



ARTICLE

Traceable Multi-Round RAG with Classified Routing for Complex Equipment Maintainability Verification

Jixin Xu, Xingxin Li^{*}, Qingqing Hu and Qingqing Song

Shijiazhuang Campus, Army Engineering University of PLA, Shijiazhuang, China

^{*}Corresponding Author: Xingxin Li. Email: xingxinli_095@163.com

Received: 02 July 2026; Accepted: 22 September 2026; Published: 22 October 2026

ABSTRACT: Maintainability verification, which validates whether equipment designs conform to relevant evaluation criteria, currently relies heavily on manual expert review. This paper proposes a traceable multi-round Retrieval-Augmented Generation framework integrating classified routing. It converts unstructured design records into a standardized information model driven by state-machine transitions. Given natural language queries as inputs, the pipeline adopts type-aware classified routing to allocate each query to dedicated retrieval branches with type-specific weight configurations. A confidence-controlled multi-round retrieval module further builds full hierarchical trace chains, linking every model reasoning output back to the original standard clauses. Evaluated on our curated domain benchmark, the proposed method yields $F1@10 = 0.743$ and Answer Correctness = 0.631, delivering a statistically significant improvement in Answer Correctness over the strongest baseline, Corrective Retrieval-Augmented Generation (CRAG), in its default configuration ($p < 0.05$) and a 0.126 Recall gain on complex queries. The traceability module achieves 96.8% completeness and 93.4% attribution accuracy, providing structured four-level auditable decision pathways. Ablation studies validate per-component contributions.

KEYWORDS: Maintainability verification; retrieval-augmented generation; classified routing; multi-round retrieval; traceability

1 Introduction

Maintainability is defined as the ability of an item, under given conditions of use, to be retained in or restored to a state in which it can perform its required function when maintenance is performed using stated procedures and resources [1]. This property directly influences the life cycle cost, operational availability, and mission readiness of complex equipment systems [2]. Defense organizations worldwide have codified maintainability requirements in standards, which prescribe qualitative design criteria covering accessibility, standardization, error-proofing, safety, and ergonomics [3,4].

Despite well-established industry standards, compliance verification for qualitative maintainability requirements still relies on manual labor. Engineers check Bills of Materials (BOM), digital mock-ups (DMU), and maintenance manuals, cross-referencing each engineering document against checklist criteria. This approach has three major limitations. First, poor scalability, as thousands of verification entries span hundreds of components, each requiring diverse supporting evidence [5,6]. Second, outcomes are inconsistent because verification quality depends heavily on individual expertise [7]. Third, traceability remains inadequate since mapping non-conformant design points back to their source standard clauses requires manual forensic tracking that impedes systematic design improvement [8].

Retrieval-augmented generation (RAG) combines the generative capabilities of large language models (LLMs) with external knowledge retrieval, which grounds domain-specific outputs in verifiable factual materials [9,10]. RAG has been deployed across engineering scenarios including fault diagnosis, manufacturing quality control, and regulatory compliance assessment [11–13]. However, conventional RAG frameworks cannot be directly adapted to maintainability verification. Verification queries exhibit highly diverse semantic expressions, and fixed retrieval pipelines tuned for a single query category degrade when processing other query variants. Complete verification also relies on integrated evidence from dispersed design artifacts, yet a single-stage retrieval pipeline fails to gather all categories of supporting evidence [14,15]. Furthermore, engineering compliance mandates traceable audit records, while existing RAG pipelines lack hierarchical structured traceability connecting reasoning outcomes, supporting evidence, and source standard clauses.

To tackle the above limitations, we develop a traceable multi-round RAG framework with classified routing for maintainability verification. This method formalizes a dedicated information model that incorporates two-dimensional requirement classification, standardized entity types and a state-machine verification workflow. Natural language inputs are routed to matched retrieval pipelines with adaptive sparse-dense weight distributions through the built-in classification module. A confidence-gated multi-round retrieval unit then resolves complex queries over two retrieval cycles and generates an end-to-end traceability chain associating all reasoning outputs with relevant source standard clauses.

2 Related Work

2.1 Maintainability Verification Methods

Maintainability verification has evolved from physical testing toward hybrid approaches combining simulation, knowledge engineering, and data-driven methods. Early standards such as MIL-STD-470 and MIL-STD-471A established systematic approaches to maintainability program management and verification, later consolidated into MIL-HDBK-470A [2–4]. MIL-HDBK-470A now provides lifecycle-oriented guidance for integrating maintainability requirements, design activities, verification, and management into the systems engineering process. Approaches based on Knowledge Graphs (KGs) have emerged for multi-source information integration in maintainability engineering [16]. Zhang et al. [17] developed a Bridge Maintenance Knowledge Graph for systematic knowledge management, while Yue et al. [18] applied KG to aircraft maintenance decision analysis. Xia et al. [19] augmented KGs with attention-based graph convolutional networks to enable predictive reasoning and interpretable fault explanation. However, existing KG approaches rely on predefined graph structures and lack the flexible retrieval and reasoning capabilities of LLMs for unstructured verification evidence.

2.2 Retrieval-Augmented Generation

RAG has emerged as a powerful paradigm for grounding LLM outputs in external knowledge. The naive RAG architecture retrieves relevant chunks for a query and conditions generation on both the query and the retrieved context, significantly reducing hallucination vs. closed-book generation on domain-specific tasks [9]. The architecture has evolved rapidly. RETRO [20] describes a retrieval-enhanced transformer that conditions auto-regressive generation on retrieved document chunks via chunked cross-attention. Self-RAG [21] introduces self-reflection mechanisms that enable the model to decide when retrieval is needed and to critique its own outputs. CRAG [22] adds a corrective retrieval loop that evaluates retrieved-document quality and triggers corrective retrieval when it is inadequate. Graph RAG [23] augments vector-based retrieval with knowledge graph traversal for multi-hop reasoning. More recently, the field has shifted toward agentic RAG, in which LLM-based agents plan multi-step retrieval strategies, invoke tools, and iteratively

refine their outputs. Comprehensive surveys by Singh et al. [24] and Mishra et al. [25] formally characterize this paradigm, with modeling agentic RAG as a partially observable Markov decision process.

2.3 Traceability in Artificial Intelligence-Assisted Engineering

Traceable Artificial Intelligence is essential in safety-critical engineering domains, where black-box decisions without evidence trails are procedurally unacceptable [26]. Masoudifard et al. [27] demonstrated auditable LLM-based compliance checking via Graph-RAG with explicit judgment-to-requirement linking. Karakurt and Akbulut [28] identified traceability and audit compliance as critical requirements for industrial RAG deployment. The present work extends these efforts in two directions. First, we integrate traceability not as a post-hoc annotation but as a structural property of the verification framework. Second, we ground traceability in a formal verification information model, so that each link in the chain, from evidence chunk to indicator to standard clause, corresponds to a semantically defined entity relationship rather than an ad hoc link.

3 Methodology

3.1 Maintainability Verification Information Model

Maintainability verification is performed progressively throughout system and equipment development to validate and update maintainability predictions, identify design deficiencies, and provide increasing assurance that the specified maintainability requirements can be achieved [2]. This process aligns with the V-model of Model-Based Reliability Systems Engineering (MBRSE) [29]: requirements are decomposed along the left arm from the system level to the component level, whereas verification activities proceed up the right arm. Horizontal traceability links connect the two arms, establishing requirement-to-verification loops (Fig. 1).

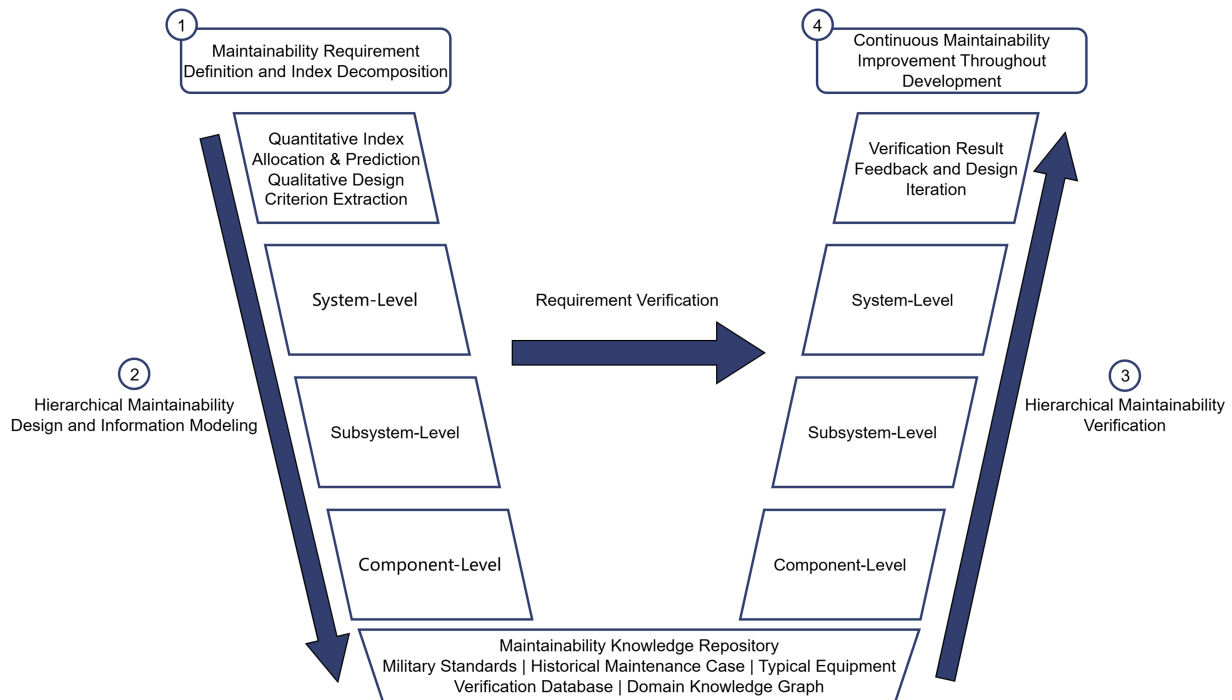


Figure 1: Maintainability verification in the product development process.

3.1.1 Two-Dimensional Classification of Qualitative Requirements

Qualitative maintainability requirements span six criteria categories: simplification, accessibility, standardization and interchangeability, error-proofing, maintenance safety, and ergonomics [30]. Their textual expressions are classified into four query types by a fine-tuned DistilBERT classifier [31]:

$$\text{Classify}(q) = (\tau, \gamma), \quad \tau \in \{Q_1, Q_2, Q_3, Q_4\}, \quad \gamma \in \{C_1, \dots, C_6\}. \quad (1)$$

where τ denotes the query type and γ denotes the criteria category. The four query types are derived from the linguistic structure of maintainability requirements.

Table 1 illustrates representative mappings between maintainability criteria and query types, which motivates our classified routing mechanism. Numeric and existence queries account for the majority of accessibility and standardization checks, while inference and fuzzy queries prevail in simplification and error-proofing evaluations.

Table 1: Examples of correspondence between maintainability requirements and query types.

Category	Q ₁ : Numeric	Q ₂ : Fuzzy	Q ₃ : Inference	Q ₄ : Existence
Simplification	Fastener count ≤ 4	Is the assembly easy to disassemble?	IF component is critical, THEN redundancy required	Is there a quick-release mechanism?
Accessibility	Access passage $\geq 200 \text{ mm} \times$ 200 mm	Is the component easy to reach?	IF component is behind panel, THEN panel must be removable	Is there a maintenance hatch?
Standardization	Standard part ratio $\geq 75\%$	Is it “Preferred”?	IF part is non-standard, THEN provide adapter	Is the part a standard component?
Error-proofing	Error rate $\leq 5\%$	Is error-proof design adopted?	IF user error occurs, THEN system must alert	Is there an error-proofing mechanism?
Safety	Voltage $\leq 36 \text{ V}$	Is it safe to operate?	IF voltage exceeds limit, THEN isolation required	Is there a safety interlock?
Ergonomics	Weight $\leq 10 \text{ kg}$	Is it comfortable to handle?	IF task is repetitive, THEN ergonomic support required	Is there ergonomic support?

3.1.2 Entity Types and State-Machine Workflow

The information model defines three entity types. A **Verification Object** $o \in \mathcal{O}$ is the product entity under inspection, characterized by product level, unique identifier, and design state. A **Verification Indicator** $i \in \mathcal{I}$ is a requirement item anchored to its source via:

$$\text{Anchor}(i) = \langle \text{source_doc}, \text{clause_id}, \text{requirement_type}, \text{product_level} \rangle \quad (2)$$

This anchor builds bidirectional trace links connecting each indicator to its source standard clause, forming the bottom two tiers of the four-layer traceability chain (Section 3.4). We define **Verification Evidence** $e \in \mathcal{E}$ as all supporting data sorted into structured, semi-structured, and unstructured resources [26]. Dedicated extraction workflows are deployed for each category, enabling a single indicator to retrieve evidence from all three groups.

For a given (o, i) pair, the entire verification procedure follows a predefined finite state machine. This machine involves four distinct states and three operational actions, namely $\Sigma = \{\text{Extract}, \text{Associate}, \text{Judge}\}$. The initial unverified state $s_0 = \langle o, i, \emptyset, \emptyset \rangle$ can transition to three subsequent states through different operations: the **Extract** action leads to the evidence-collected state s_1 , the **Associate** action yields the evidence-mapped state s_2 , and the **Judge** action produces the final judgment state s_3 . The complete transition rules are formally expressed as follows:

$$\begin{aligned}\delta(s_0, \text{Extract}) &= s_1 \\ \delta(s_1, \text{Associate}) &= s_2 \\ \delta(s_2, \text{Judge}) &= s_3\end{aligned}\tag{3}$$

A backoff mechanism reverts to s_0 with an information gap flag if *Extract* or *Associate* fails, preventing forced judgments on incomplete evidence. The verification function maps the final state to a ternary judgment:

$$V(o, i, E) \rightarrow \{\text{compliant}, \text{non-compliant}, \text{insufficient_evidence}\}\tag{4}$$

To guarantee reliable verification, the knowledge base must meet information completeness requirements. Let $M: \mathcal{I} \rightarrow 2^{\mathcal{C}}$ map each indicator i to its corresponding chunks $\mathcal{C}$. To ensure verifiability, the base must hold at least one chunk per required evidence type. This completeness constraint is defined as:

$$\forall j \in \{1, \dots, m\}, \exists c \in M(i): \text{type}(c) = E_j\tag{5}$$

where $M(i)$ denotes the set of chunks associated with indicator i , and E_j represents the j -th expected evidence type. Rather than directly triggering a verification state, this completeness condition is quantified by R_{evidence} (Eq. (8)) and incorporated into the composite confidence score C_{judge} , which governs whether the query is escalated to Round 2.

3.2 Classified Routing Mechanism

A classification routing module is designed to route each query to a retrieval pipeline with a specific type of weight distribution, combining BM25 sparse retrieval and dense semantic retrieval [32,33]. Table 2 lists the assigned weight values for all query categories. For numeric queries Q_1 , BM25 takes priority with a weight of 0.6 to support precise term matching; fuzzy queries Q_2 rely primarily on dense retrieval at 0.7; inference-oriented Q_3 adopts balanced weights of 0.5 for both retrievers; existence-check queries Q_4 are dominated by BM25 at 0.7. These weights are tuned as described in Section 5.2.

Retrieval results are augmented with entity linking [34] from a domain entity vocabulary (Table 3) spanning three categories: E_1 component entities, E_2 requirement entities, and E_3 evidence entities. Matching is case-insensitive and combines exact and partial matching against this vocabulary, which is curated from the standards and design documents in the corpus; overlapping entity spans are resolved by selecting the longest match. Recognized entities confer an additive confidence bonus $\Delta_e = 0.1$ (applied once if any domain

entity is matched). The final relevance score is computed by fusing BM25 and dense cosine similarity scores, augmented by the entity bonus:

$$S(q, c) = w_1(\tau) \cdot s_{\text{BM25}}(q, c) + w_2(\tau) \cdot s_{\text{dense}}(q, c) + \Delta_e \cdot \mathbb{I}_{\text{entity}}(q, c) \quad (6)$$

where $w_1(\tau) + w_2(\tau) = 1$, $\mathbb{I}_{\text{entity}}(q, c) = 1$ if chunk c contains the domain entities recognized in query q , and 0 otherwise. Before weighted fusion, both raw retrieval scores are scaled into the range $[0, 1]$ via min-max normalization.

Table 2: Retriever weight distributions by query type.

Query Type	Linguistic Basis	w_1 (BM25 Weight)	w_2 (Dense Weight)
Q_1 (Numeric)	Exact values with comparison operators	0.6	0.4
Q_2 (Fuzzy)	Vague qualitative descriptors	0.3	0.7
Q_3 (Inference)	IF-THEN conditional logic	0.5	0.5
Q_4 (Existence)	Property presence or absence	0.7	0.3

Table 3: Domain-specific entity vocabulary.

Type	Entity
E_1 (Component)	Fuel filter line replaceable unit (LRU), Cabin door hinge assembly, Fastener $M8 \times 20$
E_2 (Requirement)	“Simplification of function”, “Passage opening dimension”, “Standard part ratio”
E_3 (Evidence)	BOM table, DMU annotation, Maintenance manual

3.3 Multi-Round Iterative Retrieval

Inspired by Adaptive Iterative Retrieval for Retrieval-Augmented Generation (AIR-RAG) [35], we propose a multi-round mechanism that conducts a three-stage confidence-gated process with up to two retrieval rounds, as depicted in Fig. 2. Round 1 retrieves and generates a preliminary judgment. If confidence C_{judge} falls below threshold θ (or the evidence-completeness requirement is violated), Round 2 decomposes the query into a small number of atomic sub-queries with type-specific strategies. Each sub-query is retrieved independently with $k = 10$. The Round-2 retrieved chunk sets are then merged and de-duplicated, re-ranked according to the fused relevance score, and truncated to the final top-10 chunks before the final judgment. Round 3 generates a structured gap report enumerating missing evidence types and likely document sources.

3.3.1 LLM Judge Confidence Estimation

The Round-1 judge produces a composite confidence score:

$$C_{\text{judge}} = \alpha S_{\text{LLM-self}} + (1 - \alpha) R_{\text{evidence}} \quad (7)$$

where $S_{\text{LLM-self}} \in [0, 1]$ denotes the judge’s verbalized confidence in the current verification result. For each judgment, the LLM directly reports this score based on the adequacy and consistency of the retrieved evidence, explicitly avoiding token probabilities or repeated sampling. $R_{\text{evidence}} \in [0, 1]$ measures the coverage of the expected evidence types and is defined as:

$$R_{\text{evidence}} = \frac{|\{\text{type}(c) : c \in \mathcal{R}(q)\} \cap \mathcal{T}(q)|}{|\mathcal{T}(q)|} \quad (8)$$

where $\mathcal{R}(q)$ denotes the set of top- k retrieved chunks for query q , and $\mathcal{T}(q)$ denotes the set of evidence types expected for its query category, as summarized in [Table 4](#).

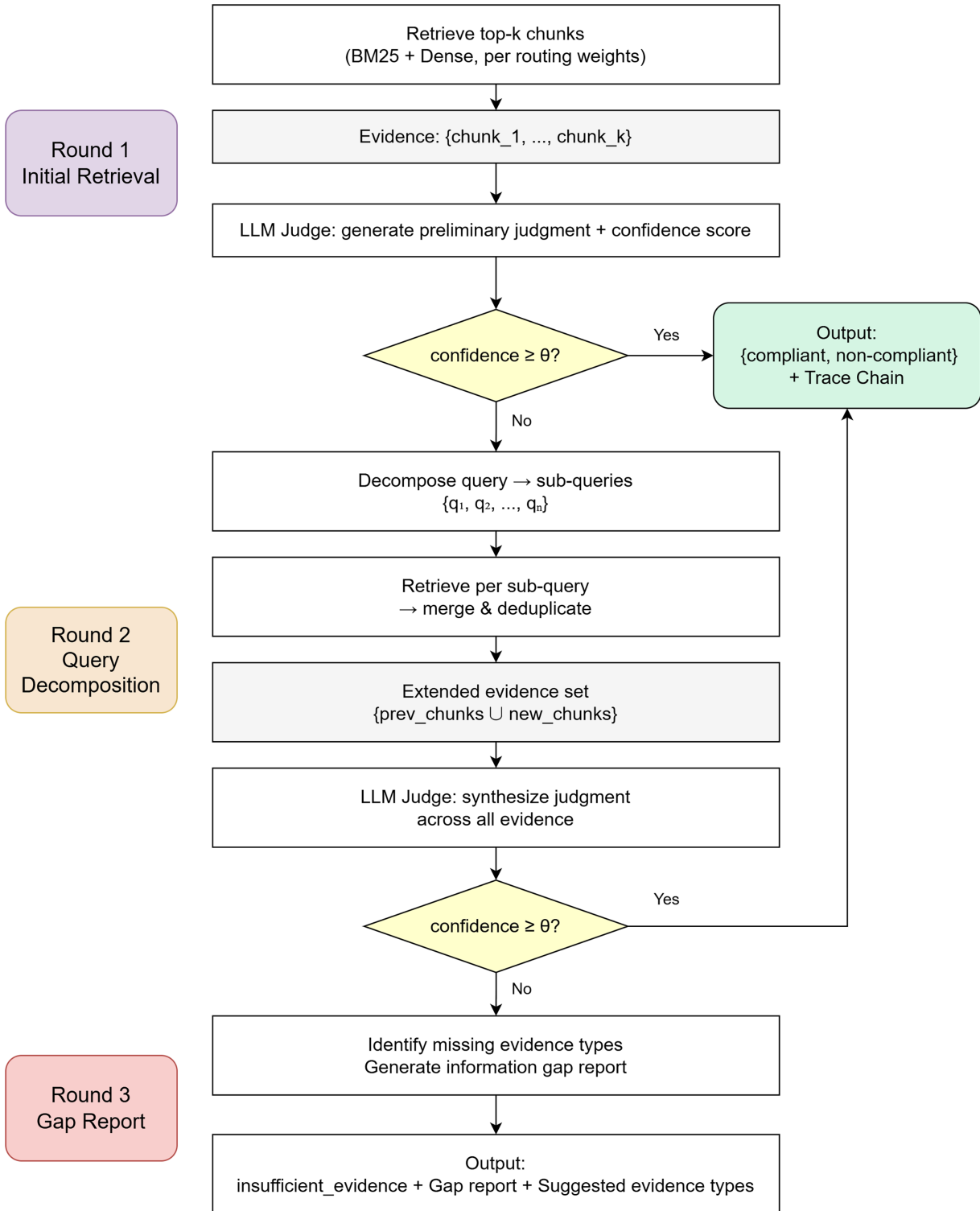


Figure 2: Multi-round retrieval overall procedure.

Table 4: Default evidence requirements by query category.

Query Type	Expected Evidence Types
Q_1 (Numeric)	Quantitative evidence + Requirement clause
Q_2 (Fuzzy)	Descriptive evidence + Requirement clause
Q_3 (Inference)	Rule/procedural evidence + State/context evidence + Requirement clause
Q_4 (Existence)	Entity/component evidence + Requirement clause

The composite confidence score directly governs the escalation decision. Specifically, if $C_{\text{judge}} \geq \theta$, the Round-1 judgment is accepted; otherwise, the query is escalated to Round 2 for supplementary retrieval. Because R_{evidence} is explicitly incorporated into C_{judge} , incomplete evidence coverage lowers the composite confidence and thereby increases the likelihood of escalation without introducing an additional independent decision rule. The selected combination was $\alpha = 0.4$ and $\theta = 0.7$, tuned as described in [Section 5.2](#).

3.3.2 LLM Judge Implementation

The LLM judge adopts the DashScope API `qwen-plus` with zero temperature, greedy decoding, and a maximum output length of 2048 tokens to minimize decoding stochasticity. One response is generated per judge invocation, with no repeated sampling or self-consistency voting. The system prompt defines the model role as a maintainability verification expert and clarifies the ternary judgement schema (compliant, non-compliant, insufficient_evidence). The user prompt template integrates the verification query, retrieved evidence chunks (each tagged with a unique chunk ID and source document), and standardized output format. The judge is constrained to generate a structured JSON object.

Judgment adheres to evidence-grounded verification criteria. A query is labeled `compliant` if retrieved evidence sufficiently supports the requirement claim, `non-compliant` if the retrieved evidence contradicts the requirement claim or demonstrates a design condition that violates the requirement, and `insufficient_evidence` if available evidence is incomplete or irrelevant for valid judgment. The LLM judge follows fixed prompt-defined decision rules, evidence usage constraints, and output specifications. Full prompt templates are provided in [Appendix A](#).

3.4 Traceability Chain

The framework establishes a four-level traceability chain linking each compliance judgment to its originating standard clause, serialized as structured JSON. As shown in [Fig. 3](#), the hierarchy consists of: (1) the LLM judgment with reasoning and evidence citations, (2) retrieved evidence chunks identified by unique ID and source document, (3) the verification indicator with Anchor metadata, and (4) the source standard clause. This navigable data structure operationalizes the V-model horizontal traceability link. It supports programmatic auditing via backward chain traversal and provides actionable design feedback by pinpointing the specific evidence and standard clause responsible for a verification failure [27].

3.5 End-to-End Pipeline Integration

[Fig. 4](#) illustrates the complete pipeline architecture. A verification query passes through the classifier for type determination, proceeds to the type-specific retrieval pipeline, enters the multi-round iterator for potential decomposition, and reaches the LLM judge for ternary judgment generation with cited evidence. The object-indicator-independent pipeline enables parallel batch verification across hundreds of checklist items.

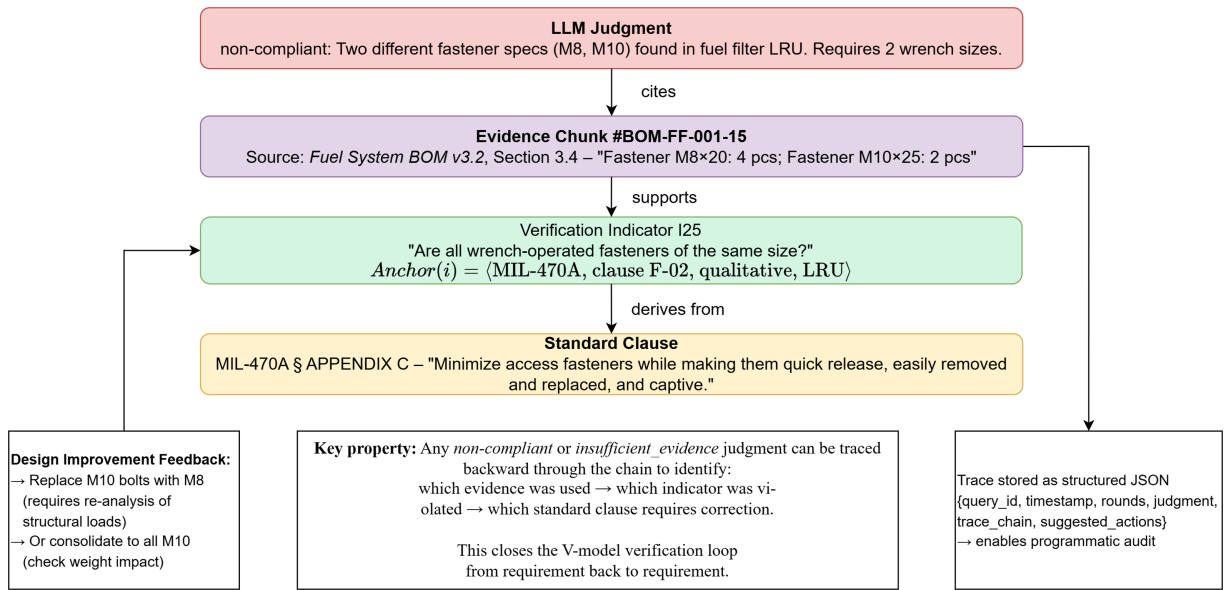


Figure 3: Traceability chain structure.

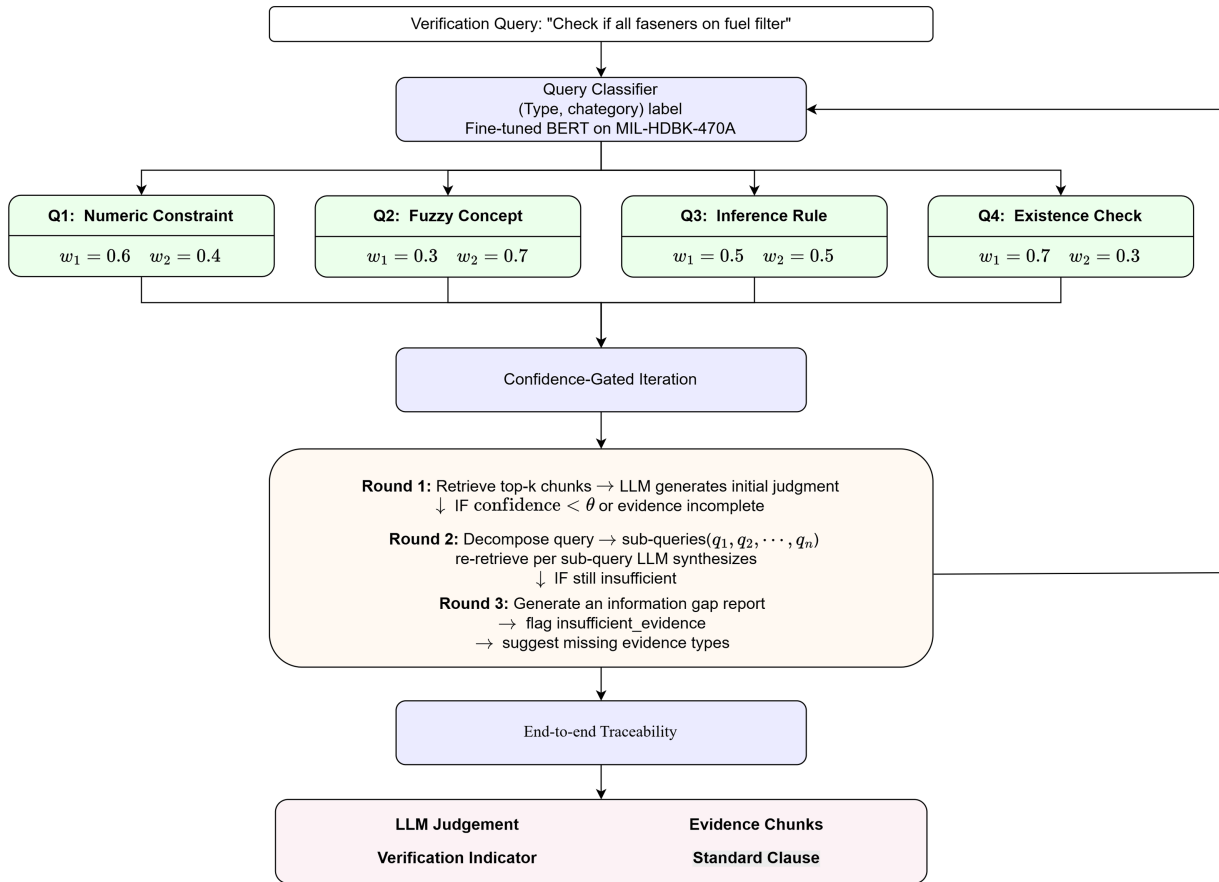


Figure 4: End-to-end pipeline architecture.

4 Experimental Setup

The experimental evaluation answers three research questions concerning routing classification performance, multi-round retrieval gains, and traceability capacity.

4.1 Retrieval Corpus

Our retrieval corpus contains 2118 text chunks generated via a 512-character sliding window with 50-character overlap. The corpus comprises 2024 Gold-tier authoritative excerpts from M109 maintenance logs, D-6A test stand manuals, and MIL-HDBK-470A specifications, together with 54 Silver-tier domain-specific distractors and 40 Bronze-tier generic engineering noise fragments. All experiments retrieve against the complete 2118-chunk corpus rather than a query-specific candidate subset.

The evaluation benchmark includes 380 test queries: 300 simple cases and 80 complex queries requiring sub-query decomposition. Each query is annotated with 10–15 gold evidence chunk IDs and 10–20 query-specific hard-negative chunk IDs, spanning five interference categories for diagnostic analysis: mismatched numeric constraints, duplicate cross-vehicle component names, generic engineering rules, synthetically rephrased safety standards with modified thresholds, and temporally out-of-scope M109 Paladin design documents. Synthetic and counterfactual distractors are excluded from validation and hyperparameter-tuning procedures. A separate 100-query validation set, disjoint from the test benchmark, is reserved exclusively for hyperparameter tuning.

4.2 Baseline Settings

We evaluate against two tiers of baselines, as summarized in Table 5. These include classical single-retriever methods and adaptive/iterative RAG approaches. Self-RAG and CRAG are domain-adapted reproductions following their original methods, with the retrieval corpus, top- k budget, and LLM backend controlled for a fair comparison rather than their raw official configurations. Ablation studies quantify per-component contributions.

Table 5: Summary of baseline approaches and implementation details.

Category	Method	Core Mechanism	Implementation Details
Single-Retriever	BM25-only	Sparse term-frequency retrieval	Okapi BM25, $k_1 = 1.2$, $b = 0.75$
	Dense-only	Dense semantic retrieval	text-embedding-v3 (1024d), FAISS Index
	ColBERT-v2	Late-interaction retrieval	Pre-trained ColBERT-v2 checkpoint, no fine-tuning
Adaptive/Iterative RAG	Self-RAG	Self-reflective adaptive retrieval	Domain-adapted reproduction
	CRAG	Corrective retrieval loop	Domain-adapted reproduction
	Ours	Classified routing RAG with multi-round iterative retrieval	Query classifier routes to type-specific retriever weights, multi-round decomposition and gap reporting

4.3 Evaluation Metrics

Retrieval metrics: Precision@10 (P@10), Recall@10 (R@10), and F1@10 are calculated against annotated gold evidence chunk IDs over the complete 2,118-chunk corpus. Context Relevancy (CR) is evaluated using the RAGAS framework [36]. Routing Accuracy (RA) counts the share of queries assigned to their correct type category.

Generation metrics: Answer Correctness (AC, a 0–1 score measuring semantic consistency with ground-truth reference answers) and Answer Faithfulness (AF, the ratio of output claims supported by retrieved context), both evaluated using RAGAS.

Efficiency metric: Average Retrieval Latency, which records end-to-end runtime from input query to final verification judgment.

Domain-Specific Metrics: Traceability Completeness (TC) quantifies the proportion of queries with valid non-empty references across all four hierarchical layers (L0–L3), defined as:

$$TC = \frac{1}{N} \sum_{i=1}^N \prod_{l=0}^3 \mathbb{I}[\text{trace_level}_l(q) \neq \emptyset] \quad (9)$$

Here, N denotes the total number of test queries, and $\mathbb{I}[\cdot]$ is the indicator function. The product enforces that all four hierarchical layers contain valid non-empty references.

Traceability Attribution Accuracy (TAA) further measures the proportion of cited evidence chunks that match gold evidence annotations:

$$TAA = \frac{\sum_{i=1}^{N_{\text{complete}}} |\mathcal{C}_i^{\text{cite}} \cap \mathcal{G}_i|}{\sum_{i=1}^{N_{\text{complete}}} |\mathcal{C}_i^{\text{cite}}|} \quad (10)$$

where N_{complete} denotes the number of fully traced queries, $\mathcal{C}_i^{\text{cite}}$ the set of chunks cited for query q_i , and $\mathcal{G}_i$ its gold evidence chunk set. TAA excludes superficial keyword overlaps by strictly validating chunk-level correspondence with annotated evidence.

Statistical Analysis: All reported metrics are averaged over three independent runs with standard deviations below 0.03 for every metric. To examine statistical differences, we conduct paired t -tests over per-query paired observations ($n = 300$ for simple queries, $n = 80$ for complex queries) with Bonferroni correction at a significance level of 0.05. The Bonferroni correction accounts for five pairwise comparisons.

4.4 Implementation Details

LLM components use the DashScope API `qwen-plus`, and embeddings use `text-embedding-v3` (1024d). To ensure fair latency comparisons, all API-based experiments were executed from the same client machine, endpoint, and region within a consistent evaluation window. Local computations were performed on Ubuntu 22.04 (Intel Core i7-12700, 32 GB RAM, NVIDIA GeForce RTX 3060 12 GB). All baselines were evaluated on the identical corpus, data splits, chunking strategy, top- k retrieval budget ($k = 10$), and LLM backend, with each method's retriever and embedding encoder as specified in Table 5. The top- k budget ($k = 10$) applies to each individual retrieval call. Hyperparameter tuning for all methods, including our two-stage search and the domain-specific baseline grids, is detailed in Section 5.2; for Self-RAG and CRAG, both default and domain-tuned configurations are reported. Implementation details of the LLM judge are provided in Section 3.3.2.

5 Results

5.1 Simple Queries Comparison

Table 6 summarizes the performance on 300 simple queries. Our method achieves the strongest retrieval effectiveness, with the highest P@10 (0.864), R@10 (0.651), F1@10 (0.743), AC (0.631), and CR (0.852). Although CRAG obtains a slightly higher AF score (0.948), its lower F1@10 (0.716) indicates that generation-level correction alone is insufficient to address retrieval incompleteness.

Table 6: Performance comparison on simple queries (300 queries).

Method	P@10	R@10	F1@10	AC	CR	AF	Latency
BM25-only	0.845	0.568	0.679	0.576	0.805	0.852	5.2 s
Dense-only	0.812	0.513	0.630	0.541	0.837	0.894	4.8 s
ColBERT-v2	0.838	0.559	0.671	0.558	0.821	0.907	6.9 s
Self-RAG	0.821	0.582	0.682	0.549	0.798	0.803	11.4 s
CRAG	0.856	0.614	0.716	0.604	0.842	0.948	7.6 s
Ours	0.864	0.651	0.743	0.631	0.852	0.937	6.3 s

Our AC improvement over the default CRAG configuration is statistically significant under the Bonferroni-corrected paired t -test ($p < 0.05$). The benchmark's high P@10 and relatively low R@10 stems from its multi-evidence setting: each query corresponds to 10–15 gold evidence chunks, while retrieval is limited to top-10 results. Models can thus retrieve predominantly relevant chunks without covering the full evidence set.

Results demonstrate the complementary strengths of distinct retrieval paradigms. BM25-only achieves high precision (0.845) via accurate matching of standardized engineering terms and numerical constraints but yields low recall (0.568) on terminologically diverse evidence. Dense-only retrieval improves semantic coverage yet introduces irrelevant candidates, degrading precision to 0.812. These findings validate the necessity of our adaptive retrieval selection, which dynamically combines sparse and dense retrieval based on query characteristics.

Our method maintains a moderate latency of 6.3 s, outperforming Self-RAG and CRAG while delivering superior retrieval effectiveness. This demonstrates that the proposed routing and verification mechanisms improve retrieval quality without introducing substantial computational overhead.

5.2 Hyperparameter Tuning

All methods are tuned exclusively on a 100-query validation set disjoint from the 300-query test set for fair comparison. Given distinct hyperparameter structures across methods, we adopt method-specific search spaces instead of uniform search-space sizes. Our two-stage tuning pipeline involves 71 validation evaluations in total. First, type-specific routing weights are optimized for all four query types by sweeping w_1 over $\{0, 0.1, \dots, 1.0\}$ with $w_2 = 1 - w_1$, producing 44 F1@10-optimized configurations. Second, the confidence parameters α and θ are jointly tuned over $\alpha \in \{0.3, 0.4, 0.5\}$ and $\theta \in \{0.50, 0.55, \dots, 0.90\}$, yielding 27 additional configurations optimized by escalation-prediction F1, as these parameters control Round-2 triggering. For baseline tuning, Self-RAG is evaluated across 48 configurations covering varying retrieval thresholds, critique thresholds, and critique passes, while CRAG is tested over 24 configurations with different confidence thresholds, correction rounds, and correction strategies. All optimal configurations

are selected via maximum F1@10 on the shared validation set, ensuring full domain-specific optimization and eliminating unfair default-tuned evaluation discrepancies.

Table 7 reports default and tuned results on the 300-query test set. Domain-specific tuning improves both baselines, with Self-RAG improving F1@10 from 0.682 to 0.705 and CRAG improving from 0.716 to 0.733. After tuning, our method achieves slightly higher F1@10 (0.743 vs. 0.733) and AC (0.631 vs. 0.618) than CRAG, while CRAG maintains the highest AF (0.953).

Table 7: Baseline performance with default vs. domain-tuned configurations on simple queries (300 queries).

Method	F1@10 (Default)	F1@10 (Tuned)	AC (Default)	AC (Tuned)	AF (Default)	AF (Tuned)
Self-RAG	0.682	0.705	0.549	0.574	0.803	0.799
CRAG	0.716	0.733	0.604	0.618	0.948	0.953
Ours	—	0.743	—	0.631	—	0.937

5.3 Effectiveness of Classified Routing

On a balanced 240-query subset of simple queries (60 per type), our approach achieves the highest per-type AC on all four categories, as shown in Table 8 and Fig. 5.

Table 8: Performance comparison on query types (240 balanced simple queries).

Type	BM25	Dense	ColBERT	Self-RAG	CRAG	Ours
Q ₁ : Numeric	0.750	0.567	0.733	0.717	0.750	0.783
Q ₂ : Fuzzy	0.517	0.567	0.533	0.533	0.583	0.600
Q ₃ : Inference	0.417	0.433	0.400	0.383	0.450	0.467
Q ₄ : Existence	0.600	0.533	0.567	0.517	0.583	0.650

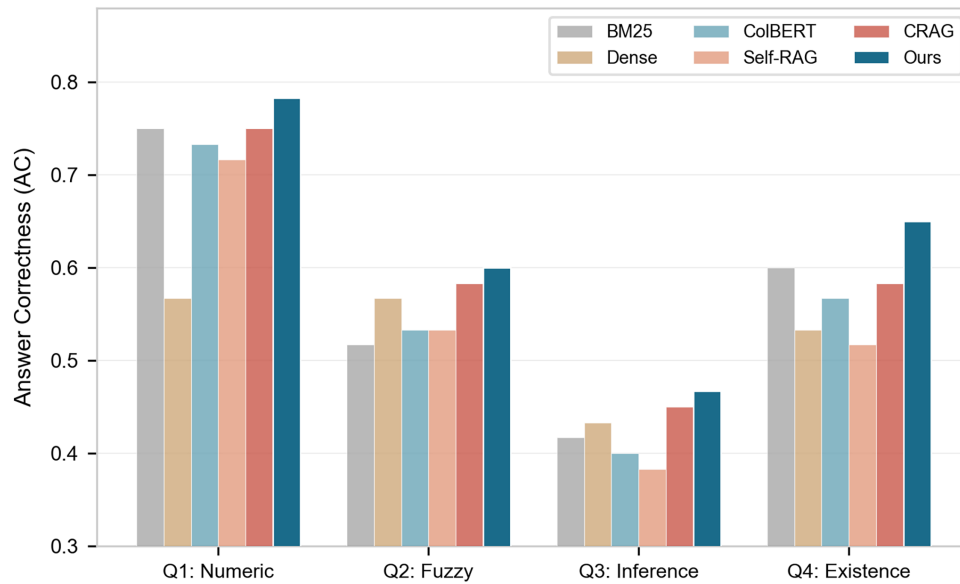


Figure 5: Answer correctness by query type (240 balanced simple queries, 60 per type).

The per-type performance distribution reveals reversed effectiveness between BM25 and dense retrievers under different linguistic structures. For numeric queries Q_1 , BM25 surpasses dense retrieval by 0.183, whereas dense retrieval outperforms BM25 by 0.050 on fuzzy Q_2 . Our type-adaptive routing strategy combines these two complementary strengths: it improves upon BM25 on Q_1 (0.783 vs. 0.750) and exceeds the dense retriever’s optimal Q_2 score (0.600 vs. 0.567).

The most prominent performance gap appears on existence queries Q_4 , where our method reaches 0.650 against CRAG’s 0.583. This improvement stems from the keyword-biased weight configuration (BM25 weight set to 0.7), which accurately fulfills entity positioning verification demands. All methods hit a unified performance bottleneck on inference-oriented Q_3 , with scores confined within the range 0.383–0.467. The implicit conditional logic embedded in procedural maintenance manuals poses intrinsic challenges for chunk-level retrieval, which inspires us to explore structured IF-THEN rule extraction as a future research direction. Overall, consistent performance gains over Q_1 , Q_2 and Q_4 validate that classified routing brings structural performance gains that cannot be achieved via simple weight tuning alone.

5.3.1 Query Classification Accuracy

The classifier is fine-tuned from a pre-trained DistilBERT checkpoint [31] on a labeled benchmark split. Only the predicted query type τ is used for routing; the criteria category γ is excluded in the current evaluation. As shown in Table 9, the classifier achieves an overall Routing Accuracy (RA) of 0.863 and a macro-averaged F1 of 0.850 on the 380-query benchmark. Q_1 and Q_4 yield the highest F1-scores (0.925, 0.890) due to prominent lexical features like numeric operators and fixed interrogative structures.

Table 9: Query classifier performance by query type.

Type	Precision	Recall	F1-Score	Support
Q_1 (Numeric)	0.917	0.933	0.925	120
Q_2 (Fuzzy)	0.821	0.800	0.810	80
Q_3 (Inference)	0.786	0.767	0.776	60
Q_4 (Existence)	0.897	0.883	0.890	120
Macro Avg.	0.855	0.846	0.850	380

Fig. 6 details the error distribution across query types. Q_3 achieves the lowest F1 (0.776), with over 55% of its misclassifications occurring between Q_3 and Q_2 , as both use complex multi-clause expressions that obscure conditional markers. Nevertheless, the reliable type guidance (RA = 0.863) and the weight similarity between adjacent types Q_2 and Q_3 (Table 2) effectively mitigate residual misclassification impacts.

Having established both the routing effectiveness and the classifier accuracy that enables it, we next examine whether multi-round iteration adds further benefit, particularly for queries requiring evidence synthesis across multiple documents.

5.4 Multi-Round Retrieval on Complex Queries

On the set of 80 complex verification queries (Table 10), our approach achieves the most significant recall improvement, reaching 0.604 compared with CRAG’s 0.478 (+0.126). This performance boost originates from the Round-2 decomposition module, which splits compound input queries into fine-grained atomic sub-queries that precisely target specific evidence categories.

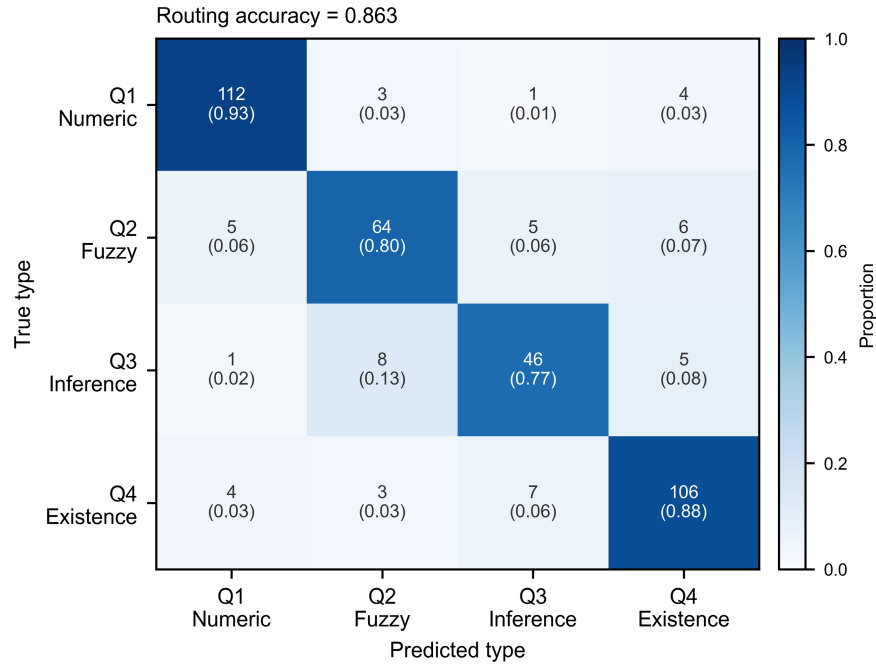


Figure 6: Confusion matrix of the fine-tuned query classifier on the 380-query benchmark.

Table 10: Performance comparison on complex queries (80 queries).

Method	P@10	R@10	F1@10	AC	CR	AF	Latency
BM25-only	0.738	0.446	0.556	0.462	0.712	0.915	6.8 s
Dense-only	0.721	0.352	0.473	0.475	0.735	0.871	5.4 s
ColBERT-v2	0.695	0.428	0.530	0.455	0.708	0.855	8.1 s
Self-RAG	0.762	0.462	0.575	0.443	0.748	0.795	12.6 s
CRAG	0.812	0.478	0.602	0.502	0.772	0.945	8.5 s
Ours	0.815	0.604	0.694	0.553	0.835	0.948	9.2 s

Table 11 summarizes the per-round performance progression on complex queries. The R2 row reports metrics after all retrieval operations (Round 1 + Round 2 supplementary retrieval); Round 3 performs gap reporting without additional retrieval and thus does not alter retrieval-based metrics. Round-2 query decomposition substantially boosts Recall (+0.142), F1 (+0.100), and AC (+0.061), with only a minor Precision drop (−0.016).

Table 11: Per-round gain on complex queries.

Round	P	R	F1	AC	Latency
R1 (Direct Retrieval)	0.831	0.462	0.594	0.492	5.1 s
R2 (+Decomposition)	0.815	0.604	0.694	0.553	9.2 s
Gain (R1 → R2)	−0.016	+0.142	+0.100	+0.061	+4.1 s

The confidence-gated architecture resolves most queries efficiently while providing a principled escalation path. In our experiments, 72.4% of queries are resolved in Round 1, 21.3% in Round 2, and 6.3% generate Round 3 gap reports. Round 3 outputs `insufficient_evidence`, scored as correct when the knowledge base genuinely lacks required information (Eq. (4)). Self-RAG degrades on complex queries (AC = 0.443 vs. BM25's 0.462) due to semantic drift in domain-specific reformulations.

5.5 Traceability Performance

Our pipeline supports four-layer auditable trace chains absent from all baselines. Traceability Completeness (TC = 0.968, $N = 380$) indicates 96.8% of queries yield fully populated, content-validated trace hierarchies. The 3.2% incomplete instances occur when our framework refuses to generate unsupported links rather than fabricate evidence. Traceability Attribution Accuracy (TAA = 0.934, $N_{\text{complete}} = 368$) shows 93.4% of referenced chunks in fully-traced queries match gold evidence, avoiding superficial keyword overlaps. Together, TC and TAA verify the audit capability: the system only records traceable conclusions backed by retrieved materials, abstaining from unsubstantiated judgments. This converts opaque black-box outputs into transparent workflows supporting independent manual auditing.

5.6 LLM Judge Reliability

To validate the reliability of the LLM judge that controls both ternary classification and multi-round escalation, we conducted a human evaluation on a stratified sample of 70 queries (20 Q_1 , 15 Q_2 , 15 Q_3 , 20 Q_4) drawn from the 380-query benchmark. Two domain experts independently assigned gold labels (`compliant`, `non-compliant`, `insufficient_evidence`) based on the same retrieved evidence provided to the LLM judge, using the unified `insufficient_evidence` definition from Eq. (5).

Inter-annotator agreement between the two human experts reached Cohen's $\kappa = 0.868$, establishing a realistic upper bound for the annotation task. The LLM judge achieved $\kappa = 0.779$ and $\kappa = 0.780$ against Human 1 and Human 2, respectively, with an overall accuracy of 0.857 and a macro-averaged F1 of approximately 0.847. Table 12 presents the confusion matrix against Human 1 as reference.

Table 12: Confusion matrix (Rows: Human 1; Columns: LLM judge) on 70 annotated queries.

	Compliant	Non-Compliant	Insufficient_Evidence
Compliant	28	1	2
Non-compliant	2	18	2
Insufficient_evidence	1	2	14

Per-class metrics (Table 13) reveal that `compliant` is the most reliably recognized category (F1 = 0.903), while `insufficient_evidence` exhibits the lowest precision (0.778). The dominant error pattern is the judge's tendency to emit a substantive judgment (`compliant` or `non-compliant`) when only partial evidence is available, rather than abstaining via `insufficient_evidence`. This systematic weakness directly motivates the confidence-gated escalation mechanism: by fusing the LLM's self-assessment with the objective evidence coverage ratio R_{evidence} (Eq. (8)), the framework compensates for the judge's imperfect calibration on evidence-incomplete queries. A secondary error pattern involves isolated design deficiencies in retrieved chunks triggering `non-compliant` judgments without full verification-object coverage, further supporting the need for evidence-type completeness checks before final judgment.

Table 13: Per-class performance of LLM judge (Human 1 as reference).

Class	Precision	Recall	F1
Compliant	0.903	0.903	0.903
Non-compliant	0.857	0.818	0.837
Insufficient_evidence	0.778	0.824	0.800

5.7 Ablation Study

Table 14 quantifies the contribution of the three core mechanisms on 120 stratified queries. Removing classified routing causes the largest AC decrease (0.073, 11.4%), followed by multi-round retrieval (0.027, 4.2%) and entity linking (0.014, 2.2%). The joint removal of classified routing and multi-round retrieval decreases AC by 0.086. Since this combined loss (0.086) is smaller than the sum of their individual losses (0.100), it indicates a partial overlap in their contributions: classified routing improves initial retrieval for all queries, whereas multi-round retrieval primarily aids the subset where type-optimized retrieval remains insufficient. This suggests a practical principle that investing in retrieval quality yields broader returns than investing in retrieval depth.

Table 14: Ablation study on core mechanisms (120 queries).

Variant	AC	F1	CR	AC Drop	Contribution
Full	0.642	0.735	0.845	—	—
w/o Classified Routing	0.569	0.651	0.782	−0.073	11.4%
w/o Multi-Round	0.615	0.710	0.825	−0.027	4.2%
w/o Entity Linking	0.628	0.724	0.838	−0.014	2.2%
w/o Routing & Multi-Round	0.556	0.624	0.760	−0.086	13.4%

Table 15 further shows that the three components benefit different query types. Classified routing has the largest effect on Q_4 and Q_1 , where exact entity, terminology, and numerical matching are important. Multi-round retrieval contributes primarily to Q_3 , with an AC drop of 0.071 when removed, indicating its importance for inference queries requiring evidence synthesis. Entity linking is mainly beneficial to Q_4 , where its removal decreases AC by 0.050 but has little effect on the other query types. These differences indicate that the modules provide complementary rather than uniform gains.

Table 15: Per-type ablation: answer correctness by query type (120 queries, 30 per type).

Variant	Q_1 Numeric	Q_2 Fuzzy	Q_3 Inference	Q_4 Existence	Overall
Full	0.781	0.621	0.492	0.674	0.642
w/o Classified Routing	0.721	0.584	0.461	0.510	0.569
w/o Multi-Round	0.777	0.610	0.421	0.652	0.615
w/o Entity Linking	0.780	0.619	0.489	0.624	0.628

6 Discussion

Combining type-aware routing, multi-round retrieval, and traceability, the framework improves verification performance while preserving evidence links required for engineering audit. Classified routing provides the largest AC gain (11.4%), indicating a uniform retrieval strategy is less effective across heterogeneous queries, whereas multi-round retrieval adds a smaller overall gain (4.2%) that mainly benefits complex cross-document cases: of the 105 queries that escalate beyond Round 1, 81 (77.1%) resolve in Round 2 and the remaining 24 (22.9%) proceed to Round 3 for unresolved evidence gaps.

Domain-specific tuning of Self-RAG and CRAG substantially narrows the performance gap: relative to CRAG, the F1@10 gap shrinks from 0.027 (default) to 0.010 (tuned) and the AC gap from 0.027 to 0.013, while CRAG retains the highest AF (0.953). Thus part of the initial gap reflected tuning effort, whereas the remainder reflects distinct strengths in retrieval, verification, and traceability. We note that our method's two-stage tuning involved more validation evaluations than the baselines (71 vs. 48 and 24), reflecting its larger set of tunable hyperparameters rather than a disproportionate optimization budget; this tuning is a one-time offline procedure that does not affect inference latency.

6.1 Positioning Relative to GraphRAG and Agentic RAG

Recent GraphRAG and agentic RAG methods provide effective mechanisms for multi-hop retrieval and adaptive reasoning. However, maintainability verification imposes different practical requirements. The source corpus consists mainly of unstructured and semi-structured engineering documents, so building and maintaining a comprehensive knowledge graph over such heterogeneous sources would introduce additional modeling and maintenance overhead. Our framework therefore uses lightweight entity linking with query-type-specific retrieval, avoiding dependence on a full graph representation.

Compared with agentic RAG, our method replaces open-ended retrieval planning with classified routing and confidence-gated multi-round retrieval. This design reduces planning variability and provides a more reproducible retrieval process, which is desirable for engineering verification. In addition, the traceability mechanism in [Section 3.4](#) explicitly links each verification result to its supporting evidence and originating standard clause, whereas GraphRAG and agentic RAG are not inherently designed to guarantee such traceability.

Thus, the use of a conventional RAG backbone is a deliberate design choice rather than a limitation. It provides a simpler and more controllable foundation for domain-specific routing, evidence traceability, and reproducible verification. Graph-based or agentic extensions remain compatible with this framework and may be explored in future work.

6.2 Limitations and Future Work

Several limitations should be acknowledged:

- **Static routing weights.** The retrieval weights are fixed for each query type after validation, and classification errors between fuzzy and inference queries may lead to suboptimal routing. Future work will investigate adaptive routing and query-dependent weight estimation.
- **Model dependence.** The current evaluation relies on a single LLM backend and an LLM-based judge. Although human evaluation shows substantial agreement with domain experts ($\kappa \approx 0.78$), lower precision is observed for `insufficient_evidence` cases. Evaluation with additional LLM backends and further judge calibration is therefore needed to assess model dependence.

- **Static knowledge base.** The current setup does not support document versioning, stale-result detection, or incremental re-verification. Incorporating these mechanisms would improve applicability to iterative engineering workflows where design artifacts are continuously updated.
- **Domain generalization.** The evaluation corpus is limited to military equipment maintenance documents. While the results demonstrate effectiveness within the evaluated setting, generalization to other engineering domains and regulatory specifications remains to be established. Future work will extend the evaluation across broader datasets, LLM backends, and application domains.

7 Conclusion

This paper presented a traceable multi-round RAG framework with classified routing for automated qualitative maintainability verification of complex equipment. The framework contributes a verification information model that formalizes two-dimensional requirement classification, entity types, and a state-machine workflow. It introduces a classified routing mechanism that leverages the linguistic structure of engineering standards to assign queries to type-specific retrieval pipelines. It further provides a confidence-gated multi-round retrieval mechanism with an end-to-end traceability chain that creates auditable decision trails from compliance judgment to originating standard clause.

Experimental evaluations confirm the framework's efficacy and robustness. Notably, even after applying comparable domain-specific tuning to competitive baselines like CRAG, our method retains a numerical advantage in AC (0.631 vs. 0.618), while ablation studies identify classified routing as the primary performance driver. The confidence-gated multi-round mechanism substantially improves evidence recall for complex queries. Crucially, unlike standard opaque RAG pipelines, the architecture generates structured four-layer auditable trace chains. Furthermore, human evaluation confirms substantial agreement between the LLM judge and domain experts, with observed calibration limitations directly motivating the integration of subjective confidence and objective evidence coverage.

Acknowledgement: Not applicable.

Funding Statement: The authors received no specific funding for this study.

Author Contributions: The authors confirm contribution to the paper as follows: Conceptualization, Xingxin Li; methodology, Jixin Xu, Xingxin Li; software, Jixin Xu; validation, Jixin Xu; formal analysis, Qingqing Hu; investigation, Qingqing Song; resources, Xingxin Li; data curation, Qingqing Hu; writing—original draft preparation, Jixin Xu; writing—review and editing, Qingqing Song, Jixin Xu; visualization, Jixin Xu; supervision, Xingxin Li; project administration, Xingxin Li. All authors reviewed and approved the final version of the manuscript.

Availability of Data and Materials: The data that support the findings of this study are available from the Corresponding Author, Xingxin Li, upon reasonable request.

Ethics Approval: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A LLM Judge Prompt Template

Table A1: Complete LLM judge prompt template.

You are a strict engineering verification judge specializing in equipment maintainability. Your task is to determine whether the retrieved evidence sufficiently verifies a given requirement claim.

Inputs:

1. Requirement Claim: The maintainability requirement to be verified.
2. Query Type: The linguistic category of the query (Numeric, Fuzzy, Inference, or Existence).
3. Expected Evidence Types: The mandatory categories of evidence required for this query type.
4. Retrieved Evidence Chunks: A list of evidence chunks formatted as:
[Chunk_ID]: Source_Doc - "Chunk_Content"

DECISION LABELS:

You must select exactly one of the following labels:

- compliant: The retrieved evidence fully covers all expected evidence types and explicitly supports the requirement claim.
- non-compliant: The retrieved evidence fully covers all expected evidence types but explicitly contradicts the claim or reveals a design deficiency.
- insufficient_evidence: One or more expected evidence types are missing, OR the retrieved evidence is ambiguous, incomplete, or irrelevant for a definitive compliant/non-compliant judgment.

JUDGMENT RULES:

1. Grounding: Use ONLY the provided retrieved evidence. Do not use external knowledge or make assumptions.
2. Type Coverage Evaluation: Explicitly evaluate whether the retrieved chunks cover ALL expected evidence types. If critical expected types are missing, you should strongly prefer the "insufficient_evidence" label rather than inferring a conclusion.
3. Citation Requirement: You must cite the exact Chunk_IDs that directly support your reasoning. Do not cite chunks that are not provided in the inputs.

CONFIDENCE SCORE CALIBRATION:

Provide a confidence score between 0.0 and 1.0 reflecting the adequacy and consistency of the retrieved evidence, as well as the coverage of expected evidence types:

- 0.90-1.00: Highly certain; evidence is direct, comprehensively covers expected types, and is unambiguous.
- 0.60-0.80: Moderately certain; evidence supports the claim but may require minor logical inference, or some secondary expected types are missing.
- 0.00-0.50: Low certainty; evidence is partial, indirect, highly ambiguous, or critical expected evidence types are missing.

OUTPUT CONSTRAINTS:

You must output a structured JSON object with the following fields:

```
{
  "judgment": "compliant" | "non-compliant" | "insufficient_evidence",
  "reasoning": "brief evidence-based explanation",
  "cited_chunk_ids": ["chunk_id_1", "chunk_id_2"],
  "confidence": 0.0-1.0
}
```

References

1. International Electrotechnical Commission. Dependability management—Part 3-10: application guide—maintainability and maintenance (IEC 60300-3-10: 2025). Geneva, Switzerland: IEC; 2025.
2. U.S. Department of Defense. Designing and developing maintainable products and systems (MIL-HDBK-470A). Washington, DC, USA: U.S. Department of Defense; 1997.
3. U.S. Department of Defense. Maintainability program requirements (MIL-STD-470). Washington, DC, USA: U.S. Department of Defense; 1966.
4. U.S. Department of Defense. Maintainability verification/demonstration/evaluation (MIL-STD-471A). Washington, DC, USA: U.S. Department of Defense; 1973.
5. Bhonde D, Zadeh P, Staub-French S. Evaluating the use of virtual reality for maintainability-focused design reviews. *J Inf Technol Constr.* 2022;27(13):253–72. doi:10.36680/j.itcon.2022.013.
6. Jiang S, Hu J, Magee CL, Luo J. Deep learning for technical document classification. *IEEE Trans Eng Manag.* 2024;71(1):1163–79. doi:10.1109/tem.2022.3152216.
7. Guo Z, Zhou D, Yu D, Zhou Q, Wu H, Hao A. An ontology-based method for knowledge reuse in the design for maintenance of complex products. *Comput Ind.* 2024;161(1):104124. doi:10.1016/j.compind.2024.104124.
8. Chen S, Turanoglu Bekar E, Bokrantz J, Skoogh A. AI-enhanced digital twins in maintenance: systematic review, industrial challenges, and bridging research-practice gaps. *J Manuf Syst.* 2025;82(9):678–99. doi:10.1016/j.jmsy.2025.07.006.
9. Lewis P, Perez E, Piktus A, Petroni F, Karpukhin V, Goyal N, et al. Retrieval-augmented generation for knowledge-intensive NLP tasks. In: *Proceedings of the 34th International Conference on Neural Information Processing Systems*; 2020 Dec 6–12; Vancouver, BC, Canada. p. 9459–74.
10. Guu K, Lee K, Tung Z, Pasupat P, Chang MW. REALM: retrieval-augmented language model pre-training. In: *Proceedings of the 37th International Conference on Machine Learning (ICML)*; 2020 Jul 13–18; Virtual. p. 3929–38. doi:10.48550/arxiv.2002.08909.
11. Ma Y, Zheng S, Yang Z, Pan H, Hong J. A knowledge-graph enhanced large language model-based fault diagnostic reasoning and maintenance decision support pipeline towards Industry 5.0. *Int J Prod Res.* 2026;64(12):5239–60. doi:10.1080/00207543.2025.2472298.
12. Heredia Álvaro JA, Barreda JG. An advanced retrieval-augmented generation system for manufacturing quality control. *Adv Eng Inform.* 2025;64(6):103007. doi:10.1016/j.aei.2024.103007.
13. Hillebrand L, Berger A, Uedelhoven D, Berghaus D, Warning U, Dilmaghani T, et al. Advancing risk and quality assurance: a RAG chatbot for improved regulatory compliance. In: *Proceedings of the 2024 IEEE International Conference on Big Data (BigData)*; 2024 Dec 15–18; Washington, DC, USA. p. 8668–70. doi:10.1109/bigdata62323.2024.10825431.
14. Laitenberger A, Manning CD, Liu NF. Stronger baselines for retrieval-augmented generation with long-context language models. In: *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*; 2025 Nov 4–9; Suzhou, China. p. 32559–69. doi:10.18653/v1/2025.emnlp-main.1656.
15. Riedler M, Langer S. Beyond text: optimizing RAG with multimodal inputs for industrial applications. *arXiv:2410.21943.* 2024.
16. Bukowski L, Werbinska-Wojciechowska S. Towards Maintenance 5.0: resilience-based maintenance in AI-driven sustainable and human-centric industrial systems. *Sensors.* 2025;25(16):5100. doi:10.3390/s25165100.
17. Zhang Y, Liu J, Hou K. Building a knowledge base of bridge maintenance using knowledge graph. *Adv Civ Eng.* 2023;2023(1):6047489. doi:10.1155/2023/6047489.
18. Yue S, Xiao L, Li J, Wang N. Research on application of knowledge graph for aircraft maintenance. *Adv Mech Eng.* 2022;14(7):16878132221107429. doi:10.1177/16878132221107429.
19. Xia L, Liang Y, Leng J, Zheng P. Maintenance planning recommendation of complex industrial equipment based on knowledge graph and graph neural network. *Reliab Eng Syst Saf.* 2023;232:109068. doi:10.1016/j.res.2022.109068.
20. Borgeaud S, Mensch A, Hoffmann J, Cai T, Rutherford E, Millican K, et al. Improving language models by retrieving from trillions of tokens. In: *Proceedings of the 39th International Conference on Machine Learning*; 2022 Jul 17–23; Baltimore, MA, USA. Vol. 162. p. 2206–40.

21. Asai A, Wu Z, Wang Y, Sil A, Hajishirzi H. Self-RAG: learning to retrieve, generate, and critique through self-reflection. In: *Proceedings of the International Conference on Learning Representations (ICLR)*; 2024 May 7–11; Vienna, Austria.
22. Yan SQ, Gu JC, Zhu Y, Ling Z. Corrective retrieval augmented generation. *arXiv:2401.15884*. 2024.
23. Edge D, Trinh H, Cheng N, Bradley J, Chao A, Mody A, et al. From local to global: a graph RAG approach to query-focused summarization. *arXiv:2404.16130*. 2024.
24. Singh A, Ehtesham A, Kumar S, Khoei TT, Vasilakos AV. Agentic retrieval-augmented generation: a survey on agentic RAG. *arXiv:2501.09136*. 2025.
25. Mishra S, Niroula S, Yadav U, Thakur D, Gyawali S, SoK GS. Agentic retrieval-augmented generation (RAG): taxonomy, architectures, evaluation, and research directions. *arXiv:2603.07379*. 2026.
26. Brundage MP, Sexton T, Hodkiewicz M, Dima A, Lukens S. Technical language processing: unlocking maintenance knowledge. *Manuf Lett*. 2021;27(1):42–6. doi:10.1016/j.mfglet.2020.11.001.
27. Masoudifard A, Mowlavi Sorond M, Madadi M, Sabokrou M, Habibi E. Leveraging graph-RAG and prompt engineering to enhance LLM-based automated requirement traceability and compliance checks. *arXiv:2412.08593*. 2024. doi: 10.48550/arxiv.2412.08593.
28. Karakurt E, Akbulut A. Retrieval-augmented generation (RAG) and large language models (LLMs) for enterprise knowledge management and document automation: a systematic literature review. *Appl Sci*. 2025;16(1):368. doi:10.3390/app16010368.
29. Wang Z. Current status and prospects of reliability systems engineering in China. *Front Eng Manag*. 2021;8(4):492–502. doi:10.1007/s42524-021-0172-2.
30. Commission of Science, Technology and Industry for National Defense. Maintainability design technique handbook (GJB/Z 91-97). Beijing, China: Commission of Science, Technology and Industry for National Defense; 1997.
31. Sanh V, Debut L, Chaumond J, Wolf T. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv:1910.01108*. 2019.
32. Robertson S, Zaragoza H. The probabilistic relevance framework: BM25 and beyond. *Found Trends Inf Retr*. 2009;3(4):333–89. doi:10.1561/15000000019.
33. Karpukhin V, Oguz B, Min S, Lewis P, Wu L, Edunov S, et al. Dense passage retrieval for open-domain question answering. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*; 2020 Nov 16–20; Online. p. 6769–81. doi:10.18653/v1/2020.emnlp-main.550.
34. Granata F, Poggi F, Mongiovi M. Enhancing retrieval-augmented generation with entity linking for educational platforms. *Big Data Cogn Comput*. 2026;10(4):120. doi:10.3390/bdcc10040120.
35. Han WH, Xiao X, Li YH, Wang J, Pechenizkiy M, Fang M. Adaptive iterative retrieval for enhanced retrieval-augmented generation. *Neurocomputing*. 2026;666(19):132272. doi:10.1016/j.neucom.2025.132272.
36. Es S, James J, Espinosa Anke L, Schockaert S. RAGAs: automated evaluation of retrieval augmented generation. In: *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*; 2024 Mar 17–22; St. Julians, Malta. p. 150–8. doi:10.18653/v1/2024.eacl-demo.16.