An Adaptive Multi-Agent RAG Architecture for Software Ecosystem Update Monitoring

Shradha Devendra Pujari
University of the Pacific
Department of Computer Science
Stockton, CA, USA
s_pujari@u.pacific.edu

Solomon Berhe
University of the Pacific
Department of Computer Science
Stockton, CA, USA
sberhe@pacific.edu

Abstract

Software ecosystems produce a diverse collection of documents, including release notes, security advisories, version control records, and online community discussions. These documents contain information about software updates, vulnerabilities, compatibility issues, and deployment considerations. Accurately answering software update–related questions often requires integrating information across multiple sources. Retrieval-Augmented Generation (RAG) has emerged as a promising approach for leveraging external knowledge to answer such questions. However, existing single-agent RAG systems face challenges related to domain specialization, vocabulary mismatch, and limited feedback mechanisms. This paper presents an Adaptive Multi-Agent RAG Architecture for answering software update questions. Evaluated on 50 real user questions, the proposed multi-agent architecture achieves a 17.2% improvement in retrieval quality over a single-agent baseline ($t(49) = 2.32$, $p = 0.020$), while producing no hallucinated version-specific outputs in our evaluation. The adaptive feedback mechanism further adds a 9.3% improvement in retrieval quality across successive interactions without human feedback.

Keywords

Multi-Agent Systems, Retrieval-Augmented Generation, RLAIF, Software Ecosystem Monitoring, Self-Improving Agents, Vendor Extraction, LangChain, smolagents

ACM Reference Format:
Shradha Devendra Pujari and Solomon Berhe. 2026. An Adaptive Multi-Agent RAG Architecture for Software Ecosystem Update Monitoring. In *Proceedings of the Workshop on Agentic Software Engineering (AgenticSE '26).* ACM, New York, NY, USA, 8 pages.

1 Introduction

Modern software ecosystems produce a diverse collection of documents, including release notes, security advisories, version control records, and online community discussions. These documents contain information about software updates, vulnerabilities, compatibility issues, and deployment considerations. Answering software update–related questions often requires integrating information from multiple sources.

Software update information is frequently accessed through natural language questions. Developers, system administrators, and users commonly ask colleagues about software updates or seek information through online community platforms. Answering such questions often requires locating relevant documents, reviewing information from multiple sources, and evaluating the consistency of available documents. Systems that generate accurate source-grounded answers can reduce the effort required to retrieve, read, and synthesize information from software ecosystem documents.

Retrieval-Augmented Generation (RAG) systems have emerged as a promising approach for answering such questions by combining document retrieval with large language models to generate answers grounded in external information [1]. Conventional single-agent RAG systems typically perform query understanding, retrieval, evaluation, and answer generation within a single reasoning process. This can result in vocabulary mismatch between user questions and retrieved documents [2], limited handling of domain-specific terminology, and the absence of feedback mechanisms for correcting retrieval errors [3].

Recent multi-agent RAG systems decompose retrieval and reasoning into multiple specialized agents [4, 5]. Most existing evaluations focus on general question-answering benchmarks, whereas software update questions require combining information from heterogeneous software ecosystem documents. In addition, the role of feedback-based adaptation in multi-agent RAG systems for software update question answering has received limited empirical evaluation [6].

This paper presents an Adaptive Multi-Agent RAG Architecture for answering software update questions [7, 8]. The architecture consists of specialized agents for query rewriting, retrieval, evaluation, and orchestration connected through a feedback loop [10–12]. The system is evaluated on 50 real user questions and compared with a single-agent baseline. The evaluation examines retrieval quality, source-grounded answer generation, and retrieval improvement through adaptive feedback without human intervention.

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 describes the proposed architecture. Section 4 presents the experimental evaluation and results. Section 5 discusses limitations and findings. Section 6 concludes the paper.

2 Related Work

This section reviews prior work on Retrieval-Augmented Generation (RAG), multi-agent architectures, corrective retrieval, and self-improving agents. The proposed architecture builds on concepts from each of these areas.

2.1 Retrieval-Augmented Generation

Retrieval-Augmented Generation (RAG) combines document retrieval with large language models to generate answers grounded

AgenticSE '26, San Jose, CA, USA
2026.

in external documents [1]. Several extensions have addressed limitations of fixed retrieval pipelines. DMQR-RAG uses query rewriting to reduce vocabulary mismatch [2]. RaFe introduces feedback-based reranking of retrieved documents [6]. Self-RAG incorporates self-evaluation during generation [13], while Adaptive-RAG selects retrieval logic based on query characteristics [14]. Surveys identify adaptive retrieval and agent-based coordination as active research directions in RAG systems [15].

2.2 Multi-Agent RAG Frameworks

Multi-agent RAG systems distribute retrieval and reasoning across specialized agents. MA-RAG employs planner, extraction, and question answering agents [5]. MAIN-RAG separates document selection and answer generation through dedicated predictor and judge agents [4]. More generally, frameworks such as AutoGen [16] and smolagents [17] provide infrastructure for agent orchestration and tool use. Most prior evaluations focus on general question-answering benchmarks, whereas this work evaluates a multi-agent architecture for software update question answering.

2.3 Corrective Retrieval and Adaptive Feedback

Corrective retrieval methods modify retrieval behavior when retrieved information is insufficient. CRAG evaluates retrieval quality and initiates additional retrieval when needed [18]. FLARE performs retrieval conditioned on uncertainty during answer generation [19]. Reflexion uses iterative self-feedback to guide future behavior [20]. The architecture presented in this paper adopts a similar feedback-oriented perspective through a dedicated evaluator agent that can trigger retrieval retries when retrieval quality falls below a predefined threshold.

2.4 Self-Improving Agents

RLAIF replaces human preference annotations with AI-generated feedback signals for system improvement [21]. Related work has explored feedback-driven adaptation in retrieval and generation systems. Unlike approaches that update model weights, the architecture proposed in this paper maintains a persistent memory of retrieval outcomes and uses those outcomes to adjust future query rewriting without human feedback or model retraining.

3 Method

This section describes the proposed adaptive multi-agent RAG architecture for software update question answering. The architecture decomposes query rewriting, retrieval, evaluation, and orchestration into specialized agents connected through an adaptive feedback loop. We also describe the software ecosystem data sources, answer synthesis process, and experimental setup used in the evaluation.

3.1 Architecture Overview

The proposed architecture separates software update question answering into four specialized agents coordinated by a central Orchestrator. Rather than performing query understanding, retrieval, evaluation, and answer generation within a single reasoning process, each task is assigned to a dedicated component with a defined responsibility. The architecture consists of a Query Rewriter, Retriever, Evaluator, and Orchestrator connected through a retrieval-level feedback loop, as shown in Figure 1.

The design follows five principles. First, each agent performs a single task. Second, retrieval is restricted to vendor-specific information when a vendor can be identified. Third, retrieval quality is evaluated through an explicit feedback mechanism that can trigger additional retrieval attempts. Fourth, failures are isolated through defined input and output contracts. Finally, retrieval outcomes are persisted across sessions to support adaptation without human feedback.

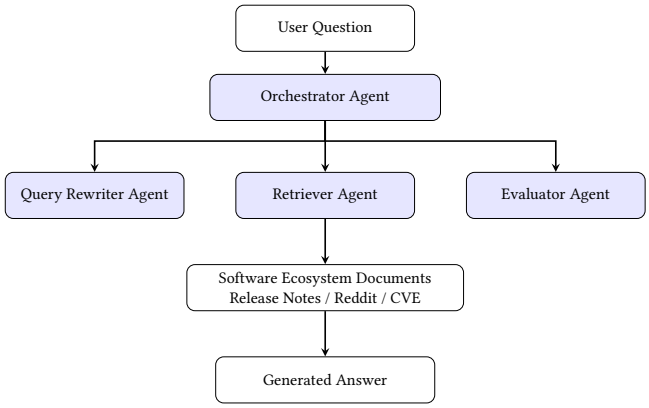

Figure 1: Adaptive multi-agent RAG architecture.

3.2 Agent Definitions

The architecture comprises four agents organized under Orchestrator coordination.

Orchestrator Agent. The Orchestrator coordinates execution of the retrieval pipeline. It receives the user question, delegates tasks to the remaining agents, and manages retrieval retries when requested by the Evaluator. The Orchestrator does not perform retrieval or answer generation directly.

Query Rewriter Agent. The Query Rewriter addresses differences between user vocabulary and document vocabulary. Llama 3.1 8B is used to transform the original question into a retrieval-oriented representation. In the self-improving configuration, learned terms from previous retrieval outcomes are incorporated before query generation.

Retriever Agent. The Retriever performs vendor-aware document retrieval. It first attempts to identify the software vendor referenced in the query using exact matching, alias lookup, and fuzzy matching against a registry of 14,223 software vendors and 628 software-related subreddits. If a vendor is detected, retrieval is restricted to vendor-specific release notes, advisories, and community discussions. Otherwise, a general retrieval logic is used. The Retriever returns ranked documents without evaluating their quality [22–24].

Evaluator Agent. The Evaluator estimates retrieval quality using a deterministic score between 0.0 and 1.0. The score combines evidence from retrieval volume, release-note matches, community matches, and CVE matches. If the score falls below a predefined

threshold ($\theta = 0.30$), the Evaluator signals the Orchestrator to perform an additional retrieval attempt. The Evaluator does not generate answers [25].

3.3 Adaptive Feedback and Self-Improvement

The architecture incorporates a retrieval-level feedback loop. After retrieval, the Evaluator assigns a quality score to the retrieved documents. If retrieval quality is insufficient, the Orchestrator re-executes the retrieval pipeline using a revised question generated by the Query Rewriter.

The system also maintains a persistent Self-Improvement Memory. Query expansion terms accumulate positive or negative scores based on retrieval outcomes across sessions. Successful retrievals increase associated term weights, while unsuccessful retrievals decrease them. These scores are subsequently used by the Query Rewriter to influence future retrieval queries. This mechanism provides adaptation without human preference labels, manual annotation, or model retraining.

3.4 Software Ecosystem Data Sources

The system retrieves information from software ecosystem documents obtained through multiple API endpoints.

Verified Sources

- `/{vendor}`: vendor registry (6,578 entries)
- `/v/`: software release notes (31,958 entries)
- `/cve`: vulnerability advisories (24,139 entries)

Community Sources

- `/reddit`: community discussions (208,466 posts)
- `/positive`: software update risk discussions (2,637 posts)
- `/by-subreddit?q={vendor}`: vendor-specific discussions
- `/questions`: software-related user questions

To support vendor-aware retrieval, a registry of software vendors and associated subreddits is loaded at startup and cached locally.

3.5 Answer Generation

Retrieved documents are synthesized into a final answer using a structured prompt. The prompt constrains generation to information contained in retrieved sources and defines explicit behavior for common retrieval outcomes.

The synthesis process follows six rules:

(1) Summarize matching community discussions and associated comments.
(2) Report version-specific information when present in release notes.
(3) Explicitly state when retrieved sources do not directly answer the question.
(4) Do not generate unsupported version numbers, fixes, or CVE identifiers.
(5) Restrict claims to information contained in retrieved sources.
(6) Report when no matching vendor-specific information is available.

3.6 Experimental Setup

The evaluation dataset consists of 50 software update questions spanning security, bug, release, community, and general software maintenance topics. The local fallback dataset contains 324 Reddit posts collected through the software ecosystem data lake and filtered using minimum quality criteria of at least 10 comments, 3 author replies, and a quality score of at least 0.30.

The single-agent baseline performs query rewriting, retrieval evaluation, and answer generation within a single agent without specialized agents, adaptive retry, or self-improvement memory. It uses the same retrieval endpoints and answer synthesis prompt as the proposed architecture, isolating the effect of architectural decomposition.

All experiments were performed using Llama 3.1 8B executed through Ollama with temperature set to zero. The evaluation platform consisted of Apple Silicon hardware with 24 GB memory and Metal GPU acceleration.

4 Results

This section presents the evaluation results of the proposed adaptive multi-agent RAG architecture. The evaluation examines retrieval quality, answer accuracy, adaptive feedback behavior, self-improvement performance, representative examples, failure modes, and implementation observations using a dataset of 50 software update questions.

4.1 Retrieval Quality

Table 1 summarizes retrieval quality across five question categories. The proposed architecture achieved higher mean retrieval quality than the single-agent baseline in all categories. The overall retrieval quality increased from 0.680 to 0.798, corresponding to a 17.2% improvement. A paired t-test across the 50 evaluation queries yielded $t(49) = 2.32$ and $p = 0.020$.

Table 1: Retrieval Quality by Question Category

Rank	Category	1-Agent	4-Agents	Improvement
1	Security	0.630	0.750	+19.0%
2	Bugs	0.756	0.833	+10.3%
3	Releases	0.667	0.723	+8.5%
4	Community	0.730	0.850	+16.4%
5	General	0.600	0.750	+25.0%
	Overall	**0.680**	**0.798**	**+17.2%**

Table 2 reports the effect of removing individual components from the architecture. Vendor-aware retrieval, multi-source retrieval, and CVE integration each reduced retrieval quality when removed. The largest reduction was observed when multi-source retrieval was disabled.

Table 2: Ablation Study

Rank	Configuration	Mean Quality	Contribution
1	Our System (all 4 agents)	0.340	—
2	– Vendor Extraction (fallback only)	0.325	+0.015
3	– Multi-Source Validation (single tier)	0.320	+0.020
4	– CVE Integration (releases+Reddit)	0.321	+0.019

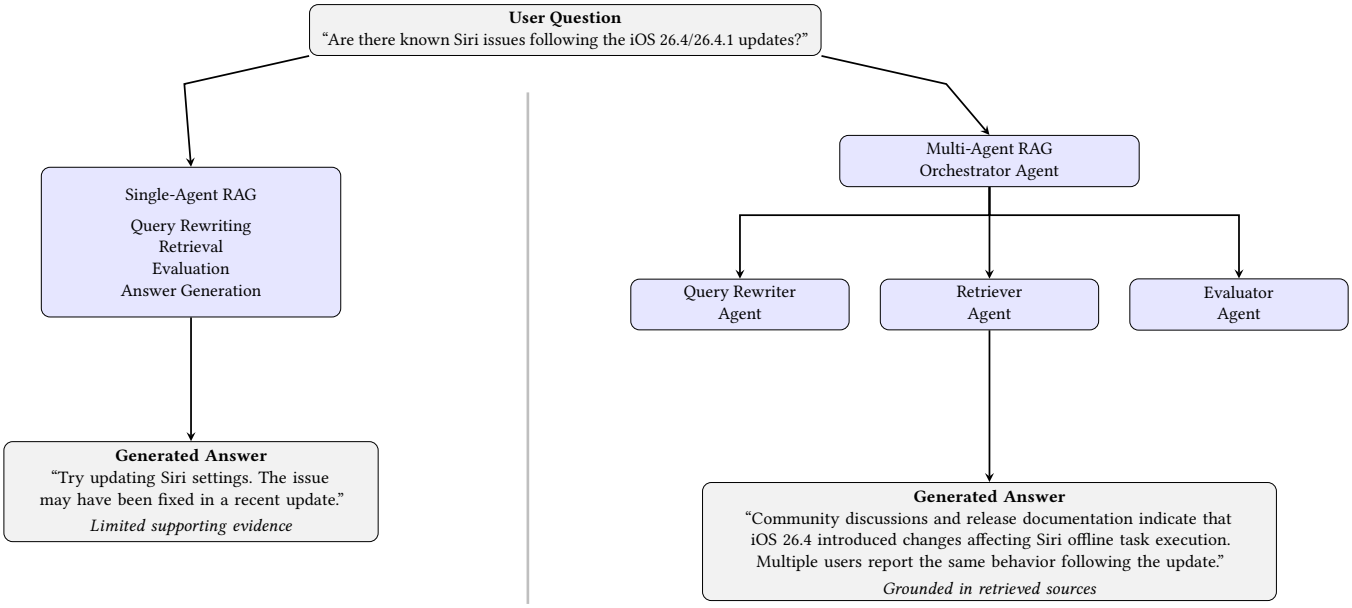

Figure 2: Illustrative comparison of single-agent and multi-agent RAG architectures on a software update question.

4.2 Answer Accuracy

To evaluate answer quality beyond retrieval metrics, the system was tested on version-specific software update questions. For example, the system correctly identified Linux v7.0.0 (released April 13, 2026) and Firefox v149.0.1 (released April 7, 2026) from the corresponding release sources. All reported version numbers were traceable to retrieved documents.

A supplementary evaluation using ten questions from the live questions endpoint produced four fully correct answers and six partially correct answers. Across these questions, the system did not generate any unsupported version numbers, release dates, or software update claims. All version-specific information appearing in generated answers could be traced to retrieved release notes or other retrieved documents.

Figure 2 presents a representative software update question. The question concerns Siri behavior following the iOS 26.4 update. The Retriever identified the relevant vendor, collected release and community information, and the generated answer reflected information contained in the retrieved sources.

Figure 3: Example execution trace of the Query Rewriter, Retriever, and Evaluator agents.

4.3 Example Multi-Agent Execution

Figure 3 illustrates a representative execution of the proposed architecture on a software update question. The Query Rewriter Agent reformulates the user question to improve retrieval coverage, the Retriever Agent gathers evidence from software ecosystem documents, and the Evaluator Agent assigns a retrieval quality score before answer generation. This example demonstrates how retrieval and evaluation are separated into specialized agents within the proposed architecture.

4.4 Adaptive Feedback

The Evaluator triggered retrieval retries when the initial retrieval quality score fell below $\theta = 0.30$. In these cases, the Orchestrator executed an additional retrieval cycle using a revised query generated by the Query Rewriter. This behavior occurred without manual intervention during evaluation and allowed retrieval to continue when the initial result set was insufficient.

4.5 Self-Improvement Behavior

Retrieval quality increased from a mean score of 0.756 in the first evaluation tertile (Q_1–Q_{17}) to 0.826 in the final tertile (Q_{34}–Q_{50}), corresponding to a 9.3% increase.

A paired t-test comparing the first and last tertiles yielded $t(16) = 2.18$ and $p = 0.043$ (two-tailed). The corresponding effect size was Cohen's $d = 0.83$ [26].

During evaluation, 27 query-expansion terms accumulated positive scores in the Self-Improvement Memory. Table 3 lists the ten highest-scoring terms.

Table 3: Highest-Scoring Learned Terms

Rank	Term	Cumulative Score	Interpretation
1	vulnerability	+2.20	Security terminology
2	patch	+1.58	Fix-related terminology
3	advisory	+1.23	Advisory terminology
4	update	+1.15	General update terminology
5	kernel	+0.98	Vendor-specific terminology
6	release	+0.87	Version-related terminology
7	bug	+0.76	Defect-related terminology
8	fix	+0.71	Resolution terminology
9	version	+0.68	Version-related terminology
10	security	+0.64	Security terminology

4.6 Representative User Questions and Answers

Table 4 presents representative examples from the evaluation dataset. The examples include correctly answered queries, partially relevant answers, and incorrect answers.

4.7 Failure Analysis

The three incorrect examples in Table 4 correspond to three recurring failure modes observed during evaluation.

Temporal Question Filtering. The question "What was the Linux version on January 1st 2026?" returned the latest Linux release rather than the release available on the specified date. The query rewriter preserved the temporal constraint, but the constraint was not passed to the retrieval endpoint.

Vendor Extraction Failure. The question "Synology NAS unreachable after upgrade" did not produce a vendor match and therefore fell back to general retrieval. This occurred because important domain-specific terms were removed during preprocessing.

Opinion-Oriented Queries. The question "What small IT task always turns into a nightmare?" contains no software vendor reference and does not correspond to a factual software update question. The system correctly failed to locate relevant documents but did not explicitly reject the question.

These three cases account for the incorrect examples reported in Table 4.

4.8 Implementation Comparison

The architecture was implemented using Pure Python, LangChain, and smolagents. All three implementations produced equivalent retrieval results on the evaluation dataset.

The Pure Python implementation provided the most direct execution trace. LangChain exposed additional intermediate execution information through callback logging. Within smolagents, ToolCallingAgent executed reliably with Llama 3.1 8B, whereas Code Agent occasionally generated invalid tool calls and repeated execution cycles.

4.9 Qualitative Analysis

Execution traces showed the Orchestrator coordinating query rewriting, retrieval, evaluation, and answer synthesis across multiple document sources. Retrieved documents were incorporated into generated answers only after evaluation. Detailed execution traces are omitted due to space constraints.

5 Discussion

Several observations emerge from the evaluation.

- **Architectural decomposition improves retrieval quality.** The proposed architecture separates query rewriting, retrieval, evaluation, and orchestration into specialized agents. The observed 17.2% improvement over the single-agent baseline suggests that this decomposition improves retrieval performance for software update questions, which often contain abbreviated product names, informal terminology, incomplete version identifiers, and user-reported symptoms. Separating query rewriting from retrieval allows terminology normalization before document search, improving retrieval coverage.
  The architecture also improves traceability by allowing failures to be associated with vendor identification, query reformulation, retrieval coverage, evaluation, or answer synthesis rather than treated as end-to-end model failures.
- **Adaptive feedback improves retrieval without retraining.** The observed 9.3% retrieval improvement demonstrates that retrieval behavior can improve through accumulated retrieval experience without modifying model weights. Successful retrieval patterns are preserved through persistent memory and influence future query reformulation.
  This result is noteworthy because the architecture operates entirely with a locally hosted LLM. Rather than relying on additional training data or parameter updates, adaptation occurs through retrieval-level feedback. Such an approach may be attractive in environments where retraining is costly, unavailable, or undesirable due to privacy, deployment, or reproducibility requirements.
  The observed improvement should not be interpreted as evidence of general learning capability. Instead, it demonstrates that retrieval behavior can be incrementally refined through accumulated retrieval experience. Larger studies are required to determine how such gains scale over longer interaction histories and larger question sets.
- **Multiple document sources provide complementary evidence.** Many questions in the evaluation required information that was not available from a single source. Release notes provide authoritative information regarding versions, fixes, and feature changes. Security advisories provide vulnerability information and mitigation guidance. Community

Table 4: Representative Questions Examples

Question	Vendor	Multi-Agent Answer	Evaluator Note	Result
(A) Correctly Answered				
Q1: What is the latest version of Firefox?	firefox	v149.0.1, released April 7 2026.	Exact release note match.	✓
Q2: Are there known Siri issues following the iOS 26.4/26.4.1 updates?	ios	iOS 26.4 changed Siri offline mode. r/applehelp confirms.	Multiple sources retrieved.	✓
Q3: G6 Bullet unstable?	Ubiquiti	Community reports instability; firmware update available.	Alias resolved to vendor.	✓
(B) Partially Relevant				
Q4: Would an IKEA hub update devices more reliably?	homeassistant	Home Assistant integration information retrieved.	Sub-vendor not represented in registry.	~
Q5: Can't upload photos on iOS to Immich	ios	iOS information retrieved; limited Immich coverage.	Third-party application not present in vendor sources.	~
Q6: Do I have to say goodbye to Windows to switch to Linux?	linux	General Linux information retrieved.	Opinion-oriented question.	~
(C) Incorrectly Answered				
Q7: What was the Linux version on January 1st 2026?	linux	Returned v7.0.0 instead of v6.18.0.	Date constraint not applied.	✗
Q8: Synology NAS unreachable after upgrade	(none)	No vendor detected.	Vendor extraction failure.	✗
Q9: What small IT task always turns into a nightmare?	(none)	No relevant documents retrieved.	Non-factual question.	✗

discussions provide deployment experiences, regressions, compatibility issues, and workarounds that may not yet appear in official documentation.
The combination of these sources is particularly valuable for software update questions because users frequently ask about operational effects of updates rather than only documented release information. A release note may confirm that a defect was fixed, while community discussions may reveal whether users continue to experience related issues after deployment. Treating these sources as complementary rather than competing forms of evidence improves coverage while preserving source attribution.

- **Source-grounded retrieval reduces unsupported update claims.** The supplementary evaluation produced four fully correct answers and six partially correct answers. No unsupported version numbers, release dates, patch identifiers, or software update claims were observed in this sample. This observation is particularly relevant for software maintenance tasks. Incorrect version information may influence update planning, security decisions, compatibility assessments, or deployment timing. While the evaluation sample remains limited, grounding answer generation in retrieved software ecosystem documents appears to reduce the likelihood of unsupported version-specific outputs. Future work should evaluate this behavior using larger benchmark datasets and longer deployment periods.

5.1 Implications for Software Update Question Answering

The evaluation suggests that software update question answering differs from traditional fact-based question answering in several respects. Questions frequently reference evolving software versions, incomplete symptom descriptions, community terminology, and information dispersed across multiple document sources. Consequently, retrieval quality becomes a central factor influencing answer quality.

The results indicate that improvements in retrieval quality translate directly into improved answer supportability. When relevant release notes, advisories, and community discussions are successfully retrieved, the generated answer is more likely to contain verifiable

evidence and less likely to rely on unsupported assumptions. This relationship motivates further investigation of retrieval-focused architectures for software ecosystem applications.

The findings also suggest that software update question answering benefits from explicit uncertainty management. Questions concerning future releases, undocumented issues, or emerging community reports may not always have definitive answers. In such situations, exposing retrieved evidence and confidence information may be more useful than forcing a categorical response. Future architectures should therefore consider retrieval confidence as a first-class component of answer generation.

Six limitations of the current system warrant discussion.

Data coverage lag. Recent community posts may not yet be indexed in the data lake. In the live evaluation, some partial answers were attributable to source availability rather than retrieval failures. Similar issues may occur whenever upstream data sources introduce delays between publication and indexing.

Vendor alias coverage. The `VENDOR_ALIASES` dictionary currently maps common aliases to canonical vendor names. Queries containing unknown vendor names or product aliases fall back to broader retrieval logic, reducing precision. Future work will replace static aliases with embedding-based lookup over the full vendor registry.

Community-source reliability. Community platforms such as Reddit provide useful information about deployment experiences, regressions, and workarounds, but posts are typically anonymous and may contain inaccurate, incomplete, or misleading information. A single post may not represent the broader experience of a software ecosystem. The current system mitigates this limitation by separating verified and community sources and considering multiple retrieved posts when available. Future work will investigate cross-post validation, agreement-based confidence estimation, and community-source credibility assessment.

Local LLM capacity. Llama 3.1 8B via Ollama performs well on structured software update questions but occasionally over-interprets ambiguous or conversational queries. Larger or domain-adapted models may improve synthesis quality, particularly for questions requiring extensive contextual reasoning.

Evaluation scope and generalization. The evaluation consists of 50 software update questions spanning five categories. Although diverse, this remains a relatively small sample compared with real-world workloads. Additionally, all evaluation questions are English-language. Generalization to larger datasets, additional languages, and different software ecosystems remains an open question.

Statistical impact. The 17.2% improvement is statistically significant ($p = 0.020$), but the sample size ($n = 50$) limits confidence in precise performance estimates. Larger evaluations are required to establish more robust effect sizes and confidence intervals.

5.2 Operational Considerations

The prototype highlights several deployment considerations. Vendor-aware caching improves retrieval efficiency by resolving vendor names, aliases, and community routes before retrieval begins. Separating verified and community sources provides useful distinctions between authoritative update information and user-reported experiences.

The Evaluator Agent serves both as a retrieval scoring component and as a control point for retry and confidence estimation. This is particularly important for version-specific questions, where returning an explicit "no matching source" response may be preferable to generating unsupported information. The ability to reject unsupported answers is an important property for software maintenance applications, where incorrect information may directly impact decisions.

Presenting retrieved evidence alongside generated answers improves answer provenance and supports auditing when errors occur. This is particularly useful when answers combine information from multiple sources because users can independently verify the supporting evidence. Local execution through Ollama also enables reproducible experiments and privacy-preserving deployment.

5.3 Reproducibility Notes

The architecture was designed so that major components can be evaluated independently. Vendor detection, retrieval, evaluation, and answer generation can each be inspected without requiring end-to-end execution. This modular structure simplifies debugging and enables systematic error analysis because failures can be associated with specific pipeline stages.

For replication, the most important artifacts are the evaluation questions, vendor registry snapshot, API responses, retrieval scoring implementation, and system configuration. Because the system relies on live software ecosystem APIs, future studies should preserve response timestamps or snapshots. Static benchmark snapshots would facilitate controlled comparison across architectures, while live evaluations would continue to measure performance under evolving ecosystem conditions.

5.4 Threats to Validity

Construct validity. Retrieval quality is measured using a deterministic score emphasizing release-note relevance, community evidence, advisory evidence, and result coverage. While this enables reproducible evaluation, it may not capture all dimensions of answer quality such as usefulness, readability, user trust, or task completion effectiveness.

Internal validity. The single-agent baseline uses the same retrieval endpoints and answer synthesis process as the proposed system, isolating the effect of architectural decomposition, adaptive feedback, and persistent memory. Implementation details such as alias coverage, endpoint freshness, query rewriting quality, and retrieval configuration may still influence measured improvements.

External validity. The evaluation focuses on English-language software update questions and publicly available software ecosystem data sources. Results may differ for other languages, software ecosystems, enterprise repositories, or domains where release documentation and community discussions are limited.

5.5 Robustness of Heuristic Design Choices

Several components rely on handcrafted heuristics including retrieval weights, vendor aliases, retry thresholds, and confidence thresholds. Similar mechanisms are commonly used in corrective and adaptive RAG systems [13, 18, 19].

The scoring structure itself remains largely domain independent. The same architecture could be adapted to other document collections by replacing domain-specific keyword sets, aliases, and retrieval sources. The separation between retrieval, evaluation, and orchestration would remain unchanged. Future work will investigate learned reward models, adaptive threshold selection, and data-driven calibration methods to reduce manual tuning while preserving interpretability.

5.6 Future Work

Future work includes comparisons with Self-RAG, CRAG, and other multi-agent RAG systems, evaluation on larger and multilingual question sets, and investigation of learned reward models, embedding-based vendor matching, and adaptive threshold selection [9, 27, 28].

Additional studies should examine retrieval adaptation across longer interaction histories and larger workloads to better understand the stability of the observed retrieval improvements. Evaluations involving multiple software ecosystems would also help determine whether the observed benefits generalize beyond the current data sources [29].

Finally, future work should investigate stronger validation mechanisms for community-derived information, including agreement analysis across multiple posts, source credibility estimation, and integration of additional software ecosystem resources. Such mechanisms may further improve answer reliability while preserving the practical benefits of community knowledge.

6 Conclusion

This paper presented a multi-agent RAG architecture for answering software update questions using software ecosystem documents. The architecture decomposes query rewriting, retrieval, evaluation, and orchestration into specialized agents connected through an adaptive feedback loop and persistent memory.

Evaluation on 50 real software update questions showed a 17.2% improvement in retrieval quality compared with a single-agent baseline ($t(49) = 2.32$, $p = 0.020$). The adaptive feedback mechanism produced an additional 9.3% improvement in retrieval quality across successive interactions without human feedback. A supplementary evaluation on ten live questions produced four fully correct answers and six partially correct answers. No unsupported version numbers, release dates, or software update claims were observed during this evaluation.

The results suggest that architectural decomposition can improve retrieval quality for software update question answering by separating query rewriting, retrieval, and evaluation into dedicated agents. The study also demonstrates that retrieval-level feedback can be used to adapt query generation without model retraining.

Future work includes evaluation against additional RAG baselines, larger-scale studies using more diverse software ecosystems, and the replacement of heuristic retrieval scoring and vendor matching mechanisms with learned approaches.

Acknowledgments

We thank the maintainers of public software ecosystem data sources, online communities, and open APIs for making software-related information accessible for research and analysis. Their continued contributions enable the collection and evaluation of real-world software update questions and supporting documents used in this study.

References

[1] P. Lewis, E. Perez, A. Piktus, H. Schwenk, F. Schwab, D. Kiela, and E. Ouz, "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 9457–9467, 2020.

[2] Y. Gao, Y. Xiong, X. Gao, and K. Jiao, "DMQR-RAG: Diverse Multi-Query Rewriting for Retrieval-Augmented Generation," *arXiv preprint arXiv:2401.12617*, 2024.

[3] T. Cook et al., "Retrieval Augmented Generation (RAG) for Fintech: Agentic Design and Evaluation," *arXiv https://arxiv.org/abs/2510.25518*, 2026.

[4] X. Wang, W. Chen, Y. Liu, and M. Zhang, "MAIN-RAG: Multi-Agent Intelligent Retrieval-Augmented Generation for Software Engineering Tasks," *arXiv preprint*, 2024.

[5] T. Nguyen et al., "MA-RAG: Multi-Agent Retrieval-Augmented Generation for Complex Question Answering," *Proceedings of the 2025 Conference of the Association for Computational Linguistics*, pp. 1–12, 2025.

[6] S. Mao et al., "RaFe: Ranking Feedback Improves Query Rewriting for RAG," *arXiv preprint https://arxiv.org/abs/2405.14431*, 2024.

[7] J. He et al., "LLM-Based Multi-Agent Systems for Software Engineering: Literature Review, Vision, and the Road Ahead," *ACM Transactions on Software Engineering and Methodology*, vol. 34, no. 5, 2025.

[8] S. Joshi, "Review of Autonomous and Collaborative Agentic AI and Multi-Agent Systems for Enterprise Applications," *International Journal of Innovative Research in Engineering & Management*, vol. 12, no. 3, pp. 65–76, 2025. doi:10.55524/ijirem.2025.12.3.9.

[9] L. Ranaldi et al., "Multilingual Retrieval-Augmented Generation for Knowledge-Intensive Task ," *arXiv preprint https://arxiv.org/abs/2504.03616*, 2025.

[10] S. Ghosh et al., "Event-Driven Approach for Monitoring and Orchestration of Cloud-Native Applications," *2025 IEEE International Conference on Cloud Computing (CLOUD)*, pp. 112–119, 2025.

[11] A. Adimulam et al., "The Orchestration of Multi-Agent Systems: Architectures, Protocols, and Enterprise Adoption," *arXiv preprint https://arxiv.org/html/2601.13671v1*, 2026.

[12] A. Roman et al., "Orchestral AI: A Framework for Agent Orchestration," *arXiv preprint https://arxiv.org/abs/2601.02577*, 2026.

[13] A. Asai et al., "Self-RAG: Learning to Retrieve, Generate, and Critique for Self-Improved Generation," *The Eleventh International Conference on Learning Representations*, 2023.

[14] M. Jeong et al., "Adaptive-RAG: Learning to Adapt Retrieval-Augmented Generation Based on Query Complexity," *arXiv preprint https://arxiv.org/abs/2403.14403*, 2024.

[15] Y. Gao et al., "Retrieval-Augmented Generation for Large Language Models: A Survey," *arXiv preprint https://arxiv.org/abs/2312.10997*, 2023.

[16] Q. Wu et al., "AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation," *arXiv preprint https://arxiv.org/abs/2308.08155*, 2023.

[17] HuggingFace, "Agents Course, Unit 1: What are Agents?" https://huggingface.co/learn/agents-course, 2024.

[18] S. Yan et al., "Corrective Retrieval Augmented Generation," *arXiv preprint https://arxiv.org/abs/2401.15884*, 2024.

[19] Z. Jiang et al., "Active Retrieval Augmented Generation," *arXiv preprint https://arxiv.org/abs/2305.06983*, 2023.

[20] N. Shinn et al., "Reflexion: Language Agents with Verbal Reinforcement Learning," *arXiv preprint https://arxiv.org/abs/2303.11366*, 2023.

[21] Y. Bai et al., "Constitutional AI: Harmlessness from AI Feedback," *arXiv preprint https://arxiv.org/abs/2212.08073*, 2022.

[22] X. Guan et al., "DeepRAG: Thinking to Retrieve Step by Step for Large Language Models," *arXiv preprint https://arxiv.org/abs/2508.01005*, 2025.

[23] Z. Yang et al., "Superintelligent Retrieval Agent: The Next Frontier of Information Retrieval ," *arXiv preprint https://arxiv.org/abs/2605.06647*, 2026.

[24] Y. Zheng et al., "SkillRouter: Skill Routing for LLM Agents at Scale ," *arXiv preprint https://arxiv.org/html/2603.22455v4*, 2026.

[25] M. Zhuge et al., "Agent-as-a-Judge: Evaluate Agents with Agents," *arXiv preprint https://arxiv.org/abs/2410.10934*, 2024.

[26] J. Cohen, "A Coefficient of Agreement for Nominal Scales," *Educational and Psychological Measurement*, vol. 20, no. 1, pp. 37–46, 1960.

[27] A. Asai et al., "Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection," *arXiv preprint https://arxiv.org/abs/2310.11511*, 2023.

[28] X. Yang et al., "CRAG – Comprehensive RAG Benchmark ," *arXiv preprint https://arxiv.org/abs/2406.04744*, 2024.

[29] X. Yang et al., "Towards Reliable Retrieval in RAG Systems for Large Legal Datasets ," *arXiv preprint https://arxiv.org/abs/2510.06999*, 2025.