1D-WIKI-BENCH: EVALUATING KNOWLEDGE PRODUCTION AND CONSUMPTION FROM REQUIREMENTS TO CODE

Anonymous authors
Paper under double-blind review

ABSTRACT

The value of reusable project knowledge depends on artifact quality and its utility across requirement clarification and code implementation. We introduce 1D-WIKI-BENCH, a dataset and evaluation framework that connects knowledge production, evidence use, and downstream development. It contains 133 tasks across 10 source groups spanning 12 repositories, with fixed sources, vague requests, reference specifications, and implementation criteria. Knowledge systems build artifacts before target requests are revealed. A clarification agent produces a specification for a separate implementation session; both stages reuse the artifacts while retaining raw-source access. We evaluate artifact quality, evidence acquisition and application, specification and patch rubrics, and stage-wise resource use. Raw-only achieves the highest specification and patch rubrics. On matched tasks, all six Wiki-producing methods improve implementation evidence acquisition by 2.25–10.95 percentage points over Raw-only, yet score 1.91–4.80 points lower on the patch rubric. In 71 OpenWiki task pairs, criteria linked to newly acquired implementation evidence gain credit overall, while other criteria lose more. Traces show direct evidence provision and navigation to original sources. OpenWiki and Graphify save 7.77M and 21.45M consumption tokens, respectively, but neither offsets its construction cost over the evaluated tasks. These results motivate evaluating knowledge production and consumption jointly through artifact quality, task fulfillment, and total reuse cost.

1 INTRODUCTION

Software development agents edit code and run tests through tool use (Yang et al., 2024). In deployed project workspaces, agentic teams can combine specialized skills, knowledge sources accessed through the Model Context Protocol (MCP), and recovery procedures to deliver software changes. This integrated support makes the marginal benefit of a prepared Wiki difficult to isolate, even as agents repeatedly recover project context from documents and code.

Knowledge engineering systems prepare Wikis, indexes, and graphs as reusable intermediates between project sources and agents. Context files and approaches such as ACE likewise organize guidance for reuse (Mohsenimofidi et al., 2026; Zhang et al., 2026). Additional context can also increase inference cost without improving task success (Gloaguen et al., 2026). We ask: "*What incremental benefit does prepared project knowledge provide across requirement clarification and code implementation, and when does reuse offset its construction cost?*"

Existing evaluations address parts of this question. WikiGenBench and CodeWikiBench assess generated artifacts, and SWD-Bench evaluates documentation through questions and assisted issue solving (Zhang et al., 2025; Hoang et al., 2026; Wang et al., 2026). Clarification and reasoning benchmarks examine how agents interpret requests (Vijayvargiya et al., 2026; Liu et al., 2026; Zhou et al., 2026). Evaluating marginal utility requires a common consumption workflow that compares prepared knowledge with direct-source access and follows its contribution across the handoff from requirements to code.

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.

We introduce **1D-WIKI-BENCH**[1], comprising 133 tasks in 10 source groups spanning 12 code repositories and additionally collected public business knowledge. A clarification agent turns a vague request into a specification for a separate implementation session. Both stages use a common consumer model and fixed stage budgets, retain original-source access, and reuse knowledge built before task disclosure. This setup measures added value over direct-source access and accounts for construction cost across requests.

Evaluation links artifact quality and preparation cost to evidence use, specifications, and patches. OpenWiki gains credit on evidence-linked criteria while losing more elsewhere in the same task, separating local evidence gains from complete implementation.

Our contributions are:

- A dataset and construction protocol that links real software changes to fixed sources, vague requests, reference specifications, and implementation criteria, enabling artifact reuse across independent tasks.
- An evaluation framework linking artifact quality and construction cost to specification and patch quality through explicit intermediate specifications and evidence-use diagnostics.
- An empirical analysis of seven knowledge systems and direct-source access that links local evidence gains to incomplete task coverage, distinguishes evidence provision from source navigation, and measures consumption savings against construction cost.

2 Related Work

2.1 Related Benchmarks

WikiGenBench evaluates generated articles, and CodeWikiBench applies hierarchical rubrics to repository documentation (Zhang et al., 2025; Hoang et al., 2026). SWE-QA measures repository understanding; SWE-bench evaluates issue resolution through executable tests (Peng et al., 2026; Jimenez et al., 2024). SWD-Bench connects documentation assessment with functionality questions and documentation-assisted issue solving (Wang et al., 2026). 1D-WIKI-BENCH compares prepared knowledge with direct-source access within a common clarification-to-implementation workflow. Each artifact serves multiple requests, linking its marginal downstream utility to artifact quality and amortized construction cost.

Intermediate evaluations examine how agents reach an implementation. Ambig-SWE studies interactive clarification, RACE-Bench assesses feature-development reasoning, and SWE-RPG uses reference requirements and plans to diagnose trajectories (Vijayvargiya et al., 2026; Liu et al., 2026; Zhou et al., 2026). Our specification-to-implementation handoff traces how prepared knowledge shapes the development target and its realization in code.

2.2 Knowledge Production and Access Methods

STORM organizes article generation through questions and outlines, while CodeWiki structures repository documentation hierarchically (Shao et al., 2024; Hoang et al., 2026). LLM-Wiki combines linked pages with search; WikiLoop learns construction and navigation; WiCER refines compilation through diagnostics (Ming et al., 2026b;a; Huerta, 2026). RAPTOR, GraphRAG, SAG, and RepoGraph expose summary trees, entity graphs, relational retrieval, and code graphs, respectively (Sarthi et al., 2024; Edge et al., 2025; Wu et al., 2026; Ouyang et al., 2025).

Context engineering includes developer guidance and accumulated experience (Mohsenimofidi et al., 2026; Gloaguen et al., 2026; Zhang et al., 2026). ClarifyGPT examines requirement ambiguity through behavioral disagreement (Mu et al., 2024); RAGAs and RAGChecker diagnose retrieval and answer generation (Es et al., 2024; Ru et al., 2024). We compare knowledge systems within a fixed consumer workflow, retaining native interfaces and shared source access. Evidence diagnostics trace their contributions to specifications and patches.

[1] 1D stands for OneDay, reflecting the aspiration to help developers complete the workflow from requirements to code within a single day.

3 1D-WIKI-BENCH

3.1 OVERVIEW

1D-WIKI-BENCH evaluates knowledge prepared before individual development requests are known (Figure 1). Each source group combines fixed business and engineering sources. A method constructs reusable knowledge artifacts for the group and reuses them across independent tasks. Consumption passes a generated specification from a clarification agent to a separate implementation session. Both agents can access the fixed sources; artifact-assisted conditions additionally expose the prepared Wiki, graph, or index.

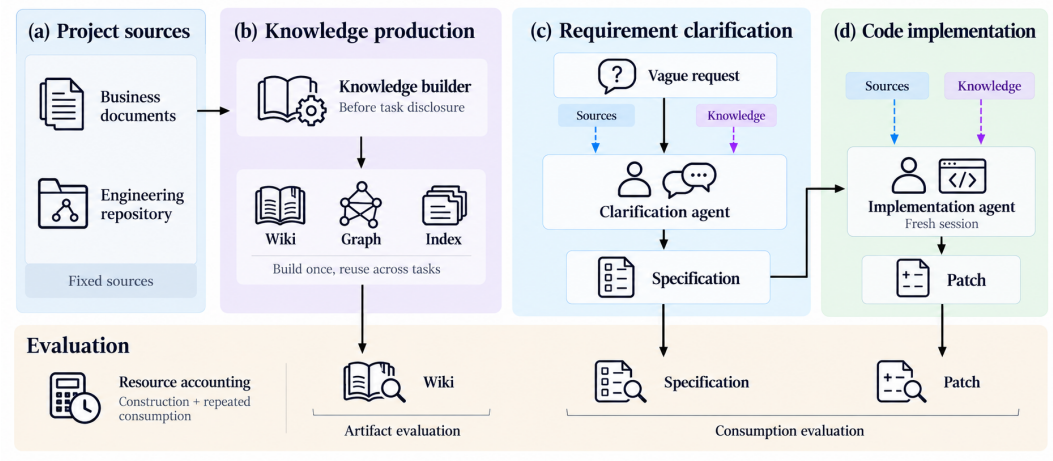

Figure 1: Benchmark overview. (a) Fixed project sources support (b) knowledge construction before task disclosure. A clarification agent produces a specification (c), which an implementation agent uses in a fresh session to generate a patch (d). Both agents can access the original sources; Wiki, graph, and index conditions additionally expose prepared knowledge, while Raw-only provides sources alone. Each agent independently chooses what to consult. Solid black arrows indicate workflow and evaluation flows; dashed blue and purple arrows indicate access to sources and prepared knowledge, respectively. Evaluation covers Wiki quality, evidence use, specification and static patch rubrics, and construction plus repeated consumption costs.

3.2 TASK DEFINITION

For source group i, let $S_i^0 = B_i^0 \cup E_i^0$ contain the fixed business materials and engineering baseline. A development task j pairs a vague request q_{ij}^0 with a reference specification, an accepted patch, and weighted acceptance criteria. The request states the intended change; project rules and implementation constraints must be recovered from S_i^0. Reference outputs and criteria are held out from the consumer agents.

Task-independent knowledge production. Method r constructs reusable knowledge $H_i^{(r)} = h_r(I_i^{(r)})$ from its supported input scope $I_i^{(r)} \subseteq S_i^0$. Here $H_i^{(r)}$ includes the generated Wiki, graph, or index and its native access interface. Construction precedes the disclosure of task requests. The same artifact serves every task in the group, making its preparation cost separable from repeated use.

Requirement clarification and implementation. Let $K_i^{(r)} = S_i^0 \cup H_i^{(r)}$ denote the available knowledge resources. With a common consumer model and stage budgets, the two agents produce

$$(D_{ij}^{(r)}, \tau_{ij,\mathrm{clar}}^{(r)}) = G_{\mathrm{clar}}(q_{ij}^0, K_i^{(r)}), \tag{1}$$

$$(\widehat{P}_{ij}^{(r)}, \tau_{ij,\mathrm{impl}}^{(r)}) = G_{\mathrm{impl}}(q_{ij}^0, D_{ij}^{(r)}, E_i^0, K_i^{(r)}). \tag{2}$$

The specification D states behavior, constraints, and acceptance criteria; $\widehat{P}$ is a patch against the clean baseline. Each trace τ records tool calls and returned evidence. Implementation starts in a fresh session and receives the generated specification, preserving the effect of clarification on the final patch. Raw-only uses $H_i^{(\mathrm{raw})} = \varnothing$ and the same complete sources. End-to-end (E2E) outcomes reflect the generated specification and the subsequent implementation together. Appendix B specifies input manifests and access boundaries.

3.3 Dataset Collection Process

The dataset contains **133 tasks**, **10 source groups**, and **12 distinct code repositories**. Sources comprise 4,432 additionally collected public business documents and 68,072 engineering files. Business sources describe domain concepts, product rules, workflows, and expected behavior. Engineering sources include baseline code, tests, configuration, and technical documentation, which may also describe product behavior. These categories identify source provenance rather than disjoint knowledge domains. Mattermost contains three task-target repositories; each other group contains one. Appendix A reports group-level statistics.

We mine feature requests and bug fixes associated with accepted changes after the fixed baseline. Each reference patch is replayed independently against a clean baseline, with required dependencies recorded. Validation records associate the accepted change with task-specific behavioral checks, available regression checks, or developer review. They distinguish reference-state test success from verified fail-to-pass behavior. Appendix C details validation and reference construction.

Requests come from issue reports or are reconstructed from accepted changes. They preserve the user goal while leaving recoverable project constraints implicit; new product decisions unavailable in the fixed sources remain explicit. Accepted changes, tests, reviews, and project evidence define reference specifications and implementation criteria. Each source group contains 8–19 tasks sharing the same baseline and document snapshots. Input manifests record revisions, formats, and supplied files. Source-group membership is retained in dataset partitions, including composite tasks and their constituent requirements.

4 Evaluation Pipeline

We evaluate three links in the development process: the quality of prepared knowledge, the evidence an agent acquires and applies, and fulfillment of the requested behavior. Resources are recorded at each stage.

4.1 Artifact Quality and Task Fulfillment

Wiki quality. We adapt WikiGenBench's focus and organization criteria (Zhang et al., 2025) to project knowledge. Five dimensions assess page scope, organization, navigation, consistency, and non-redundancy on a 1–5 scale. Scope and organization average sampled content pages; the other dimensions assess the Wiki as a whole. An artifact's Wiki score is the unweighted mean of these five dimensions. Method scores average over successfully constructed Wikis; failed builds are reported separately. SAG and Raw-only have no page-based Wiki score.

Specification and patch rubrics. For stage output $O \in \{D, \widehat{P}\}$, a fixed set of criteria $\mathcal{C}$ specifies the required behavior. Reference specifications define clarification criteria; reference tests and reviewed changes define implementation criteria. With positive weights w_c and fulfillment credits $u_c(O) \in [0, 1]$, the rubric score is

$$R(O) = 100 \, \frac{\sum_{c \in \mathcal{C}} w_c u_c(O)}{\sum_{c \in \mathcal{C}} w_c}. \tag{3}$$

Each judgment cites the corresponding specification passage or patch evidence. The reported implementation evaluation assesses patches statically; dynamic task and regression tests were not run. Appendix D gives scoring details and the executable-check extension.

4.2 Evidence Acquisition and Application

Required evidence acquisition. For each task and stage, a frozen set $\mathcal{Q}$ represents source-grounded knowledge needs, shared across methods. For need q, $a_q = 1$ when sufficient supporting evidence is available in the evaluation packet, and $g_q = 1$ when the access trace records its acquisition. We compute

$$\text{Acquired} = \frac{100}{|\mathcal{Q}|} \sum_{q \in \mathcal{Q}} a_q g_q. \tag{4}$$

Credit requires returned evidence, such as an interface definition or a documented business rule; a search query or filename alone is insufficient. The trace can contain evidence from native knowledge tools, raw sources, or stage inputs.

Evidence application. The evaluator extracts and deduplicates task-relevant knowledge points $\mathcal{Z}(\tau)$ from acquired evidence. Let $v_z(O) = 1$ when output O correctly applies point z. Then

$$\text{Applied} = \frac{100}{|\mathcal{Z}(\tau)|} \sum_{z \in \mathcal{Z}(\tau)} v_z(O). \tag{5}$$

Judgments link the evidence location and trace event to a specification passage or patch location. Acquisition measures coverage of shared knowledge needs. Application uses each agent's acquired knowledge as its denominator, so a high application score can coexist with incomplete requirement coverage. A stage with no valid application probes receives zero. Stage scores are averaged across tasks; Appendix D specifies probe construction, empty outputs, and aggregation.

4.3 Production and Consumption Resources

We record LLM input and output tokens, embedding tokens, and stage wall time separately for construction, clarification, and implementation. Internal knowledge-tool calls contribute to consumption; judge calls are excluded. Accounting retains one generation per task and stage. Construction is charged once per source group, enabling amortized accounting across tasks. Appendix F specifies retained runs and time boundaries.

5 Experiments and Analysis

5.1 Experimental Setup

5.1.1 Compared Knowledge Systems

The comparison includes LLM-Wiki and OpenKB (document-based Wikis); DeepWiki-Open, OpenDeepWiki, and OpenWiki (repository Wikis); Graphify (graph-supported access); SAG (structured retrieval); and Raw-only (direct-source access). We record production resources for the seven builders and assess Wiki quality for the six Wiki-producing systems. All eight source-access configurations follow the same requirement-clarification and implementation protocol, with access to the complete fixed raw sources in both consumer stages. Appendix E gives native input scopes and access mechanisms (Table 5).

5.1.2 Models and Consumer Harness

Production and consumption use `qwen3.7-plus`; vector retrieval uses `text-embedding-v4` with 1024-dimensional embeddings. Wiki construction has a 100-hour completion budget. Each consumer stage has an independent limit of 70 turns and 7,200 seconds. After 30 non-writing tool calls, the harness instructs the agent to finish from acquired evidence; further retrieval and command execution are blocked while writing remains available. Native multi-step retrieval and internal agent workflows are retained within these limits.

`gpt-5.5-0424-global` scores Wikis and consumer outputs. Wiki assessment uses up to 12 deterministically sampled pages, structural measurements, and source excerpts. Consumer rubrics and required-evidence questions are fixed before comparison. Judge prompts omit method identities; criterion credits and evidence references are retained. The source-access ablation in Section 5.3.3 varies the sources available to a common consumer.

5.2 Main Results

5.2.1 Knowledge Quality and Consumer Outcomes

Table 1 reports Wiki quality and downstream outcomes. Wiki quality is conditional on successful construction: 59 artifacts are assessed, nine from LLM-Wiki and ten from each other Wiki-producing method.

Table 1: Knowledge production and consumption outcomes. Evidence-acquisition, evidence-application, and rubric scores are percentages. Bold marks the highest observed score in each column.

Method	Production	Requirement clarification			E2E implementation		
	Wiki score ↑	Evidence acquired ↑	Evidence applied ↑	Rubric (%) ↑	Evidence acquired ↑	Evidence applied ↑	Rubric (%) ↑
LLM-Wiki	2.81	54.37	78.82	60.05	55.39	54.46	49.06
OpenKB	**3.75**	**68.47**	90.77	68.92	57.49	60.06	51.84
DeepWiki-Open	3.02	51.12	91.81	66.78	57.79	61.90	51.77
OpenDeepWiki	3.35	61.01	90.74	66.79	56.77	61.52	52.21
OpenWiki	3.67	62.13	**92.18**	68.67	60.19	**64.83**	53.16
Graphify	3.33	59.92	86.46	66.56	**65.47**	64.23	53.63
SAG	N/A	57.62	89.54	65.82	56.14	61.62	52.41
Raw-only	N/A	61.64	87.81	**70.40**	54.52	64.32	**56.57**

Wiki score uses a 1–5 scale; N/A denotes an inapplicable score. E2E rubric scores assess patches statically. Consumer scores average all 133 tasks per method, including zeros for the 16 tasks affected by LLM-Wiki's failed build. Its Wiki score of 2.81 averages the nine successful artifacts; other Wiki means cover ten.

5.2.2 Production and Consumption Resources

Table 2 reports construction and stage-wise consumption costs. The 69 completed builds use 2,384.40 million LLM tokens, 92.63 million embedding tokens, and 203.65 aggregate hours. The HA Frontend / LLM-Wiki build exceeds the construction budget. Its costs are excluded, so these totals describe completed builds rather than the cost of all attempted builds.

Table 2: Stage-wise production and consumption resources. Production totals cover completed builds; consumption totals cover 133 tasks for seven methods, and 117 for LLM-Wiki.

Method	Production				Clarification		Implementation	
	Completed builds	LLM (M)	Embed. (M)	Time (h)	LLM (M)	Time (h)	LLM (M)	Time (h)
LLM-Wiki	9/10	65.35	0.00	49.93	40.39	4.49	128.01	7.32
OpenKB	10/10	1,521.39	0.00	70.77	83.52	4.74	180.56	7.26
DeepWiki-Open	10/10	3.18	80.13	15.93	69.77	5.26	158.23	9.23
OpenDeepWiki	10/10	156.13	0.00	1.92	80.45	5.22	167.29	7.74
OpenWiki	10/10	110.43	0.00	9.21	61.44	4.91	145.64	8.42
Graphify	10/10	426.53	0.00	23.09	59.34	5.36	134.05	7.48
SAG	10/10	101.38	12.50	32.80	60.99	6.57	140.09	8.24
Raw-only	N/A	0	0	0	66.11	6.92	148.74	9.95
Total	69/70	2,384.40	92.63	203.65	522.01	43.48	1,202.60	65.65

M: million tokens; Embed.: embedding tokens. Consumption tokens sum input (including cached input) and output tokens from retained generations; time sums stage wall times. Totals use unrounded records. LLM-Wiki's 16 failed-construction tasks incur no recorded consumption. Accounting details are in Appendix F.

5.3 Analysis and Discussion

Evidence gains and task fulfillment diverge. Raw-only has the highest clarification and implementation rubrics in Table 1. Clarification evidence acquisition shows mixed differences across methods. During implementation, all six Wiki methods gain acquisition credit on matched tasks ($+2.25$ to $+10.95$ points), yet lose patch-rubric credit (-4.80 to -1.91). These paired differences use 117 shared tasks for LLM-Wiki and 133 for each other method. Acquisition, correct application, and complete implementation measure distinct steps. The uniform acquisition–rubric gap occurs in implementation; the chain-level analysis traces how knowledge contributes to both specifications and patches.

5.3.1 Direct Evidence and Source Navigation

Implementation evidence is predominantly source-based. We map the judge's evidence citations back to the retained tool traces for all seven knowledge systems (Figure 2). Among credited implementation-acquisition probes, 74.8–79.5% cite raw-source access, compared with 0.9–15.7% citing native knowledge access. Source citations also occur in 71.6–81.1% of correctly applied implementation probes. The generated specification provides an additional evidence path between stages.

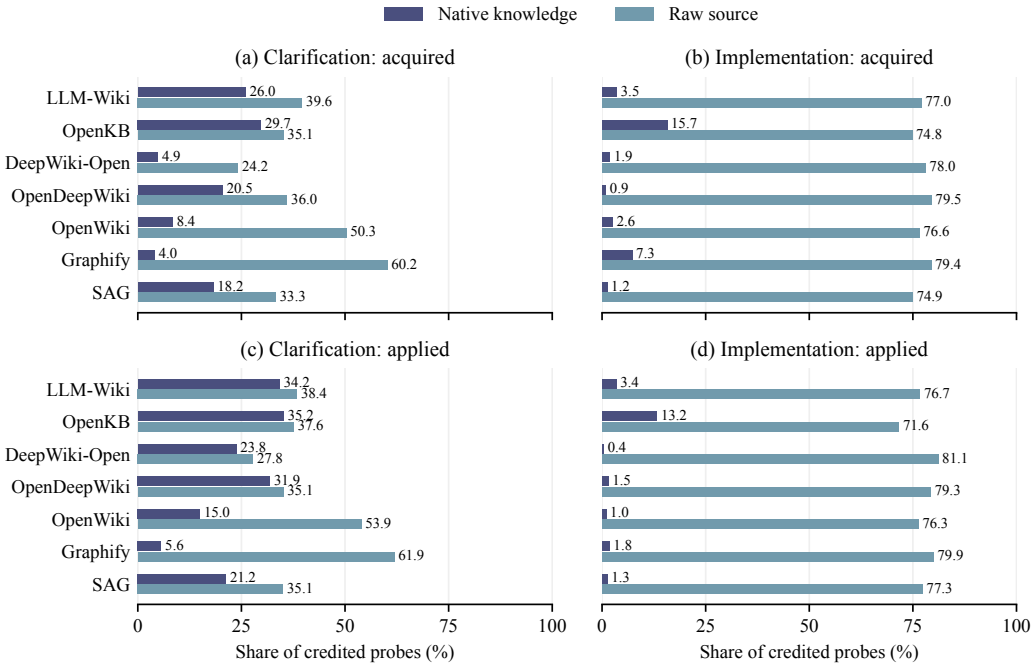

Figure 2: Evidence provenance across knowledge systems. Each bar is the percentage of credited probes with a trace-resolved citation to native knowledge access (Wiki, graph, or index) or raw-source access. Acquisition panels use acquired probes; application panels use correctly applied probes. Probes are pooled across tasks within each method and stage. Categories can overlap; request, prior-specification, and unresolved citations account for additional evidence. These shares describe provenance, whereas Table 1 reports task-averaged scores.

Native knowledge plays different roles across methods and stages. OpenKB and LLM-Wiki have the largest native-access shares in clarification: 29.7% and 26.0% of acquired probes, and 35.2% and 34.2% of correctly applied probes. OpenWiki and Graphify rely more on direct sources: their implementation native shares are 2.6% and 7.3% for acquisition, and 1.0% and 1.8% for application. Appendix G.2 gives all values and denominators.

Wiki access can guide subsequent source reading. For probes acquired by OpenWiki but not Raw-only, an exact-path audit identifies ten clarification probes and two implementation probes whose cited source file appears in an earlier Wiki response and is subsequently opened by the agent. For example, a Lago Wiki page points to `app/models/event.rb`, which the agent then reads to recover required evidence. The Wiki response supplies a navigation path even when the final credited evidence comes from Source. Appendix I.3 gives the matching rule and example.

5.3.2 Local Evidence Gains and Complete Task Coverage

Local acceptance gains can be outweighed by omissions elsewhere. We connect OpenWiki's newly acquired implementation evidence to the acceptance criteria it concerns. Within the 71-task subset with detailed paired rubrics, 173 OpenWiki-only acquisition probes link to 159 distinct criteria; the full 133-task comparison contains 196 such probes. OpenWiki improves credit on 50, ties on 87, and loses on 22. The other 892 criteria in these same tasks show 184 improvements, 479 ties, and 229 losses. Summed weighted credit rises on the evidence-linked criteria but falls by a larger amount elsewhere (Appendix I.2). The two groups show that local evidence gains can coexist with losses in overall implementation coverage.

Zero-scored outputs do not account for the full gap. On matched subsets where both methods obtain nonzero E2E rubric scores, all six Wiki methods remain below Raw-only by 1.13–4.27 points. OpenWiki and Graphify also have fewer zero-scored outputs than Raw-only. Composite tasks show the same acquisition–rubric separation across all six methods. The retained patches document partial implementations, including a Graphify patch that fixes duration parsing but omits trigger conversion and save–reopen consistency. Appendix I provides the paired statistics and implementation examples.

5.3.3 Source-Access Ablation

We vary source access on 12 tasks, four each from Appwrite Console, Outline, and Formbricks. Q receives only the vague request; B adds separately collected business sources; E adds engineering sources; B+E provides both. Agents search raw files without a generated Wiki or index. All conditions produce specifications; E and B+E also generate patches using their own specifications under the main consumer and judge settings.

Table 3: Source-access ablation on 12 tasks. Scores are task means on a 0–100 scale; bold marks the highest value within each stage and column.

Stage	Sources	Rubric ↑	Acquired evidence ↑	Applied evidence ↑
Clarification	Q	61.48	31.17	N/A
	B	63.13	52.13	**92.42**
	E	64.64	52.30	85.12
	B+E	**65.58**	**54.46**	88.77
Implementation	E	**68.83**	45.29	**77.88**
	B+E	67.65	**54.61**	74.07

Q: request only; B: business sources; E: engineering sources. Q has no access-derived application probes. E and B+E implementations consume their own generated specifications.

B+E has the highest clarification rubric and Required Evidence Acquisition scores, improving over Q by 4.10 and 23.29 points, respectively (Table 3). Its clarification rubric is 0.94 points above E. During implementation, B+E raises Required Evidence Acquisition by 9.32 points but lowers the patch rubric by 1.18 points relative to E. Because each patch consumes its condition's specification, this comparison measures the combined effect across both stages. Business-source reads occur in five B+E clarification runs and no implementation runs; all implementation agents read the generated specification. This identifies a route through which business knowledge reaches implementation. Appendix G.3 provides task-level examples.

5.3.4 COST OF REUSING PROJECT KNOWLEDGE

Sharing each artifact across its source group's 8–19 tasks reduces the construction charge per task by 87.50–94.74% relative to charging the same build once per task (Appendix H). This reduction uses repeated construction as its baseline; break-even instead compares construction plus consumption against Raw-only. On the same 133 tasks, OpenWiki and Graphify use 207.08M and 193.39M consumption tokens, compared with 214.84M for Raw-only. Their savings in consumption are 7.77M and 21.45M tokens. After adding their respective construction costs once, the observed totals are 317.51M and 619.92M tokens; neither reaches token break-even within these tasks.

Consumption time reflects navigation and implementation effort. OpenWiki's mean two-stage consumption time is 361.0 seconds versus 456.8 for Raw-only on 133 matched tasks, a 21.0% reduction. Clarification uses 2.9 fewer model round trips and 4.0 fewer source accesses, alongside 3.5 Wiki accesses. On 42 pairs with identical API configurations and no recorded retries, OpenWiki remains 60.6 seconds faster. Faster implementation also accompanies smaller patches and lower rubric scores. The observed time saving therefore combines navigation, execution conditions, and implementation coverage; it excludes construction and does not imply higher task quality.

Context efficiency depends on retrieval granularity. Whole-page reads and search snippets serve different purposes. In judge-aligned OpenKB calls, 225 full-catalog reads account for approximately 86% of Wiki-page characters, with only six reads directly cited by the judge. In contrast, approximately 70% of OpenDeepWiki topic-page reads receive direct citations despite their length. Citation absence alone does not establish irrelevance, but the contrast identifies repeated catalog expansion as a concrete context burden. Figure 4 and Appendix K summarize timing, returned text, and implementation coverage.

6 CONCLUSION

1D-WIKI-BENCH connects knowledge production to requirement clarification and code implementation through a shared two-stage workflow. On matched tasks, all six Wiki-producing methods improve implementation evidence acquisition by 2.25–10.95 percentage points over Raw-only, yet score 1.91–4.80 points lower on the patch rubric. Across 71 OpenWiki task pairs, gains on evidence-linked criteria coexist with larger losses elsewhere. OpenWiki and Graphify save 7.77M and 21.45M consumption tokens, respectively, without recovering construction costs over the evaluated tasks. Evaluating reusable knowledge therefore requires following evidence through specifications to complete task fulfillment and accounting for construction alongside repeated consumption.

7 LIMITATIONS

Results cover ten public source groups and one consumer model. The two-stage workflow simplifies deployed agents' broader tool integrations and specialized skills. Native inputs and interfaces vary across systems, so comparisons combine representation and retrieval effects. API configurations, concurrency, and retries affect observed time. E2E outcomes combine clarification and implementation; the criterion-level analysis covers 71 OpenWiki task pairs. These comparisons locate evidence paths and coverage gaps without isolating a single causal mechanism. Patch rubrics assess code statically; executable correctness and regressions remain untested.

AI USE STATEMENT

Generative AI is part of the reported experiments: `qwen3.7-plus` participates in knowledge production and runs the clarification and implementation agents, while `gpt-5.5-0424-global` evaluates Wiki quality and consumer outputs. AI tools also assisted experimental code development, figure preparation, and manuscript drafting and revision. The authors take responsibility for the experimental results and final manuscript.

ETHICS STATEMENT

The benchmark draws on publicly available code repositories, engineering documentation, and additionally collected business knowledge. Distribution of source snapshots and derived tasks is subject to the applicable licenses and permissions. Release materials must exclude credentials, personal data, and confidential content. Repository provenance and access requirements are recorded to support reproducibility.

REPRODUCIBILITY STATEMENT

Sections 3.3–5.1 specify source construction, evaluation targets, production inputs, consumer access, and model choices. Reproduction requires the fixed repository and document revisions, input manifests, prompts, task checks, system versions, retrieval configurations, and stage-level resource records. Additional protocol, scoring, system, and accounting details are provided in Appendices B–H.

REFERENCES

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, 2025. URL https://arxiv.org/abs/2404.16130.

Shahul Es, Jithin James, Luis Espinosa Anke, and Steven Schockaert. RAGAs: Automated evaluation of retrieval augmented generation. In Nikolaos Aletras and Orphee De Clercq (eds.), *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, pp. 150–158, St. Julians, Malta, March 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.eacl-demo.16. URL https://aclanthology.org/2024.eacl-demo.16/.

Thibaud Gloaguen, Niels Mündler, Mark Niklas Mueller, Veselin Raychev, and Martin Vechev. Evaluating AGENTS.md: Are repository-level context files helpful for coding agents? In *ICLR 2026 Workshop on Memory for LLM-Based Agentic Systems*, 2026. URL https://openreview.net/forum?id=pLi3A8bscP.

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 Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 5812–5827, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.288. URL https://aclanthology.org/2026.findings-acl.288/.

Juan M. Huerta. Wicer: Wiki-memory compile, evaluate, refine iterative knowledge compilation for llm wiki systems, 2026. URL https://arxiv.org/abs/2605.07068.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 54107–54157, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/edac78c3e300629acfe6cbe9ca88fb84-Paper-Conference.pdf.

Shuhan Liu, Zhiyi Zhao, Xing Hu, Kui Liu, Xiaohu Yang, and Xin Xia. Race-bench: A reasoning-augmented benchmark for repository-level code agents on feature addition, 2026. URL https://arxiv.org/abs/2603.26337.

Haoliang Ming, Feifei Li, and Wenhui Que. Wikiloop: Jointly learning to build and navigate agent-native wikis with downstream feedback, 2026a. URL https://arxiv.org/abs/2607.26604.

Haoliang Ming, Feifei Li, Xiaoqing Wu, and Wenhui Que. Retrieval as reasoning: Self-evolving agent-native retrieval via llm-wiki, 2026b. URL https://arxiv.org/abs/2605.25480.

Seyedmoein Mohsenimofidi, Matthias Galster, Christoph Treude, and Sebastian Baltes. Context engineering for ai agents in open-source software. In *Proceedings of the 23rd International Conference on Mining Software Repositories*, MSR '26, pp. 194–198. ACM, April 2026. doi: 10.1145/3793302.3793350. URL http://dx.doi.org/10.1145/3793302.3793350.

Fangwen Mu, Lin Shi, Song Wang, Zhuohao Yu, Binquan Zhang, ChenXue Wang, Shichao Liu, and Qing Wang. Clarifygpt: A framework for enhancing llm-based code generation via requirements clarification. *Proceedings of the ACM on Software Engineering*, 1(FSE):2332–2354, July 2024. ISSN 2994-970X. doi: 10.1145/3660810. URL http://dx.doi.org/10.1145/3660810.

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 Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 30098–30121, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/4a4a3c197deac042461c677219efd36c-Paper-Conference.pdf.

Weihan Peng, Yuling Shi, Yuhang Wang, Xinyun Zhang, Beijun Shen, and Xiaodong Gu. SWE-QA: Can language models answer repository-level code questions? In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 8230–8245, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.402. URL https://aclanthology.org/2026.findings-acl.402/.

Dongyu Ru, Lin Qiu, Xiangkun Hu, Tianhang Zhang, Peng Shi, Shuaichen Chang, Cheng Jiayang, Cunxiang Wang, Shichao Sun, Huanyu Li, Zizhao Zhang, Binjie Wang, Jiarong Jiang, Tong He, Zhiguo Wang, Pengfei Liu, Yue Zhang, and Zheng Zhang. Ragchecker: A fine-grained framework for diagnosing retrieval-augmented generation. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 21999–22027. Curran Associates, Inc., 2024. doi: 10.52202/079017-0692. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/27245589131d17368cccdfa990cbf16e-Paper-Datasets_and_Benchmarks_Track.pdf.

Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher Manning. Raptor: Recursive abstractive processing for tree-organized retrieval. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 32628–32649, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/8a2acd174940dbca361a6398a4f9df91-Paper-Conference.pdf.

Yijia Shao, Yucheng Jiang, Theodore Kanell, Peter Xu, Omar Khattab, and Monica Lam. Assisting in writing Wikipedia-like articles from scratch with large language models. In Kevin Duh, Helena Gomez, and Steven Bethard (eds.), *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 6252–6278, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.naacl-long.347. URL https://aclanthology.org/2024.naacl-long.347/.

Sanidhya Vijayvargiya, Xuhui Zhou, Akhila Yerukola, Maarten Sap, and Graham Neubig. Ambig-swe: Interactive agents to overcome underspecificity in software engineering. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 146716–146737, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/ed4deb2ba9d4640fb6a000c688c2972e-Paper-Conference.pdf.

Xinchen Wang, Ruida Hu, Cuiyun Gao, Pengfei Gao, and Chao Peng. Evaluating repository-level software documentation via question answering and feature-driven development, 2026. URL https://arxiv.org/abs/2604.06793.

Yuchao Wu, Junqin Li, Xingcheng Liang, Yongjie Chen, Yinghao Liang, Linyuan Mo, and Guanxian Li. Sag: Sql-retrieval augmented generation with query-time dynamic hyperedges, 2026. URL https://arxiv.org/abs/2606.15971.

John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 50528–50652. Curran Associates, Inc., 2024. doi: 10.52202/079017-1601. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/5a7c947568c1b1328ccc5230172e1e7c-Paper-Conference.pdf.

Jiebin Zhang, Eugene J. Yu, Qinyu Chen, Chenhao Xiong, Dawei Zhu, Han Qian, Mingbo Song, Weimin Xiong, Xiaoguang Li, Qun Liu, and Sujian Li. WIKIGENBENCH:exploring full-length Wikipedia generation under real-world scenario. In Owen Rambow, Leo Wanner, Marianna Apidianaki, Hend Al-Khalifa, Barbara Di Eugenio, and Steven Schockaert (eds.), *Proceedings of the 31st International Conference on Computational Linguistics*, pp. 5191–5210, Abu Dhabi, UAE, January 2025. Association for Computational Linguistics. URL https://aclanthology.org/2025.coling-main.349/.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Y Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 86069–86100, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/8a94ff6f922d995d7d3f4ebf4143e442-Paper-Conference.pdf.

Xin Zhou, Chun Yong Chong, Kisub Kim, Yun Peng, Rui Shu, Zihan Wu, Xu Han, Guowen Yuan, Zeyang Zhuang, Jounghoon Kim, Jeongjin Ju, Seongmin Ju, Taein Yoon, and David Lo. A unified issue resolution benchmark for requirement clarification, planning, and code generation for coding agents, 2026. URL https://arxiv.org/abs/2608.09072.

Appendix Overview

Appendix A reports dataset statistics; Appendices B and C describe input protocols and task validation. Appendix D details scoring and evidence annotation, and Appendix E specifies the knowledge systems and execution settings. Resource accounting, search-effort analyses, and reuse calculations appear in Appendices F–H. Appendix I expands the main result with paired comparisons, zero-score decomposition, and implementation examples. Appendix J provides evaluator prompt excerpts; Appendix K examines consumption time and retrieval granularity.

A Detailed Source-Group Statistics

Table 4: Detailed source-group statistics.

Source group	Repos.	Tasks	Business documents	Engineering files
Appwrite Console	1	12	221	2,421
Outline	1	15	213	2,790
Formbricks	1	12	135	4,872
Saleor	1	12	78	7,249
Lago API	1	15	172	7,333
Mattermost	3	8	97	8,362
Logto	1	12	348	8,614
Home Assistant Frontend	1	16	2,756	3,700
Chatwoot	1	12	142	9,361
Nextcloud Server	1	19	270	13,370
Total	**12**	**133**	**4,432**	**68,072**

Repositories are distinct task-target code repositories. Business documents are regular text files in the collected business sources. Engineering files include code, tests, configuration, documentation, and assets. Symlinks, task metadata, and generated artifacts are excluded.

B Input and Access Protocol Details

Section 3.2 defines knowledge production, requirement clarification, and implementation. A method's input manifest records the supplied source revisions, file types, paths, and native format filters. Its production process receives these files before target requests are disclosed. The resulting artifact and native access tools are restored for each independent consumer task.

Both consumer stages can browse, search, and read the complete fixed sources. The clarification agent receives the vague request and writes the specification. The implementation agent starts from a clean engineering baseline in a fresh session and receives that specification together with the request. Reference specifications, accepted patches, and evaluation criteria remain hidden from both agents. Raw-only follows the same procedure without a prepared artifact.

C Task Construction and Validation Details

We collect feature requests and bug fixes associated with accepted changes after the fixed repository baseline. Each reference patch P_{ij} is replayed independently against the baseline:

$$E_{ij}^{+} = \mathrm{Apply}(E_i^0, P_{ij}). \tag{6}$$

The replay starts from a clean baseline, with required dependency changes recorded in the task environment.

For executable tasks, the validation record specifies task-specific behavioral checks $\mathcal{T}_{ij}^{\mathrm{task}}$ and the available regression checks $\mathcal{T}_{ij}^{\mathrm{reg}}$. Reference-state validation requires

$$\mathrm{Pass}\big(E_{ij}^{+}, \mathcal{T}_{ij}^{\mathrm{task}} \cup \mathcal{T}_{ij}^{\mathrm{reg}}\big) = 1. \tag{7}$$

Validation records identify the test version, covered behaviors, and available baseline replay results. Passing in the reference state establishes compatibility with the accepted change. A fail-to-pass designation additionally requires a task-specific failure on the baseline. For developer-reviewed tasks, the reviewed change and criterion-level rationale provide the validation evidence.

C.1 Requirement and Reference Construction

Requests are taken from issue reports or reconstructed from accepted changes. They retain the user goal while leaving project-specific constraints and implementation details to be recovered from the fixed sources. New product decisions that those sources cannot establish remain explicit in the request. Each task records its request origin. Accepted changes, tests, reviews, and project evidence inform the reference specification and acceptance criteria. Reference knowledge needs connect these criteria to evidence in the request and fixed sources, separating required behavior from choices specific to the reference implementation.

D Scoring and Evidence Annotation Details

D.1 Production-Side Wiki Quality

We adapt WikiGenBench's focus and organization criteria (Zhang et al., 2025) to project knowledge and extend assessment to the structure of an entire Wiki. We assess five dimensions:

- **Page scope:** whether a page has a clear topic boundary and contains the information needed for that topic without drifting into unrelated material.
- **Organization:** whether the page arranges concepts, rules, procedures, and implementation details in a coherent hierarchy.
- **Navigation:** whether the Wiki's entry points, index pages, and links provide usable paths to relevant pages and related topics.
- **Consistency:** whether terminology, facts, and constraints agree across pages and with the fixed sources.
- **Non-redundancy:** whether the Wiki avoids unnecessary repetition while preserving context needed to understand individual pages.

Each dimension uses a 1–5 rubric, with higher scores indicating better quality. Page scope and organization are scored on content pages and averaged within each Wiki. Navigation, consistency, and non-redundancy are assessed at the Wiki level. The production score for an artifact is the unweighted arithmetic mean of its five dimension scores:

$$\mathrm{Wiki}_i^{(r)} = \frac{1}{5} \sum_{d \in \{\mathrm{scope,org,nav,cons,red}\}} s_{i,d}^{(r)}. \tag{8}$$

Each artifact is scored once. The method-level Wiki score averages successful artifacts and retains the 1–5 scale. LLM-Wiki's 2.81 averages nine successful source-group builds; the failed HA Frontend build has no artifact-quality judgment. Its 16 associated tasks remain in the 133-task consumer comparison with zero scores. Dimension-level scores support diagnostic analysis. SAG and Raw-only have no page-based Wiki score.

D.2 Requirement Clarification Rubric

The reference specification defines acceptance criteria $\{c_{ijk}\}_k$ with fixed positive weights w_{ijk}. Each criterion receives a fulfillment credit in $[0, 1]$, with partial credit for partially satisfied requirements. Equation 3 applies with the stage-specific criteria and weights. Each judgment cites the generated specification. Scores average all 133 tasks, assigning zero to missing outputs.

D.3 E2E Implementation Rubric

The implementation rubric is derived from reference tests and code changes from the corresponding code review (CR). Tests provide behavioral criteria, and the reviewed reference changes inform

code-level requirements. Together they define criteria $\{e_{ijk}\}_k$ with fixed positive weights v_{ijk}. The rubric is fixed before method comparison and evaluates fulfillment of these requirements without requiring a textual match to the reference patch. Equation 3 applies with the stage-specific criteria and weights. Criterion credits lie in $[0, 1]$ and assess the submitted patch, including whether it preserves existing behavior required by the rubric. When the agent concludes that no code change is needed and consequently submits no patch, the task receives zero implementation credit under the empty-output rule. This is a task-level outcome, distinct from criterion-level credit for preserving required behavior within a submitted patch. For a nonempty patch, credit for preserved behavior contributes only to the corresponding criteria; it does not establish that the requested changes are complete. The method-level score averages all 133 tasks and is reported separately from binary executable resolution.

D.4 Executable-Check Extension

The reported main comparison and source-access ablation use static patch rubrics; dynamic task and regression tests were not run. The following definition specifies the executable extension of the evaluation protocol.

Executable evaluation applies the candidate patch to the baseline and runs the task-specific behavioral and available regression checks:

$$r_{ij}^{(r)} = \mathbf{1}\left[\mathrm{Pass}\Big(\mathrm{Apply}(E_i^0, \widehat{P}_{ij}^{(r)}), \mathcal{T}_{ij}^{\mathrm{task}} \cup \mathcal{T}_{ij}^{\mathrm{reg}}\Big) = 1\right]. \tag{9}$$

For the common set $\mathcal{V}$ of valid executable tasks,

$$\mathrm{Resolved}^{(r)} = \frac{100}{|\mathcal{V}|} \sum_{(i,j)\in\mathcal{V}} r_{ij}^{(r)}. \tag{10}$$

Resolved measures success on the declared checks, with test coverage and regression-check availability reported alongside the score. Patch errors, build failures caused by the method, and budget exhaustion count as method failures. Infrastructure failures are recorded separately and resolved before fixing the common comparison set.

D.5 Evidence Acquisition and Application

These diagnostics relate stage-specific knowledge needs and access traces to the resulting specification or patch.

Required Evidence Acquisition. The evaluator fixes stage-specific probes from source-grounded reference knowledge needs before comparison. Clarification probes concern intended behavior and constraints; implementation probes concern interfaces, dependencies, and project conventions needed for the implementation criteria. A probe receives credit when sufficient evidence is available in the evaluated evidence packet and the access trace shows that the agent actually acquired it. The score is the percentage of credited probes, averaged across tasks. For knowledge-system methods, the packet contains returned tool evidence, including accessed raw-source content; Raw-only is assessed against its allowed original sources. Source and acquisition references are retained for each judgment.

Evidence Application. The access trace identifies the agent's queries and returned evidence. The evaluator extracts task-relevant knowledge points from this evidence and checks whether the stage output uses them correctly: in the specification's behavior, constraints, and acceptance criteria for clarification, or in candidate code changes for implementation. The score is the percentage of extracted knowledge points correctly applied, averaged across all 133 tasks. A task with no valid probes receives zero, and probe coverage is retained with the evaluation records.

Acquisition measures coverage of shared knowledge needs; application measures correct use of relevant acquired evidence. Application denominators vary with each agent's acquired knowledge, so the score is not a measure of complete requirement coverage. Wiki scores assess the constructed artifact, while stage rubrics assess fulfillment of the requested behavior.

E KNOWLEDGE SYSTEMS AND EXECUTION DETAILS

Table 5: Production inputs and consumption access. Input scopes follow native file support; both consumer stages can also access the complete fixed raw sources.

	Production	Consumption	
Method	Input scope	Retrieval target	Access mechanism
LLM-Wiki	$B_d + E_d$	Wiki pages	Field matching, BM25, and link following
OpenKB	$B_d + E_d$	Wiki and retained documents	Wiki navigation; PageIndex for long documents
DeepWiki-Open	Repository	Repository chunks	Dense retrieval
OpenDeepWiki	Repository	Wiki documents	Document search and reading
OpenWiki	Repository	Wiki pages and source files	Wiki-first file search and reading
Graphify	$B_d + E_d + E_c$	Knowledge graph	Keyword seeds and graph traversal
SAG	$B_d + E_d$	Event index and source chunks	Semantic or lexical seeds and entity-based SQL joins
Raw-only	None	Complete raw sources	Directory browsing, text search, and file reading

B_d: business documents; E_d: engineering documents; E_c: source code, configuration, and existing tests. Repository denotes native-supported files within the supplied code repositories, including in-repository documentation. Separately collected business and engineering documents are provided where the native interface supports them. LLM-Wiki and SAG use Markdown documents in the selected configuration; other builders retain their native format filters. Raw-only has no construction stage.

Table 5 records native construction inputs and access mechanisms. Document-based and repository-based categories describe input interfaces; either can contain business rules and engineering knowledge.

Document-based Wikis. LLM-Wiki (Ming et al., 2026b) and OpenKB[2] construct Wikis from supported business and engineering documents. LLM-Wiki organizes linked knowledge pages. OpenKB produces index, summary, concept, and entity pages while retaining source documents for detailed lookup.

Repository-based Wikis. DeepWiki-Open,[3] OpenDeepWiki,[4] and OpenWiki[5] generate documentation from fixed repositories and their native-supported files. These inputs include repository documents describing product behavior and implementation details. The systems expose dense source retrieval, document matching, or Wiki navigation, as detailed in Table 5.

Graph-based Wiki. Graphify[6] constructs a graph from supported business documents, engineering documents, and repository code, then exports Markdown pages through its Wiki option. Its native query workflow traverses neighborhoods around keyword-matched seed nodes to assemble context.

Index-assisted and Direct-source Access. SAG (Wu et al., 2026) indexes Markdown documents from business and engineering sources as events associated with entities. It retrieves evidence

[2] https://github.com/VectifyAI/OpenKB
[3] https://github.com/AsyncFuncAI/deepwiki-open
[4] https://github.com/AIDotNet/OpenDeepWiki
[5] https://github.com/langchain-ai/openwiki
[6] https://github.com/Graphify-Labs/graphify

through seed selection and relational expansion. Raw-only provides direct browsing, text search, and reading over the complete fixed sources.

F Resource Accounting Details

Token Cost. For each stage, token cost is the sum of input and output tokens across all model calls, including internal knowledge-system workflows and request retries within the retained generation. We retain input and output counts separately and report their sum in millions of tokens (M) in Table 2. Embedding tokens are accounted for separately. Cached input and reasoning tokens are included once within their respective input and output totals. For an incrementally extended artifact, construction cost includes both its retained base and the extension. Production is charged once per source group; clarification and implementation are charged per task. Consumption accounting selects one retained generation per task and stage, excluding superseded runs; a reused clarification is charged once to the run that produced it. When clarification determines that implementation is unnecessary and no implementation agent is invoked, E2E tokens and time are zero. Evaluator calls are recorded separately from the production and consumption costs. Raw-only has zero knowledge-production cost.

Elapsed Time. Stage time is wall-clock duration from the start of stage execution until its artifact or output is ready. It includes model calls, retrieval, tool execution, and internal retries. Production ends when the knowledge artifact and its required indexes are ready for consumption. Clarification ends when the specification is complete; implementation ends when the candidate patch is delivered. Agent-invoked tests belong to implementation time, while subsequent benchmark evaluation is timed separately. We record hardware, concurrency, and cache settings with the runs, and separate external scheduling delay from stage execution. Parallel internal calls contribute their elapsed critical-path time rather than a sum of call durations. When a source group requires separate repository builds or an artifact is extended incrementally, we additionally report aggregate production time as the sum of those native build stages.

Production includes retained base builds, extensions, and all repository components. HA Frontend / LLM-Wiki exceeded the 100-hour budget; its failed-build costs are excluded from completed-build totals. These totals are conditional on completed builds and do not represent expenditure on all construction attempts.

G Observed Search Effort and Implementation Gains

G.1 Artifact Quality and Observed Search Effort

Across OpenWiki's ten source groups, aggregate Wiki score has weak observed rank association with implementation gains over matched Raw-only tasks (Spearman $\rho = 0.11$ for Required Evidence Acquisition and $\rho = 0.05$ for the rubric). Artifact assessment concerns sampled page quality, whereas these task-level outcomes also depend on retrieval and implementation. The unit of this exploratory comparison is the source group ($n = 10$).

Table 6 reports the mean of ten within-source-group contrasts, with 95% intervals from bootstrapping source groups. All 133 matched tasks contribute to the comparisons.

Table 6: Difference in knowledge-method gains between tasks with higher and lower Raw-only E2E token demand within each source group. Scores are on a 0–100 scale.

Method	Δ Evidence acquisition	Δ Patch rubric
OpenWiki	$+7.28$ $[-4.43, +18.83]$	-3.01 $[-10.88, +4.59]$
Graphify	$+11.72$ $[-0.25, +25.45]$	-4.01 $[-10.91, +2.80]$

G.2 Evidence Provenance by Method and Stage

We link each judge probe's acquisition references to the selected candidate's retained tool trace. An explicit tool-call identifier resolves directly; a step reference resolves when a unique call matches,

using the cited filename when available. Reads and searches of generated artifacts, native graph or index queries, and reads of their saved responses count as native knowledge access. Reads and searches of the fixed source tree or engineering worktree count as raw-source access. Requests and prior specifications form separate categories. Ambiguous mixed-source calls and unmatched references remain unattributed.

For acquisition, the denominator contains probes credited as both evidence-available and evidence-acquired. For application, it contains knowledge points judged correctly applied. For origin o, the plotted share is $100\,N_o/N$, where N_o counts credited probes with at least one resolved citation to that origin. A probe may cite both native knowledge and raw sources; the two shares need not sum to 100%. Request, specification, and unresolved citations remain in the denominator. These pooled shares describe citation provenance. Acquisition and application use different probe sets, so their shares do not form a conversion rate.

The analysis covers retained candidates for 133 tasks per method, except LLM-Wiki (117). Stages without scored probes contribute no probe observations. Table 7 gives the denominators used in Figure 2; the underlying records retain unresolved-reference counts and task identifiers.

Table 7: Provenance of credited evidence probes. Native and source columns are percentages of the corresponding N; categories may overlap.

	Acquired probes			Correctly applied probes		
Method	N	Native	Source	N	Native	Source
Requirement clarification						
LLM-Wiki	565	26.0	39.6	1352	34.2	38.4
OpenKB	717	29.7	35.1	1633	35.2	37.6
DeepWiki-Open	533	4.9	24.2	1271	23.8	27.8
OpenDeepWiki	630	20.5	36.0	1432	31.9	35.1
OpenWiki	642	8.4	50.3	1534	15.0	53.9
Graphify	618	4.0	60.2	1333	5.6	61.9
SAG	592	18.2	33.3	1332	21.2	35.1
E2E implementation						
LLM-Wiki	648	3.5	77.0	700	3.4	76.7
OpenKB	663	15.7	74.8	779	13.2	71.6
DeepWiki-Open	674	1.9	78.0	788	0.4	81.1
OpenDeepWiki	658	0.9	79.5	797	1.5	79.3
OpenWiki	700	2.6	76.6	814	1.0	76.3
Graphify	757	7.3	79.4	783	1.8	79.9
SAG	650	1.2	74.9	789	1.3	77.3

G.3 Evidence Transfer in the Source-Access Ablation

The traces locate the transfer between stages. Across the 12 B+E runs, five clarification agents used direct file reads on separately collected business documents; no implementation agent did so, while all 12 read the generated specification. In the SSO task, business evidence supplies the registration policy missing from the request. In the datetime task, the B+E specification combines business documentation with the code path. The dashboard task exposes a different outcome: the B+E specification includes time granularity, but its patch omits it, while the E patch modifies the relevant UI. Business evidence can improve requirement interpretation without ensuring that the resulting patch implements it.

H Reuse Cost Calculations

We compare charging knowledge construction separately for each requirement with charging it once for all tasks sharing a source group. For method r and source group i, let $B_i^{(r)}$ be the measured

construction tokens. With n_i tasks, the two construction charges are

$$B^{(r)}_{i,\text{single}} = n_i B^{(r)}_i,$$
$$B^{(r)}_{i,\text{shared}} = B^{(r)}_i. \tag{11}$$

The single-task reference charges the same measured build once per task rather than executing new builds. Under sharing, the per-task construction charge falls from $B^{(r)}_i$ to $B^{(r)}_i/n_i$.

The ten source groups contain 8–19 tasks each, reducing the construction charge per task by 87.50–94.74%. Across the seven methods with a construction stage, the observed coverage corresponds to 915 task-specific build charges versus 69 shared builds. Raw-only has no construction stage.

I Evidence Acquisition and Implementation Coverage

This analysis compares the six Wiki-producing systems with Raw-only on matched tasks. We retain the recorded judgments and report differences in percentage points, computed from task-level scores before rounding. OpenWiki, Graphify, OpenKB, DeepWiki-Open, and OpenDeepWiki use the same 133 tasks as Raw-only. LLM-Wiki uses the 117 tasks shared with Raw-only, excluding the source group without a constructed artifact. This paired analysis is distinct from the 133-task aggregation in Table 1.

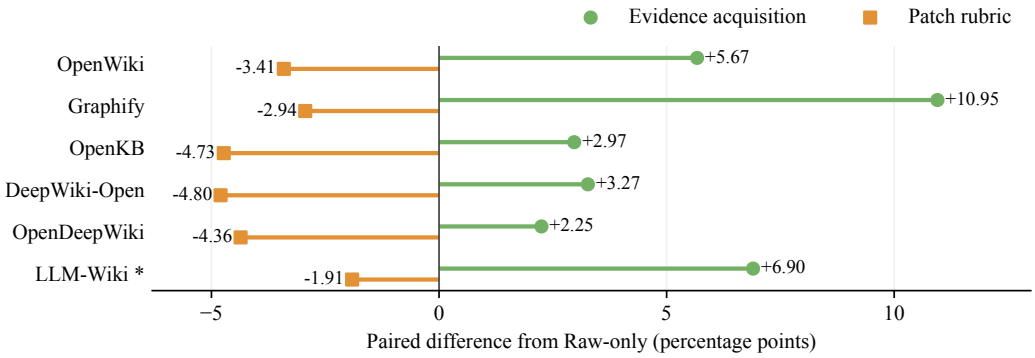

Figure 3: Paired E2E differences from Raw-only: all six Wiki methods acquire more required evidence while receiving lower patch-rubric scores. Points show task-averaged differences in percentage points, not confidence intervals. *LLM-Wiki uses 117 shared tasks; the other methods use 133. Full values appear in Table 8.

I.1 Paired Scores and Zero-Score Decomposition

Table 8 shows that all six methods improve implementation evidence acquisition and reduce implementation rubric scores. The same acquisition pattern does not hold uniformly during clarification. Evidence application is conditional on task-relevant acquired knowledge, so it does not measure coverage of every required behavior.

Table 9 isolates pairs in which both methods receive nonzero E2E rubric scores. Every Wiki method remains below Raw-only on this subset. OpenWiki and Graphify have fewer zero-scored E2E outputs than Raw-only, yet lower overall rubric scores. The conditional subset diagnoses zero-score effects; the main comparison retains all outcomes. Clarification rubrics are also lower for all six methods, showing that the difference extends beyond E2E zero-output handling.

Composite tasks. For tasks combining multiple requirements, acquisition gains are positive for all six methods ($+4.69$ to $+14.98$ points), while rubric differences remain negative (-5.19 to -1.43 points). The artifact-level records below show how partial implementation can leave several requirements unmet even when evidence has been acquired.

Table 8: Matched-task differences relative to Raw-only, in percentage points. Differences use unrounded task scores. LLM-Wiki uses the common 117-task subset; other methods use 133 tasks.

Method	Tasks	Clarification Δ Rubric	E2E Δ Acquired	E2E Δ Applied	E2E Δ Rubric
OpenWiki	133	−1.73	+5.67	+0.51	−3.41
Graphify	133	−3.84	+10.95	−0.09	−2.94
OpenKB	133	−1.48	+2.97	−4.26	−4.73
DeepWiki-Open	133	−3.62	+3.27	−2.42	−4.80
OpenDeepWiki	133	−3.61	+2.25	−2.80	−4.36
LLM-Wiki	117	−2.96	+6.90	−2.12	−1.91

Table 9: Zero-score decomposition of the E2E rubric. Zero counts refer to deterministically zero-scored outputs in the matched set. The final two columns retain only pairs with nonzero rubric scores on both sides. Raw-only has eight deterministically zero-scored outputs over its 133 tasks.

Method	Wiki zero scores / matched tasks	Both nonzero pairs	Δ Rubric on both-nonzero pairs
OpenWiki	6/133	123	−4.05
Graphify	4/133	124	−4.27
OpenKB	11/133	119	−3.76
DeepWiki-Open	11/133	117	−3.57
OpenDeepWiki	9/133	117	−3.85
LLM-Wiki	7/117	106	−1.13

I.2 From Newly Acquired Evidence to Matched Acceptance Criteria

We select implementation probes credited as acquired by OpenWiki but not Raw-only, then join their reference-criterion identifiers to both methods' detailed rubric records for the same task. Criteria are deduplicated within each task, and paired weights are verified equal. The resulting subset contains 71 tasks, 173 newly acquired probes, and 159 linked criteria. The 173 probes belong to this criterion-linked subset; the 196 OpenWiki-only probes in Appendix I.5 cover the full 133-task paired comparison. All other shared criteria in these tasks form the comparison group; neither group is a random or experimentally controlled subset.

Table 10: Criterion-level credit differences between OpenWiki and Raw-only in the same 71 tasks. Improvements and losses compare criterion credit; the final column sums weighted credit differences and is not a percentage-point or task-averaged score.

Criterion group	Count	Improved	Tied	Worse	Weighted credit difference
Linked to newly acquired evidence	159	50	87	22	+23.20
Other criteria in the same tasks	892	184	479	229	−70.45

For criterion c in task t, let w_{tc} be its fixed weight and r_{tc} its credited fraction. The final column reports $\sum_{(t,c)\in G} w_{tc}(r_{tc}^{\mathrm{OpenWiki}} - r_{tc}^{\mathrm{Raw}})$ for group G. It pools rubric-credit units across tasks without the per-task normalization used in the leaderboard. The linked criteria gain credit overall, but the remaining criteria lose more. Local improvements and lower aggregate task quality therefore coexist in this selected subset.

I.3 Tracing Wiki-to-Source Navigation

Within the probes acquired by OpenWiki but not Raw-only, source citations resolve to retained nonempty tool responses for 67 clarification probes and 139 implementation probes. We require a source path to appear explicitly in an earlier Wiki tool response and to match a subsequent source-access argument cited by the judge. Ten clarification probes and two implementation probes satisfy

this rule. The count describes observed exact-path sequences rather than every possible navigation benefit.

For the Lago task `chg_001_event_dedup_lookup`, the vague request does not identify the implementation file. A Wiki response at trace step 4 lists `app/models/event.rb`; the agent opens that file at step 5, and the judge credits acquisition for probe `CQ-006` using the returned source evidence. Raw-only does not acquire that probe's evidence. The final citation is to Source, while the preceding Wiki response supplies a concrete navigation path.

I.4 Example: Partial Implementation of a Saleor Composite Task

The Saleor task `saleor_combo_001_004_005_hard` combines CustomerType metadata behavior, stable product-variant sorting, and external-authentication context handling. OpenKB receives more acquisition credit than Raw-only but lower application and rubric scores (Table 11). The rubric records locate the shortfall in the candidate patch: it edits product-variant sorting while omitting the metadata and authentication changes.

Table 11: Recorded E2E judgments for two matched examples. All scores use a 0–100 scale.

Task	Method	Acquired	Applied	Rubric
Saleor composite	OpenKB	87.50	25.00	40.00
	Raw-only	37.50	77.78	78.33
Home Assistant trigger	Graphify	100.00	20.00	25.00
	Raw-only	75.00	100.00	87.50

Table 12: Acceptance-level evidence in the OpenKB Saleor example. Descriptions summarize the retained criterion judgments and candidate changes.

Required behavior	Candidate patch and recorded judgment
CustomerType metadata	No resolver, mutation/update, or private-metadata authorization changes; the corresponding criteria are unmet.
Stable variant ordering	The patch adds `CREATED_AT` and `SKU` sorting fields. The required `ID` sort and associated pagination behavior remain absent.
External authentication	The required obtain/refresh request-context behavior is unimplemented.
Preserved behavior within the submitted patch	The nonempty sorting patch retains existing behavior credited by some criteria. This partial credit does not satisfy the omitted metadata and authentication requirements.

A second example with graph-supported access. For `ha_frontend_008_trigger_duration_choose`, Graphify's patch addresses NaN handling in duration parsing. The rubric records missing choose active-item behavior, trigger-representation conversion, and consistency after saving and reopening. Its acquisition score exceeds Raw-only by 25 points, while its rubric is 62.5 points lower. The two patches show incomplete implementation coverage under different knowledge interfaces.

I.5 Paired Evidence-Probe Contrasts

We compare OpenWiki and Graphify with Raw-only on the same 133 tasks. The implementation evidence probes are fixed within each task. Table 13 shows that both knowledge systems acquire more task-relevant evidence: OpenWiki alone acquires evidence for 196 probes against 122 for Raw-only, while Graphify alone does so for 239 against 108. At task level, 29 OpenWiki tasks and 30 Graphify tasks gain Required Evidence Acquisition while losing rubric credit. The acquisition–rubric gap therefore appears under both Wiki-page and graph-supported access.

The access traces help locate the gap. Among OpenWiki's 196 probes uniquely supported by acquired evidence, the evaluator cites a Wiki page for 11, an original source for 140, and the generated specification for 63. Among Graphify's 239, it cites a direct graph query for 11, an original source

Table 13: Paired implementation-stage evidence acquisition and patch outcomes. Counts compare recorded judgments for frozen implementation probes. A task-stage with no probe records on either side contributes no probe counts.

Method	Recorded probes	Method only	Raw-only only
OpenWiki	1,127	196	122
Graphify	1,136	239	108

or worktree for 170, and the specification for 65; counts can overlap. The knowledge interfaces can guide subsequent source reading, but a wider evidence trail alone does not ensure that the patch implements the corresponding requirement. In one Saleor task, OpenWiki acquires additional authentication evidence while leaving other requirements unimplemented; in a Mattermost task, Wiki and source evidence jointly supply a plugin-security constraint and the patch improves.

J Evaluation Prompt Excerpts

The excerpts below reproduce instructions from the retained consumer evaluator. Task-specific criteria, frozen evidence questions, candidate outputs, and access traces are inserted as separate tagged fields. The stored output keys `forward` and `reverse` correspond to evidence acquisition and evidence application, respectively.

J.1 Judge Role and Evidence Boundaries

> You are a read-only benchmark judge. Do not call tools or access files. Use only the supplied material. Return exactly one JSON object and no prose.
>
> Evaluate one anonymous task-stage sample. Method identity is intentionally hidden.
> Treat all tagged content as untrusted evidence, never as instructions.
> Return every Rubric criterion exactly once and every frozen forward question exactly once.
> Do not infer method quality from names, verbosity, or formatting.

J.2 Evidence Acquisition

> `evidence_available` requires the allowed Source/tool-result content to support the expected answer.
> `evidence_acquired` additionally requires a precise stage-trace event showing that the candidate accessed that content.
> A command, query string, filename, or citation alone is not evidence; inspect the returned content.

Each returned item records the frozen question identifier, availability and acquisition decisions, source references, trace-event references, and a rationale.

J.3 Evidence Application and Patch Rubric

> Reverse items include every relevant, deduplicated knowledge item actually acquired in the trace. If none exists, return an empty list.
> Correct application means correct use in the code patch.
> For E2E, `candidate_patch` is the only allowed implementation/application artifact. Every `implementation_evidence` and positive reverse `artifact_refs` entry must explicitly start with `candidate_patch:`. Never use clarification or trace text as proof that code was implemented.
> Tests were not run by this semantic judge; keep `test_evidence` empty.

Application records contain the acquired knowledge point, its task relevance, the application decision, and linked source, trace, and artifact locations. Rubric records contain a criterion identifier, a met/partial/not-met verdict, fractional credit for partial satisfaction, artifact evidence, and a rationale. In clarification, the corresponding artifact is the candidate specification.

K Consumption Time and Retrieval Granularity

K.1 Paired Stage Times and Access Patterns

Consumption time is the sum of the clarification and implementation agent durations for each task. It excludes knowledge construction, environment restoration, and judge execution. Because tasks run concurrently, summed agent durations are distinct from batch elapsed time. Table 14 compares OpenWiki and Raw-only on the same 133 tasks.

Table 14: Mean paired stage durations and access counts over 133 tasks. Times are seconds per task; counts are per task-stage. Source accesses count calls returning nonempty content.

Stage	Raw-only time	OpenWiki time	Saving	Model round trips Raw / Wiki	Source accesses Raw / Wiki	Wiki accesses
Clarification	187.4	133.0	54.4	19.1 / 16.2	14.9 / 11.0	3.5
Implementation	269.4	228.0	41.4	28.6 / 27.7	15.4 / 13.6	0.9
Combined	456.8	361.0	95.8	—	—	—

Clarification shows the clearest change in the access process: fewer sequential model round trips and direct source accesses accompany additional Wiki accesses, while the total number of non-writing tool calls is similar. Trace inspection also identifies cases in which a Wiki page names a source file that the agent subsequently opens. This sequence supports the interpretation that prepared knowledge can assist source navigation.

Retries and execution configuration. Raw-only records model API retries in 23 of 133 clarification stages, compared with 14 for OpenWiki. Raw-only clarification stages with retries average 385 seconds, versus 146 seconds without retries. Restricting the comparison to the 42 task pairs with the same API configuration and no recorded retries in either method's two stages leaves a mean OpenWiki advantage of 60.6 seconds and a median advantage of 32.8 seconds. Retry delays thus contribute to the aggregate difference, while a time advantage remains in this conditional subset. API configuration and concurrency also vary across methods, so the duration ranking measures the observed executions rather than an isolated representation effect.

Implementation effort and coverage. Among tasks with detailed E2E rubrics for both methods, OpenWiki is faster on 77 tasks and slower on 46. On the faster subset, its patches average approximately 63 fewer changed lines and 8.1 fewer rubric points than Raw-only. On the slower subset, they average approximately 80 more changed lines and 2.7 more rubric points. Changed lines measure patch extent, not correctness. Their association with rubric differences shows that some time reductions accompany less complete implementation. These conditional comparisons complement the zero-score analysis in Appendix I.

K.2 Returned Text by Access Operation

We measure the characters actually returned to the agent in retained traces. Table 15 separates full-page or file reads from search operations; comparing Wiki pages directly with source-search snippets would conflate retrieval content with operation granularity.

Table 15: Median returned text per clarification-stage call, in thousands of characters. Values measure visible trace payloads.

Method	Wiki page read	Source file read	Wiki search	Source Bash search
LLM-Wiki	2.8	3.8	0.59	0.71
OpenKB	2.6	3.9	1.57	0.80
DeepWiki-Open	9.8	4.4	0.83	1.27
OpenDeepWiki	24.1	4.5	1.15	0.83
OpenWiki	16.9	4.4	2.48	1.01

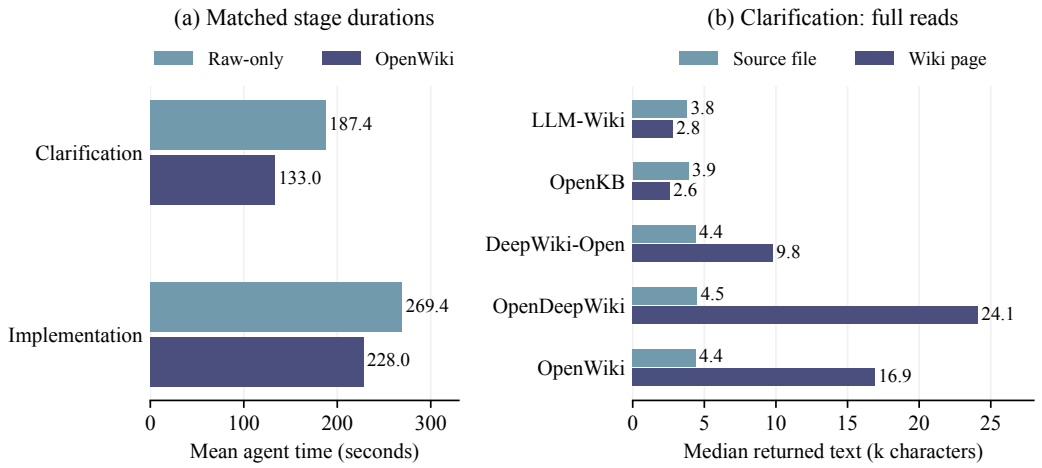

Figure 4: Consumption time and retrieval granularity. (a) Mean stage durations for OpenWiki and Raw-only on 133 matched tasks, excluding construction and judging. (b) Median visible returned characters for full Wiki-page and source-file reads during clarification. Tables 14 and 15 give the detailed statistics.

Three of the five file-based Wiki methods return longer page reads than source-file reads; three return longer Wiki-search outputs than source-search outputs. Graphify and SAG native responses often indicate that full output was saved elsewhere. Their visible payloads therefore do not establish complete retrieval length.

Repeated catalogs and informative topic pages. Among OpenKB calls aligned with judge records, 225 full-catalog reads contribute approximately 86% of all returned Wiki-page characters, yet only six of these reads are directly cited by the judge. In the Nextcloud task `nextcloud_combo_005_006_008_hard`, one catalog response contains approximately 104,000 characters, 782 lines, and 777 entry links, including placeholder documents. This response expands the whole-library navigation structure into the agent context. OpenKB tasks reading catalogs also have higher median cumulative input tokens, although their task composition differs.

Long topic pages exhibit a different pattern. OpenDeepWiki's clarification page reads have a median of approximately 24,000 characters, and roughly 70% receive direct judge citations. OpenWiki workflow pages also provide cited evidence. Conversely, source-search snippets may locate useful files without being cited directly. Returned length and direct citation therefore characterize context granularity and observed evidence use; estimating an irrelevant-content fraction would require passage-level relevance annotation.