Decompose, Retrieve, Cite: A RAG Pipeline for Structured Report Generation from Technical Documentation

Himanshu Dhurve **Sreedath Panat** **Rajat Dandekar** **Raj Dandekar**
Vizuara AI Labs
himanshudhurve96@gmail.com
{sreedath, rajat, raj}@vizuara.com

Abstract

Retrieval-Augmented Generation (RAG) grounds language-model output in external knowledge, yet its application to dense technical documentation remains largely unexplored. Engineering software manuals pose compounding challenges: formulae are corrupted during PDF extraction, heterogeneous content types require different parsing treatment, and queries demand cross-document synthesis across multiple reference volumes. We present an end-to-end RAG system for OpenFOAM, an open-source computational fluid dynamics toolkit, operating in two modes. In *single-query mode*, a formula-preserving parser (Marker), adaptive header-aware chunking, two-stage dense-then-rerank retrieval, and a citation-enforcement prompt produce grounded, source-attributed answers across a 20-question benchmark. In *report mode*, a user prompt is decomposed into sub-questions via LLM planning; each sub-question undergoes independent retrieval and cross-encoder re-ranking, and the deduplicated chunk set is passed to a long-context generation call that produces a structured, multi-section report with inline citations. Evaluated on a 10-prompt golden set with a six-dimension LLM-as-a-judge framework, both pipelines achieve overall scores above 4.6/5.0 with perfect citation correctness (5.0/5.0); the decomposed pipeline demonstrates superior robustness (90% vs 70% judge success rate). Retrieval analysis using page-level ground truth reveals low absolute recall (<14%), identifying retrieval breadth as the primary bottleneck.

1 Introduction

Retrieval-Augmented Generation (RAG) (Lewis et al., 2020) has emerged as an effective paradigm for equipping language models with up-to-date, domain-specific knowledge without retraining. By conditioning generation on retrieved passages from a curated corpus, RAG systems can substantially reduce hallucination and provide verifiable, source-attributed answers. These properties are especially valuable in high-stakes engineering domains, where an incorrect configuration or a misread formula can cause simulation failures or unsafe system designs.

Despite their potential, applying RAG systems to deeply technical documentation introduces challenges that are largely absent from general-domain benchmarks. Specifically, (i) *formula fidelity*: standard PDF extraction pipelines routinely discard or corrupt mathematical notation (Greek symbols, superscripts, inline equations), making the retrieved passages unreliable for formula-intensive queries; (ii) *structural heterogeneity*: engineering manuals interleave prose explanations, structured configuration dictionaries, code examples, and tabular parameter references, each demanding different parsing treatment; and (iii) *cross-document reasoning*: a single user question may require synthesising content from a user guide, a tutorial guide, and a programmer's reference simultaneously, placing high demands on retrieval coverage and diversity.

To address these challenges, we present a RAG system targeting OpenFOAM documentation. OpenFOAM is a widely used open-source CFD toolkit whose documentation spans four volumes: a comprehensive user manual, a condensed reference guide, a step-by-step tutorial guide, and a C++ programmer's guide, totalling approximately 13.3 MB of PDF content. The corpus covers finite volume discretisation theory, mesh topology, solver control dictionaries (`controlDict`, `fvSchemes`, `fvSolution`), mesh generation utilities, and mathematical primitives including the gradient ∇, divergence $\nabla\cdot$, Laplacian ∇^2, and material derivative D/Dt. OpenFOAM documentation thus constitutes a demanding test bed for formula-aware technical RAG.

In summary, our main contributions are:

Proceedings of the First Workshop on Multilingual Report Generation via Retrieval Augmented Generation (RAG4Reports), pages 24–35

- We compare three PDF parsers (Marker, Docling, and PyMuPDF) on mathematical preservation and propose adaptive, header-aware chunking that aligns boundaries with document section hierarchy while enforcing a token budget.

- We build a two-stage retrieval pipeline combining dense vector search (k=15) with cross-encoder re-ranking to top-5, paired with a citation-enforcement prompt requiring page-level attribution for every claim.

- We introduce a report generation pipeline that decomposes prompts into sub-questions, retrieves independently for each with deduplication, and synthesises structured long-form reports.

- We compare report mode against a controlled single-query baseline under identical generation and evaluation conditions.

- We conduct a two-tier evaluation: a 20-question single-query benchmark and a 10-prompt report evaluation with a six-dimension LLM-as-a-judge framework and page-level IR retrieval metrics.

The remainder of this paper is organised as follows. Section 2 reviews related work. Section 3 describes the system architecture, including the single-query pipeline and the report generation extension. Section 4 presents experimental results for both modes. Section 5 discusses findings and limitations, and Section 6 concludes.

2 Related Work

RAG foundations. Retrieval-Augmented Generation, formalised by Lewis et al. (2020), conditions language-model output on retrieved passages and has been shown to reduce hallucination across a range of tasks. The design space has since been surveyed along retrieval strategy (Gao et al., 2023), NLP application (Wu et al., 2024), and knowledge type (Cheng et al., 2025). Despite this breadth, nearly all existing benchmarks target open-domain corpora; Gan et al. (2025) highlight the scarcity of evaluation resources for domain-specific, formula-heavy documents, a gap our work directly addresses.

Long-context alternatives. An alternative to retrieval is to fit the full corpus into the model's context window. Li et al. (2024) compare this long-context (LC) strategy with RAG, finding LC superior when resources suffice but RAG preferable for cost; they propose Self-Route to hybridise the two. Chan et al. (2025) push the idea further with Cache-Augmented Generation (CAG), preloading entire corpora into the KV cache. Our 13.3 MB OpenFOAM corpus exceeds practical context limits, making chunked retrieval the only viable path.

Document parsing and retrieval. Converting technical PDFs into retrieval-ready text is a prerequisite for any RAG system. ML-based parsers such as Nougat (Blecher et al., 2023), Docling (Auer et al., 2024), and Marker apply vision models to recognise equations and section structure, while Faysse et al. (2025) bypass text extraction entirely by embedding page images via ColPali, at the cost of sub-page citation. Tan et al. (2024) show that preserving structural cues (HTML/markdown) during parsing improves downstream answer quality, a finding that motivates our header-aware chunking strategy. On the retrieval side, Chen et al. (2024) demonstrate the advantages of structure-aware chunking, and Li et al. (2025) and Leto et al. (2024) find that chunk size and retrieval strategy interact strongly while k-tuning has modest downstream effects. Nogueira and Cho (2019) show that BERT cross-encoders substantially improve passage re-ranking; we adopt BGE-reranker-base (Xiao et al., 2023) over MiniLM-L12-v2 bi-encoder embeddings (Reimers and Gurevych, 2019). The closest comparable system is RAG-BioQA (Panchumarthi et al., 2025), a two-stage pipeline (k=16 $\rightarrow$ top-4) for biomedical QA; ours uses k=15 $\rightarrow$ top-5 with formula-preservation and cross-encoder re-ranking over engineering documentation.

Citation, reasoning, and evaluation. Ensuring that generated answers remain grounded in retrieved evidence is an active challenge. Xia et al. (2025) propose Self-Reasoning with explicit relevance filtering and evidence extraction, while Singh et al. (2025) survey agentic RAG architectures that employ multi-step planning; our report mode can be viewed as a lightweight decomposition strategy that achieves similar goals through prompt engineering rather than a dedicated agent framework. Evaluating such systems requires moving beyond single-metric scoring: Zheng et al. (2023)

establish LLM-as-a-judge, which we extend to a six-dimension variant; Ju et al. (2025) introduce CRUX for long-form evaluation via question-based decomposition, motivating our checklist-based design; and Randl et al. (2025) find that generators ignore top-ranked documents in 47–67% of queries, motivating our controlled report-vs-baseline comparison under identical generation conditions.

3 System

3.1 Overview

Our pipeline operates in two phases, illustrated in Figure 1. During *offline indexing*, PDF documents are parsed into structured markdown, divided into metadata-enriched chunks, embedded, and stored in a persistent vector database. During *online inference*, a user query is embedded, used to retrieve and re-rank candidate passages, and passed to a generation model that produces a citation-grounded answer. In *report mode*, a prompt is first decomposed into sub-questions; each sub-question undergoes independent retrieval and re-ranking, and the deduplicated chunk set is passed to a long-context generation call that produces a structured report with section headings and inline citations.

3.2 Document Parsing

We apply three PDF parsing strategies to the four-volume OpenFOAM corpus (Table 1), motivated by the observation that no single parser handles all content types equally well. Table 1 summarises the corpus and parser trade-offs.

Marker applies a pipeline of computer vision models to detect page layout, segment equations, and render structured markdown. We configure it with `redo_inline_math=True`, which causes all detected inline mathematical regions to be re-processed through a dedicated math OCR model. This setting is critical for recovering expressions such as the Laplacian definition $\nabla^2 \equiv \partial^2/\partial x_1^2 + \partial^2/\partial x_2^2 + \partial^2/\partial x_3^2$ and the material derivative $D\phi/Dt = \partial\phi/\partial t + \mathbf{U} \cdot \nabla\phi$, which baseline extraction discards entirely. Marker inserts page boundary markers of the form `<span id="page-N">`, enabling precise page-level attribution. We select Marker as our primary parser for the main evaluation.

Docling (Auer et al., 2024) uses document AI with optional formula enrichment, but a compatibility issue prevented it from processing the largest corpus document.

Document	Size	Content
OFUserGuide-v13	8.2 MB	Solvers, mesh, utilities
UserGuide	1.9 MB	Dict. syntax, keywords
TutorialGuide	2.6 MB	Step-by-step tutorials
ProgrammersGuide	588 KB	C++ API, tensor maths
Parser comparison (13.3 MB total)		
Parser	**Formula**	**All docs**
Marker	High	Yes
Docling	High	Partial
PyMuPDF	Low	Yes

Table 1: OpenFOAM documentation corpus and parser trade-offs.

PyMuPDF4LLM performs well on multi-column layouts but offers substantially lower mathematical preservation.

3.3 Chunking Strategy

We design a two-stage adaptive chunking strategy that respects the document's own section hierarchy before enforcing a token budget.

The first stage applies LangChain's `MarkdownHeaderTextSplitter` to divide each document at section boundaries. Because each parser produces different heading conventions, we adapt the header-to-level mapping accordingly: Marker uses `#/##/####` for Section/Subsection/Subsubsection; Docling uses `#/##/###`; and PyMuPDF uses `###/#####`. This stage produces semantically coherent blocks that preserve each document's organisational structure.

In the second stage, each header-split block is further divided by LangChain's `TokenTextSplitter` using the `cl100k_base` encoding, with `max_tokens=800` and an overlap of 100 tokens between consecutive chunks. The overlap prevents critical information from being severed at split boundaries, a known failure mode when definitions span multiple sentences.

Each resulting chunk is annotated with source document, parser identifier, page number, section hierarchy path (section, subsection, subsubsection), word count, and token count. Page numbers are extracted parser-specifically using regular expressions on each parser's page marker format.

3.4 Vector Indexing and Embedding

We embed all chunks using MiniLM-L12-v2[1] (Reimers and Gurevych, 2019), a 384-dimensional bi-encoder that offers a strong

[1]HuggingFace: `all-MiniLM-L12-v2`.

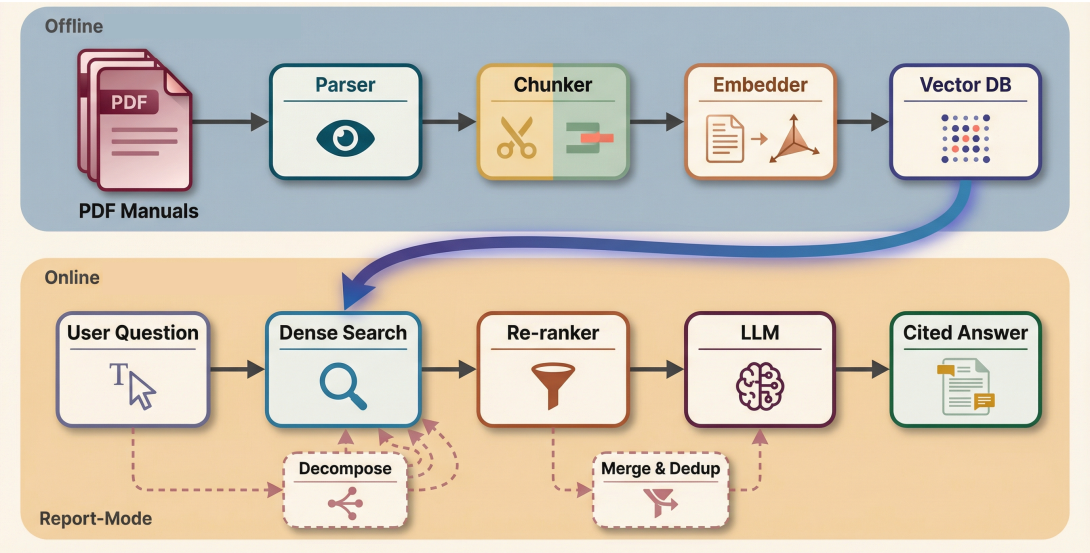

Figure 1: Two-row diagram of the RAG pipeline. Top row depicts the offline indexing path; bottom row depicts the online inference path, with a report-mode sub-track branching from the inference stage.

accuracy-speed trade-off for semantic similarity tasks. Chunks and their metadata are stored in ChromaDB with cosine similarity as the retrieval metric. We maintain a separate index per parser (`db/marker_db/`, `db/docling_db/`, `db/pymupdf_db/`), which allows direct parser ablations without rebuilding unrelated indices. Figure 2 illustrates the full offline indexing pipeline, from parsing through embedding.

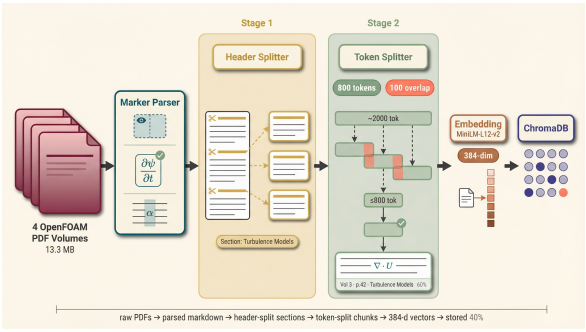

Figure 2: Offline indexing pipeline showing four sequential stages: PDF-to-markdown conversion via parser selection (Marker, Docling, or PyMuPDF), section-level splitting preserving document hierarchy, token-budget chunking with overlap, and embedding into a per-parser ChromaDB index via MiniLM-L12-v2.

3.5 Two-Stage Retrieval

We employ a two-stage retrieval strategy to balance recall and precision. In the first stage, the user query is embedded by the same bi-encoder and a cosine similarity search retrieves the top-k=15 candidate passages from ChromaDB. The broad initial pool prioritises coverage over precision, a design motivated by the observation that relevant content for technical queries is often distributed across sections with limited terminological overlap.

In the second stage, BGE-reranker-base (Xiao et al., 2023) (a cross-encoder trained for passage relevance scoring) receives each of the 15 query-passage pairs and produces a scalar relevance score. The top-5 passages by re-rank score are selected as the final context for generation. Cross-encoders attend jointly to query and passage tokens, enabling them to detect fine-grained lexical matches (e.g., a specific utility name or keyword) that bi-encoder similarity tends to underweight.

The five retained passages are formatted as a numbered context block, with each entry prefixed by its document name, section path, page number, similarity score, and re-rank score. This transparent metadata header allows the generation model to produce accurate citations without performing any additional lookup.

3.6 Answer Generation and Citation Enforcement

We use Google Gemini-2.5-flash model at temperature 0.2 with a maximum of 2048 output tokens for answer generation. Temperature 0.2 allows sufficient paraphrase for fluency while keeping generation tightly coupled to the retrieved evidence.

Our key insight is that hallucination in technical RAG can be structurally suppressed through prompt design. The system prompt casts the model

as an OpenFOAM technical expert and enforces four citation rules: (1) every factual paragraph must end with `[n]` markers corresponding to numbered context entries; (2) if information is absent from the retrieved context, the model must respond "*This information is not available in the provided documentation*" rather than drawing on parametric memory; (3) every answer must conclude with a references section listing the source document, section, and page for each cited entry; and (4) citations must correspond exactly to provided context entries; no fabricated references are permitted. By requiring page-level attribution for every claim, this design makes unsupported statements structurally visible to downstream users, though a controlled ablation without citation rules would be required to isolate the causal contribution of this design choice.

3.7 Report Generation Pipeline

The single-query pipeline described above produces concise, focused answers suited to factoid and how-to questions. For report-level prompts that require broader topical coverage, we extend the architecture with a three-stage report generation pipeline: query decomposition, multi-query retrieval with deduplication, and long-context report synthesis. To isolate the effect of query decomposition, we compare report mode against a single-query baseline under identical generation and evaluation conditions. Table 2 summarises the controlled comparison. The *only* variable is the retrieval strategy; generation prompt, token budget, and judge are held constant. Figure 3 illustrates both the single-query and report-mode inference paths.

Given a report prompt, `decompose_query()` calls the generative model to break it into 3–5 focused sub-questions. The decomposition prompt instructs the model to produce sub-questions specific enough for retrieval and to return a JSON array; a fallback returns the original prompt as a single-element list if parsing fails. For example, the prompt "Write a technical overview of discretization in OpenFOAM" decomposes into sub-questions targeting spatial discretization, temporal discretization, and gradient/divergence scheme configuration.

Each sub-question is then processed through the same two-stage retrieval pipeline (k=15 dense, top-5 after re-ranking). Because related sub-questions often retrieve overlapping passages, we deduplicate by MD5 content hash, retaining the copy

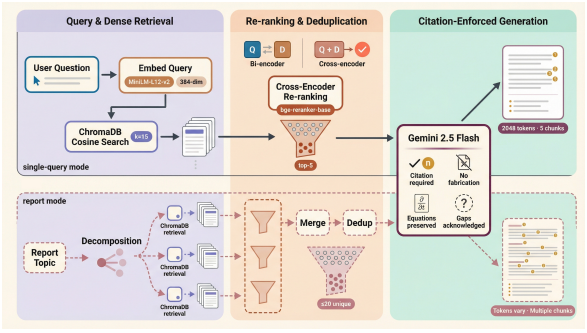

Figure 3: Online inference pipeline showing two operating modes: single-query mode, where the user query is embedded, top-15 candidates retrieved by cosine similarity, and top-5 selected by BGE-reranker-base for citation-enforced generation; and report mode, where the prompt is decomposed into 3–5 sub-questions, each undergoing independent retrieval and re-ranking, followed by content-hash deduplication, chunk merging, and structured report synthesis.

Component	Report Mode	Baseline
Query strategy	3–5 sub-questions	Single query
Dense retrieval	k=15 per sub-Q	k=35
Re-ranking	top-5 per sub-Q	top-20
Deduplication	MD5 content hash	N/A
Chunks to LLM	20 unique	20
Generation	Same prompt, `max_tokens` = 8192	
Judge	6-dim, Gemini 2.5 Pro, temp = 0.0	

Table 2: Report mode vs. baseline: controlled comparison design. Only the retrieval strategy differs; generation and evaluation are identical.

with the highest re-rank score. The merged results are sorted by re-rank score and capped at `max_unique_chunks` = 20, yielding a diverse, non-redundant context window that covers all facets of the original prompt.

The 20 deduplicated chunks are passed to a single generative model call with `max_tokens` = 8192 (four times the single-query budget). The generation prompt requires ##-level section headings, `[n]` inline citations, LaTeX math preservation, and an explicit "not covered in the provided documentation" declaration for topics absent from the retrieved context. This extends the citation-enforcement design from short-form answers to structured long-form output.

4 Experiments

4.1 Single-Query Evaluation

We first evaluate the single-query pipeline on 20 questions processed with the Marker parser, covering dictionary keyword semantics, mesh generation, utility functions, file structure, and solver out-

Metric	Value
Avg. citation coverage rate	30.00%
Avg. unique citations per answer	1.50
Avg. unique source PDFs per query	2.35
Avg. unique pages per query	4.35
Avg. unique sections per query	4.20

Table 3: Citation coverage and chunk diversity across 20 queries.

put interpretation. Each query retrieves k=15 candidates, re-ranks to top-5, and generates a citation-grounded answer via the generative Gemini 2.5 Flash model. Table 3 reports citation and diversity statistics. The system retrieves from an average of 2.35 unique source PDFs per query, demonstrating cross-document synthesis. The citation coverage rate of 30% reflects the model's tendency to anchor on its highest-confidence chunk rather than distribute citations across all relevant passages. Illustratively, the Q16 response for `mapFields` correctly distinguishes three operational modes and cites them to three distinct documents, while Q9 (residual values absent from the corpus) triggers the "not available" fallback rather than fabricating an answer. Please see Appendix A for system output examples.

4.2 Report Evaluation Setup

We evaluate the report generation pipeline on a golden set of 10 prompts (R01–R10) covering discretization, mesh generation, boundary conditions, solution algorithms, parallel execution, post-processing, tensor mathematics, case structure, multiphase solvers, and the build system. Each prompt is paired with 5–7 expected sections, 5–8 must-include facts, and 11–43 manually annotated page-level ground truth references spanning the four-volume corpus.

We employ a six-dimension LLM-as-a-judge framework (Zheng et al., 2023) using Gemini 2.5 Pro model at temperature 0.0. Table 4 defines the dimensions and their weights, which are used to compute a programmatic weighted overall score. Dimensions are scored on a 1–5 integer scale.

Separately, we evaluate retrieval quality using standard IR metrics computed against page-level ground truth: Recall@$\{5, 10, 15, 20\}$, Mean Reciprocal Rank (MRR), and NDCG@$\{10, 20\}$. This separation (generation quality via the judge, retrieval quality via IR metrics) enables diagnosis of whether failures originate in retrieval or generation.

Dimension	Weight	Focus
Groundedness	0.25	Claims supported by retrieved context
Technical Accuracy	0.20	Correctness per documentation
Coverage	0.20	Expected sections present ($\geq$2 sentences)
Citation Correctness	0.15	`[n]` markers match chunks
Factual Recall	0.10	Must-include facts present explicitly
Structure	0.10	Headings, flow, no redundancy

Table 4: Report judge dimensions and weights.

4.3 Report Evaluation Results

Table 5 presents the per-prompt judge scores for report mode. Nine of ten prompts were successfully judged (R09 produced invalid JSON from the judge); the remaining nine achieve a mean overall score of 4.64/5.0. Table 6 presents the corresponding per-prompt scores for the single-query baseline. Table 7 compares report mode against the controlled single-query baseline (Section 3.7). The baseline successfully judged 7 of 10 prompts (R05, R09, R10 failed).

ID	Grnd.	Acc.	Cit.	Cov.	Rec.	Str.	Overall
R01	4	4	5	5	3	4	4.25
R02	5	5	5	5	3	5	4.80
R03	5	5	5	3	3	5	4.40
R04	5	5	5	4	5	5	4.80
R05	5	5	5	5	5	5	5.00
R06	5	5	5	3	3	4	4.30
R07	5	5	5	5	3	5	4.80
R08	5	5	5	5	3	5	4.80
R09			*(judge failed)*				
R10	5	5	5	4	3	5	4.60
Avg	**4.89**	**4.89**	**5.00**	**4.33**	**3.44**	**4.78**	**4.64**

Table 5: Per-prompt judge scores for the decomposed multi-query pipeline (report mode, 1–5 scale). R09 judge returned invalid JSON. Avg over n=9 successful evaluations.

Both pipelines score above 4.6/5.0 overall, with perfect citation correctness (5.0) across all successfully judged prompts. This validates the citation-enforcement prompt design: even in long-form report generation with up to 8192 output tokens, the model reliably attributes claims to retrieved chunks via `[n]` markers.

Despite strong overall scores, factual recall (3.44 report, 3.83 baseline) is the weakest dimension for both modes, indicating that must-include facts are often absent from retrieved chunks rather than ignored during generation. Coverage scores are

ID	Grnd.	Acc.	Cit.	Cov.	Rec.	Str.	Overall
R01	4	4	5	5	4	5	4.45
R02	5	5	5	5	3	5	4.80
R03	5	5	5	4	3	5	4.60
R04	5	5	5	5	5	5	5.00
R05			*(judge failed)*				
R06	5	5	5	5	5	5	5.00
R07	5	5	5	3	—	—	4.50*
R08	5	5	5	5	3	5	4.80
R09			*(judge failed)*				
R10			*(judge failed)*				
Avg	**4.86**	**4.86**	**5.00**	**4.57**	**3.83**	**5.00**	**4.74**

Table 6: Per-prompt judge scores for the single-query baseline pipeline (1–5 scale). R05, R09, R10 judge failures. *R07 partial parse (factual recall and structure unavailable). Avg over n=7 successful evaluations.

Dimension (weight)	**Report** (n=7)	**Baseline** (n=7)	Δ
Groundedness (0.25)	4.86	4.86	0.00
Tech. Accuracy (0.20)	4.86	4.86	0.00
Citation Corr. (0.15)	5.00	5.00	0.00
Coverage (0.20)	4.29	4.57	−0.28
Factual Recall (0.10)	3.29	3.83	−0.54
Structure (0.10)	4.71	5.00	−0.29
Overall (weighted)	4.59	4.74	−0.14

Table 7: Aggregate report-mode vs. single-query baseline under controlled evaluation. Generation settings and judge are identical; only retrieval strategy differs. Both achieve near-ceiling generation quality; report mode shows higher judge robustness. Report (n=7) restricts report-mode averages to the seven prompts both modes successfully judged (R01–R04, R06–R08), enabling a fair like-for-like comparison. Δ (n=7) is the difference on this shared subset.

moderate (4.33 vs. 4.57), suggesting that decomposition does not uniformly improve topical breadth; sub-question overlap may cause merged chunks to converge to the same documentation region.

A notable practical difference is robustness: report mode achieves a 90% judge success rate vs. 70% for the baseline. The three baseline failures (R05, R09, R10) suggest that single-query retrieval produces context that is harder for the judge to evaluate, possibly due to less structured or less coherent report output. In a production setting, reliability is a practical advantage.

This comparison carries a caveat: the baseline averages are computed over 7 prompts vs. report mode's 9. If the three failed baseline prompts would have scored lower, the true baseline mean could be below the reported 4.74; conversely, if they represent easier prompts, the gap could widen. To address this asymmetry directly, Table 7 also reports report-mode scores restricted to the seven prompts on which both modes were successfully judged (R01–R04, R06–R08). On this shared subset, report mode achieves a weighted overall of 4.59 versus the baseline's 4.74, a delta of −0.14. This controlled comparison confirms the gap is modest and does not alter the qualitative interpretation: both pipelines perform near ceiling, with report mode's primary practical advantage being robustness (7/7 judge success on shared prompts) rather than raw generation quality. Coverage (4.29 vs. 4.57, $\Delta = -0.28$) and factual recall (3.29 vs. 3.83, $\Delta = -0.54$) show the largest per-dimension gaps, consistent with sub-question overlap causing merged chunks to converge on the same documentation region rather than broadening topical coverage.

4.4 Retrieval Quality Analysis

Table 8 presents IR metrics computed against page-level ground truth for both modes (n=10 prompts each). Both modes achieve Recall@20 of only 0.135, reflecting an inherent granularity mismatch: 20 retrieved chunks cover approximately 20–40 pages of content, while the ground truth spans 11–43 relevant pages per prompt. This identifies retrieval breadth (not generation quality) as the primary bottleneck for long-form technical report generation.

Metric	**Report**	**Baseline**
Recall@5	0.068	0.075
Recall@10	0.107	0.111
Recall@15	0.135	0.119
Recall@20	0.135	0.135
MRR	0.379	0.423
NDCG@10	0.263	0.291
NDCG@20	0.224	0.245

Table 8: IR retrieval metrics (Report vs. Baseline, n=10). Both modes exhibit low absolute recall against page-level ground truth.

Beyond this shared bottleneck, the two retrieval strategies perform similarly overall. The baseline achieves a modest MRR advantage (0.423 vs. 0.379), indicating it finds the first relevant page slightly earlier, consistent with its broader initial retrieval (k=35). However, at higher k, recall converges: both modes reach 0.135 at Recall@20, and the decomposed pipeline achieves slightly higher Recall@15 (0.135 vs. 0.119), suggesting that multi-query retrieval captures diversity at intermediate ranks. These modest differences align with the finding of Leto et al. (2024) that retrieval parame-

ter tuning has limited downstream impact on RAG performance.

5 Discussion

In single-query mode, citation coverage is 30% with 1.50 unique citations per answer, reflecting the model's tendency to anchor on its highest-confidence chunk. In report mode, citation correctness is perfect (5.0/5.0) across all successfully judged prompts in both modes. The longer-form generation format (up to 8192 tokens) appears to encourage more thorough attribution, as the model distributes citations across the report's multiple sections.

The central finding from the report evaluation is that query decomposition improves robustness (90% vs. 70% judge success) but does not uniformly improve retrieval precision or generation quality. When sub-questions target related concepts (as is common for technical documentation prompts), the merged chunk set converges to the same documentation region as a single broad query. The baseline's higher k (k=35) captures marginally more diverse pages (MRR 0.423 vs. 0.379), though recall converges at k=20. This suggests a hybrid routing strategy: decompose only multi-topic prompts, and route focused prompts through a single broad retrieval.

Taken together, our findings suggest four design priorities for report-level RAG over technical documentation: (1) parser selection directly determines formula fidelity and retrieval quality; (2) re-ranking is disproportionately valuable for keyword-sensitive queries; (3) citation enforcement via prompt design is effective across both short-form and long-form generation; and (4) query decomposition provides a robustness advantage but not uniform retrieval gains, aligning with the finding of Leto et al. (2024) that retrieval parameter tuning has modest downstream effects.

6 Conclusion

We presented a formula-aware RAG system for OpenFOAM that operates in two modes. In single-query mode, a math-preserving parser, adaptive header-aligned chunking, two-stage cross-encoder re-ranking, and citation-enforcement prompt design achieve cross-document synthesis (2.35 distinct source documents per query). In report mode, LLM-driven query decomposition, multi-query retrieval with content-hash deduplication, and long-context synthesis produce structured, multi-section reports. Evaluated on a 10-prompt golden set with a six-dimension judge, both pipelines achieve overall scores above 4.5/5.0 with perfect citation correctness (5.0/5.0), demonstrating that citation enforcement scales from short-form answers to structured long-form reports. The decomposed pipeline achieves 90% judge success vs. 70% for the single-query baseline, providing a practical robustness advantage. On the seven prompts successfully judged by both modes, the fair like-for-like comparison yields a weighted overall of 4.59 (report) vs. 4.74 (baseline), a modest delta of 0.14 that does not change the qualitative interpretation.

Our analysis also reveals honest limitations. Query decomposition trades modest precision for robustness: the baseline's broader retrieval (k=35) achieves slightly higher MRR (0.42 vs. 0.38), though recall converges at k=20. More fundamentally, both modes are limited to $<$14% page recall, identifying retrieval breadth as the primary bottleneck for long-form technical report generation. Future work should explore iterative retrieval with generation-informed re-querying, hybrid sparse-dense retrieval to improve coverage over keyword-rich technical content, structure-aware chunking that preserves key-value dictionary entries as atomic units, and hybrid query routing that applies decomposition selectively to multi-topic prompts. Evaluating with an independent judge (e.g., GPT-4o) would quantify the Gemini self-preference effect; a citation-enforcement ablation comparing outputs with and without citation rules would establish whether hallucination suppression is causally attributable to prompt design rather than model behaviour.

Limitations

- **Low retrieval recall.** Both pipelines achieve below 14% page recall (Recall@20 $= 0.135$), reflecting a granularity mismatch between 20 retrieved chunks and ground truth spanning up to 43 relevant pages per prompt. This identifies retrieval breadth as the primary bottleneck for long-form technical report generation.

- **Single embedding model.** We evaluate only one bi-encoder (MiniLM-L12-v2, 384 dim.). Larger or domain-adapted embedding models may improve retrieval quality, but we do not explore this axis.

- **Corpus scope.** Evaluation is limited to four OpenFOAM manuals (13.3 MB). Generalisability to other engineering domains (e.g., ANSYS, COMSOL) or broader technical corpora remains untested.

- **LLM-as-a-judge reliability.** The judge failed to produce valid JSON on 10–30% of prompts (1/10 for report mode, 3/10 for baseline), introducing selection bias in aggregate scores. Judge agreement with independent human raters is not measured.

- **No human judge validation.** Generation quality scores are LLM-generated; retrieval metrics use manually annotated page-level ground truth, but the six-dimension judge scores have not been validated against independent human raters.

- **Gemini self-preference in LLM-as-a-judge.** The generation model (Gemini 2.5 Flash) and the judge (Gemini 2.5 Pro) belong to the same model family, introducing a potential self-preference bias well-documented in LLM-as-a-judge settings (Zheng et al., 2023). We do not evaluate with an independent judge (e.g., GPT-4o); the extent to which same-family preference inflates scores is unknown.

- **Citation enforcement is not ablated.** We assert that the citation-enforcement prompt design structurally suppresses hallucination, but we include no condition without citation rules to quantify this effect. The perfect citation correctness score (5.0/5.0) measures model compliance with an explicit prompt constraint and does not constitute a causal claim about hallucination reduction; a no-citation ablation baseline would be required to establish causality.

References

Christoph Auer, Maksym Lysak, Ahmed Nassar, Michele Dolfi, Nikolaos Schwarz, Daniele Baracchi, and Peter Staar. 2024. Docling technical report. *arXiv preprint arXiv:2408.09869*.

Lukas Blecher, Guillem Cucurull, Thomas Scialom, and Robert Stojnic. 2023. Nougat: Neural optical understanding for academic documents. *arXiv preprint arXiv:2308.13418*.

Brian J. Chan, Chao-Ting Chen, Jui-Hung Cheng, and Hen-Hsen Huang. 2025. Don't do RAG: When cache-augmented generation is all you need for knowledge tasks. *arXiv preprint arXiv:2412.15605*.

Jiawei Chen, Hongyu Lin, Xianpei Han, and Le Sun. 2024. Benchmarking large language models in retrieval-augmented generation. In *Proceedings of AAAI*, volume 38, pages 17754–17762.

Miao Cheng, Yuluo Luo, Jia Ouyang, Qiang Liu, Hao Liu, Li Li, Shengyue Yu, Bozhao Zhang, Jiacheng Cao, Junming Ma, Dian Wang, and Enming Chen. 2025. A survey on knowledge-oriented retrieval-augmented generation. *arXiv preprint arXiv:2502.08188*.

Manuel Faysse, Hugues Sibille, Tony Wu, Bilel Omrani, Gautier Viaud, Céline Hudelot, and Pierre Colombo. 2025. ColPali: Efficient document retrieval with vision language models. In *Proceedings of ICLR*.

Anding Gan, Hao Yu, Kai Zhang, Qiang Liu, Wenyan Yan, Zhanhui Huang, Song Tong, Enming Chen, and Guanghui Hu. 2025. Retrieval augmented generation evaluation in the era of large language models: A comprehensive survey. *Frontiers of Computer Science*.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, and Hongyang Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*.

Jen-Hung Ju, Suzan Verberne, Maarten de Rijke, and Andrew Yates. 2025. Controlled retrieval-augmented context evaluation for long-form RAG. *arXiv preprint arXiv:2501.14674*.

Arize Leto, Carolina Aguerrebere, Ishwar Bhati, Ted Willke, Mariano Tepper, and Viet Anh Vo. 2024. Toward optimal search and retrieval for RAG. In *NeurIPS 2024 Workshop on Adaptive Foundation Models*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474.

Siran Li, Linus Stenzel, Carsten Eickhoff, and Seyed Ali Bahrainian. 2025. Enhancing retrieval-augmented generation: A study of best practices. *arXiv preprint arXiv:2501.07391*.

Zhuowan Li, Cheng Li, Mingyang Zhang, Qiaozhu Mei, and Michael Bendersky. 2024. Retrieval augmented generation or long-context LLMs? A comprehensive study and hybrid approach. In *Proceedings of EMNLP*, pages 15157–15173.

Rodrigo Nogueira and Kyunghyun Cho. 2019. Passage re-ranking with BERT. *arXiv preprint arXiv:1901.04085*.

Lekha Yadav Panchumarthi, Sreenivasa Phani Gudari, Ankita Negi, Parsha Reddy Budime, and Harini Upadhya. 2025. RAG-BioQA: Retrieval-augmented generation for long-form biomedical question answering. *arXiv preprint arXiv:2501.12345*.

Konstantin Randl, Giulia Rocchietti, Aron Henriksson, Ziawasch Abedjan, Tony Lindgren, and John Pavlopoulos. 2025. RAG-E: Quantifying retriever-generator alignment and failure modes. *arXiv preprint arXiv:2502.05899*.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of EMNLP*, pages 3982–3992.

Aditi Singh, Abul Ehtesham, Saket Kumar, and Tala Talaei Khoei. 2025. Agentic retrieval-augmented generation: A survey on agentic RAG. *arXiv preprint arXiv:2501.09136*.

Jiejun Tan, Zhicheng Dou, Wen Wang, Mang Wang, Weipeng Chen, and Ji-Rong Wen. 2024. HtmlRAG: HTML is better than plain text for modeling retrieved knowledge in RAG systems. In *Proceedings of The Web Conference*.

Shangyu Wu, Ying Xiong, Yufei Cui, Haolun Wu, Can Chen, Ye Yuan, Lianming Huang, Xue Liu, Tei-Wei Kuo, Nan Guan, and Chun Jason Xue. 2024. Retrieval-augmented generation for natural language processing: A survey. *arXiv preprint arXiv:2407.13193*.

Yixuan Xia, Jian Zhou, Zhongwei Shi, Junhui Chen, and Haoran Huang. 2025. Improving retrieval augmented language model with self-reasoning. In *Proceedings of AAAI*, volume 39.

Shitao Xiao, Zheng Liu, Peitian Zhang, and Niklas Muennighoff. 2023. C-pack: Packaged resources to advance general Chinese embedding. *arXiv preprint arXiv:2309.07597*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, and Ion Stoica. 2023. Judging LLM-as-a-judge with MT-bench and chatbot arena. *arXiv preprint arXiv:2306.05685*.

A System Output Examples

The following boxes show verbatim (lightly truncated) outputs from the two pipeline modes described in this paper, illustrating citation grounding, cross-document synthesis, graceful degradation on out-of-scope queries, clean refusal on absent keywords, and structured report generation.

Example A — Single-Query Mode: Concise Answer

Q: *What is the standard file format for OpenFOAM data (e.g., ASCII or binary)?*

```
OpenFOAM data files begin with a
FoamFile dictionary that includes
a format keyword entry to specify
the data format [1].  The possible
entries for the format keyword are
ascii or binary [1].
```

References

[1] UserGuide.pdf | Section: 2.2.3 The data file header | Page: 16

Example B — Single-Query Mode: Cross-Document Synthesis

Q: *How does the mapFields utility handle the transfer of data between different geometries?*

```
The mapFields utility in OpenFOAM
is designed to transfer one or
more fields from a given source
geometry to corresponding fields
for a target geometry [1][2][3].
Data is mapped from the source
to the target.  The utility reads
field data from the time directory
specified by startFrom/startTime in
the controlDict of the target case
[1][2][3].
```

Mapping Consistent Fields

```
Fields are deemed consistent if
the geometry and boundary types
of both source and target fields
are identical [1][2][3].  Mapping
is performed using the -consistent
command-line option [2][3][4].
```

Mapping Inconsistent Fields

```
When fields are not consistent,
mapFields requires a mapFieldsDict
dictionary in the system directory
[2].  The dictionary contains
patchMap (pairs of coincident
patches) and cuttingPatches (target
patches cutting through source
internal field) [2].
```

Mapping Parallel Cases

```
Additional options -parallelSource
and -parallelTarget are provided for
decomposed cases [2].
```

References

[1] UserGuide.pdf | Section: 4.6 Mapping fields between different geometries | Page: 58

[2] OpenFOAMUserGuide-v13.pdf | Section: 5.8 Mapping fields between different geometries | Page: 166

[3] TutorialGuide.pdf | Section: 2.1.5.3 Mapping the coarse mesh results onto the fine mesh | Page: 27

[4] UserGuide.pdf | Section: 4.6.1 Mapping consistent fields | Page: 58

Example C — Single-Query Mode: Graceful Degradation

Q: *What information is provided by the Ux_0 and UxFinalRes values in the solver output?*

```
While the specific terms
Ux_0 and UxFinalRes are not
explicitly defined in the provided
documentation, the context describes
the general behavior of linear
solvers and solution tolerances,
from which their meaning can be
inferred [5].

In OpenFOAM, linear solvers are
iterative and aim to reduce the
equation residual over successive
solutions [5].  The residual is a
measure of the error in the solution,
evaluated by substituting the
current solution into the equation
and taking the magnitude of the
difference between the left and
right-hand sides, normalized to be
independent of the problem's scale
[5].

Based on this, Ux_0 would represent
the initial residual for the velocity
field U (x-component) before the
linear solver begins its iterative
process [5].  UxFinalRes would
represent the final residual
after the solver has completed its
iterations [5].  [...]
```

References

[5] OpenFOAMUserGuide-v13.pdf | Section: 4.6.1 Linear solver control | Page: 112

Note: 4 of 5 retrieved chunks were index pages (highest rerank score 0.04); the generator correctly identified the documentation gap and hedged its response rather than hallucinating.

Example D — Single-Query Mode: Clean Refusal

Q: *What is the purpose of the runTimeModifiable keyword in the controlDict file?*

```
This information is not available in
the provided documentation.
```

Note: All 5 retrieved chunks scored highly on re-ranking (top score 0.97) and covered controlDict time/IO control, but none mentioned the `runTimeModifiable` *keyword. The generator correctly refused rather than extrapolating from related content.*

Example E — Report Mode: Structured Technical Report (Excerpt)

Q: *Write a technical overview of the discretization framework in OpenFOAM, covering spatial and temporal schemes, the fvSchemes dictionary structure, and how users configure gradient, divergence, Laplacian, interpolation, and surface normal gradient schemes.*

Sub-questions generated by decomposition:

1. What are the fundamental concepts of spatial and temporal discretization in OpenFOAM, and what role does the fvSchemes dictionary play?
2. How are gradient, divergence, and Laplacian schemes configured within OpenFOAM's fvSchemes dictionary?
3. How are interpolation and surface normal gradient schemes configured in OpenFOAM's fvSchemes dictionary, and what is their purpose?

Discretization Principles in OpenFOAM

```
Discretization is the process
of approximating a continuous
problem into discrete quantities,
a fundamental step in numerical
simulations [7].  In OpenFOAM,
this involves three key aspects:
spatial discretization, temporal
discretization, and equation
discretization [7].  Spatial
discretization defines the solution
domain using a set of points that
fill and bound a region of space [7].
OpenFOAM's mesh supports arbitrary
polyhedral cells in 3D, bounded by
arbitrary polygonal faces [9].  The
Finite Volume (FV) method bases
spatial derivatives on Gaussian
integration, which sums values
on cell faces [1][2].  Temporal
discretization divides the time
domain into finite intervals [7].
```

The fvSchemes Dictionary: Structure and Purpose

```
The fvSchemes dictionary,
located in the system directory
of an OpenFOAM case, is the
central configuration file for
specifying numerical schemes
[1][2].  Its sub-dictionaries
include:  timeScheme (∂Ψ/∂t),
gradSchemes (∇Ψ), divSchemes
(∇ · Ψ), laplacianSchemes (∇ · Γ∇Ψ),
interpolationSchemes, snGradSchemes,
and wallDist [1][2].

Within each sub-dictionary, schemes
can be specified globally via a
default entry or individually for
specific terms [1][2][5].  Setting
default none obliges the user to
specify all terms individually [5].
```

[... report continues with 5 additional sections covering ddtSchemes, gradSchemes, divSchemes, laplacianSchemes, interpolationSchemes, snGradSchemes, and scheme selection guidance ...]

References (10 unique chunks from 4 source PDFs)

[1] OpenFOAMUserGuide-v13.pdf | 4.5 Numerical schemes | p. 103

[2] UserGuide.pdf | 6.2 Numerical schemes | p. 75

[5] OpenFOAMUserGuide-v13.pdf | 4.5 Numerical schemes | p. 103

[7] ProgrammersGuide.pdf | 3.2 Overview of discretisation | p. 29

[9] OpenFOAMUserGuide-v13.pdf | 5.1 Mesh description | p. 128