Service Business RAG:
Fast Retrieval and Trustworthy Answers in Multi-Client Environments

Guilherme Fábio Vieira

January 30, 2026

Abstract

Service providers typically operate across multiple clients and continuously produce and consume large volumes of heterogeneous documents (manuals, wikis, contracts, procedures, reports, support tickets, and API integration artifacts). In this setting, knowledge seeking by team members—as well as understanding client-specific rules and business logic—tends to be slow, error-prone, and operationally expensive.

This paper proposes the *Service Business RAG* approach: an operational data model and a Retrieval-Augmented Generation (RAG) architecture for fast and accurate retrieval over multiple sources, with logical isolation per client, metadata governance, and auditable evidence for each generated answer. The data model is defined as minimal, chunk-level metadata persisted in JSONL and associated with per-tenant vector indices.

In addition, we describe an end-to-end operational pipeline for chunking, embedding generation, multi-index vector search (e.g., FAISS), contextual metadata filtering, similarity-based deduplication, and optional reranking. The goal is to reduce time-to-knowledge and improve answer accuracy in service operations.

1 Introduction

Service businesses operate across multiple clients, teams, and processes, generating documentation continuously: operational procedures, reports, manuals, internal wikis, deployment records, contracts, and interaction histories. The fragmentation of such information across repositories and formats makes it difficult to obtain fast and precise answers at decision time.

This work introduces *Service Business RAG* as a practical pattern to organize, index, and retrieve multi-source, multi-client knowledge with traceability, enabling precise search and grounded answers supported by document excerpts.

Contributions. This paper makes the following contributions:

- We propose an operational *Retrieval-Augmented Generation* pattern tailored to service businesses operating in multi-client environments, emphasizing tenant isolation and traceability rather than model benchmarking.
- We define a minimal, chunk-level metadata schema persisted in JSONL and coupled with per-tenant vector indices, enabling auditable retrieval without analytical data modeling.

- We describe a reference retrieval pipeline integrating query expansion, metadata-based filtering, similarity-based deduplication, and optional reranking under explicit cost–quality trade-offs.
- We introduce an operational evaluation perspective based on system instrumentation and retrieval-quality signals, suitable for real-world deployments where controlled experiments are not always feasible.

1.1 Research question

How can an operational metadata model, combined with a multi-tenant RAG architecture, enable fast, traceable, and accurate knowledge retrieval processes in service businesses operating across multiple clients?

2 Problem statement and objectives

2.1 Problem statement

In service organizations, the information required to execute a task (e.g., configuring a client, validating a process, answering a question, meeting a contractual requirement) is often distributed across multiple sources and affected by version conflicts. Without a unified retrieval and governance layer, search becomes slow and inaccurate, rework increases, and there is a higher risk of using outdated information or content belonging to the wrong client.

2.2 Objectives

- Define a general RAG approach for service businesses operating across multiple clients and heterogeneous data sources.
- Propose an operational, metadata-based data model for retrieval and auditing that enables per-tenant isolation, traceability, and source auditing.
- Describe an efficient ingestion/indexing and retrieval pipeline (embeddings, vector search, filters, and reranking).
- Discuss quality criteria (accuracy, coverage, deduplication) and governance aspects (versioning, recency, and access control).
- Address the research question stated in Section 1.1.

3 Methodology

3.1 Architecture overview

The *Service Business RAG* architecture follows a generic and replicable flow: (i) a user request from an external application to an API (e.g., FastAPI), (ii) question validation and extraction, (iii) content normalization, (iv) optional and configurable *query expansion* for semantic variations, (v) embedding generation, (vi) multi-index vector search (by

tenant, by source, and/or by logical domain), (vii) contextual filtering based on metadata and similarity-based deduplication, (viii) heuristic and/or LLM-based reranking, and (ix) construction of the final context for the language model call, with optional return of sources and quality metrics.

4 Pipeline implementation in Service Business RAG

This section describes the operationalization of *Service Business RAG* as implemented in the project, covering: construction of vector indices (FAISS), the retrieval core (embeddings and local search), context assembly (query expansion, filters, deduplication, and metrics), and API-based exposure.

4.1 Index construction (FAISS and metadata)

The indexing process persists two primary artifacts per *tenant* (client) and/or logical domain:

- `faiss.index`: a vector structure for similarity search using inner product;
- `meta.jsonl`: per-*chunk* metadata (e.g., `source_type`, `tenant`, `title/path`, `chunk_id`, `text`).

During indexing, texts are segmented into overlapping *chunks* and embeddings are generated in batches. Embeddings are L2-normalized before insertion into a `IndexFlatIP` index, ensuring that FAISS similarity scores correspond to cosine similarity (inner product between unit vectors).

4.1.1 Formulation (embeddings and similarity)

Given a query q and a document *chunk* d, the embedding model produces dense vectors:

$$\mathbf{e}_q = f(q), \qquad \mathbf{e}_d = f(d)$$

L2 normalization is then applied:

$$\hat{\mathbf{e}}_q = \frac{\mathbf{e}_q}{\|\mathbf{e}_q\|_2}, \qquad \hat{\mathbf{e}}_d = \frac{\mathbf{e}_d}{\|\mathbf{e}_d\|_2}$$

Vector similarity is computed using the inner product:

$$s(q, d) = \hat{\mathbf{e}}_q^{\top} \hat{\mathbf{e}}_d$$

Because both vectors are normalized, $s(q, d)$ is equivalent to cosine similarity, simplifying interpretation and enabling the use of relevance thresholds.

4.2 Retrieval core and per-tenant isolation

The RAG core implements:

- automatic discovery of available indices by traversing `FAISS_BASE` following the pattern `/tenant[/project]`;

- index and metadata caching to reduce I/O overhead;
- query embedding generation;
- local FAISS Top-k search per index.

Multi-tenant isolation is enforced by construction: indices are stored in separate directories per client (and optionally per project). This separation reduces the search space and mitigates the risk of cross-tenant context leakage.

4.3 Context construction (query expansion, filters, deduplication, and metrics)

Final context assembly follows these steps:

1. **Attachments**: text extraction from attachments and construction of an `attachment_ctx` block;
2. **Query expansion**: generation of semantically equivalent variations of the original question;
3. **Vector search**: execution of Top-k search for each query variant across all available indices;
4. **Thresholding**: removal of candidates with normalized scores below a minimum value;
5. **Contextual filtering**: prioritization or filtering based on metadata (e.g., source type and recency);
6. **Deduplication**: removal of redundant excerpts based on high textual similarity;
7. **Reranking**: reordering of candidates using heuristic and/or LLM-based rerankers prior to prompt assembly;
8. **Quality metrics**: computation of lightweight signals (e.g., coverage and diversity) aggregated into `quality_metrics`;
9. **Construction of `full_ctx`**: concatenation of conversation history (windowed), attachments, and selected excerpts into the final context passed to the language model.

4.3.1 Coverage and diversity (lightweight metrics)

To audit the retrieved set, two simple metrics are considered.

Query term coverage. Let $T(q)$ denote the normalized terms of the query and $T(D)$ the terms of the concatenated retrieved excerpts:

$$\mathrm{cov}(q, D) = \frac{|T(q) \cap T(D)|}{|T(q)|}$$

Inter-excerpt diversity. For retrieved excerpts $\{d_i\}_{i=1}^{k}$ with normalized embeddings $\hat{\mathbf{e}}_{d_i}$, diversity can be defined as:

$$\mathrm{div}(D) = 1 - \frac{2}{k(k-1)} \sum_{i<j} \hat{\mathbf{e}}_{d_i}^{\top} \hat{\mathbf{e}}_{d_j}$$

4.4 API exposure and evidence auditing

The API exposes an endpoint compatible with `/v1/chat/completions`, receiving messages in the Chat Completions format. The pipeline extracts the user's latest question, assembles the context, and invokes the configured language model. When enabled, the service returns evidence (sources and quality metrics), allowing operational auditing and inspection.

> **Note:** This section describes the pipeline as implemented in the project and does not claim quantitative performance gains.

4.5 Flow diagram (ingestion → retrieval → answer)

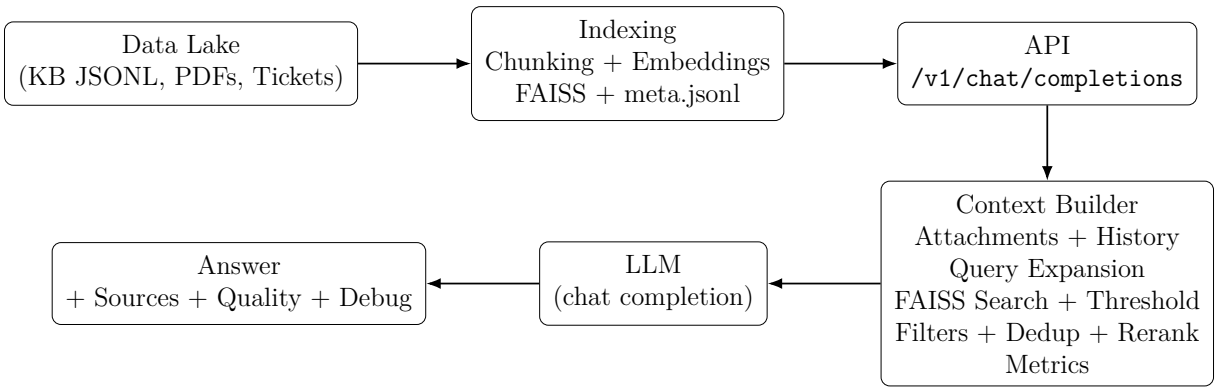

Figure 1: Operational flow of *Service Business RAG*: consolidated data in a data lake, FAISS indexing per tenant/domain, and API-based serving with auditable evidence.

5 Related work

5.1 Query Expansion

Query Expansion is a classic technique in Information Retrieval used to mitigate the *vocabulary mismatch* problem, where different terms can express the same semantic intent. In Retrieval-Augmented Generation (RAG) pipelines, this problem becomes particularly relevant because short or underspecified queries may limit the retrieval of adequate context.

Traditional approaches based on *Pseudo-Relevance Feedback* (PRF) expand the query using initially retrieved documents, but they are sensitive to noise when early results are not relevant. Recent work explores the use of *Large Language Models* (LLMs) to generate semantically equivalent expansions directly from the query, leveraging the model's latent knowledge while reducing dependence on the initial corpus [4].

However, LLM-generated expansions may introduce irrelevant content or hallucinations and increase computational cost. To address these limitations, methods such as *Progressive*

Query Expansion combine controlled expansion with iterative relevance assessment, reporting consistent gains in retrieval efficiency and quality, especially under cost constraints or when accessing data sources via APIs [3].

In this work, we adopt a *query expansion* strategy based on semantically equivalent reformulations of the original query generated by an LLM, aiming to enlarge the semantic search space explored by the retriever without changing the user's informational intent. This approach is compatible with both lexical and dense retrievers and aligns with recent evidence that controlled, semantically consistent expansions improve retrieval robustness [5].

5.2 Reranking

In Retrieval-Augmented Generation (RAG) systems, *reranking* plays a critical role by refining the candidates retrieved in the first stage and mitigating the inclusion of irrelevant documents that can degrade generation quality. Recent studies show that poorly ranked documents not only reduce factual accuracy but also contribute to *lost-in-the-middle* effects, where relevant information is diluted within long contexts [6].

Modern reranking approaches based on *Large Language Models* (LLMs) can yield substantial quality improvements by leveraging deeper semantic capabilities. However, purely LLM-based reranking often incurs high computational cost or relies on static selection strategies such as a fixed Top-k, which may not adapt to query complexity [6]. In contrast, recent findings indicate that signals derived from generation quality can serve as indirect feedback to improve reranking decisions, reinforcing the value of context-aware relevance assessment [6].

In this work, we adopt a hybrid reranking strategy that combines lightweight lexical-overlap heuristics with an optional LLM-based reranker. This choice is motivated by evidence that heuristic rerankers provide an efficient initial filter, while LLM-based rerankers deliver stronger semantic precision when applied to a reduced candidate set [7]. This design also aligns with results showing that efficient rerankers can remain competitive while reducing the volume and complexity of processed information [7].

Moreover, explicitly separating fast heuristics from semantic reranking provides operational flexibility, enabling dynamic tuning of cost versus quality depending on the query context. This architectural choice improves pipeline robustness and fits practical RAG scenarios, where efficiency, cost control, and generation quality must be carefully balanced.

5.3 Contextual filtering, deduplication, and quality analysis

After vector search, candidate documents are filtered using metadata-based contextual constraints, including source type and recency. Next, similarity-based deduplication reduces redundancy and prevents the inclusion of near-duplicate excerpts in the final context.

We also compute lightweight quality metrics for the retrieved set, including query-term coverage and document diversity, enabling inspection and auditability of the retrieval process.

Unlike work primarily focused on algorithmic gains, this paper emphasizes architectural and governance decisions for multi-tenant RAG systems with operational traceability.

6 Data sources and ingestion

The system assumes the existence of a corporate *data lake* as a storage layer ("a data lake is simply storage") [8]. This work does not treat ingestion as a research problem; instead, ingestion is considered an operational step to enable indexing and retrieval.

The pipeline consumes: (i) knowledge bases (e.g., wikis in JSONL), (ii) technical documentation in PDF, and (iii) support history (tickets).

6.1 Data lake organization and sources

The *data lake* organizes data by source type and by client (*tenant*), enabling logical separation of content across different business contexts. The main sources consumed by the pipeline are:

- **Knowledge bases**: content extracted from corporate wiki systems, stored in JSONL format.
- **Technical documentation**: manuals and guides in PDF format, organized per client.
- **Support history**: tickets extracted from ITSM/CRM systems, containing full descriptions and basic metadata.

These sources are treated as immutable inputs; any update requires explicit rebuilding of the corresponding indices, preserving governance and auditability.

6.2 Content normalization and segmentation

Before indexing, documents undergo textual normalization, including removal of structural markup (HTML, Markdown), cleanup of visual artifacts, and whitespace standardization. Text is then segmented into smaller units (*chunks*) with controlled size and overlap between consecutive segments.

This strategy balances granularity and context preservation, reducing the risk of semantic loss during retrieval without relying on external semantic structures or supervised learning.

Each *chunk* is associated with minimal metadata, including source type, client, origin identifiers, and position in the original document. These fields are later used for contextual filtering and auditing.

6.3 Embedding generation and vector indexing

After segmentation, *chunks* are mapped into dense vectors using a pretrained embedding model. Vectors are normalized and inserted into independent vector indices per client and per logical domain (e.g., knowledge base, support), using an inner-product similarity structure.

This separation enables:

- logical isolation across clients;
- scope control during retrieval;

- reduced search space and computational cost.

Vector indices are persisted together with their metadata, enabling full reconstruction of retrieved context and tracking of sources used in each answer.

6.4 Scope and limitations

The described ingestion pipeline focuses on operationalizing *Service Business RAG* and does not explore advanced techniques such as semantic enrichment, incremental learning, or adaptive indexing optimization. These extensions are considered out of scope and may be addressed as future work.

7 Operational retrieval data model

Service Business RAG adopts a strictly operational data model, defined exclusively by the needs of context retrieval and auditing of evidence used in generated answers. The system does not employ classical analytical modeling (e.g., fact and dimension tables) nor supervised learning structures.

The fundamental unit is the text *chunk*, treated as an atomic retrieval entity. Each *chunk* is indexed and retrieved independently, enabling precise control over traceability, multi-client isolation, and contextual filtering.

7.1 Multi-tenant isolation

Isolation across clients (*tenants*) is enforced by construction through physical separation of vector indices and metadata into distinct directories. Each client maintains its own FAISS indices and metadata files, eliminating complex logical filters and reducing the risk of context leakage.

7.2 Multi-tenant routing and access control

Multi-tenant isolation in *Service Business RAG* is enforced by construction rather than by post-retrieval filtering. Each request is explicitly routed to a tenant-scoped retrieval context based on metadata resolved at the API layer (e.g., client identifier, project, or domain).

Vector indices and metadata files are physically separated per tenant and optionally per logical domain. As a result, retrieval operations are constrained to a bounded search space, preventing cross-tenant leakage and reducing computational overhead. Access control is therefore implemented as a combination of request routing and storage-level separation, eliminating reliance on complex logical filters during retrieval.

7.3 Persisted artifacts

Indexing produces two primary artifacts per *tenant* (and optionally per logical domain):

- `faiss.index`: the vector index used for similarity search;
- `meta.jsonl`: textual records containing *chunk* content and minimal metadata.

7.4 Metadata fields

Each record in `meta.jsonl` contains only information directly used by the retrieval pipeline:

- logical client identifier (*tenant*);
- source type (*kb*, *pdf*, *ticket*);
- source identifier or path;
- *chunk* position in the original document;
- *chunk* textual content.

These metadata fields are used exclusively for contextual filtering, deduplication, reranking, and answer auditing, without reliance on analytical schemas or external semantic enrichment.

7.5 Audit trail and evidence schema

To support traceability and operational auditing, each retrieved *chunk* can be mapped back to its origin through a minimal evidence schema. Metadata are persisted alongside vector indices and exposed optionally at inference time.

A simplified example of the persisted schema is shown below:

```
{
  "tenant": "client_a",
  "source_type": "kb",
  "source_id": "operations/onboarding.md",
  "chunk_id": 42,
  "offset": [1280, 1536],
  "text": "All client onboarding requests must be approved..."
}
```

This schema enables reconstruction of the retrieval path used to generate an answer, supporting debugging, compliance verification, and human inspection without requiring external logging systems or analytical schemas.

8 Evaluation

This paper does not aim to report final quantitative results. Evaluation is treated as observational and operational, based on internal retrieval process metrics (coverage, diversity, and latency) collected via logging. In addition, we propose configuration-based comparisons (with vs. without Query Expansion) to inspect cost–quality trade-offs without claiming universal improvements.

These metrics characterize pipeline behavior and enable analysis of trade-offs between computational cost, coverage, and context quality without asserting absolute performance gains. The choice of an observational evaluation aligns with the paper scope, which focuses on architecture and operationalization rather than model benchmarking or statistical validation.

8.1 Experimental protocol

Although this work does not report final quantitative results, we outline an experimental protocol suitable for evaluating the proposed pipeline in operational environments.

The protocol assumes multiple tenants with heterogeneous sources (knowledge bases, technical documentation, and support tickets). Typical evaluation tasks include procedural lookup, incident resolution, and validation of client-specific rules.

Metrics collected through instrumentation include retrieval latency (P50/P95), number of retrieved chunks per query, similarity-based duplication rate, and query-term coverage over the final context. Comparisons are performed across pipeline configurations (e.g., with vs. without query expansion or reranking) to analyze cost–quality trade-offs rather than absolute performance.

This protocol reflects the practical constraints of service operations, where controlled benchmarks are often secondary to reliability, traceability, and operational efficiency.

9 Conclusion

This paper presented *Service Business RAG* as a practical pattern for multi-client service environments, featuring: (i) per-tenant and per-domain isolated vector indices, (ii) JSONL-based metadata for traceability, (iii) a retrieval pipeline with query expansion, thresholding, contextual filtering, deduplication, and optional reranking, and (iv) an API-based interface returning auditable evidence and quality metrics.

As a complement, we proposed an operational evaluation approach based on system instrumentation, enabling plots and controlled comparisons across execution modes (with vs. without Query Expansion) without inferring quantitative results in the absence of measurement.

As a limitation, this work does not claim absolute quantitative gains nor compare retrieval models, focusing instead on architectural and operational decisions.

References

[1] R. Kimball and M. Ross. *The Data Warehouse Toolkit*. Wiley, 3rd ed., 2013.

[2] D. Jurafsky and J. H. Martin. *Speech and Language Processing*. 3rd ed. (draft), 2023.

[3] M. S. Rashid, J. A. Meem, Y. Dong, and V. Hristidis. Progressive Query Expansion for Retrieval Over Cost-constrained Data Sources. *arXiv preprint* arXiv:2406.07136, 2024.

[4] R. Jagerman, H. Zhuang, Z. Qin, X. Wang, and M. Bendersky. Query Expansion by Prompting Large Language Models. *arXiv preprint* arXiv:2305.03653, 2023.

[5] S. Yao, P. Huang, Z. Liu, Y. Gu, Y. Yan, S. Yu, and G. Yu. ExpandR: Teaching Dense Retrievers Beyond Queries with LLM Guidance. *arXiv preprint* arXiv:2502.17057, 2025.

[6] J. Sun, X. Zhong, S. Zhou, and J. Han. DynamicRAG: Leveraging Outputs of Large Language Model as Feedback for Dynamic Reranking in Retrieval-Augmented Generation. *arXiv preprint* arXiv:2505.07233, 2025.

[7] H. Déjean and S. Clinchant. Reranking with Compressed Document Representation. *arXiv preprint* arXiv:2505.15394, 2025.

[8] J. Serra. *Decifrando Arquiteturas de Dados*. Novatec Editora Ltda., 2024. ISBN 978-85-7522-921-7.