

RAFT: A Stateful Retrieval-Augmented Framework for Troubleshooting Agents

Mingxuan Zhang, Xiaowen Wang, Anupma Sharan, Zhengyi Chen,
Chenyu Diana Zhang, Shanshan Yang, Chittibabu Pacharu

Microsoft, Redmond, WA, USA

Correspondence: mingxuzhang@microsoft.com

Abstract

Effective troubleshooting agents in enterprise customer support depend on retrieving actionable guidance from similar historical resolved cases, yet existing retrieval-augmented generation (RAG) systems treat support cases as static documents and overlook their multi-stage, stateful nature. We introduce RAFT (Retrieval-Augmented Framework for Troubleshooting Agents), a stateful RAG framework that abstracts each resolved case into a directed chain of timeline entries and retrieves at the entry level, surfacing cases whose intermediate states match the active case and returning the parent-case trajectory anchored at the matched state; an optional case-level graph links cases by shared root cause and resolution. We evaluate this retrieval layer directly, which, unlike evaluating a full agent system, requires no production deployment. Because public multi-stage troubleshooting data is extremely rare, we pair a synthetic benchmark built from Microsoft Learn Windows Server documentation with real Apache Jira issues carrying human-created duplicate labels. RAFT improves Case Hit over vanilla RAG and GraphRAG baselines at every stage of case progress, with statistically significant gains over the strongest baseline; the Jira results provide directional evidence that the advantage transfers to real case histories. We release our benchmark, implementation, and the Apache Jira evaluation set.¹

1 Introduction

Large language models (LLMs) now power intelligent agents across domains from software engineering (Jimenez et al., 2023) to customer support. In enterprise customer support, an effective agent must reason over a private corpus of historically resolved cases, whether it resolves incoming tickets or assists support engineers in doing so. Fine-tuning (Hu et al., 2022; Ouyang et al., 2022) can

inject such domain knowledge, but it is computationally expensive, restricted to open-weight models, prone to catastrophic forgetting (Luo et al., 2023), and requires periodic retraining as new and more recent cases emerge. Retrieval-augmented generation (RAG) (Lewis et al., 2020; Gao et al., 2023; Singh et al., 2025) is a more practical alternative, grounding the LLM in a private knowledge base at inference time without altering its parameters. Yet as knowledge bases grow in scale and complexity, traditional RAG struggles: retrieved context is often extensive, poorly organized, and noisy, degrading both retrieval accuracy and the agent’s ability to reason over it (Han et al., 2025b; Edge et al., 2024; Xiang et al., 2025; Chen et al., 2024).

GraphRAG (Edge et al., 2024; Zhang et al., 2025b; Zhuang et al., 2025; Chen et al., 2025; Yang et al., 2026) responds by imposing explicit relational structure over the knowledge base. However, existing pipelines, GraphRAG variants included, are general-purpose plug-ins for arbitrary document collections and overlook three properties that distinguish IT and technical support data from typical text corpora: (1) cases bundle heterogeneous artifacts (logs, traces, dialogue) of uneven actionable value; (2) resolution unfolds across multiple stages where the problem framing itself can shift; and (3) the actionable signal is sparse and scattered across many turns (Section 3). For a troubleshooting agent, effective guidance heavily depends on efficiently surfacing similar cases and learning from their resolution steps and investigation strategies. Existing GraphRAG methods make offline, multi-hop reasoning over LLM-extracted entity graphs the primary retrieval backbone, adding construction cost without addressing this core need (Zhuang et al., 2025; Xiang et al., 2025; Chen et al., 2025).

The quality of this retrieval layer is thus key to effective troubleshooting regardless of the agent’s harness, underlying models, or workflow, and the

¹<https://github.com/MingxuanZhangPurdue/RAFT>; artifacts will be published following internal review.

layer can be evaluated independently of the agent built on top of it. A full agent system, in contrast, is nearly impossible to evaluate without a real-world deployment, which precludes academic evaluation, cross-system comparison, and extension by others. We therefore evaluate the retrieval layer directly: given the current information of an active case, it must surface similar resolved cases so that the agent gains concrete ideas for diagnosis and resolution.

We propose RAFT (Retrieval-Augmented Framework for Troubleshooting Agents), a stateful RAG framework built around the multi-stage nature of troubleshooting. RAFT abstracts each resolved case into a directed chain of timeline entries, each recording the technical state at one meaningful stage of the investigation. Embedding and retrieving at the entry level rather than the case level surfaces cases whose intermediate states most closely match the active case, giving the agent both tactical guidance for the current stage and strategic context on where comparable cases led. A complementary, optional case-level graph links cases by shared root cause and resolution, enabling expansion to relevant cases that do not match at the entry level.

Public data meeting our requirements (resolved multi-stage troubleshooting cases with their resolution histories) is extremely rare (Section 2), and the best real source we identified is Apache Jira. We therefore evaluate on two complementary datasets: a synthetic benchmark constructed from Microsoft Learn Windows Server troubleshooting documentation (MicrosoftDocs, 2024) for controlled demonstration and development, and real Apache Jira (Apache Software Foundation, 2026) issues with human-created duplicate labels to check that the advantage transfers to real data. RAFT improves Case Hit over vanilla RAG and GraphRAG baselines at every stage of case progress (Sections 5.4 and 5.5).

Contributions.

- A **stateful RAG architecture** that represents resolved cases as directed chains of timeline entries, retrieves over their evolving intermediate troubleshooting states, and returns the parent-case trajectory anchored at the matched state.
- A **case-level graph** linking cases by shared root cause and resolution, enabling principled expansion beyond initial entry-level matches.
- A **public synthetic development benchmark** of 826 support cases from Microsoft Learn

Windows Server documentation, with a reproducible protocol that probes retrieval at multiple stages of an active case.

2 Related Work

For broad surveys of RAG, Agentic RAG, and GraphRAG, we refer readers to Fan et al. (2024); Zhang et al. (2025b); Singh et al. (2025). These pipelines target generic document corpora, and applying them to historical customer support cases remains underexplored, largely due to the lack of suitable public datasets. To our knowledge, no public customer support dataset (Abdellatif, 2025; Qu et al., 2018; Yang et al., 2018) satisfies all of the following criteria: (1) rich, multi-turn interactions rather than single-turn question answering; (2) key entities (error codes, products, services) preserved without redaction; and (3) labels grouping similar cases together. Beyond the data gap, most work in this area originates in industry settings where evaluation relies on proprietary data and production metrics, severely limiting reproducibility. Existing academic work also focuses largely on question answering over static domain documents (Su et al., 2025; Patel, 2025; Zhao et al., 2025) rather than leveraging historical case interactions for multi-stage troubleshooting. The closest work to ours, Xu et al. (2024), constructs a knowledge graph from historical cases for question answering and shows that graph structure can benefit support retrieval; however, their system is closed-source, their dataset private, and their evaluation confined to a production environment, making direct comparison infeasible. Evaluating the retrieval layer directly removes this dependence on production deployments and lets others reproduce and extend the comparison. We therefore pair a public synthetic benchmark for controlled demonstration and development with a transfer evaluation on Apache Jira (Apache Software Foundation, 2026), alongside an open implementation and a reproducible protocol.

3 Problem Statement

In enterprise customer support, particularly for IT services, resolving a ticket is rarely a one-shot process. Resolution unfolds across multiple stages: an initial symptom report, hypothesis formation and iterative information gathering across logs, system outputs, and diagnostic tools, and finally identifying the root cause and issuing targeted remediation.

Across this trajectory, surfacing similar previously resolved cases and learning how they were diagnosed and fixed is critical for an agent to resolve the active case effectively.

We therefore seek a retrieval system, built on a corpus of resolved cases, that identifies similar cases given a query reflecting the current state of an active case and surfaces guidance on both the immediate next step and the broader resolution trajectory. Critically, this retrieval must be **state-aware**: rather than treating the active case as a static lookup, the agent issues a sequence of updated queries as the case evolves, and the system must return the most pertinent cases at each stage. Early on, a ticket describes only surface symptoms (an error code or a brief account of the behavior), and the agent should retrieve cases with similar initial symptoms to identify productive directions; as more context is gathered, updated queries let the system surface cases whose intermediate states match the current investigative state, giving finer-grained insight into the next steps.

We deliberately scope this retrieval problem to surfacing similar cases and the evidence needed to diagnose and act on them. Higher-order reasoning, over how cases relate or how root causes cluster, is valuable but is best performed by the agent online against the specific active case rather than frozen into the offline index. We therefore keep retrieval focused on this task and defer relational inference to the agent, a stance our metrics (Section 5) reflect.

This problem exposes four fundamental limitations of existing RAG approaches:

1. **Noisy, unstructured data.** Raw case data interleaves substantial non-technical content with the diagnostically relevant details, and the signal is scattered across many turns. Because existing RAG baselines operate directly on this raw data, the noise degrades their ability to identify genuinely similar cases in the first place.
2. **Incoherent retrieved chunks.** Even when a similar case is retrieved, an agent cannot reason effectively from isolated chunks. To judge whether a historical case is truly similar and to learn from how it was resolved, the agent must see the case as a coherent whole, which requires reassembling the raw case data by mapping retrieved chunks back to their parent cases. Given the size of real cases and the practical output-token limits of RAG tools, this both

wastes tokens and sharply limits the number of similar cases an agent can examine per retrieval. GraphRAG methods that return chunks together with their entity relationships partially mitigate this, but the construction process is unstable and offers no guarantee that the chunks needed to form a coherent picture are retained (Zhuang et al., 2025; Han et al., 2025a; Zhou et al., 2025; Xiang et al., 2025).

3. **Non-actionable cases.** In practice, a non-trivial fraction of closed tickets contain no actionable resolution intelligence at all. Many are closed prematurely for administrative reasons such as customer unresponsiveness or duplication, and without an explicit mechanism to filter them out they inject noise that materially degrades retrieval performance.
4. **Privacy constraints.** Enterprise customer data often contains personally identifiable information that should not be exposed directly to retrieval or to the agent, requiring an additional processing layer to abstract or redact such content before indexing.

RAFT addresses these limitations through the two-level architecture described next.

4 RAFT

RAFT organizes resolved cases at two levels: a per-case extracted resolution trajectory and a case-level graph $\mathcal{G}$ that links cases by root cause and resolution.

4.1 Indexing

Let $\mathcal{H} = \{h_i\}_{i=1}^N$ be a corpus of N resolved cases. Each raw case h_i has a unique identifier u_i (e.g., ticket number), metadata m_i (e.g., category, created and closed times), and a time-ordered turn sequence $(x_1^{(i)}, \dots, x_{T_i}^{(i)})$ of emails, notes, logs, and similar artifacts. We process each case independently using an LLM to obtain a structured representation $\tilde{h}_i$:

$$h_i \xrightarrow{\text{LLM}} \tilde{h}_i \stackrel{\text{def}}{=} (v_i, \{\phi_k^{(i)}\}_{k=1}^{K_i}, r_i, a_i, e_i), \quad (1)$$

where $v_i \in \{0, 1\}$ is an actionability flag, $\{\phi_k^{(i)}\}_{k=1}^{K_i}$ is the chronologically ordered sequence of timeline entries, r_i is the confirmed root cause, a_i the canonical resolution steps, and e_i a list of troubleshooting-relevant entities (error codes, environments, product versions, and the like). The full extraction schema is provided in Appendix B.

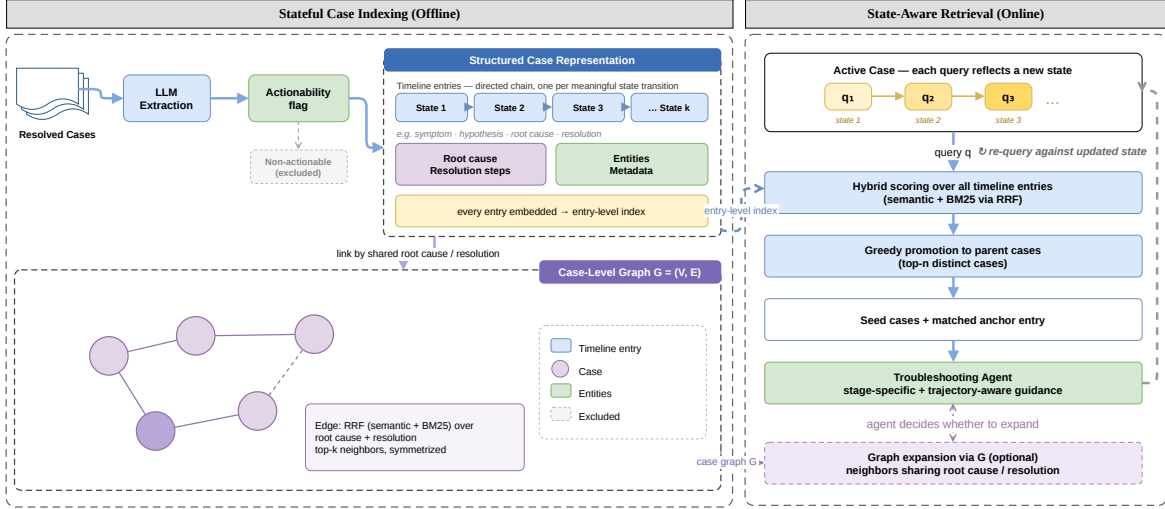


Figure 1: Overview of **RAFT**. RAFT contains no autonomous loop; the circular arrow denotes the external agent re-querying RAFT as the active case advances.

Actionability flag. As discussed in Section 3, a potentially substantial fraction of closed tickets carry no actionable resolution intelligence. We set $v_i = 0$ for such cases (the LLM additionally produces a brief justification); the remaining fields are left empty, and the case is excluded from the index. Throughout the rest of this paper, we consider only actionable cases ($v_i = 1$).

Timeline entries. Intuitively, each timeline entry $\phi_k^{(i)}$ is a distilled representation of a batch of consecutive turns $(x_j^{(i)}, x_{j+1}^{(i)}, \dots, x_{j+z}^{(i)})$ that together advance the case to a new state. Segmentation follows a single principle: a new entry begins at each meaningful state transition, where the framing of the problem or the current understanding is materially updated. These transitions correspond to the natural phases of an investigation, for example the opening symptom report, a hypothesis being added, discarded, or confirmed, the root cause being confirmed, and a resolution being proposed and verified. Acknowledgments and minor updates that add no new insight are absorbed into the current entry. This keeps the timeline compact ($K_i \ll T_i$) while ensuring every entry carries actionable information. Each entry records the technical state of its segment: the actions taken, the hypotheses under investigation, and the current understanding of the issue.

Labels and entities. The root cause r_i and the resolution steps a_i summarize how the case ultimately resolved and serve as the basis for case-level linking. The entities e_i together with m_i support

optional metadata filtering at retrieval time, allowing agents to narrow the search space by error code, category, product version, or timestamp.

Case-level graph. Once all cases have been processed, we construct an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with vertex set $\mathcal{V} = \{\tilde{h}_i\}$. For each case h_i , we score its similarity to every other case in $\mathcal{V}$ using a hybrid score that combines semantic similarity and BM25 lexical similarity (Lù, 2024) over the concatenation of root-cause and resolution texts via Reciprocal Rank Fusion (RRF) (Cormack et al., 2009). We connect each case to its top- k highest-scoring neighbors and symmetrize the result to obtain $\mathcal{E}$. Metadata and entity-based filters can further constrain the neighbor set during graph construction. This graph supports principled expansion beyond initial entry-level matches at retrieval time.

4.2 Retrieval

We embed every timeline entry $\phi_k^{(i)}$ across all indexed cases. Given a query q , we apply the same hybrid score (semantic plus lexical via RRF) over all timeline entries, then greedily promote ranked entries to their parent cases until n distinct cases are selected. The full procedure is given in Algorithm 1.

For each selected case $c \in C$, k_c^* is the index of the highest-scoring timeline entry, so the agent receives both the full case representation $\tilde{h}_c$ and the anchor entry that triggered the match. This lets the agent see which investigation state was matched, not just which case.

Algorithm 1 RAFT Entry-Level Retrieval

Require: Query q , indexed timeline entries $\{\phi_k^{(i)}\}$, number of cases n

Ensure: Retrieval result R

- 1: Compute hybrid score $s(q, \phi_k^{(i)})$ for all entries
- 2: Sort entries in descending order of s
- 3: Initialize $C \leftarrow \emptyset$, $k^* \leftarrow \{\}$
- 4: **for** each entry $\phi_k^{(i)}$ in sorted order **do**
- 5: **if** parent case $i \notin C$ **then**
- 6: $C \leftarrow C \cup \{i\}$; $k_i^* \leftarrow k$
- 7: **end if**
- 8: **if** $|C| = n$ **then**
- 9: **break**
- 10: **end if**
- 11: **end for**
- 12: **return** $R = \{(\tilde{h}_c, k_c^*) : c \in C\}$

Once the agent has identified seed cases via entry-level retrieval, the case-level graph $\mathcal{G}$ enables expansion to neighboring cases that share a similar root cause and resolution pattern but may not match at the entry level. The rationale is that cases sharing a root cause need not exhibit similar intermediate states or symptoms, so increasing the initial n tends to introduce noise rather than uncover such neighbors (Zhang et al., 2025a). Keeping initial entry-level retrieval selective and expanding through $\mathcal{G}$ recovers such neighbors without inflating n .

5 Experiments

We evaluate RAFT’s retrieval performance against vanilla RAG and GraphRAG baselines on a synthetic development benchmark, and then test whether the results transfer to real cases with human-created labels (Section 5.5).

5.1 Synthetic Development Benchmark

As discussed in Section 2, no existing public dataset meets the requirements for evaluating retrieval in multi-stage troubleshooting. We therefore construct a synthetic corpus grounded in Microsoft Learn Windows Server troubleshooting documentation (MicrosoftDocs, 2024). The synthesis pipeline first organizes source articles into a structured wiki, then generates 2–4 support cases per documented root cause, injecting context from related articles to produce realistic diagnostic ambiguity. For evaluation, we hold out one case per root-cause group as the test query, with the remaining cases forming the indexed corpus. The synthetic data generation pipeline is detailed in Appendix A; Table 1 reports per-category statistics.

Category	Cases	Messages	Tokens
Active Directory	177	11.0 ± 1.8	3297 ± 795
Windows Security	121	9.7 ± 1.8	2556 ± 628
Remote Desktop	72	10.7 ± 1.9	2794 ± 657
Group Policy	92	9.8 ± 2.2	2948 ± 642
Licensing and Activation	62	11.1 ± 1.9	2550 ± 496
Networking	245	10.4 ± 1.7	2346 ± 579
Backup and Storage	57	10.4 ± 2.2	3292 ± 897
Overall	826	10.4 ± 1.9	2767 ± 770

Table 1: Synthetic dataset statistics per category. Number of Messages and Tokens report per-case mean $\pm$ std.

5.2 Baselines

We compare RAFT against vanilla RAG and two recent GraphRAG methods: HippoRAG2 (Gutiérrez et al., 2025) and Fast-GraphRAG (Circlemind AI, 2024). HippoRAG2 builds an open-relation knowledge graph over the corpus and retrieves via personalized PageRank seeded by query-linked entities; Fast-GraphRAG extracts an entity graph and returns a budgeted mix of entities, relations, and chunks. We run both in their default configurations. To ensure a fair comparison, all methods share the same embedding model (text-embedding-3-large) and the same LLM for indexing (gpt-5.2); evaluation uses gpt-5.4. No metadata filtering is applied, isolating the effect of each method’s retrieval mechanism. Moreover, every case in our corpus is actionable, so RAFT’s actionability filtering excludes no cases and provides no advantage in this comparison. In production, retrieval tools exposed to an agent typically enforce a per-call token cap on returned context. Real support cases are token-intensive, often running into tens of thousands of tokens, whereas our synthetic cases are considerably shorter, averaging 2767 tokens (Table 1). To reflect this constraint, we cap retrieved context at 6000 tokens for all methods. For methods that return case-level units (RAFT, Vanilla RAG, HippoRAG2), we additionally cap retrieval at 5 distinct cases; Fast-GraphRAG returns entity-linked chunks rather than case-level units, so only the token cap applies. Detailed configurations are provided in Appendix C.2.

5.3 Evaluation Metrics

A key property of troubleshooting is that useful guidance depends on how far the investigation has progressed. To capture this, we construct queries from prefixes of each test case at three progress points: 0%, 30%, and 60% of turns. The 0% query

contains only the initial symptom report; later cut-offs reveal progressively more diagnostic context. We report three metrics:

- **Case Hit:** whether at least one retrieved passage belongs to a ground-truth similar case, i.e., one sharing the same root cause and resolution.
- **Root Cause Coverage:** the fraction of atomic claims in the gold root-cause explanation that are entailed by the retrieved context, as judged by an LLM.
- **Resolution Steps Coverage:** the analogous fraction for the gold remediation procedure.

These metrics form a hierarchy: without a Case Hit, the retrieved context cannot support diagnosis; given a hit, coverage measures whether the context contains coherent evidence for both understanding and acting on the issue. Full metric definitions appear in Appendix C.1.

5.4 Results on the Synthetic Benchmark

Table 2 reports retrieval performance across all methods and progress levels. RAFT achieves the best scores on every metric, with the largest and most reliable gains on Case Hit. At 0% progress (initial symptom only), RAFT achieves 84.2% Case Hit compared to 67.3% for vanilla RAG and 65.0% for HippoRAG2. This advantage persists as more context becomes available, with RAFT reaching 88.8% Case Hit at 60% progress. To statistically evaluate this claim, we compare RAFT against vanilla RAG, the strongest baseline, using bootstrap resampling clustered by root-cause group; the Case Hit gains are statistically significant at all three progress points, with full confidence intervals in Appendix C.3. Coverage gains are smaller and concentrated at earlier stages.

Fast-GraphRAG underperforms vanilla RAG at all progress levels. This aligns with recent findings that complex entity extraction and graph-based reasoning offer little benefit, and can even degrade retrieval, when the task does not demand hierarchical knowledge retrieval or deep contextual reasoning across documents (Xiang et al., 2025). In our setting, surfacing similar cases and supplying actionable insights matter more than abstract relational inference. A similar pattern holds for HippoRAG2, which also fails to surpass vanilla RAG in this setting.

To probe *why* entry-level retrieval helps, we examine where within a matched case the hit occurs.

For each test case with a correct retrieval, we record the timeline entry that triggered the match and report its absolute index and its depth as a percentile of the number of timeline entries in the extracted case, averaged over hits, in Table 3. The match moves steadily deeper as the query reflects a later stage, from 9.1% depth at 0% progress to 54.0% at 60%. This is the intended behavior: early queries carry only the symptom and match the opening entries of past cases, whereas later queries align with the corresponding intermediate states rather than re-matching symptoms.

To assess robustness to noisy queries, we perturb test queries with off-topic content, typos, and dropout, and find RAFT degrades less than vanilla RAG (Appendix C.4). We also measure the contribution of graph expansion under a matched retrieval budget (Appendix C.5).

Additional experiments. We provide two additional experiments in Appendix C.6: an ablation on indexing-model capacity, and an agentic case study in which the agent composes its own queries and filtering conditions, further improving retrieval quality.

5.5 Human-Grounded Evaluation on Apache Jira

To test whether RAFT transfers to real datasets, we build an evaluation set from public Apache Jira projects, where engineers link duplicate issues in their normal workflow (construction details in Appendix D). The final evaluation set contains 30 audited duplicate groups, each evaluated against 570 additional resolved (*Fixed*) issues as distractors, over contributor-written, unredacted issue histories.

We apply RAFT as is, with no modifications to the extraction prompt, schema, models, or retrieval procedure from the synthetic experiments, and compare against vanilla RAG, the strongest baseline in Section 5.4. RAFT improves Case Hit by +16.7, +17.3, and +10.5 percentage points at the 0%, 30%, and 60% progress points (Table 4), providing directional evidence that the advantage transfers to real case histories.

6 Conclusion

We presented RAFT, a stateful retrieval-augmented framework that addresses the limitations of generic RAG and GraphRAG pipelines on enterprise customer support data. RAFT abstracts each resolved

Method	Case Hit			Root Cause Cov.			Resolution Steps Cov.		
	0%	30%	60%	0%	30%	60%	0%	30%	60%
Vanilla RAG	0.673	0.719	0.769	0.597	0.625	0.672	0.528	0.548	0.590
HippoRAG2 (Gutiérrez et al., 2025)	0.650	0.688	0.711	0.574	0.609	0.637	0.507	0.541	0.559
Fast-GraphRAG (Circlemind AI, 2024)	0.421	0.442	0.583	0.294	0.299	0.448	0.208	0.213	0.341
RAFT (Ours)	0.842	0.871	0.888	0.649	0.675	0.689	0.563	0.587	0.605

Table 2: Retrieval evaluation results averaged over 5 independent runs, reported at three progress points (0%, 30%, 60%). For each run, we randomly shuffle all cases and select one test case per root-cause group; the remaining cases form the indexed corpus. Best result per column in **bold**.

Case Progress	Matched-Entry Depth (%)	Matched-Entry Index
0%	9.1	0.28
30%	20.0	0.59
60%	54.0	1.58

Table 3: Position of the retrieval match within a case, averaged over hits. *Depth* = position as a percentile of the number of timeline entries in the extracted case (0% = first entry); *Index* = absolute position.

Method	Case Hit		
	0%	30%	60%
Vanilla RAG	0.667	0.667	0.789
RAFT (Ours)	0.833	0.840	0.895

Table 4: Case Hit on the Apache Jira evaluation set (30 human-audited duplicate groups, 570 distractors), averaged over five runs.

case into a directed chain of timeline entries and connects cases at the graph level by shared root cause and resolution, enabling state-aware retrieval that returns coherent, stage-specific evidence. On a synthetic benchmark built from Microsoft Learn Windows Server documentation, RAFT substantially improves Case Hit over vanilla RAG and recent GraphRAG baselines at every progress level, and an evaluation on real Apache Jira issues provides directional evidence that the advantage transfers to real case histories.

Limitations

While our experiments demonstrate the effectiveness of RAFT for retrieval, three limitations should be noted. First, our main evaluation is conducted on a synthetic dataset of moderate scale, whereas production corpora typically contain far more cases with substantially higher token counts per case. Second, the Apache Jira evaluation comprises 30 audited duplicate groups and carries no confidence intervals, so we treat it as directional transfer evidence rather than a comprehensive real-world evaluation. Third, this work does not evaluate final diagnosis, resolution success, engineer productivity, or other end-to-end troubleshooting outcomes; we scope the work to the retrieval layer for the reasons given in Section 1.

References

- Mohammad Abdellatif. 2025. [Help desk tickets](#).
- Apache Software Foundation. 2026. Apache JIRA issue tracker. <https://issues.apache.org/jira>. Accessed August 2026.
- Jiawei Chen, Hongyu Lin, Xianpei Han, and Le Sun. 2024. Benchmarking large language models in retrieval-augmented generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*.
- Shengyuan Chen, Chuang Zhou, Zheng Yuan, Qinggang Zhang, Zeyang Cui, Hao Chen, Yilin Xiao, Jiannong Cao, and Xiao Huang. 2025. You don't need pre-built graphs for rag: Retrieval augmented generation with adaptive reasoning structures. *arXiv preprint arXiv:2508.06105*.
- Circlemind AI. 2024. Streamlined and promptable fast GraphRAG framework designed for interpretable, high-precision, agent-driven retrieval workflows. <https://github.com/circlemind-ai/fast-graphrag>.
- Gordon V Cormack, Charles LA Clarke, and Stefan Buettcher. 2009. Reciprocal rank fusion outperforms condorcet and individual rank learning methods. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 758–759.
- Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitanansky, Robert Osazuwa Ness, and Jonathan Larson. 2024. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*.
- Wenqi Fan, Yujuan Ding, Liangbo Ning, Shijie Wang, Hengyun Li, Dawei Yin, Tat-Seng Chua, and Qing Li. 2024. A survey on rag meeting llms: Towards retrieval-augmented large language models. In *Proceedings of the 30th ACM SIGKDD conference on knowledge discovery and data mining*, pages 6491–6501.
- Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*.
- Bernal Jiménez Gutiérrez, Yiheng Shu, Weijian Qi, Sizhe Zhou, and Yu Su. 2025. From rag to memory: Non-parametric continual learning for large language models. *arXiv preprint arXiv:2502.14802*.
- Haoyu Han, Li Ma, Yu Wang, Harry Shomer, Yongjia Lei, Zhisheng Qi, Kai Guo, Zhigang Hua, Bo Long, Hui Liu, Charu C Aggarwal, and Jiliang Tang. 2025a. Rag vs. graphrag: A systematic evaluation and key insights. *arXiv preprint arXiv:2502.11371*.
- Haoyu Han, Yu Wang, Harry Shomer, Kai Guo, Jiayuan Ding, Yongjia Lei, Mahantesh Halappanavar, Ryan A Rossi, Subhabrata Mukherjee, Xianfeng Tang, et al. 2025b. Retrieval-augmented generation with graphs (graphrag). *arXiv preprint arXiv:2501.00309*.
- Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*.
- Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2023. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*.
- Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474.
- Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yue Zhang. 2023. An empirical study of catastrophic forgetting in large language models during continual fine-tuning. *arXiv preprint arXiv:2308.08747*.
- Xing Han Lù. 2024. [Bm25s: Orders of magnitude faster lexical search via eager sparse scoring](#). *Preprint*, arXiv:2407.03618.
- MicrosoftDocs. 2024. SupportArticles-docs: A public version to sync with SupportArticles-docs-pr. <https://github.com/MicrosoftDocs/SupportArticles-docs>. Accessed: 2026-04-11.
- Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al.

2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744.
- Piyushkumar Patel. 2025. Graph-enhanced retrieval-augmented question answering for e-commerce customer support. *arXiv preprint arXiv:2509.14267*.
- Chen Qu, Liu Yang, W Bruce Croft, Johanne R Trippas, Yongfeng Zhang, and Minghui Qiu. 2018. Analyzing and characterizing user intent in information-seeking conversations. In *The 41st international acm sigir conference on research & development in information retrieval*, pages 989–992.
- Aditi Singh, Abul Ehtesham, Saket Kumar, Tala Talaei Khoei, and Athanasios V Vasilakos. 2025. Agentic retrieval-augmented generation: A survey on agentic rag. *arXiv preprint arXiv:2501.09136*.
- Hanchen Su, Wei Luo, Yashar Mehdad, Wei Han, Elaine Liu, Wayne Zhang, Mia Zhao, and Joy Zhang. 2025. Llm-friendly knowledge representation for customer support. In *Proceedings of the 31st International Conference on Computational Linguistics: Industry Track*, pages 496–504.
- Zhishang Xiang, Chuanjie Wu, Qinggang Zhang, Shengyuan Chen, Zijin Hong, Xiao Huang, and Jinsong Su. 2025. When to use graphs in rag: A comprehensive analysis for graph retrieval-augmented generation. *arXiv preprint arXiv:2506.05690*.
- Zhentao Xu, Mark Jerome Cruz, Matthew Guevara, Tie Wang, Manasi Deshpande, Xiaofeng Wang, and Zheng Li. 2024. Retrieval-augmented generation with knowledge graphs for customer service question answering. In *Proceedings of the 47th international ACM SIGIR conference on research and development in information retrieval*, pages 2905–2909.
- Chang Yang, Chuang Zhou, Yilin Xiao, Su Dong, Luyao Zhuang, Yujing Zhang, Zhu Wang, Zijin Hong, Zheng Yuan, Zhishang Xiang, et al. 2026. Graph-based agent memory: Taxonomy, techniques, and applications. *arXiv preprint arXiv:2602.05665*.
- Liu Yang, Minghui Qiu, Chen Qu, Jiafeng Guo, Yongfeng Zhang, W Bruce Croft, Jun Huang, and Haiqing Chen. 2018. Response ranking with deep matching networks and external knowledge in information-seeking conversation systems. In *The 41st international acm sigir conference on research & development in information retrieval*, pages 245–254.
- Guibin Zhang, Muxin Fu, Guancheng Wan, Miao Yu, Kun Wang, and Shuicheng Yan. 2025a. G-memory: Tracing hierarchical memory for multi-agent systems. *arXiv preprint arXiv:2506.07398*.
- Qinggang Zhang, Shengyuan Chen, Yuanchen Bei, Zheng Yuan, Huachi Zhou, Zijin Hong, Hao Chen, Yilin Xiao, Chuang Zhou, Junnan Dong, et al. 2025b. A survey of graph retrieval-augmented generation for customized large language models. *arXiv preprint arXiv:2501.13958*.
- Cen Zhao, Tiantian Zhang, Hanchen Su, Yufeng Zhang, Shaowei Su, Mingzhi Xu, Yu Liu, Wei Han, Jeremy Werner, Claire Na Cheng, et al. 2025. Agent-in-the-loop: A data flywheel for continuous improvement in llm-based customer support. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 1919–1930.
- Yingli Zhou, Yaodong Su, Youran Sun, Shu Wang, Tao-tao Wang, Runyuan He, Yongwei Zhang, Sicong Liang, Xilin Liu, Yuchi Ma, et al. 2025. In-depth analysis of graph-based rag in a unified framework. *arXiv preprint arXiv:2503.04338*.
- Luyao Zhuang, Shengyuan Chen, Yilin Xiao, Huachi Zhou, Yujing Zhang, Hao Chen, Qinggang Zhang, and Xiao Huang. 2025. Linearrag: Linear graph retrieval augmented generation on large-scale corpora. *arXiv preprint arXiv:2510.10114*.

A Microsoft Learn Synthetic Dataset

Microsoft Learn is Microsoft’s public documentation site for product guidance, troubleshooting articles, and learning resources. Each troubleshooting article typically documents a specific error or issue, including its symptoms, root cause, and resolution steps. Understanding these articles requires technical knowledge spanning multiple products, concepts, and technologies, which makes them a strong foundation for synthetic support case generation. We construct the dataset in two phases: we first build a **knowledge base** from the source articles, and then generate synthetic cases grounded in that knowledge base and troubleshooting articles.

Knowledge base construction. Rather than asking agents to read individual articles in isolation and generate cases from them, we first organize the source material into a wiki-style knowledge base. We focus on Windows Server troubleshooting and select seven categories: Active Directory, Group Policy, Licensing and Activation, Remote Desktop, Windows Security, Backup and Storage, and Networking. We use the Claude Code CLI with `claude-opus-4-7` to construct this knowledge base, running one session per category. Each session spawns sub-agents that work on individual subcategories; each sub-agent reads all source files within its scope and classifies them as troubleshooting, informational, or general knowledge. The orchestrating agent then consolidates these results and produces a category overview.

Case generation. From the indexed knowledge base, we generate synthetic cases on a per-article basis, again with one Claude Code session per category. To refine the generation process, we apply a simple self-reflection strategy executed by Claude Code itself. Starting from a high-level generation instruction file in Markdown, the agent produces 15 pilot cases by spawning generation sub-agents, then reviews the resulting cases, identifies their shortcomings, and revises the instruction file accordingly. This loop continues until no meaningful improvement is observed, at which point the full generation session is launched for the category.

Role of the wiki. The key benefit of generating from the wiki is that agents are not confined to the single article they are working from; they also gain awareness of related articles within the same topic area, including their symptoms, root causes, and points of overlap. This cross-article context

is what makes the generated cases diagnostically challenging: agents can introduce realistic red herrings drawn from sibling issues, construct multi-step diagnostic paths in which plausible alternative causes are explored and ruled out, and ensure that no two conversations follow the same troubleshooting sequence. Without the wiki, agents operate in isolation and produce cases that are technically correct but diagnostically straightforward. With the wiki, an initial symptom can plausibly point to several documented root causes, requiring the troubleshooting conversation to disambiguate among them.

Summary. Using this two-phase approach, first building a structured knowledge base and then generating cases informed by it, we processed over 800 source articles across seven Windows Server categories and produced 826 synthetic support cases. We generate 2–4 cases per root cause so that, during evaluation, each cause group can be split into an indexing set and a test set; all cases within the same group share the same root cause and resolution steps. Below is a short example of a synthetic case.

Example Case TKT-ADAUDIT-05 (Group: SEC — Permissions, Access Control, and Auditing)

[Customer → Support] *Subject: Auditing logonHours changes for service accounts*

We have an OU of service accounts (OU=ServiceAccounts,DC=tailspintoys,DC=local). A pen-test finding flagged that someone could remove the logonHours restriction on a service account and use it outside maintenance windows. We want to log any change to logonHours on accounts in that OU. Directory Service Access and Directory Service Changes auditing are already on. Just need the right SACL to capture logonHours changes specifically (not every attribute). Server 2019 DCs.

[Agent → Customer] *Subject: RE: SACL for logonHours auditing*

For narrow attribute-specific auditing you want a SACL targeting just the logonHours property write. On OU=ServiceAccounts:

- ADUC > View > Advanced Features.
- Right-click the OU > Properties > Security > Advanced > Auditing > Add.
- Principal: Everyone; Type: All; Applies to: Descendant User objects.
- Leave Write all properties unchecked; tick only Write logonHours.

You will get a 5136 event only when logonHours is modified, with the old/new byte array—keeping noise down.

[Customer → Agent] *Subject: RE: SACL for logonHours auditing*

Test passed first try. 5136 with

AttributeLDAPDisplayName = logonHours and the new byte value (OperationType 14674). Exactly what we wanted. Thanks.

Synthetic Annotation

Root cause: Customer needed narrow attribute-specific auditing of logonHours changes under OU=ServiceAccounts; broad SACLs would be too noisy, so the SACL had to be scoped to the logonHours property only.

Resolution: Targeted SACL via ADUC Auditing entry (Principal Everyone, Type All, Descendant User objects), ticking only Write logonHours. Validated via a logonHours toggle producing event 5136.

B RAFT

B.1 Schema Definitions

Listing 1: Simplified schema for the interaction graph extracted from each case.

```
class TimelineEntry(BaseModel):
    narrative: str = Field(
        min_length=200,
        max_length=4000,
    )

class Entity(BaseModel):
    name: str
    context: str

class InteractionGraph(BaseModel):
    extractable: bool
    non_extractable_reasoning: str | None = \
        Field(default=None)
    entities: list[Entity] | None = \
        Field(default=None, max_length=10)
    timeline: list[TimelineEntry] | None = \
        Field(default=None, min_length=1)
    root_cause: str | None = \
        Field(default=None)
    resolution_steps: str | None = \
        Field(default=None)
```

B.2 Indexing Process

Each case is indexed by a single, self-contained LLM (extractor) call that depends only on that case and not on the rest of the corpus. This independence is advantageous at scale: indexing parallelizes trivially and can be dispatched through the discounted batch APIs offered by common LLM providers (e.g., OpenAI, Anthropic), which trade latency for substantially lower per-token cost. Adding new cases therefore requires only running the same extraction step on the new cases and updating the adjacency structure of $\mathcal{G}$.

C Experiments

C.1 Metrics

We evaluate retrieval quality with three complementary metrics. Let $\mathcal{D}$ denote the evaluation set. Each

example $i \in \mathcal{D}$ consists of a query q_i , a ground-truth *shared id* s_i^* identifying the underlying issue, gold root-cause text r_i^* , gold resolution-steps text a_i^* , the retrieved context string C_i , and the set of retrieved case ids $\mathcal{R}_i$. We write $\text{sid}(c)$ for the shared id of a retrieved case c .

Case Hit. Measures whether retrieval surfaces *any* case belonging to the same underlying issue as the query.

$$\text{CH}_i = \mathbb{1}[s_i^* \in \{\text{sid}(c) : c \in \mathcal{R}_i\}], \quad (2)$$

$$\text{CH} = \frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} \text{CH}_i. \quad (3)$$

Root Cause Coverage. Quantifies how much of the gold root-cause explanation is actually *supported* by the retrieved context. An LLM judge (i) atomizes r_i^* into a set of claims $\mathcal{K}(r_i^*) = \{k_1, \dots, k_{n_i}\}$, and (ii) verdicts each claim k_j against C_i as $v_{ij} \in \{0, 1\}$, where $v_{ij} = 1$ iff k_j is entailed by C_i :

$$\text{RCC}_i = \frac{1}{n_i} \sum_{j=1}^{n_i} v_{ij}, \quad (4)$$

$$\text{RCC} = \frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} \text{RCC}_i. \quad (5)$$

This rewards retrieving evidence that actually diagnoses the issue, rather than merely co-occurring with the correct case.

Resolution Steps Coverage. Analogously measures how much of the gold remediation procedure is supported by the retrieved context. The gold steps a_i^* are atomized into $\mathcal{K}(a_i^*) = \{k_1, \dots, k_{m_i}\}$ and each claim is verdicted against C_i to obtain $u_{ij} \in \{0, 1\}$:

$$\text{RSC}_i = \frac{1}{m_i} \sum_{j=1}^{m_i} u_{ij}, \quad (6)$$

$$\text{RSC} = \frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} \text{RSC}_i. \quad (7)$$

This rewards retrieving the evidence needed to *act on* the issue.

C.2 Baseline Configurations

- **Vanilla RAG:** We use a chunk size of 256 tokens with a 64-token overlap for indexing. At retrieval time, we return the unique cases associated with the top-ranked chunks, in rank order.

- **HippoRAG2**: We use the default settings and operate directly on each case. Their implementation applies no chunking by default; this suffices because all cases fit within the token limit of our chosen embedding model, and preliminary experiments with a chunked variant yielded worse performance for this baseline.
- **Fast-GraphRAG**: We adopt the default chunking configuration from the implementation. For retrieval, we preserve the implementation’s default budget ratios and apply them to our 6,000-token budget, allocating 1,500 tokens to entities, 1,125 tokens to relations, and 3,375 tokens to chunks.

C.3 Uncertainty Analysis

For the paired uncertainty analysis, we rerun RAFT and vanilla RAG on identical held-out groups, progress points, and context budgets across the five splits, producing a paired result for every query. We then bootstrap the underlying issue-group clusters across all splits, rather than treating only five split-level averages as the inferential sample, and report percentile 95% confidence intervals for the paired differences in Table 5. All three Case Hit gains are statistically supported. Every unconditional coverage point estimate also favors RAFT; the early-stage coverage gains and the 30% root-cause gain are statistically supported, the 30% resolution gain is small and near the interval boundary, and we make no late-stage coverage claim.

Metric	Progress	RAFT – Vanilla (pp) [95% CI]
Case Hit	0%	+16.79 [+13.91, +19.78]
Case Hit	30%	+14.79 [+12.02, +17.73]
Case Hit	60%	+12.19 [+9.75, +14.68]
Root cause	0%	+5.69 [+3.35, +8.07]
Root cause	30%	+4.15 [+1.88, +6.56]
Root cause	60%	+1.88 [−0.07, +3.89]
Resolution	0%	+2.98 [+1.03, +4.99]
Resolution	30%	+2.01 [+0.09, +3.97]
Resolution	60%	+0.57 [−1.18, +2.32]

Table 5: Paired differences between RAFT and vanilla RAG in percentage points (pp), with issue-group-clustered bootstrap 95% CIs.

C.4 Query Robustness

We run a paired controlled-noise study at 30% and 60% progress across the same five held-out splits. For each clean query, both RAFT and vanilla RAG receive the identical deterministic perturbation: (i)

adjacent-character swaps in 2% of words of length at least five, (ii) removal of 40% of observed updates while preserving the initial report, or (iii) insertion of one turn from an unrelated issue category. We then measure each method’s Case Hit change relative to its result on the corresponding clean query; Table 6 reports the changes in percentage points, with positive paired differences favoring RAFT.

Perturbation	Progress	RAFT (pp)	Vanilla (pp)	Paired diff. (pp) [95% CI]
Light typos	30%	−0.11	−0.22	+0.11 [−0.44, +0.66]
Light typos	60%	0.00	+0.33	−0.33 [−1.05, +0.39]
40% update dropout	30%	−0.39	−1.05	+0.66 [−0.89, +2.22]
40% update dropout	60%	−0.83	−1.27	+0.44 [−1.39, +2.33]
One unrelated turn	30%	−11.14	−25.93	+14.79 [+11.52, +18.06]
One unrelated turn	60%	−9.36	−49.31	+39.94 [+35.73, +44.10]

Table 6: Case Hit change under paired, deterministic query perturbations, in percentage points (pp) relative to the corresponding clean query. Positive paired differences favor RAFT.

The changes under typos and missing updates are small for both methods, and the between-method differences are not statistically resolved. The unrelated turn is substantially harder and degrades vanilla RAG much more than RAFT. These are controlled query-robustness and within-benchmark results, not evidence of noisy-corpus robustness, production-scale behavior, or cross-domain generalization.

C.5 Graph Contribution and Sensitivity

In graph construction, k is a pre-symmetrization sparsity cap: we rank cross-case candidates using the hybrid semantic/lexical score, retain the top k , remove links below a fixed 0.6 embedding-similarity threshold, and then symmetrize the retained links, so a case can acquire more than k final neighbors through incoming links.

To isolate the graph’s marginal contribution, we compare pure entry-level retrieval with a budget-matched graph condition that reserves one of the five case slots for one-hop expansion from the directly retrieved seeds. *Full-sibling recovery* asks, for issue groups with two correct sibling cases available in the index, whether both are retrieved. Table 7 reports the sparsest tested setting, $k = 3$.

At $k = 3$, graph expansion modestly improves sibling diversity at early and intermediate progress; at 60%, direct retrieval is already high and the result is effectively unchanged. Overall Case Hit changes by only +0.22, −0.28, and −0.06 pp at 0%, 30%, and 60%, respectively. As a sensitivity check, $k = 5$ and $k = 10$ produce nearly identical

Progress	Direct recovery	With expansion	Change (pp)
0%	73.71%	75.65%	+1.94
30%	79.84%	82.74%	+2.90
60%	88.71%	88.55%	-0.16

Table 7: Full-sibling recovery without and with budget-matched graph expansion ($k = 3$; four direct anchors plus one graph-expanded slot).

results: across the three graph settings, Case Hit varies by at most 0.17 pp and full-sibling recovery by at most 0.81 pp. The conclusion is therefore not sensitive to k in this benchmark, and we use $k = 3$ throughout, presenting graph expansion as an optional evidence-diversification mechanism rather than the source of RAFT’s main retrieval gain.

C.6 Additional Experiments

We conduct two additional experiments. First, we examine how indexing-model capacity affects RAFT’s retrieval quality. Second, we evaluate an agentic setting in which the agent composes its own search queries and filtering conditions given the same 0%, 30%, and 60% case context used in the main experiments.

Note that in this paper we extract general-purpose entities. For domain-specific deployments, we recommend replacing these with application-specific entities (e.g., error codes, Windows version) that uniquely characterize a case; such entities enable agents to write more targeted queries and further narrow the search space.

Table 8 reports an ablation on indexing-model capacity, in which we replace the default indexing model gpt-5.2 with the smaller gpt-5.4-mini and gpt-5.4-nano, and separately reduce the reasoning effort from medium to low. Both gpt-5.4-mini and the low-reasoning setting yield only modest drops, whereas gpt-5.4-nano degrades more noticeably, as expected for a substantially less capable model. Extraction quality therefore matters, but smaller models remain viable under cost constraints, offering a practical trade-off for large-scale deployments.

In the agentic setting, we expose RAFT to the agent as a tool that filters cases by the category of the active case and retrieves similar cases from agent-supplied search queries. The agent is instructed to compose queries that reflect its current understanding of the issue, using the same case context at the 0%, 30%, and 60% progress points as in the main experiments. As shown in Table 9,

Method	Case Hit	Root Cause Coverage	Resolution Steps Coverage
0% Case Progress			
RAFT (low reasoning)	0.827	0.649	0.573
RAFT (gpt-5.4-mini)	0.823	0.603	0.536
RAFT (gpt-5.4-nano)	0.772	0.590	0.492
30% Case Progress			
RAFT (low reasoning)	0.868	0.686	0.597
RAFT (gpt-5.4-mini)	0.855	0.627	0.555
RAFT (gpt-5.4-nano)	0.800	0.611	0.545
60% Case Progress			
RAFT (low reasoning)	0.886	0.705	0.629
RAFT (gpt-5.4-mini)	0.871	0.643	0.575
RAFT (gpt-5.4-nano)	0.855	0.631	0.554

Table 8: Ablation on indexing-model capacity. We replace the default gpt-5.2 with smaller models (gpt-5.4-mini, gpt-5.4-nano) to assess sensitivity to extraction quality.

this improves Case Hit across all progress points.

Case Hit	Case Progress		
Method	0%	30%	60%
RAFT-Agent	0.910	0.910	0.919

Table 9: Agentic retrieval with RAFT. The agent composes its own queries and filtering conditions; Case Hit improves over the non-agentic baseline (Table 2).

C.7 Deployment Considerations

RAFT’s indexing cost is one self-contained LLM call per case. GraphRAG-family pipelines also run LLM extraction over the corpus during index construction, and some add further levels of processing such as community summarization (Edge et al., 2024), so RAFT’s construction cost is comparable to or lower than theirs. Relative to vanilla chunking, which needs no LLM at indexing time, several properties temper the added cost. First, distilling a case that has already been resolved is an easier task than solving it, so smaller and cheaper extraction models remain viable, as the indexing-model ablation above shows (Table 8). Second, the actionability filter drops unsolved and request-for-information cases that vanilla chunking ingests as noise. Third, distilled cases are compact, so a fixed context budget carries more cases at query time. Offline indexing still costs more than vanilla chunking overall, and whether this cost is repaid depends on query volume and model prices.

D Apache Jira Evaluation Set

We freeze public issue histories from Apache Cassandra, Hadoop, HBase, and Spark. Jira contributors mark a later report as a *Duplicate* of an older

issue in their normal workflow; we use the later report as the held-out query and the older issue as the exact target only if it was already resolved *Fixed* before the query opened. Other cases serve as distractors, not certified semantic negatives, because Jira links may be incomplete.

The construction pipeline proceeds as follows. Starting from 2,082 raw duplicate pairs, metadata filters retain 85 pairs whose older target was already *Fixed* and whose histories were sufficiently substantive. Pre-disclosure, leakage, length, near-copy, and one-pair-per-component filters reduce these to 41 candidate groups. Manual audit then removes 8 semantically mismatched or ambiguous links and 3 leakage-affected groups, leaving 30 audited groups. The final corpus contains 600 cases: the 30 gold targets and 570 *Fixed* distractors. By project, the 30 groups comprise 12 from Cassandra, 5 from Hadoop, 2 from HBase, and 11 from Spark; the distractors comprise 143 from Cassandra, 143 from Hadoop, 142 from HBase, and 142 from Spark. All 30 groups are evaluable at the 0% and 30% progress points; at 60%, the 19 groups with sufficient pre-disclosure history (6 Cassandra, 4 Hadoop, 2 HBase, 7 Spark) are evaluated.

We apply the same extraction prompt, schema, and retrieval procedure as the synthetic experiments; the one configuration difference is a 5,000-token context cap instead of 6,000. Each query retrieves five cases, and results are averaged over five indexing seeds. We report Case Hit only, because the dataset provides no gold root-cause or resolution-step annotations from which to compute coverage metrics. Given the 30-group scale, we report no confidence intervals and treat the result as directional transfer evidence.