, and HybridQA requires combining tables with linked text. Both tasks require evidence from different locations, and their larger gains suggest that probing helps models retrieve and combine this evidence more effectively.

LLM-Wiki achieves the highest document Macro F1 among the baselines, while RAPTOR and GraphRAG do not consistently outperform Dense RAG. Adding hierarchical or graph structure therefore does not necessarily improve QA. On code QA, direct source search remains competitive. Probing and repair further improve the code Wiki, allowing OG-ClaimWiki to lead on both

code benchmarks. Although baseline performance differs between document and code QA, probing brings further gains in both domains, showing that its benefits extend across source types.

Compared with the no-probing variant, probing improves document Macro F1 by 2.6–4.3 points across answering models. In both domains, models with lower initial scores generally gain more, while those with higher initial scores also improve. All answering models use the same refined Wiki, showing that the improvements benefit different models without requiring a separately optimized Wiki for each.

On the document QA test sets, gains are not limited to questions that share evidence with probes (Appendix D.4). For GPT-5.6 Sol on HotpotQA, MuSiQue, and 2WikiMultiHopQA, directly adding probe questions and answers improves F1 by 0.3 points on test questions that share no evidence with probes, compared with 1.7 points from diagnosis and repair (Appendix D.5). Repairing problems in knowledge organization thus improves performance on these test questions more than simply adding probe QA content.

Table 2: **Code QA results.** SWE-QA reports Overall (5–100), and SWE-QA-Pro reports Score (5–50).

| Method | Qwen3.6 | | GPT-5.6 Sol | | Claude Opus 5 | |
|---|---|---|---|---|---|---|
| | SWE-QA | SWE-QA-Pro | SWE-QA | SWE-QA-Pro | SWE-QA | SWE-QA-Pro |
| Agentic Grep | 60.2 | 32.9 | 66.7 | 37.3 | 67.4 | 36.8 |
| Function Chunking | 56.4 | 31.6 | 62.6 | 34.7 | 62.3 | 36.4 |
| RepoAgent | 58.3 | 32.2 | 63.6 | 36.9 | 64.3 | 37.1 |
| CodeWiki | 62.3 | 33.7 | 66.5 | 37.2 | 66.8 | 38.9 |
| Raw MCP Graph | 60.1 | 34.0 | 65.8 | 37.1 | 67.2 | 38.6 |
| RepoDoc | 61.4 | 35.6 | 67.0 | 38.2 | 68.5 | 38.8 |
| OG-ClaimWiki w/o probing | 62.0 | 35.4 | 68.5 | 38.8 | 68.2 | 39.8 |
| **OG-ClaimWiki** | **66.2** +4.2 | **38.1** +2.7 | **71.4** +2.9 | **40.7** +1.9 | **70.7** +2.5 | **41.4** +1.6 |

### 4.3 Probing Strategy Analysis

**Probe construction.** We compare three strategies with matched question counts and distributions of required Claim counts per dataset. Random-QG randomly samples source Claims to generate questions. Page-QG selects source pages or code files and follows page links to gather related content for question generation. Claim-QG selects and combines evidence through source-supported relations in the Claim graph. All strategies start from the same initial Wiki and use the same diagnosis and repair settings.

Page-QG outperforms Random-QG on document QA and SWE-QA, but not on SWE-QA-Pro, while Claim-QG performs best in all three settings (Table 3). Page-QG provides related pages but leaves the generator to decide which facts to combine into a question. Claim-QG uses explicit Claim relations to select and combine evidence, directing probes toward whether that evidence can jointly support an answer. After probing and repair with Claim-QG, 99.4% of previously correct document answers remain correct.

Table 3: **Effect of probe construction on document and code QA.** Ret. is the percentage of previously correct document answers that remain correct after probing.

| Strategy | Document QA | | | SWE-QA | | SWE-QA-Pro | |
|---|---|---|---|---|---|---|---|
| | Macro F1 | Δ | Ret. | Overall | Δ | Score | Δ |
| No probing | 82.3 | – | – | 68.5 | – | 38.8 | – |
| Random-QG | 83.0 | +0.7 | 99.0 | 69.3 | +0.8 | 39.9 | +1.1 |
| Page-QG | 84.2 | +1.9 | 98.8 | 70.1 | +1.6 | 39.6 | +0.8 |
| **Claim-QG (ours)** | **85.4** | **+3.1** | **99.4** | **71.4** | **+2.9** | **40.7** | **+1.9** |

**Probing budget.** Increasing the number of probing questions improves QA performance for all three strategies (Figure 3). At matched budgets, Claim-QG consistently outperforms Random-QG

and Page-QG. Across all three settings, Claim-QG with half the budget also achieves larger gains than either alternative with the full budget. Within the evaluated range, increasing question counts therefore does not close the gap between probe construction strategies. For Claim-QG, additional gains diminish beyond half the full budget, particularly on document QA and SWE-QA. In these two settings, half the probing budget achieves gains close to those of the full budget.

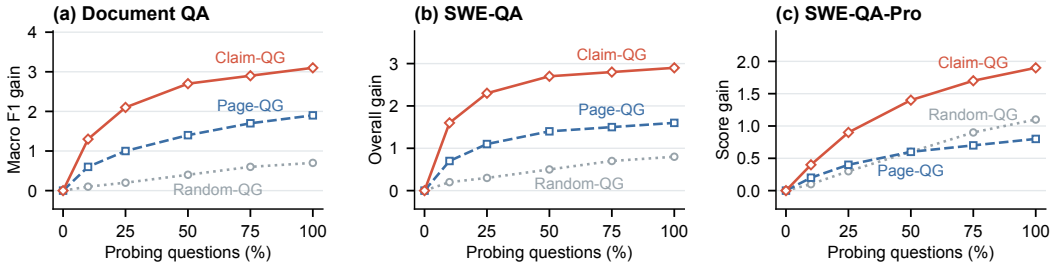


Figure 3: Absolute QA score gains over the no-probing Wiki at different probe budgets.

**Diagnosis and repair.** In controlled-defect experiments, we introduce known defects into the Wiki to test whether diagnosis can locate and repair them. Removing access traces lowers both localization accuracy and repair success, showing that records of actual access help identify failure causes and guide repair (Appendix D.2). OG-ClaimWiki also recovers 83.5% of the no-probing Wiki's initial errors on the original MuSiQue questions, showing that diagnosis and local repair can address QA failures beyond system-generated probes (Appendix B.3).

On MuSiQue and SWE-QA-Pro, having three models answer each probe once identifies more distinct defects and yields higher post-repair QA scores than having one model answer each probe three times (Appendix D.8). Different models thus help uncover problems missed by one another, enabling a more thorough check of the Wiki.

### 4.4 Ablation Study

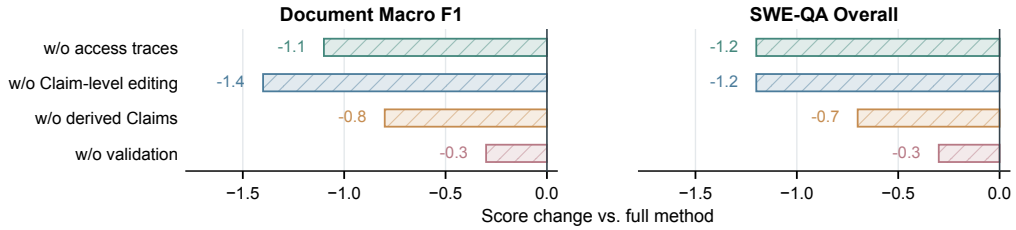


Figure 4: Ablation results showing changes in document Macro F1 and SWE-QA Overall relative to OG-ClaimWiki.

With the probe pool and editing budget fixed, removing any component lowers performance (Figure 4). The ablations of evidence-chain guidance and access traces show that both the required evidence and the content actually accessed provide useful information for diagnosis, with benefits reflected in final QA performance. SWE-QA is most sensitive to removing evidence-chain guidance, highlighting the importance of specifying the evidence needed for code QA.

On document QA, replacing Claim-level edits with full-page rewriting causes the largest drop in Macro F1, at 1.4 points. This indicates that edits targeted at the identified problem are more effective than rewriting the entire page. Removing derived Claims lowers performance in both domains, showing the benefit of storing conclusions inferred from multiple source facts. Removing validation also lowers final scores. Changes proposed to fix an individual failure do not necessarily improve overall QA, and validation helps reject ineffective changes.

### 4.5 Source Update Evaluation

When sources change, both the changed facts and knowledge that depends on them may need updating. This experiment tests whether OG-ClaimWiki can make these updates while preserving correct answers to unaffected questions. We use entity-attribute changes from MQuAKE-T (Zhong et al., 2023) and code version changes from CoDocBench (Pai et al., 2025) to compare QA performance after updating. Knowledge bases remain fixed during evaluation, and answers are assessed against the new source versions. Accuracy measures correctness on affected questions. Retention measures how often previously correct answers to unrelated questions remain correct. No update leaves the Wiki unchanged, while Local update modifies only directly changed content. Other baselines use their own update procedures. Details appear in Appendices B.4 and B.5.

Table 4: **Document and code QA after source updates (%).** Accuracy covers all affected questions. Retention is measured on previously correct, unrelated questions.

**(a) Document updates**

| Method | Accuracy | Retention |
|---|---|---|
| No update | 18.9 | – |
| Local update | 75.1 | 99.2 |
| LLM-Wiki | 86.6 | 98.6 |
| **OG-ClaimWiki** | **92.8** | 98.9 |

**(b) Code updates**

| Method | Accuracy | Retention |
|---|---|---|
| No update | 15.2 | – |
| Local update | 75.6 | 99.1 |
| RepoAgent | 78.4 | 98.5 |
| CodeWiki | 82.1 | 98.7 |
| RepoDoc | 87.6 | 98.9 |
| **OG-ClaimWiki** | **93.1** | 99.0 |

OG-ClaimWiki achieves the highest post-update accuracy in both domains while preserving correct answers to most previously correct, unrelated questions (Table 4). Local update has slightly higher retention but substantially lower accuracy on affected questions. Effective updating therefore requires both preserving unaffected knowledge and correcting knowledge affected by source changes.

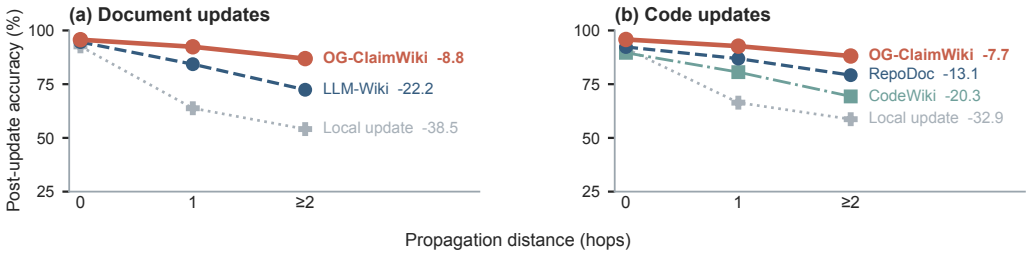


Figure 5: Post-update accuracy by dependency distance. Endpoint labels show changes from 0 hops (percentage points).

**Dependency distance.** To examine whether updates reach knowledge that depends on changed facts, we group questions by the dependency distance from changed Claims to the answer-target Claim, as defined in Appendix B.5. Local update already achieves high accuracy on questions that directly ask about changed facts. Its gap to OG-ClaimWiki is substantially larger in groups whose questions concern dependent knowledge. OG-ClaimWiki outperforms the baselines in every distance group (Figure 5), showing that updates need to correct changed facts and check knowledge that depends on them.

## 5 Conclusion

We present OG-ClaimWiki, which improves knowledge organization through evidence-chain probing before real user queries are available. Comparing source-supported evidence chains with agents' access traces helps locate defects and guide validated local repairs. Experiments on document and code QA show gains over the evaluated baselines, additional improvements from offline probing, and higher accuracy after dependency-aware updates. These findings support improving knowledge organization based on how agents acquire evidence to strengthen QA performance.

### AI USE STATEMENT

Generative AI tools are integral to the studied method and experiments. They generate probe questions, construct and update knowledge representations, act as answering agents, and support model-based evaluation, as detailed in Sections 3 and 4 and the appendices. During research and manuscript preparation, generative AI tools also assisted with discussions of experimental settings and design, interpretation of experimental results, organization of the analysis, language editing, translation, LaTeX and table formatting, and coding. The authors reviewed and verified AI-assisted text, code, results, citations, and other materials against the source materials and experimental outputs. The authors take responsibility for the final content of this work, including material produced with the aid of generative AI.

### ETHICS STATEMENT

Organizing knowledge at a fine level of detail may make sensitive information easier to retrieve. Deployments should therefore preserve the access controls, privacy protections, and deletion requirements of the original sources. A citation alone does not establish that the cited evidence supports a statement. Automated updates should retain their sources, validation results, and version history to support auditing and rollback.

### REPRODUCIBILITY STATEMENT

The appendices provide implementation rules, experimental protocols, prompt templates, and supplementary results. The planned release will include data construction scripts, model and prompt versions, random seeds, tool-call limits, probe questions, access traces, evidence chains, update logs, and cost statistics.

## REFERENCES

Anthropic. Introducing claude opus 5. Anthropic product announcement, July 2026a. URL https://www.anthropic.com/news/claude-opus-5.

Anthropic. Introducing claude sonnet 5. Anthropic product announcement, June 2026b. URL https://www.anthropic.com/news/claude-sonnet-5.

Songcheng Cai, Zhiheng Lyu, Yuansheng Ni, Xiangchao Chen, Baichuan Zhou, Shenzhe Zhu, Yi Lu, Haozhe Wang, Chi Ruan, Benjamin Schneider, Weixu Zhang, Xiang Li, Andy Zheng, Yuyu Zhang, Ping Nie, and Wenhu Chen. SWE-QA-Pro: A representative benchmark and scalable training recipe for repository-level code understanding. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 16958–16999. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.findings-acl.837. URL https://aclanthology.org/2026.findings-acl.837/.

Wenhu Chen, Hanwen Zha, Zhiyu Chen, Wenhan Xiong, Hong Wang, and William Yang Wang. HybridQA: A dataset of multi-hop question answering over tabular and textual data. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 1026–1036. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.findings-emnlp.91.

Zhaoling Chen, Robert Tang, Gangda Deng, Fang Wu, Jialong Wu, Zhiwei Jiang, Viktor Prasanna, Arman Cohan, and Xingyao Wang. LocAgent: Graph-guided LLM agents for code localization. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8697–8727. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.426. URL https://aclanthology.org/2025.acl-long.426/.

DeepSeek-AI. DeepSeek V4 - model card, April 2026. URL https://fe-static.deepseek.com/chat/transparency/deepseek-V4-model-card-EN.pdf.

DeusData. codebase-memory-mcp. Computer software, 2026. URL https://github.com/DeusData/codebase-memory-mcp.

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. URL https://arxiv.org/abs/2404.16130.

Google DeepMind. Gemini 3.5 Flash, May 2026. URL https://ai.google.dev/gemini-api/docs/models/gemini-3.5-flash.

Zirui Guo, Lianghao Xia, Yanhua Yu, Tu Ao, and Chao Huang. LightRAG: Simple and fast retrieval-augmented generation. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 10746–10761. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-emnlp.568. URL https://aclanthology.org/2025.findings-emnlp.568/.

Bernal Jiménez Gutiérrez, Yiheng Shu, Weijian Qi, Sizhe Zhou, and Yu Su. From RAG to memory: Non-parametric continual learning for large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 21497–21515. PMLR, 2025. URL https://proceedings.mlr.press/v267/gutierrez25a.html.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics*, pp. 6609–6625. International Committee on Computational Linguistics, 2020. doi: 10.18653/v1/2020.coling-main.580. URL https://aclanthology.org/2020.coling-main.580/.

Anh Nguyen Hoang, Minh Le-Anh, Bach Le, and Nghi D. Q. Bui. CodeWiki: Evaluating AI's ability to generate holistic documentation for large-scale codebases. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 5812–5827, 2026. doi: 10.18653/v1/2026.findings-acl.288. URL https://aclanthology.org/2026.findings-acl.288/.

Haoyu Huang, Jiaxin Bai, Shujie Liu, Yang Wei, Huihao Jing, Hong Ting Tsang, Yisen Gao, Zhongwei Xie, Yufei Li, and Yangqiu Song. DeepRefine: Agentic knowledge refinement via reinforcement learning. *arXiv preprint arXiv:2605.10488*, 2026. URL https://arxiv.org/abs/2605.10488.

Juan M. Huerta. WiCER: Wiki-memory compile, evaluate, refine iterative knowledge compilation for LLM wiki systems. *arXiv preprint arXiv:2605.07068*, 2026. URL https://arxiv.org/abs/2605.07068.

Andrej Karpathy. LLM Wiki. GitHub Gist, 2026. URL https://gist.github.com/karpathy/442a6bf555914893e9891c11519de94f. An abstract design pattern for incrementally building and maintaining a persistent Wiki.

Jeongsoo Lee, Daeyong Kwon, and Kyohoon Jin. GRADE: Generating multi-hop QA and fine-gRAined Difficulty matrix for RAG evaluation. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 4405–4424, 2025. doi: 10.18653/v1/2025.findings-emnlp.236.

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*, volume 33, pp. 9459–9474. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html.

Xiangyan Liu, Bo Lan, Zhiyuan Hu, Yang Liu, Zhicheng Zhang, Fei Wang, Michael Qizhe Shieh, and Wenmeng Zhou. CodexGraph: Bridging large language models and code repositories via code graph databases. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 142–160. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.naacl-long.7. URL https://aclanthology.org/2025.naacl-long.7/.

Qinyu Luo, Yining Ye, Shihao Liang, Zhong Zhang, Yujia Qin, Yaxi Lu, Yesai Wu, Xin Cong, Yankai Lin, Yingli Zhang, Xiaoyin Che, Zhiyuan Liu, and Maosong Sun. RepoAgent: An LLM-powered open-source framework for repository-level code documentation generation. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 436–464. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.emnlp-demo.46. URL https://aclanthology.org/2024.emnlp-demo.46/.

Haoliang Ming, Feifei Li, Xiaoqing Wu, and Wenhui Que. Retrieval as reasoning: Self-evolving agent-native retrieval via LLM-Wiki. *arXiv preprint arXiv:2605.25480*, 2026. URL https://arxiv.org/abs/2605.25480.

OpenAI. GPT-5.6: Frontier intelligence that scales with your ambition, July 2026. URL https://openai.com/index/gpt-5-6/.

Siru Ouyang, Wenhao Yu, Kaixin Ma, Zilin Xiao, Zhihan Zhang, Mengzhao Jia, Jiawei Han, Hongming Zhang, and Dong Yu. RepoGraph: Enhancing AI software engineering with repository-level code graph. In *International Conference on Learning Representations*, pp. 30098–30121, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/4a4a3c197deac042461c677219efd36c-Abstract-Conference.html.

Kunal Pai, Premkumar Devanbu, and Toufique Ahmed. CoDocBench: A dataset for code-documentation alignment in software maintenance. In *2025 IEEE/ACM 22nd International Conference on Mining Software Repositories (MSR)*, pp. 451–455. IEEE, 2025. doi: 10.1109/MSR66628.2025.00077. URL https://ieeexplore.ieee.org/document/11025763/.

Weihan Peng, Yuling Shi, Yuhang Wang, Xinyun Zhang, Beijun Shen, and Xiaodong Gu. SWE-QA: Can language models answer repository-level code questions? In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 8230–8245. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.findings-acl.402. URL https://aclanthology.org/2026.findings-acl.402/.

Qwen Team. Qwen3.6-35B-A3B: Agentic coding power, now open to all, April 2026. URL https://qwen.ai/blog?id=qwen3.6-35b-a3b.

ripgrep contributors. ripgrep. Computer software, 2026. URL https://github.com/BurntSushi/ripgrep. Accessed September 24, 2026.

Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389, 2009. doi: 10.1561/1500000019. URL https://www.nowpublishers.com/article/Details/INR-019.

Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D. Manning. RAPTOR: Recursive abstractive processing for tree-organized retrieval. In *International Conference on Learning Representations*, pp. 32628–32649, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/8a2acd174940dbca361a6398a4f9df91-Abstract-Conference.html.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. MuSiQue: Multi-hop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022. doi: 10.1162/tacl_a_00475. URL https://aclanthology.org/2022.tacl-1.31/.

Hong Ting Tsang, Jiaxin Bai, Haoyu Huang, Qiao Xiao, Tianshi Zheng, Baixuan Xu, Shujie Liu, and Yangqiu Song. AutoGraph-R1: End-to-end reinforcement learning for knowledge graph construction. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 23351–23374. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.1070. URL https://aclanthology.org/2026.acl-long.1070/.

Baoyi Wang, Xingliang Wang, Guochang Li, Chen Zhi, Junxiao Han, Xinkui Zhao, Nan Wang, Shuiguang Deng, and Jianwei Yin. Better call grep: Evaluating and improving grep-like

lexical retrieval for repository-level code completion. *Proceedings of the ACM on Software Engineering*, 2026. URL https://conf.researchr.org/details/issta-2026/issta-2026-research-papers/45/Better-Call-Grep-Evaluating-and-Improving-Grep-Like-Lexical-Retrieval-for-Repository. Accepted for ISSTA 2026.

Dong Xu, Mingwei Liu, Xiwen Wang, Jianfeng Zhong, and Zibin Zheng. RepoDoc: A knowledge graph-based framework to automatic documentation generation and incremental updates. *arXiv preprint arXiv:2604.26523*, 2026. URL https://arxiv.org/abs/2604.26523.

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 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2369–2380. Association for Computational Linguistics, 2018. doi: 10.18653/v1/D18-1259. URL https://aclanthology.org/D18-1259/.

Ding-Chu Zhang, Xiaowen Zhang, Yue Fei, Renjun Hu, Xiao-Wen Yang, Zhi Zhou, Baixuan Li, Yu-Feng Li, Xing Shi, and Wei Lin. AutoEvolve: Automatically evolving queries for applicable and scalable retrieval-augmented generation benchmarking. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 7624–7639, 2025a. doi: 10.18653/v1/2025.findings-emnlp.403.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. Qwen3 Embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*, 2025b. URL https://arxiv.org/abs/2506.05176.

Xinkui Zhao, Haode Li, Yifan Zhang, Guanjie Cheng, and Yueshen Xu. TRAIL: Joint inference and refinement of knowledge graphs with large language models. *arXiv preprint arXiv:2508.04474*, 2025. URL https://arxiv.org/abs/2508.04474.

Zexuan Zhong, Zhengxuan Wu, Christopher D. Manning, Christopher Potts, and Danqi Chen. MQuAKE: Assessing knowledge editing in language models via multi-hop questions. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 15686–15702. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.971. URL https://aclanthology.org/2023.emnlp-main.971/.

Kunlun Zhu, Yifan Luo, Dingling Xu, Yukun Yan, Zhenghao Liu, Shi Yu, Ruobing Wang, Shuo Wang, Yishan Li, Nan Zhang, Xu Han, Zhiyuan Liu, and Maosong Sun. RAGEval: Scenario specific RAG evaluation dataset generation framework. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8520–8544, 2025. doi: 10.18653/v1/2025.acl-long.418.

# A Implementation Details

**Source parsing and page compilation.** Document Claims retain paragraph, table-row, or infobox locations and are independently verified against those locations. Entities are aligned by page identifiers, internal links, and canonical names. Dates use ISO dates or intervals. For code, `codebase-memory-mcp` (MCP) (DeusData, 2026) resolves symbols, locations, signatures, and static relations at a frozen repository commit. Code Claims retain bindings to the commit and the corresponding file, symbol, or syntax node. Unresolved nodes and edges are removed.

Claims are assigned to pages by entity, topic, or code symbol. Page generation retains only text supported by current Claims and records paragraph-level Claim references. Verified relations form page links. Names, aliases, symbols, and paths populate the shared index used by probing and final QA.

Table 5: Relations in the Claim graph and the deterministic rules used to construct them.

| Modality | Relation | Construction condition |
|---|---|---|
| Document | Shared entity | Mentions resolve to the same canonical entity. |
| Document | Explicit relation | A sentence, infobox, or table explicitly connects subject and object. |
| Document | Temporal condition | Normalized dates or intervals satisfy ordering, equality, or containment. |
| Document | Table structure | Claims share a row or a header hierarchy. |
| Code | Static call | Syntax and symbol resolution map a call site to its target. |
| Code | Return or exception | Return values, throw sites, or static paths establish a propagation relation. |
| Code | Configuration | A configuration read, default, or branch constrains target behavior. |
| Code | Test | A test invokes the target and an assertion or fixture constrains its behavior. |
| Code | Dependency | Imports, build manifests, or resolved symbol references establish a dependency. |

**Evidence chains and local edits.** Chain enumeration follows directed relations without revisiting nodes and stops when the terminal Claim determines a unique answer. Derived Claims retain their premises, relations, and source versions for revalidation after changes.

Table 6: Observed conditions and candidate local edits.

| **Edit target** | **Observed condition** | **Candidate edit** |
|---|---|---|
| Claim content | Required Claim is missing, outdated, or conflicts with the current source | Add, update, or deactivate the Claim |
| Source binding | Recorded location no longer points to supporting source content | Update the source binding |
| Search index | Search terms in the evidence chain fail to retrieve an existing page | Update titles, aliases, or index entries |
| Page link | Visited page lacks a source-supported link to a required evidence page | Add the page link |
| Context organization | Required Claims are acquired and cited, but the answer remains incorrect | Propose a derived Claim |

**Edit acceptance.** Each candidate edit group is tested on a Wiki copy using the following checks:

1. **Source support:** New or modified Claims, relations, and source bindings must pass source verification.
2. **Probe recovery:** The original probe must be answered correctly in three fresh sessions.
3. **Independent validation:** All three validation questions must be answered correctly. They concern affected knowledge but require different Claims or evidence chains from the probe.
4. **Regression:** All twenty previously correct questions must remain correct. Ten concern the edit location and ten do not overlap with it.

Validation and regression questions are generated from sources and frozen before candidate generation. Edits are committed only if every check passes. No edit is committed without an independent validation set.

# B Supplementary Experimental Protocol

## B.1 Data, Baselines, and Models

**Data.** We sample 1,000 questions from each official development or evaluation split of HotpotQA distractor, MuSiQue answerable, 2WikiMultiHopQA, and HybridQA, stratified by question type. The first three datasets share a merged document corpus. HybridQA uses its official tables and linked passages. Retrieval is corpus-wide, without question-specific candidate evidence. SWE-QA contains 720 questions from 15 repositories, and SWE-QA-Pro contains 260 from 26 repositories, each fixed at the sample-specified commit.

**Document baseline implementations.** Document baselines use their official implementations and recommended hyperparameters, with the shared corpus, answering models, and budgets specified here. Methods requiring text embeddings use Qwen3-Embedding-8B (Zhang et al., 2025b).

**BM25 and Dense RAG.** BM25 (Robertson & Zaragoza, 2009) ranks passages by lexical relevance; Dense RAG (Lewis et al., 2020) ranks them by embedding similarity. Each supplies its five highest-ranked passages to the answering model.

**RAPTOR.** RAPTOR (Sarthi et al., 2024) recursively groups and summarizes passages into a tree, allowing retrieval from both source passages and summaries.

**GraphRAG and LightRAG.** GraphRAG (Edge et al., 2024) extracts an entity graph and organizes community summaries hierarchically. LightRAG (Guo et al., 2025) combines retrieval of specific entities with broader topic-level graph retrieval.

**HippoRAG2.** HippoRAG2 (Gutiérrez et al., 2025) builds a graph from extracted triples and uses Personalized PageRank to identify evidence for answering.

**LLM-Wiki.** LLM-Wiki (Ming et al., 2026) compiles sources into linked pages. Compilation selects at most five existing pages per source passage and rechecks Error Book entries every ten compilation batches. Answering uses a patience threshold of three consecutive empty searches and the shared 20-operation limit.

**Code baseline implementations.** Agentic Grep provides directory browsing, `ripgrep` (ripgrep contributors, 2026), and source reading. Function Chunking (Peng et al., 2026) splits at function boundaries. RepoAgent v0.2.0 (Luo et al., 2024) generates object descriptions. CodeWiki (Hoang et al., 2026) retains module hierarchies, cross-references, and Markdown pages. RepoDoc (Xu et al., 2026) retains module pages and graph-supported links. Raw MCP Graph (DeusData, 2026) exposes the same frozen symbols, static relations, and source locations as OG-ClaimWiki, without Claim or Wiki generation.

**Models and budgets.** The main comparison evaluates Qwen3.6-35B-A3B, DeepSeek-V4-Pro, Gemini 3.5 Flash, GPT-5.6 Sol, and Claude Opus 5 (Qwen Team, 2026; DeepSeek-AI, 2026; Google DeepMind, 2026; OpenAI, 2026; Anthropic, 2026a). All answering models share one refined Wiki. Claude Sonnet 5 (Anthropic, 2026b) generates probes, and GPT-5.6 Sol constructs and maintains the Wiki. The probing analysis, source-update evaluation, and ablations use GPT-5.6 Sol as the answering model. All methods allow at most 20 operations per question and a 64k-token answering context.

**Evaluation protocol.** For downstream QA evaluation, test questions are excluded from construction, probing, and candidate validation. Evaluation uses a frozen Wiki, with each question answered

independently. Methods are compared on the same test samples. The MuSiQue failure-repair analysis is conducted separately on isolated Wiki copies, which are excluded from all other experiments (Appendix B.3).

### B.2 Probe Construction and Ablations

Random-QG samples candidate Claims from the source Claim pool for the evaluated corpus or repository, without using Claim relations to organize them. Page-QG samples a starting source page or code file and expands to related pages through links in the initial Wiki. It generates questions from the selected page content without a preselected Claim path. Claim-QG selects Claim paths and their source-supported relations to generate single-Claim, relational, and compositional questions. All strategies may generate questions requiring one or multiple Claims.

All strategies use the same generation model, source verification, closed-book filtering, and deduplication. Required Claim counts refer to the evidence needed to derive the answer, rather than all Claims supplied to the generator. Each retained evidence set must support the answer and become insufficient when any Claim is removed. After filtering, strategies match the total question count and the distribution of required Claim counts within each dataset. Invalid candidates are resampled. Strategies start from separate copies of the same initial Wiki and share the diagnosis, editing, and validation procedure in Section 3.

Table 7: **Probe counts retained by Claim-QG after source verification, closed-book filtering, and deduplication.** Random-QG and Page-QG match total counts and required Claim-count distributions, rather than question-type counts.

| Source | Single-Claim | Relational | Compositional | Total |
|---|---|---|---|---|
| HotpotQA | 597 | 681 | 533 | 1,811 |
| MuSiQue | 512 | 549 | 447 | 1,508 |
| 2WikiMultiHopQA | 706 | 793 | 618 | 2,117 |
| HybridQA | 851 | 940 | 737 | 2,528 |
| SWE-QA | 683 | 654 | 549 | 1,886 |
| SWE-QA-Pro | 247 | 309 | 281 | 837 |

**Filtering and separation.** Retained questions must have a unique, source-supported answer. Removing a necessary Claim must make the evidence chain insufficient. For closed-book filtering, each of the five answering models makes three attempts in independent sessions with tools disabled. A question is retained only if all fifteen answers are incorrect against the source-supported answer. Questions whose answers change after source updates undergo filtering again.

Generated questions are checked against the public benchmark test questions, and duplicates are excluded. Appendix D.4 reports near-duplicate rates and the overlap between probes and test evidence. Validation questions that duplicate probes or reuse their complete evidence chains are also excluded. Test answers and per-question decisions are withheld from generation and editing modules.

Audits of 600 document and 600 code probes yield pass rates of 94%, 93%, 92%, and 96% for source support, answer uniqueness, evidence necessity, and closed-book judgments, respectively.

Table 8: Component ablation settings. Variants share the probe pool, sequence, and editing budget.

| Variant | Change from OG-ClaimWiki |
|---|---|
| w/o evidence-chain | Diagnosis omits structured evidence-chain guidance, including required Claims, relations, and their page or retrieval-entry mappings. Other procedures remain unchanged. |
| w/o access traces | Diagnosis omits access histories and trace-derived feedback but retains the question, generated answer, expected evidence, and Wiki access. Failure detection still uses traces. |
| w/o Claim-level editing | Rewrite the target page in full instead of editing individual Claims. |
| w/o derived Claims | Omit derived Claims. |
| w/o validation | Remove independent-question validation and regression checks. |

**Probe injection.** This control tests whether the probing gain arises from probe coverage alone. It starts from a separate copy of the initial Wiki and uses the same retained Claim-QG probes as OG-ClaimWiki (Table 7). Each probe is written as a QA entry containing the question, the source-verified expected answer, and references to its evidence Claims. The entry is attached to every Wiki page or code entry in the probe's evidence chain, and the question text is added to the search index. Diagnosis, editing, and validation are omitted. Evaluation uses GPT-5.6 Sol and the frozen-Wiki protocol above.

### B.3 Diagnosis and Repair Protocols

**Controlled defects.** We inject one defect per Wiki copy for document questions answered correctly and consistently by the initial Wiki, constructing 200 copies per type. Only cases that become incorrect are retained. Interventions remove or alter a required Claim, remove its source location, break search-term matching, break an evidence-chain page link, or add a topically related but answer-irrelevant Claim. These correspond to Claim content, source binding, search index, page link, and context organization defects, respectively.

Type macro-F1 compares predicted and injected types. Object localization accuracy measures identification of the modified object. End-to-end recovery counts cases that regain the correct answer and pass edit acceptance (Appendix A), divided by all post-injection failures.

**Benchmark-failure repair.** Initial failures are identified by EM with official answer normalization. Each case starts from an isolated initial-Wiki copy. The repair module receives only the question and access trace and constructs its evidence chain from sources. Reference answers serve only failure selection and evaluation. Benchmark supporting annotations and precomputed chains are withheld. Recovery requires edits to pass the acceptance checks in Appendix A and the original question to be answered correctly in all three fresh sessions. The recovery rate is the fraction of all initial failures that meet these criteria. Unresolved cases and rejected edits count as unrecovered. Revised copies are excluded from all other experiments.

### B.4 Metrics and Statistical Evaluation

**QA scoring.** Document EM and F1 use official answer normalization. SWE-QA uses the official Claude Sonnet 4.5 judge for correctness, completeness, relevance, clarity, and coherence, each scored 1–20 and summed as Overall. SWE-QA-Pro uses the official GPT-5 judge for correctness, completeness, relevance, clarity, and reasoning, each scored 1–10 and summed as Score. Method identities are hidden and answer order is randomized. Human blind review of 980 stratified answers yields Spearman $\rho = 0.85$ with judge scores.

**Source-update scoring.** Document answers in the source-update evaluation use EM against new-version answers. Code questions have fixed short answers derived from version differences and use exact match with manual verification. The affected question set is fixed before updating, regardless of old-version performance. An unrelated question's evidence chain neither contains changed content nor reaches it through static dependencies.

**Runs and uncertainty.** The main comparison reports one run with temperature 0. The probing analysis uses five question-selection seeds. The source-update evaluation reports means over three independent runs. We estimate 95% confidence intervals from 10,000 stratified paired bootstrap resamples. Resampling units are source clusters within document datasets, repositories for code QA, and events for source updates.

### B.5 Source Update Events

**Event construction.** Document events use MQuAKE-T factual differences between the April 2021 and April 2023 Wikidata dumps. Its six relations concern office or position changes, limiting this evaluation to entity-attribute updates. Events require both values to be supported by their respective dumps and all other multi-hop premises to remain unchanged.

Code events reconstruct adjacent repository snapshots from CoDocBench version pairs and commit metadata, excluding unrecoverable commits, formatting-only changes, and changes without observable behavioral effects. Source differences and static dependencies guide affected-question construction. Three annotators independently assess semantic changes and impact scope (Fleiss' $\kappa = 0.78$). Code-event verification yields Fleiss' $\kappa = 0.81$. Affected and unrelated questions are frozen before any method runs.

Table 9: Source-update evaluation sizes.

| Domain | Events | Affected questions | Unrelated questions |
|---|---|---|---|
| Documents | 1,248 | 3,864 | 5,792 |
| Code | 1,036 | 3,107 | 4,655 |

**Update procedures.** Update inputs comprise new sources and addition, modification, and deletion markers. Information boundaries are specified in Appendix C. No update retains the old Wiki. Local update replaces directly changed units without updating dependencies. LLM-Wiki re-ingests new sources, merges them with relevant existing pages, and checks pages, links, and indexes, without receiving our Claim graph or diagnoses. RepoAgent, RepoDoc, and CodeWiki use their incremental update procedures. Evaluation begins only after updates finish and knowledge bases are frozen.

**Dependency distance.** For each question, we identify the relevant changed Claims and the answer-target Claim, which directly states the fact requested by the question. Dependency distance is the length of the shortest directed path from a relevant changed Claim to the answer-target Claim. If multiple changed Claims are involved, we take the minimum distance to the answer target. A changed Claim used as intermediate evidence is not a path endpoint unless the question directly asks about that Claim.

We group questions into zero, one, and at least two hops. Original benchmark questions and additionally constructed questions follow the same grouping rule, rather than being grouped directly by their total reasoning hops. All methods share fixed groups, which are unaffected by links or derived Claims added during updating. Question composition may differ across groups, so between-group performance differences are not attributed solely to dependency distance.

## C Information Isolation Protocol

Table 10: Information available to update modules and to the isolated validator or evaluator.

| **Stage** | **Available to update modules** | **Reserved for the validator or evaluator** |
|---|---|---|
| Initial construction | Sources, versions, and frozen source structure, including MCP output for code | Test questions, answers, and supporting annotations |
| Offline probing | Source-generated questions, expected answers, and access traces | Independent validation and regression questions and their answers |
| Candidate validation | Candidate edits and the original probe | Frozen independent validation and regression questions, answers, and per-question results |
| Source update | New source records or code snapshots with addition, modification, and deletion markers | Edited triples, impact annotations, and update questions |
| Final evaluation | Frozen Wiki interfaces, without online MCP access | All test inputs, labels, and evaluation results |

During final evaluation, the answering agent obtains evidence through the Wiki interfaces. It does not receive the Claim graph, expected answers, or candidate-generation records. Test supporting annotations are reserved for the evaluator.

# D SUPPLEMENTARY RESULTS

## D.1 ADDITIONAL MAIN RESULTS

Table 11: **Document QA results for the remaining answering models.**

| Model | Method | HotpotQA | | MuSiQue | | 2WikiMultiHopQA | | HybridQA | | Macro F1 |
|---|---|---|---|---|---|---|---|---|---|---|
| | | EM | F1 | EM | F1 | EM | F1 | EM | F1 | |
| DeepSeek-V4 | BM25 | 64.6 | 73.5 | 46.5 | 55.9 | 72.1 | 79.0 | 46.5 | 56.5 | 66.2 |
| | Dense RAG | 70.4 | 77.9 | 50.9 | 59.6 | 75.4 | 82.6 | 54.0 | 62.8 | 70.7 |
| | RAPTOR | 66.9 | 75.7 | 48.1 | 55.7 | 68.7 | 74.8 | 56.6 | 65.8 | 68.0 |
| | GraphRAG | 67.6 | 76.0 | 55.8 | 64.0 | 68.6 | 76.2 | 54.2 | 64.1 | 70.1 |
| | LightRAG | 73.9 | 81.5 | 51.7 | 61.9 | 83.8 | 89.4 | 57.8 | 66.1 | 74.7 |
| | HippoRAG2 | 71.0 | 80.0 | 56.9 | 65.1 | 81.6 | 87.3 | 61.1 | 70.8 | 75.8 |
| | LLM-Wiki | 74.2 | 83.7 | 58.7 | 69.0 | 83.3 | 90.7 | 61.2 | 70.1 | 78.4 |
| | OG-ClaimWiki w/o probing | 74.9 | 83.1 | 65.5 | 74.6 | 85.6 | 91.3 | 61.6 | 70.6 | 79.9 |
| | **OG-ClaimWiki** | **77.5** | **86.4** +3.3 | **70.3** | **79.3** +4.7 | **88.0** | **93.5** +2.2 | **66.6** | **76.0** +5.4 | **83.8** +3.9 |
| Gemini 3.5 | BM25 | 64.5 | 73.6 | 47.8 | 56.8 | 74.1 | 80.3 | 51.5 | 59.6 | 67.6 |
| | Dense RAG | 67.1 | 75.9 | 50.8 | 59.8 | 74.1 | 81.6 | 54.0 | 64.7 | 70.5 |
| | RAPTOR | 69.3 | 77.0 | 49.9 | 58.8 | 67.7 | 76.0 | 58.0 | 67.1 | 69.7 |
| | GraphRAG | 70.8 | 78.0 | 56.0 | 64.1 | 70.7 | 77.5 | 54.9 | 63.9 | 70.9 |
| | LightRAG | 72.6 | 82.6 | 54.6 | 63.8 | 81.9 | 88.8 | 56.8 | 66.3 | 75.4 |
| | HippoRAG2 | 71.6 | 80.4 | 58.2 | 66.4 | 82.9 | 89.1 | 58.4 | 68.1 | 76.0 |
| | LLM-Wiki | 73.5 | 82.3 | 60.4 | 70.6 | 84.7 | 92.2 | 64.4 | 72.5 | 79.4 |
| | OG-ClaimWiki w/o probing | 73.6 | 83.3 | 66.5 | 74.4 | 83.5 | 91.8 | 64.8 | 72.9 | 80.6 |
| | **OG-ClaimWiki** | **77.4** | **86.5** +3.2 | **69.4** | **78.0** +3.6 | **87.1** | **93.9** +2.1 | **69.9** | **78.4** +5.5 | **84.2** +3.6 |

Table 12: **Complete code QA results.**

| Method | Qwen3.6 | DeepSeek-V4 | Gemini 3.5 | GPT-5.6 | Claude Opus 5 |
|---|---|---|---|---|---|
| *SWE-QA* | | | | | |
| Agentic Grep | 60.2 | 63.8 | 64.9 | 66.7 | 67.4 |
| Function Chunking | 56.4 | 59.8 | 60.3 | 62.6 | 62.3 |
| RepoAgent | 58.3 | 61.5 | 61.2 | 63.6 | 64.3 |
| CodeWiki | 62.3 | 63.2 | 64.7 | 66.5 | 66.8 |
| Raw MCP Graph | 60.1 | 62.9 | 62.7 | 65.8 | 67.2 |
| RepoDoc | 61.4 | 63.6 | 64.6 | 67.0 | 68.5 |
| OG-ClaimWiki w/o probing | 62.0 | 65.4 | 64.5 | 68.5 | 68.2 |
| **OG-ClaimWiki** | **66.2** +4.2 | **68.9** +3.5 | **68.2** +3.7 | **71.4** +2.9 | **70.7** +2.5 |
| *SWE-QA-Pro* | | | | | |
| Agentic Grep | 32.9 | 34.3 | 35.7 | 37.3 | 36.8 |
| Function Chunking | 31.6 | 33.8 | 34.4 | 34.7 | 36.4 |
| RepoAgent | 32.2 | 34.7 | 35.4 | 36.9 | 37.1 |
| CodeWiki | 33.7 | 35.2 | 36.4 | 37.2 | 38.9 |
| Raw MCP Graph | 34.0 | 35.6 | 36.5 | 37.1 | 38.6 |
| RepoDoc | 35.6 | 36.5 | 36.4 | 38.2 | 38.8 |
| OG-ClaimWiki w/o probing | 35.4 | 37.3 | 37.8 | 38.8 | 39.8 |
| **OG-ClaimWiki** | **38.1** +2.7 | **39.6** +2.3 | **40.0** +2.2 | **40.7** +1.9 | **41.4** +1.6 |

## D.2 CONTROLLED DEFECT DIAGNOSIS

We inject the five defect types into separate copies of a Wiki that initially answers the document questions correctly. Table 13 reports defect type identification, object localization, and end-to-end recovery. We also report the *w/o access traces* variant of Figure 4.

Table 13: **Diagnosis and repair of five controlled defect types in document QA (%).** The *w/o access traces* row uses the ablation setting of Figure 4.

| Variant | Type Macro-F1 | Object localization | End-to-end recovery |
|---|---|---|---|
| OG-ClaimWiki | 90.4 | 93.6 | 88.7 |
| w/o access traces | 82.1 | 79.8 | 76.3 |

### D.3 Probing Strategies by Dataset

Table 14: **Probing-strategy comparison by dataset: document F1 and code QA scores.**

| Strategy | HotpotQA | MuSiQue | 2WikiMultiHopQA | HybridQA | SWE-QA | SWE-QA-Pro |
|---|---|---|---|---|---|---|
| No probing | 86.4 | 76.3 | 91.9 | 74.6 | 68.5 | 38.8 |
| Random-QG | 86.8 | 77.2 | 92.8 | 75.2 | 69.3 | 39.9 |
| Page-QG | 88.4 | 78.6 | 92.2 | 77.6 | 70.1 | 39.6 |
| **Claim-QG** | **89.1** | **80.0** | **93.5** | **79.0** | **71.4** | **40.7** |

### D.4 Overlap between Probes and Test Evidence

Probes are generated from the same corpus that contains the test evidence. We therefore measure how much of the probing gain depends on probes that reproduce test evidence chains. The analysis uses HotpotQA, MuSiQue, and 2WikiMultiHopQA, whose annotations identify the supporting paragraphs of each test question. HybridQA is excluded because its annotations do not identify complete evidence chains over tables and passages. Supporting annotations are used only in this post hoc analysis and remain unavailable to construction, probing, and editing (Appendix C).

**Overlap groups.** A probe covers a supporting paragraph if its evidence chain contains a Claim located in that paragraph. Each test question is assigned to one of three groups according to the Claim-QG probes in Table 7. *None*: no probe covers any of its supporting paragraphs. *Partial*: at least one supporting paragraph is covered, but no single probe covers all of them. *Full*: at least one probe covers all of its supporting paragraphs. The analysis reuses the GPT-5.6 Sol answers from Table 1 and requires no additional runs. Confidence intervals follow Appendix B.4, with bootstrap resampling within each group.

**Near-duplicate check.** A probe counts as a near-duplicate of a test question if their cosine similarity under the Dense RAG encoder is at least 0.9, or if they share the normalized answer and the set of supporting paragraphs.

Table 15: **Probing gains by overlap between probes and test evidence (F1, GPT-5.6 Sol).** Groups pool the 3,000 test questions of HotpotQA, MuSiQue, and 2WikiMultiHopQA. Share is each group's contribution to the overall gain.

| Group | Questions (%) | w/o probing | OG-ClaimWiki | Gain | 95% CI | Share (%) |
|---|---|---|---|---|---|---|
| None | 60.0 | 85.6 | 87.3 | +1.7 | [1.1, 2.3] | 39 |
| Partial | 28.0 | 84.3 | 87.7 | +3.4 | [2.4, 4.4] | 36 |
| Full | 12.0 | 82.7 | 88.2 | +5.5 | [3.9, 7.1] | 25 |
| All | 100.0 | 84.9 | 87.5 | +2.6 | [2.1, 3.2] | 100 |

Full-overlap questions account for 12% of test questions and 25% of the probing gain (Table 15). Their gain is the largest, as expected when probes directly exercise their evidence chains. Questions whose supporting paragraphs are covered by no probe still gain 1.7 F1 points, with a confidence interval excluding zero, and contribute 39% of the overall gain. Near-duplicates account for 0.4% of retained probes (21 of 5,436). The probing gain therefore does not arise mainly from probes that reproduce test evidence chains, and it extends to questions whose supporting evidence was never probed.

### D.5 Probe Injection without Repair

Probe injection (Appendix B) receives the same probes, expected answers, and evidence references as OG-ClaimWiki but makes no diagnosis-guided edits. Table 16 compares the two methods, and Table 17 decomposes their document gains by the overlap groups of Appendix D.4.

Table 16: **Probe injection versus diagnosis-guided repair (GPT-5.6 Sol).** Both methods use the same Claim-QG probes. Ret. is the percentage of previously correct document answers that remain correct. Objects count Wiki entries added or modified on document and SWE-QA sources.

| | | | | | | Objects | |
|---|---|---|---|---|---|---|---|
| **Method** | **Macro F1** | **Δ** | **Ret.** | **SWE-QA** | **Δ** | **Doc** | **Code** |
| No probing | 82.3 | – | – | 68.5 | – | – | – |
| Probe injection | 83.2 | +0.9 | 98.1 | 69.2 | +0.7 | 7,964 | 1,886 |
| **OG-ClaimWiki** | **85.4** | **+3.1** | **99.4** | **71.4** | **+2.9** | 3,412 | 1,027 |

Table 17: **Document gains of probe injection and OG-ClaimWiki by overlap group (F1, GPT-5.6 Sol).** Groups follow Table 15.

| **Group** | **Questions (%)** | **w/o probing** | **Injection** | **Gain** | **OG-ClaimWiki** | **Gain** |
|---|---|---|---|---|---|---|
| None | 60.0 | 85.6 | 85.9 | +0.3 | 87.3 | +1.7 |
| Partial | 28.0 | 84.3 | 85.5 | +1.2 | 87.7 | +3.4 |
| Full | 12.0 | 82.7 | 86.9 | +4.2 | 88.2 | +5.5 |
| All | 100.0 | 84.9 | 85.9 | +1.0 | 87.5 | +2.6 |

Probe injection improves document Macro F1 by 0.9 points and SWE-QA Overall by 0.7 points, less than one third of the gains from OG-ClaimWiki. Its gain concentrates on Full-overlap questions (+4.2), whose evidence chains are directly restated by injected entries, and nearly vanishes on questions with no probe coverage (+0.3, compared with +1.7 for OG-ClaimWiki). Injection also lowers retention to 98.1%, as injected entries compete with page content in search. It adds 7,964 document entries, whereas OG-ClaimWiki commits 3,412 edits. Probe coverage alone therefore explains only a small part of the probing gain, and most of the gain on unprobed evidence comes from diagnosis-guided repairs to Claims, indexes, and links.

### D.6 Offline Cost

Table 18 reports LLM tokens and requests for building and refining each knowledge base. Requests count individual model calls, including each step of an agent session. Answering-time cost is excluded because all methods share the same budget (Appendix B).

Probing and repair increase document cost by 3.0× over the no-probing Wiki. Candidate validation is the largest stage, accounting for 52% of document tokens, because each edit group is re-answered on the original probe, validation questions, and regression questions. The four checks run in order, and a candidate proceeds to the next check only after passing the previous one, so rejected candidates skip the regression check, the most expensive step. Probe injection shares probe generation and filtering and costs 39% of OG-ClaimWiki. The additional cost of diagnosis and validation yields 2.2 more Macro F1 points than injection together with higher retention (Table 16). All costs are incurred offline once per source version, before real user queries are available.

Table 18: **Offline cost of knowledge-base construction and refinement.** Tokens in millions and requests in thousands, summing input and output tokens. Code costs cover SWE-QA repositories. Probe injection comprises the first three stages.

| | Documents | | SWE-QA | |
|---|---|---|---|---|
| **Stage or method** | **Tokens (M)** | **Requests (K)** | **Tokens (M)** | **Requests (K)** |
| *OG-ClaimWiki stages* | | | | |
| Claim extraction and page compilation | 560 | 180 | 220 | 80 |
| Probe generation and verification | 60 | 24 | 17 | 6 |
| Closed-book filtering | 36 | 180 | 11 | 44 |
| Probe answering and diagnosis | 151 | 52 | 48 | 12 |
| Candidate validation | 890 | 410 | 344 | 144 |
| *Totals* | | | | |
| OG-ClaimWiki w/o probing | 560 | 180 | 220 | 80 |
| Probe injection | 656 | 384 | 248 | 130 |
| **OG-ClaimWiki** | 1,697 | 846 | 640 | 286 |
| LLM-Wiki | 880 | 260 | – | – |
| HippoRAG2 | 310 | 150 | – | – |
| CodeWiki | – | – | 260 | 95 |
| RepoDoc | – | – | 190 | 70 |

### D.7 Uncertainty in Performance Differences

The intervals in Table 19 use GPT-5.6 Sol. Only the SWE-QA-Pro comparison without probing includes zero. The probing gain on that benchmark excludes zero.

Table 19: Confidence intervals for performance differences.

| Metric | Comparison | 95% CI |
|---|---|---|
| Document Macro F1 | w/o probing vs. LLM-Wiki | $[0.7, 2.2]$ |
| Document Macro F1 | OG-ClaimWiki vs. w/o probing | $[2.5, 3.8]$ |
| SWE-QA Overall | w/o probing vs. RepoDoc | $[0.4, 2.4]$ |
| SWE-QA Overall | OG-ClaimWiki vs. w/o probing | $[2.0, 3.7]$ |
| SWE-QA-Pro Score | w/o probing vs. RepoDoc | $[-0.2, 1.3]$ |
| SWE-QA-Pro Score | OG-ClaimWiki vs. w/o probing | $[1.2, 2.7]$ |
| Document update accuracy | OG-ClaimWiki vs. local update | $[15.8, 19.3]$ |
| Code update accuracy | OG-ClaimWiki vs. RepoDoc | $[4.0, 7.3]$ |

### D.8 Complementarity across Probing Models

For each of MuSiQue and SWE-QA-Pro, we fix 300 probe questions generated from the sources by Claude Sonnet 5. We compare three attempts per question by a single model with one attempt each by Qwen3.6, GPT-5.6 Sol, and Claude Opus 5. Every configuration performs 900 question-answering runs with the same tool-call limit. All runs use independent sessions and the same frozen Wiki. Edits are applied only after all probes in a configuration have been completed.

We deduplicate defects by target object and type, counting only source-verified defects in Claim content, search indexes, page links, and source bindings. Candidate edits are applied in the same target-object order across configurations, using the same editing and validation procedure. After editing, GPT-5.6 Sol answers an independent test set using the revised Wiki.

Compared with repeating the strongest single model, using three models identifies 34 additional defects on MuSiQue and 16 on SWE-QA-Pro, improving QA scores by 0.6 and 0.3 points, respectively (Table 20). Although most defects are found by more than one model, each model also identifies defects missed by the other two (Figure A1).

Table 20: **Multi-model probing with equal execution counts.** Defect counts include only distinct, verified defects. QA scores are measured on the revised Wiki.

| Probing configuration | MuSiQue | | SWE-QA-Pro | |
|---|---|---|---|---|
| | Unique defects | F1 | Unique defects | Score |
| No probing | – | 76.3 | – | 38.8 |
| Qwen3.6 ×3 | 142 | 78.6 | 102 | 39.7 |
| GPT-5.6 Sol ×3 | 164 | 79.2 | 115 | 40.2 |
| Claude Opus 5 ×3 | 158 | 79.0 | 121 | 40.5 |
| **Three models** ×1 | 198 | 79.8 | 137 | 40.8 |

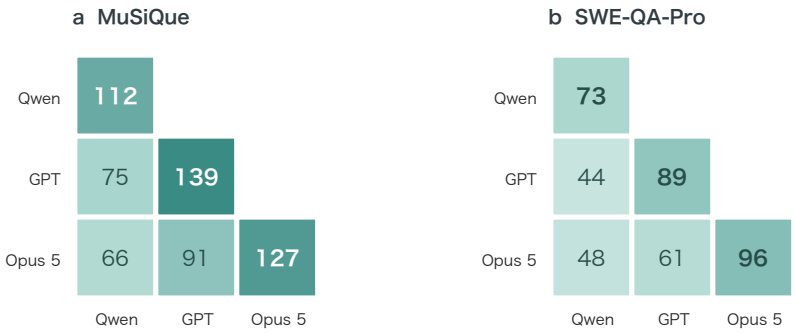


Figure A1: Overlap of defects found in one execution per model. Diagonal entries count defects found by each model. Lower-triangular entries count pairwise intersections.

# E Prompt Templates

## E.1 Knowledge Construction

**Claim Extraction**

**Instruction**
Extract independently verifiable atomic statements from the supplied sources.

1. Split separate facts and preserve names, values, units, dates, and conditions needed to interpret each statement.
2. For code, describe only behavior supported by the supplied code. Do not add facts from outside knowledge.
3. Bind each statement to its supporting source location. Copy the supplied identifier or location into `source` rather than inventing a reference.

**Input**

```
Source text, identifiers, locations and versions: {sources}
```

**Output**

```
{"claims": [{"claim": "...", "source": "..."}]}
```

**Page Compilation**

**Instruction**
Create or update the page body using the supplied Claims.

1. Retain existing content supported by current-source Claims and incorporate the assigned Claims. If existing content is empty, create the page from the assigned Claims.
2. Organize the content into concise paragraphs and preserve factual conditions. Do not add unsupported statements.

**Page Compilation (continued)**

3. Cite the supporting input Claim IDs for each paragraph and return the complete page body.

**Input**

```
Page title: {page_title}
Existing content and Claim references: {existing_content}
Current-source Claims supporting it: {existing_claims}
Assigned Claims and source locations: {claims}
```

**Output**

```
{"paragraphs": [{"text": "...", "evidence": ["C1"]}]}
```

### E.2 Probe Generation

**Question Generation**

**Shared instruction**

Generate one self-contained question, its answer, and the input Claim IDs needed to support that answer. Apply only the instruction for the selected strategy. Use only the supplied evidence and preserve necessary conditions. The answer must require exactly the requested number of Claims. Select a sufficient evidence set in which removing any Claim prevents deriving the answer from that set. Return only supplied Claim IDs. Do not reveal the answer in the question. If no valid question meets these requirements, return `null` for `question` and `answer`, with an empty `evidence` list.

**Claim-QG**

1. Follow the supplied Claim path or paths and their source-supported relations. Apply the specified answer operation to obtain the expected answer.
2. Match the requested type. A single-Claim question requires one Claim. A relational question connects evidence from two pages or code entries. A compositional question combines at least three Claims.
3. Retain an identifiable starting entity or code entry and the conditions needed to interpret the question. For multi-hop questions, omit intermediate entities and code symbols that the answering agent must discover.

**Page-QG**

Generate a question from the supplied source pages or code files, including pages reached through links. Select the required evidence from their content. The question may use one or multiple pages, and must satisfy the shared required Claim count.

**Random-QG**

Generate a question from the randomly sampled source Claims. Do not assume that co-occurring Claims are related. Return an invalid result if they cannot support a question satisfying the shared required Claim count.

**Input**

```
Strategy: {strategy}
Required Claim count: {required_claim_count}
Claims, IDs and source passages: {claims_and_sources}
```

**Additional input for Page-QG**

```
Selected pages or code files and Claim mappings: {page_context}
```

**Additional input for Claim-QG**

```
Claim paths and supported relations: {evidence_chain}
Claim-to-page or code-entry mapping: {page_correspondence}
Answer operation and expected answer: {answer_specification}
Question type: {question_type}
```

**Output**

```
{"question": "...", "answer": "...",
 "evidence": ["C1", "C2"]}
```

### Independent Validation Questions

**Instruction**
Generate one validation question about the supplied affected knowledge.

1. Use only the supplied Claims and source-supported relations. Select evidence sufficient to determine a unique answer, with every selected Claim needed to derive it.
2. Require different Claims or a different evidence chain from each excluded probe. Do not paraphrase an excluded question or reuse its complete evidence chain.
3. Make the question self-contained and preserve necessary conditions. For multi-hop questions, omit intermediate entities and code symbols that must be discovered. Do not reveal the answer.

Return the answer and its supporting input Claim IDs. If no valid question can be formed, return `null` for `question` and `answer`, with an empty `evidence` list.

**Input**

```
Affected Claims, relations and source passages:
{affected_knowledge}
Excluded questions and their evidence chains:
{excluded_probes}
```

**Output**

```
{"question": "...", "answer": "...",
 "evidence": ["C1", "C2"]}
```

### Question Quality Verification

**Instruction**
Check the question, expected answer, and selected evidence against the sources.

1. Verify that the selected Claims and relations support a unique answer and that it matches the expected answer.
2. Check that the evidence is sufficient and that removing any selected Claim makes that evidence set insufficient.
3. If a question type is specified, check its evidence requirements. Single-Claim questions use one Claim, relational questions connect two pages or code entries, and compositional questions combine at least three Claims.
4. Reject questions that reveal the answer or required intermediate entities or code symbols, omit necessary conditions, or depend on unsupported assumptions.

Return `valid` as true only if all applicable checks pass. Give a short reason for the decision.

**Input**

```
Question: {question}
Expected answer: {answer}
Requested question type, if any: {question_type}
Selected Claims, IDs and source locations: {evidence}
Claim-to-page or code-entry mapping: {page_correspondence}
Relations and source passages: {relations_and_sources}
```

**Output**

```
{"valid": true, "reason": "..."}
```

### Closed-Book Answering

**Instruction**
Answer the question without tools or external context. Give a concise answer, or `null` if you cannot determine it.

**Input**

```
Question: {question}
```

**Output**

```
{"answer": "..."}
```

### E.3 QA AND EDIT VALIDATION

**Wiki Question Answering**

**Instruction**
Answer the question using evidence obtained through the Wiki tools.

1. Search for relevant pages and read their content. Follow page links or search again to obtain additional evidence needed for the answer.
2. Read code only through source references returned by the Wiki. Cite only pages and paragraphs obtained through the tools.
3. Give a concise answer supported by the collected evidence and cite the support for each conclusion. If evidence is insufficient, return a `null` answer and retain relevant citations.

Use at most 20 tool calls.

**Input**

```
Question: {question}
```

**Output**

```
{"answer": "...",
 "citations": [{"page": "...", "paragraph": "..."}]}
```

**Derived Claim Generation**

**Instruction**
Propose one compact, self-contained Claim that makes explicit a relation or combined conclusion needed to answer the question.

1. Derive the statement jointly from the supplied Claims and verified relations, preserving all applicable conditions.
2. Do not merely restate one Claim or add external facts or causal assumptions.
3. List the supporting input Claim IDs and select a suitable page from the supplied pages. Copy its identifier into `page`.

If no supported statement or suitable page exists, return `null` for `claim` and `page`, with an empty `evidence` list.

**Input**

```
Question: {question}
Claims, IDs and source passages: {claims_and_sources}
Verified relations: {relations}
Existing related pages and IDs: {pages}
```

**Output**

```
{"claim": "...", "evidence": ["C1", "C2"],
 "page": "..."}
```

**Source Verification**

**Instruction**
Check whether the candidate is supported by the supplied current-source evidence. Apply the check for its type.

1. A source Claim must follow from the cited source passage without changing its factual conditions.
2. A relation must connect the stated endpoints and have supporting source evidence.
3. A derived Claim must follow jointly from its recorded premises and verified relations without losing conditions.
4. A source binding must point to the supporting content at the supplied current-source location.

Do not rewrite the candidate. Return `supported` as true or false and a short reason.

**Input**

```
Candidate and type: {candidate}
Current source passages and locations: {sources}
Supporting premises and relations, if applicable: {support}
```

**Output**

**Source Verification (continued)**

```
{"supported": true, "reason": "..."}
```

**Answer Judging**

**Instruction**
Judge the candidate answer using the question, reference answer, and source evidence. Accept equivalent wording, but require all requested conclusions and conditions. Mark null, contradictory, or incomplete answers as incorrect. Return `correct` as true or false and a short reason.

**Input**

```
Question: {question}
Reference answer: {answer}
Candidate answer: {candidate_answer}
Source evidence: {support}
```

**Output**

```
{"correct": true, "reason": "..."}
```

**Edit Reliance Judgment**

**Instruction**
Decide whether answering the question from its gold evidence requires each candidate modification.

1. A Claim is required if the answer or a necessary intermediate step depends on the fact it states, regardless of wording.
2. A link or index entry is required if it connects or retrieves pages whose evidence must be combined to reach the answer.
3. A derived Claim is required if it states a conclusion composed from facts in the gold evidence.
4. Judge each candidate independently using only the supplied text.

**Input**

```
Question: {question}
Reference answer: {answer}
Gold supporting evidence: {supporting_evidence}
Candidate modifications (ID, type, content): {candidates}
```

**Output**

```
{"judgments": [{"id": "E1", "required": true,
                "reason": "..."}]}
```

# F Probe and Local Repair Examples

## F.1 Documents: Restoring a Relation in a Multi-hop Chain

**Source Claims and Progressive Probes**

**Sources.** The entries for *Arena of Khazan*, *Tunnels & Trolls*, and Ken St. Andre.

**C1** *Arena of Khazan* is an adventure for *Tunnels & Trolls*. *Source: sentence 1 of the adventure entry.*

**C2** *Tunnels & Trolls* was designed by Ken St. Andre. *Source: sentence 1 of the game entry.*

**C3** Ken St. Andre was born in Ogden, Utah. *Source: sentence 1 of the person entry.*

**Evidence chain.** Adventure → game → designer → birthplace.

**Source Claims and Progressive Probes (continued)**

| Probe | Question | Expected answer |
|---|---|---|
| Single-Claim | In which city was Ken St. Andre born? | Ogden (C3). |
| Relational (2-hop) | In which city was the designer of *Tunnels & Trolls* born? | Ogden (C2, C3). |
| Compositional (3-hop) | In which city was the designer of the game system used by *Arena of Khazan* born? | Ogden (C1–C3). |

**Diagnosis and Candidate Repair**

**State to repair.** C1 links the adventure page to the game page, but the game page omits C2. The person page retains C3.

**Diagnosis.** The single-Claim probe is answerable. Both multi-hop probes require the designer relation on the game page. If access stops there while the source Claim graph contains C2, the candidate edit restores that relation.

**Local edit.** Add C2 to the game page, bind it to sentence 1 of the source entry, and link to the designer's page through this relation.

**Expected re-answer.** Starting from the adventure page, identify *Tunnels & Trolls*, then Ken St. Andre, and finally retrieve Ogden from the person page.

### F.2 Code: Determining Which Adapter Handles a Request

**Source Claims and Progressive Probes**

**Sources.** Adapter registration, selection, and sending in Requests' `Session`.

**C1** Adapters are stored in descending order of URL-prefix length. *Source:* `Session.mount`.

**C2** Prefixes are checked in stored order, returning the adapter for the first prefix matching the target URL. *Source:* `Session.get_adapter`.

**C3** Sending obtains the target URL's adapter through `get_adapter`, then calls that adapter's `send`. *Source:* `Session.send`.

**Question setup.** Suppose a Session first registers adapter A for `https://files.example/`, then adapter B for `https://`. A request to `https://files.example/a.csv` matches both prefixes.

**Evidence chain.** Prefix ordering → adapter selection → request sending.

| Probe | Question | Expected answer |
|---|---|---|
| Single-Claim | After both registrations, is A's prefix or B's prefix stored first? | A's prefix (C1). |
| Relational (2-hop) | Does `get_adapter` select A or B for the target URL? | A (C1, C2). |
| Compositional (multi-hop) | When downloading this file, is the request actually sent by A or by B, which was registered later? | A (C1–C3). |

**Diagnosis and Candidate Repair**

**State to repair.** All three Claims are present. The sending page links to the selection page, but the selection page lacks a link to the registration page.

**Diagnosis.** If the agent follows the sending page to the selection page, it can establish that both prefixes match but lacks evidence for their checking order. C1 already records that order on the registration page.

**Local edit.** Link the selection page's description of `self.adapters` to the `mount` registration page.

**Diagnosis and Candidate Repair (continued)**

**Expected re-answer.** The registration page establishes that A's longer prefix comes first, the selection page therefore identifies A as the first match, and the sending page confirms that A handles the request.