CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

# Retrieval-Augmented Generation for Sustainable Material Data Handling in Automotive Value Chain

Master's thesis in Computer Science and Engineering

JOHN VICTOR ZHENG TRAN
DANIEL KLAS HENRY LARSSON

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2026

Master's thesis 2026

# Retrieval-Augmented Generation for Sustainable Material Data Handling in Automotive Value Chain

JOHN VICTOR ZHENG TRAN
DANIEL KLAS HENRY LARSSON

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2026

A Chalmers University of Technology Master's thesis template for LaTeX
JOHN VICTOR ZHENG TRAN
DANIEL KLAS HENRY LARSSON



Supervisor: Oana Geman, Department of Computer Science and Engineering
Supervisor: Adam Ek, AI Sweden
Examiner: Matti Karppa, Department of Computer Science and Engineering

Master's Thesis 2026
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2026

## Abstract

Applying large language models (LLMs) to industrial material data workflows has the potential to improve efficiency. However, conventional LLMs are limited by hallucinations, depend on proprietary training data, and are costly to update. This thesis explores Retrieval-Augmented Generation (RAG) as an alternative approach, in which an LLM generates responses grounded in a restricted, domain-specific corpus of documents and databases and provides source citations for them. The study is carried out in an industrial setting with two main data domains: an internal SQL materials database with tabular material properties, and a corpus of unstructured textual documents, including supplier documents, corporate standards, and environmental product declarations. A RAG system is developed that (1) indexes both textual and tabular data, (2) retrieves relevant chunks via dense vector search, and (3) generates source-grounded responses.

The work investigates whether a RAG model that explicitly integrates both domains can outperform a baseline tuned for unstructured text and explores which tabular serialization format yields the most semantically informative embeddings for pretrained embedding models. To achieve this, we first constructed LLM-based pipelines to generate document- and table-based test sets with ground-truth chunk annotations, and implemented a modular RAG pipeline with separate indices for textual and tabular data. Then, we experimented with multiple retrieval strategies, ranging from concatenating the retrieval results to using cross-encoders to weigh them. In addition, several fusion strategies were tested to evaluate whether they could improve retrieval accuracy when operating across different domains. Experiments are conducted comparing nine tabular serialization strategies, studying performance as a function of index size, chunk size, and top-k, and evaluating different fusion modes and embedding models. The evaluation metrics used are Hit Rate, Recall, Precision, F1-score, and Mean Reciprocal Rank.

Results show that enriched serialization, which converts tabular rows into natural-language statements, yields stronger tabular retrieval performance than a standard key-value-based format, without degrading performance on document retrieval. Larger chunk sizes and higher top-k values systematically improve retrieval metrics, highlighting both the difficulty of relying solely on similarity search and the benefits of cross-encoder reranking on larger candidate sets. A domain-aware weighted fusion retriever further improves overall retrieval performance over the optimized baseline with only moderate computational overhead. These findings demonstrate that semantically rich tabular representations and domain-aware fusion can enhance RAG performance on heterogeneous industrial material data.



Keywords: Sustainable Material Selection, Retrieval-Augmented Generation (RAG), Applied Artificial Intelligence, Information Retrieval Systems, Large Language Models, Multi-domain RAG

## Acknowledgements

We would like to express our sincere gratitude to Oana Geman and Matti Karppa for their guidance and support as supervisor and examiner at Chalmers. Their feedback and academic insight have been invaluable throughout this thesis work.

We would also like to give a special thank you to our supervisor at AI Sweden, Adam Ek, for his continuous support and deep expertise in the subject area. His guidance has been essential in shaping both the direction and quality of this work.

We would also like to thank Helena Theander and Katarina Klevell for their engaging support and for enabling our work to become a meaningful and impactful contribution to the joint project.

## Abbreviations

| | |
|---|---|
| **AI** | Artificial Intelligence |
| **ANN** | Approximate Nearest Neighbour |
| **CMD** | Common Materials Database |
| **CSD** | Corporate Standards Database |
| **CSV** | Comma-Separated Values |
| **DOCX** | Microsoft Word Open XML Document |
| **EPD** | Environmental Product Declaration |
| **GPU** | Graphics Processing Unit |
| **HTML** | HyperText Markup Language |
| **JSON** | JavaScript Object Notation |
| **LLM** | Large Language Model |
| **MRR** | Mean Reciprocal Rank |
| **NLP** | Natural Language Processing |
| **PDF** | Portable Document Format |
| **QA** | Question Answering |
| **RAG** | Retrieval-Augmented Generation |
| **RRF** | Reciprocal Rank Fusion |
| **SQL** | Structured Query Language |
| **UUID** | Universally Unique Identifier |
| **XML** | eXtensible Markup Language |
| **YAML** | YAML Ain't Markup Language |

# Contents


**1 Introduction** **1**
1.1 Industrial Background . . . 1
1.2 Technical Background and Research Context . . . 2
1.3 Problem Statement . . . 2
1.4 Research Questions . . . 3
1.5 Limitations . . . 3

**2 Literature Review** **5**
2.1 Architectures for Retrieval-Augmented Generation . . . 5
2.2 RAG over Structured Data . . . 6
2.3 Hybrid Retrieval for Structured and Unstructured Data . . . 7
2.4 Multi-Modal and Multi-Corpus RAG . . . 7
2.5 Summary and Research Gap . . . 8

**3 Theory** **11**
3.1 Representation Learning and Embeddings . . . 11
3.1.1 Vector Space Semantics . . . 11
3.1.2 Embedding Models . . . 12
3.1.3 Representations of Tabular Data . . . 14
3.2 Dense Retrieval . . . 15
3.2.1 Dense Representation for Retrieval . . . 15
3.2.2 Bi-Encoder Architecture . . . 15
3.2.3 Approximate Nearest Neighbour Search . . . 16
3.2.4 Limitations of Dense Retrieval . . . 17
3.3 Retrieval-Augmented Generation . . . 17
3.3.1 Document Chunking and Context Representation . . . 18
3.3.2 Fusion Retrieval from Multiple Sources . . . 19
3.3.3 Reranking Models . . . 21
3.4 Evaluation Metrics . . . 22

**4 Methodology** **25**
4.1 Data . . . 26
4.1.1 Data Domains and Sources . . . 26
4.1.2 Data Preparation . . . 27
4.1.3 Data Preprocessing . . . 28
4.2 User Study . . . 29
4.3 Test Set Generation Pipeline . . . 29
4.3.1 Document-Based Question Generation . . . 29

4.3.2 Table-Based Question Generation . . . 32
4.4 Retrieval-Augmented Generation Pipeline . . . 33
4.4.1 Pipeline Overview . . . 33
4.4.2 Model and Retrieval Parameter Configuration . . . 33
4.4.3 Embedding Model . . . 34
4.4.4 Query Processing . . . 34
4.4.5 Retrieval Methods . . . 34
4.4.6 Reranking . . . 36
4.4.7 Response Generation . . . 37
4.5 Evaluation . . . 37
4.5.1 Evaluation Metrics . . . 37
4.5.2 Evaluation Pipeline . . . 37
4.6 Experimental Setup . . . 38
4.6.1 Tabular Serializer Performance . . . 38
4.6.2 Vector Index Scaling . . . 39
4.6.3 Chunk Size and top-k Influence . . . 39
4.6.4 Fusion Modes . . . 39
4.6.5 Weighted retriever . . . 39
4.6.6 Retrieval Methods . . . 39

**5 Results** **41**
5.1 Tabular Serializer Performance . . . 41
5.2 Vector Index Scaling . . . 42
5.3 Chunk Size and Top-k Influence . . . 43
5.4 Fusion Modes . . . 44
5.5 Weighted Retriever . . . 45
5.6 User studies . . . 47

**6 Discussion** **49**
6.1 Tabular Serializer Performance . . . 49
6.2 Vector Index Scaling . . . 50
6.3 Chunk Size and top-$k$ Influence . . . 50
6.4 Fusion Modes . . . 50
6.5 Weighted Retriever . . . 50
6.6 Usability . . . 51
6.7 Future Work . . . 52
6.8 Conclusion . . . 52

**Bibliography** **55**

**A Appendix - User studies** **I**
A.1 User study questions . . . I
A.2 User study responses . . . II

**B Tabular conversion dictionaries** **III**

**C Appendix - Table serialization** **V**

**D Appendix - System prompts** **VII**
D.1 Document-question generation . . . VII
D.2 Table-question generation . . . VIII

D.3 Response generation . . . . . IX

**E Appendix - Results** **XI**
E.1 Vector Index Scaling Results . . . . . XI
E.2 Chunk Size and Top-k experiment results . . . . . XII

# 1

# Introduction

Applying the computational power of Large Language Models (LLMs) to the task of industrial material data handling has the potential to both improve efficiency and allow for automated integration of sustainability indicators. Today, standard LLM implementations have persisting issues such as hallucinations and costly model updates, even with fine-tuning methods. A potential solution is the implementation of Retrieval-Augmented Generation (RAG), in which an LLM's response generation is restricted to and augmented by a specific knowledge corpus, while also providing references to the sources it ends up using. This provides a valuable opportunity as a tool for industry actors, where extensive material databases are subject to conventionally time-consuming handling processes.

In this thesis, we develop a modular RAG architecture capable of handling multi-domain data sources, specifically integrating structured databases and unstructured documents into a unified retrieval pipeline. As tabular retrieval poses the more difficult domain task, methods for representing structured material data are explored in this work. Given the limited availability of annotated industry data, emphasis is also placed on designing evaluation strategies that remain robust under constrained data conditions.

## 1.1 Industrial Background

The resource intensiveness of the automotive industry implies a responsibility in material selection for its components, which is reflected by increasing directives at the EU level [1] and strengthened internal corporate policies driven by the desire to be a 'green' company. Meanwhile, material selection within product development processes is a knowledge-intensive task involving trade-offs between environmental, durability, and economic factors. A wide range of professionals, here referred to as *personas*, are involved in material selection. *Persona* roles range from designers designing for circularity, material experts choosing materials that fulfill requirements on hazardous substances and material properties, and purchasers assessing and including new sustainable materials. Although all *personas* have different knowledge needs about the materials, the necessity of reliable and up-to-date information is shared. This is traditionally obtained by manual searches through company databases and extensive communication with suppliers. There is an expressed growing need for automated processing to keep up with the demands of modern development processes.

To address these emerging industry needs, a collaboration project has been initiated with the automotive manufacturer Volvo Group and the national center for applied AI, AI Sweden, as key partners. The automotive company provides all components of the use-case, including a material database, external sustainability-related supplier documents, software resources, as well as the perspective and input from the various personas. AI Sweden contributes with domain knowledge in the field of data science, in particular with expertise

in implementation of RAG models. This thesis contributes to the project by constructing and evaluating the performance of a RAG system. The system specifically aims to integrate sustainability data with regular physical properties data, and the model should take the form of an AI agent with a natural language interface, capable of providing all necessary information for the personas to accomplish their work tasks.

## 1.2 Technical Background and Research Context

To meet industry needs, implemented solutions must operate reliably on large, heterogeneous material datasets, expose this information through a natural language interface, and while adhering to restrictions on traceability, confidentiality, all while being efficient and keeping response times at a minimum. In addition, material selection processes often require combining descriptive documentation with structured material data stored in relational databases. Effective decision support therefore depends on retrieval mechanisms capable of reasoning across both textual explanations and quantitative specifications.

LLMs have introduced new possibilities for industrial data workflows by enabling natural language interaction with complex data sources, thereby lowering technical barriers and improving efficiency [2]. However, despite their versatility, LLMs exhibit fundamental limitations. Their outputs are not inherently interpretable, updating their internal knowledge requires costly retraining, and they are prone to hallucinations, where generated information is not grounded in verifiable sources [3]. These characteristics make standalone LLMs unsuitable for high-risk industrial applications such as operating over large corporate databases.

RAG has emerged as a technique to address these challenges. By incorporating an external retrieval mechanism, RAG enables LLMs to condition their responses on explicitly retrieved documents or database entries. This grounding mechanism improves factual reliability and has been shown to reduce hallucinations compared to standard LLM usage [4, 5]. As a result, RAG systems present significant potential value for industrial knowledge access, bridging the gap between industrial data workflows and trustworthy AI-assisted decision support.

A significant challenge still arises in domains where structured and unstructured data must be jointly utilized. While modern retrieval methods perform well on free-form text, structured data such as tables encode meaning differently, through rows, columns, hierarchical relations, and numerical values. The semantic relationships within tabular data are therefore not always captured effectively by standard text-based embedding approaches. Integrating these heterogeneous data modalities into a unified retrieval pipeline remains a non-trivial problem.

## 1.3 Problem Statement

The central issue being investigated in this thesis is how RAG systems can be adapted to effectively handle information originating from two fundamentally different data domains. In industrial contexts, material data is typically stored in structured formats such as SQL databases containing material properties and performance indicators measured in laboratories. In contrast, sustainability-related supplier data often exists in unstructured forms, including PDF documents, PowerPoint presentations, and even structured CSV reports. These two domains differ not only in structure and format, but also in the type of

reasoning required to extract relevant information.

Merging these heterogeneous sources within a single RAG framework presents a significant technical challenge. Traditional RAG models are primarily designed for purely textual data and therefore rely on embedding and retrieval mechanisms that assume unstructured input. Structured, relational data such as SQL tables cannot be directly represented using the same methods without substantial loss of contextual or relational information. At the same time, methods optimized for tabular reasoning do not directly address free-form text retrieval or document understanding. In addition to the representational challenge, there are also practical demands related to the database and model interaction. Industrial material databases are dynamic and frequently updated as new laboratory results or sustainability reports from suppliers become available. For a RAG system to remain reliable in an industrial application, its embedding index must be able to update, allowing new or modified entries to be periodically re-encoded and incorporated into the vector database. Ensuring this kind of update mechanism in the RAG system introduces further complexity, but is essential for real-world deployment.

## 1.4 Research Questions

This work aims to develop a RAG model that integrates two distinct domains in which one contains a structured SQL material database, while the other one includes both structured and unstructured supplier data. The model generates responses exclusively based on the provided data, ensuring factual grounding and source transparency by including references for each answer. Based on these objectives, the thesis investigates the following research questions:

*Q1: Can a RAG model be implemented to achieve higher retrieval performance than a standard baseline RAG in the context of multi-domain (tabular and textual) retrieval?*

*Q2: Given the feasibility of Q1, what tabular data formatting yields the embeddings providing highest retrieval scores for an embedding model primarily pre-trained on text corpora?*

In providing evidence criteria for the fulfillment of said research objectives, selected metrics are used to define baseline performances and the accomplished performance of the developed RAG system. Additionally, a limited user study including testing and reviewing of the model with a selection of *personas* will be performed, intended to qualitatively evaluate whether the model could successfully be implemented to accomplish the intended use-case.

## 1.5 Limitations

This thesis tackles a breadth of tasks involved with building a RAG system, stretching between technical implementation, user-needs adaptation, and data architecture. Each factor is subject to time and resource constraints related to the conditions of the thesis. The main limitations are related to *persona* adaptation and data readiness.

The number and diversity of user personas included during the project affects the learning opportunities and final usability of the system. The industrial context encompasses a wide range of roles, each with distinct objectives and information needs when interacting with material data. The time budget for user studies, as well as the number of employees that could participate, is limited and includes four people, each with a different role, such that they represent a specific predefined *persona* each. This means that feedback and insights

may be narrow and associated only with that specific person or role, and less representative for the wider *persona* description targeted.

Another limitation lies in the restriction on what types of data the system can handle. While some multimodal RAG systems are capable of processing video, audio, and image data, this thesis focuses exclusively on textual and tabular data. This decision reflects both time constraints and the need for a controlled experimental setup. However, the high variability of industrial document formatting makes achieving sufficient accuracy for arbitrary tables and figures a considerable technical challenge and beyond the main scope of this work.

Each component of a RAG system, including the embedding model, retrieval mechanism, and generation module, represents a substantial research area on its own. To accomplish the objectives, this thesis focuses on enabling and evaluating the integration of textual and tabular modalities rather than extensively optimizing every component as a subsystem. Topics like indexing structures, re-ranking strategies, or prompt optimization are only addressed to the extent necessary to achieve the main objective. They thus receive default choices which have limited tuning and adjustment as well as no systematic ablation.

The performance of the system further depends on the quality, consistency, and representativeness of the available datasets. Industrial material data is often fragmented, inconsistently formatted, and highly confidential, which may limit both training coverage and the model's ability to generalize to unseen cases. System evaluation is additionally challenged by the absence of established benchmarks for material decision-making tasks.

Ideally, evaluation data would be constructed and validated in collaboration with domain experts to ensure high fidelity and relevance. However, due to time constraints, it was not feasible within the scope of this thesis to involve expert users in the manual creation and annotation of evaluation data. Instead, synthetic evaluation data generated using LLMs is employed. While this approach enables scalable and consistent evaluation, it may not fully capture the nuances of real industrial decision-making processes. Consequently, potential biases and limitations introduced by synthetic data generation should be considered when interpreting the reported results.

# 2

# Literature Review

As described in Chapter 1, RAG systems offer a promising approach to enable reliable interaction with large industrial knowledge sources. However, many practical applications involve heterogeneous data sources that combine structured databases with unstructured documents. Integrating these different forms of information within a single retrieval pipeline remains a challenging research problem.

Therefore, recent research has explored extensions of the RAG paradigm that enable retrieval and reasoning across structured tables, relational databases, and heterogeneous document collections. Approaches differ in terms of how structured information is being represented, how multi-modal retrieval is approached, and how evidence from multiple sources is combined prior to generation.

This chapter reviews research directions addressing these challenges. First, work on general RAG systems is briefly discussed to illustrate the broader research landscape. The chapter then examines approaches designed for structured data retrieval, followed by methods that integrate structured and unstructured information within hybrid document settings. Additionally, research on multi-modal and multi-corpus RAG frameworks is reviewed. Finally, the state of the research field is summarized and the identified research gap is formulated, which is addressed by this thesis.

## 2.1 Architectures for Retrieval-Augmented Generation

RAG was originally introduced by Lewis et al. as a framework for combining neural retrieval with conditional text generation for knowledge-intensive tasks [6]. In this architecture, a retriever first identifies relevant documents from an external knowledge corpus, after which a sequence-to-sequence language model generates a response conditioned on the retrieved information.

The idea of augmenting language models with external retrieval mechanisms builds upon earlier work such as REALM [7], which incorporates neural retrieval into language model pretraining. Later architectures further explored large-scale retrieval integration, including RETRO [8], which retrieves relevant text chunks from extremely large corpora during generation. More recent systems such as Atlas [9] demonstrate the effectiveness of retrieval-augmented models for knowledge-intensive question answering tasks.

Recent research has continued to extend the RAG paradigm by improving retrieval control and reliability. For example, Self-RAG [10] introduces mechanisms that allow the model to dynamically decide when retrieval is necessary and critique its own generated responses. Related work such as ReAct [11] integrates retrieval with iterative reasoning and external tool usage, enabling language models to gather information during multi-step problem

solving. Surveys of the field highlight retrieval quality, information integration, and robustness as key challenges for the continued development of RAG systems [3].

These approaches illustrate several strategies for integrating retrieval into language models. REALM [7] incorporates retrieval during pretraining, allowing models to learn representations grounded in external knowledge. The original RAG architecture instead separates retrieval and generation into modular components. RETRO [8] integrates retrieval more deeply into the decoding process by retrieving relevant text during generation, while systems such as Atlas [9] demonstrate the effectiveness of retrieval-augmented models on downstream knowledge-intensive tasks. More recent approaches, including Self-RAG and ReAct [11], introduce mechanisms that allow models to dynamically control when retrieval should occur and how retrieved information is incorporated during reasoning. Together, these developments reflect a shift from static retrieval pipelines toward more adaptive retrieval mechanisms in which retrieval becomes an interactive component of the generation process.

## 2.2 RAG over Structured Data

Integrating structured data into language model pipelines presents challenges that differ from those encountered in traditional text retrieval. Early research focused on enabling neural models to directly reason over tabular data. For example, TAPAS [12] extends transformer architectures to operate on structured tables, allowing question answering over tabular inputs without explicitly generating database queries.

Another line of work focuses on translating natural language queries into executable database queries. Text-to-SQL models such as RAT-SQL [13] generate SQL statements conditioned on database schema representations, enabling precise retrieval of structured information from relational databases.

More recent work has begun integrating these capabilities within RAG pipelines. Systems such as StructRAG [14] introduce structure-aware retrieval mechanisms that incorporate schema information when retrieving relevant table entries. Similarly, frameworks such as DB-GPT [15] enable natural language interaction with relational databases by combining SQL generation with LLM-based reasoning.

Building on these developments, Yu et al. propose a RAG architecture named TableRAG [16], specifically designed for structured databases. Instead of embedding serialized tables into a shared vector space, the framework treats relational data as a symbolic knowledge source and translates natural language queries into executable SQL statements. The retrieved rows are then provided to the language model as grounded context for response generation. By leveraging the database engine for operations such as filtering, aggregation, and joins, TableRAG enables precise retrieval of structured information that may be difficult to capture through purely semantic similarity search. The authors further argue that this design better preserves relational structure compared to approaches that flatten tables into text before embedding. Experimental results show improved accuracy on table question answering tasks compared to baseline RAG pipelines that rely on textual representations of tabular data. However, such approaches assume direct access to structured database schemas and query execution capabilities, which may not always be available when tabular information is embedded within heterogeneous document collections.

These models demonstrate different strategies for integrating structured data into language model pipelines. Neural table reasoning models such as TAPAS [12] attempt to directly

encode tabular structure within neural architectures. In contrast, text-to-SQL systems such as RAT-SQL [13] treat relational databases as symbolic knowledge sources that can be queried through executable SQL statements. More recent frameworks including StructRAG [14], DB-GPT [15], and TableRAG [16] extend these ideas by integrating database querying capabilities within RAG pipelines. While these approaches improve the ability of language models to interact with structured data, they often assume direct access to database schemas and query execution mechanisms, which may limit their applicability in settings where structured data must be integrated with heterogeneous document collections.

## 2.3 Hybrid Retrieval for Structured and Unstructured Data

In many real-world knowledge bases, structured and unstructured information coexist and must be jointly utilized to answer user queries. For example, technical documentation may contain descriptive text alongside tabular data, while relational databases are often complemented by external reports or supplier documents. Retrieving relevant information in such environments therefore requires systems capable of integrating evidence across both structured and unstructured sources.

Early research addressing this challenge focused on hybrid question answering tasks that require reasoning over both tables and textual documents. HybridQA [17] introduced a benchmark dataset in which answering a question may require retrieving information from a table and following hyperlinks to supporting textual passages. The results demonstrate that systems optimized for either table reasoning or text retrieval alone struggle to effectively combine evidence across these heterogeneous sources.

More recent work has extended RAG architectures to hybrid document settings. HD-RAG [18] proposes a framework specifically designed for documents containing both long-form text and hierarchical tables. The system preserves table structure through a hierarchical representation and employs a two-stage retrieval pipeline that combines traditional retrieval methods with LLM-based document selection. Experimental results show improved performance compared to baseline RAG pipelines that rely on flattened textual representations of tables.

Taken together, these studies highlight different aspects of hybrid retrieval. HybridQA [17] emphasizes the difficulty of reasoning across tabular and textual information, while HD-RAG [18] focuses on hybrid documents that contain both modalities within the same document structure. However, these approaches typically assume that structured and unstructured information are available within a shared document context. In many practical settings, including industrial knowledge bases, structured data and textual documentation are instead stored in separate repositories. As a result, integrating these heterogeneous sources within a unified retrieval pipeline remains an open challenge.

## 2.4 Multi-Modal and Multi-Corpus RAG

As RAG systems are increasingly applied to heterogeneous knowledge bases, recent research has explored mechanisms that allow models to dynamically select between multiple information sources. In such settings, different corpora or modalities may require distinct retrieval strategies, making static retrieval pipelines insufficient.

UniversalRAG [19], proposed by Yeo et al., introduces a routing-based architecture designed to retrieve knowledge across multiple modalities and corpus granularities. Instead of

embedding all information into a single shared representation, the framework maintains modality-specific corpora and employs a router that predicts the most appropriate modality and retrieval granularity for a given query.

Related work has explored similar ideas of dynamic retrieval control. Self-RAG [10] enables language models to decide whether retrieval is necessary during generation and to critique their own responses. Meanwhile, Toolformer [20] demonstrates that language models can learn to invoke external tools, including search systems, suggesting a broader paradigm in which retrieval sources can be dynamically selected during reasoning.

Collectively, these approaches reflect a shift toward more adaptive RAG architectures in which retrieval is treated as a controllable component of the generation process rather than a fixed preprocessing step. Such mechanisms are particularly relevant in heterogeneous knowledge environments where multiple corpora or modalities must be accessed dynamically.

## 2.5 Summary and Research Gap

The reviewed literature demonstrates the rapid development of RAG systems and their extension beyond traditional text-based retrieval tasks. Early work focused on integrating retrieval mechanisms within language models, progressing from architectures that incorporate retrieval during pretraining, such as REALM [7], to modular pipelines combining retrievers and generators as in RAG. More recent developments have explored adaptive retrieval mechanisms that dynamically control when and how external knowledge sources are accessed, as demonstrated by systems such as Self-RAG [10] and Toolformer [20].

Research has also investigated how structured data can be integrated into language model pipelines. Neural table reasoning models such as TAPAS [12] attempt to encode tabular structures directly within neural architectures, while text-to-SQL approaches such as RAT-SQL [13] translate natural language queries into executable database queries. More recent frameworks including StructRAG [14], DB-GPT [15], and TableRAG [16] extend these ideas by integrating database querying capabilities into RAG pipelines. In parallel, work on hybrid retrieval tasks, such as HybridQA [17] and HD-RAG [18], highlights the challenges of combining tabular and textual information within a single reasoning process. Furthermore, recent multi-modal and multi-corpus RAG architectures introduce routing mechanisms that dynamically select the most relevant knowledge source among multiple corpora or modalities.

Despite these advances, several challenges remain when applying RAG systems in real-world industrial environments. Much of the existing research assumes well-structured datasets, clearly defined schemas, or hybrid documents where tabular and textual information coexist within the same context. In practice, however, industrial knowledge bases are often fragmented across multiple heterogeneous repositories. Structured data such as material properties may be stored in relational databases, while complementary information such as sustainability reports, regulatory requirements, or supplier documentation exists as unstructured textual documents. Establishing reliable cross-references between these different sources therefore remains a non-trivial task.

Another challenge concerns the availability and quality of data for system development and evaluation. Industrial datasets are often exclusive, sparsely annotated, and not originally designed for machine learning or RAG-based retrieval systems. Creating high-quality annotated benchmarks for such data can be overly expensive, limiting the ability to train

or fine-tune models on domain-specific tasks. Consequently, many practical applications must rely on pretrained models and limited domain-specific data, raising questions about the feasibility and reliability of RAG systems in such constrained settings.

These challenges highlight a gap between existing research on RAG architectures and the requirements of industrial knowledge environments. In particular, there remains limited work investigating how RAG systems can effectively integrate structured relational databases with domain-specific document collections within a unified retrieval pipeline. Furthermore, evaluation strategies for such systems remain underexplored when only limited annotated data is available.

This thesis addresses these challenges by investigating a modular RAG architecture designed to operate across heterogeneous industrial data sources. Specifically, the work explores how structured material data stored in relational databases can be retrieved and combined with domain-specific document corpora, such as sustainability and requirements documentation. Additionally, the thesis examines how different tabular data serialization strategies influence retrieval performance when using pretrained embedding models. Given the restricted availability of annotated industrial data, the study also evaluates the feasibility of constructing reliable evaluation pipelines under constrained data conditions.

Nevertheless, RAG remains an active research area with unresolved challenges. Recent evaluations show that modern RAG systems still struggle with negative rejection, information integration, counterfactual robustness, and robustness to noise [4]. While access to external corpora substantially improves performance compared to relying solely on pretraining [21], full reliability is not yet guaranteed. Successful applications in domains such as clinical healthcare demonstrate the promise of the approach [22], yet industrial deployment requires careful design, evaluation, and validation to meet the high standards of accuracy and trust demanded by corporate environments.

# 3

# Theory

This chapter presents the theoretical foundations required to understand the methodology and experiments in this thesis. It introduces the key concepts underlying RAG, dense vector embeddings, and modern information retrieval systems. Note that this chapter assumes a general knowledge of machine learning, but not any expertise within natural language processing.

The discussion begins with the principles of representation learning and semantic vector spaces, followed by an overview of classical and dense retrieval methods that motivate the use of vector databases. The chapter then outlines the theoretical formulation of RAG, explaining how retrieved context influences model predictions and why this approach is effective for factual, domain-specific tasks. Concepts related to document chunking, multi-source fusion retrieval, and reranking models are also summarized to provide the necessary background for the system architecture described later.

Finally, the evaluation metrics used throughout the thesis, such as ranking-based retrieval metrics and measures of response quality, are introduced. This chapter therefore establishes the essential framework on which the subsequent methodology and experimental analysis are built.

## 3.1 Representation Learning and Embeddings

As a foundation for the concepts explored in this chapter, we start by introducing representation learning, which is a key framework that enables machine learning models to encode the semantic content of raw data into vector spaces suitable for machine learning tasks such as retrieval and inference.

Formally, representation learning seeks to construct mappings:

$$f : \mathcal{X} \to \mathbb{R}^d,$$

where $\mathcal{X}$ corresponds to the input space and $\mathbb{R}^d$ is a continuous vector space of dimension $d$. The goal is to encode each input $\mathcal{X}$ into a vector $f(x)$, also referred to as an embedding, such that the geometric structure of the embedding space reflects the semantic structure of the data domain.

### 3.1.1 Vector Space Semantics

The construction of semantic vector spaces is based on the *distribution hypothesis*, which states that words that occur in similar contexts tend to have similar meanings [23]. This

hypothesis provides the theoretical motivation for embedding models, which seek to approximate semantic relatedness through geometric proximity in $\mathbb{R}^d$.

Formally, a semantic vector space can be defined as a metric space

$$\left(\mathbb{R}^d, d(\cdot,\cdot)\right),$$

where $d : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a distance function satisfying the properties of a metric.

In practice, similarity is often computed using functions such as cosine similarity:

$$\texttt{sim}(u, v) = \frac{u^\top v}{\|u\|\|v\|}, \tag{3.1}$$

which can be related to a distance measure such as cosine distance.

These similarity measures form the computational basis for nearest-neighbor retrieval, clustering, and alignment of heterogeneous data domains.

### 3.1.2 Embedding Models

Having established the geometric foundations of semantic vector spaces, we now turn to the models that learn these representations. Embedding models provide the parametric functions that map raw inputs into the continuous spaces described above.

Formally, an embedding model is a parameterized function:

$$f_\theta : \mathcal{X} \to \mathbb{R}^d,$$

where $\mathcal{X}$ corresponds to the input domain, $\mathbb{R}^d$ is a $d$-dimensional vector space and $\theta$ represents the learnable parameters of the model. The objective is to learn representations such that semantically similar inputs yield vectors that are near each other under a chosen similarity function. Based on Equation (3.1), this requirement can be expressed as:

$$x_i \sim x_j \Rightarrow \texttt{sim}\left(f_\theta(x_i), f_\theta(x_j)\right) \text{ is high.}$$

For textual inputs $x = (w_1, ..., w_n)$, transformer-based models such as those introduced by Vaswani et al. in *Attention is All You Need* [24], compute token embeddings using a learned matrix

$$\mathbf{W} \in \mathbb{R}^{V \times dim},$$

where $V$ is the vocabulary size and $dim$ represents the embedding dimension. Before applying the embedding matrix, the input sequence $x = (w_1, ..., w_n)$ consisting of textual units is processed by a tokenizer. The tokenizer maps each word or subword unit to a discrete integer index:

$$w_i \longrightarrow \mathrm{ID}(w_i) \in \{1, ..., V\}.$$

Therefore, the model does not operate on raw strings, but on a sequence of token IDS:

$$x = \left(\mathrm{ID}(w_1), ..., \mathrm{ID}(w_n)\right).$$

Each token embedding is thus computed as:

$$e_i = \mathbf{W}[\mathrm{ID}(w_i)]$$

where the operation selects the row of $\mathbf{W}$ associated with that token.

Transformers are inherently permutation-invariant and therefore require an additional mechanism to encode token order. Positional encodings address this by assigning each position $i$ a vector $p_i$ that is added to the token embedding $e_i$, which produces:

$$h_0^{(i)} = e_i + p_i.$$

These encodings may be fixed, such as the sinusoidal scheme introduced in [24], or learned parameters optimized during training. By incorporating positional structure, the model can form meaningful contextual embeddings.

After incorporating positional information, the sequence $\{h_0^{(i)}\}_{i=1}^n$ is processed by a stack of transformer layers. Each layer consists of a multi-head self-attention mechanism and a feed-forward subnetwork. Self-attention enables each token to incorporate information from all other tokens in the sequence. Given query, key and value matrices Q, K and V, the attention mechanism is defined as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V,$$

where $d_k$ denotes the key dimensionality used to scale the dot-product attention [24]. This operation produces contextualized representations that capture semantic relationships across the sequence. After L transformer layers, the final contextual representation is given by:

$$h_L = \text{Transformer}_\theta(h_0).$$

Since retrieval systems require a fixed-size vector representation, sequence-level embeddings are obtained by applying a pooling operation to the contextual token representations. Sentence-BERT [25] systematically evaluates several pooling strategies, including the *CLS-token embedding*, *mean pooling*, and *max pooling* methods, and demonstrates that mean pooling often yields the most robust semantic representations for retrieval.

Formally, let $h_L^{(i)}$ correspond to the contextualized embedding of the $i$-th token from the final transformer layer. Mean pooling computes the sequence-level embedding as the average of all token embeddings:

$$f_\theta(x) = \frac{1}{n}\sum_{i=1}^{n} h_L^{(i)}.$$

This operation aggregates all the information across all positions in the sequence, which produces stable and semantically smooth representation suitable for similarity-based retrieval.

Embedding models are central to RAG, as they provide the dense vector representations used during similarity search. Understanding how embeddings encode semantic relations is therefore crucial for interpreting retrieval performance and designing effective fusion strategies.

### 3.1.3 Representations of Tabular Data

Following the formulation of embedding models for unstructured text, we consider the problem of representing tabular data within the same embedding framework. This problem is central to RAG, where heterogeneous data sources are encoded into a common vector space to enable similarity-based retrieval [6].

Embedding models are predominantly trained on large-scale natural language corpora. As a consequence, tabular data presents unique challenges in RAG models, since its semantics are encoded not only in the content of individual cells, but also in the relationships between rows, columns, and headers.

Unlike unstructured text, tables exhibit an explicit schema that conveys meaning through column names, data types, and positional alignment. For example, a numerical value may represent entirely different concepts depending on the column header under which it appears, while relationships between rows often encode comparative or categorical information. Consequently, treating tabular data as a flat collection of text tokens can lead to a loss of structural semantics and overly abstract representations.

Formally, a table can be represented as a set of rows:

$$T = \{r_1, r_2, \ldots, r_m\},$$

where each row $r_i$ consists of attribute–value pairs:

$$r_i = \{(a_1, v_{i,1}), (a_2, v_{i,2}), \ldots, (a_k, v_{i,k})\}.$$

The attribute names $a_j$ define the schema, while the values $v_{i,j}$ may be numerical, categorical, or textual. An adequate tabular representation must therefore capture both the lexical content of the values and their association with the corresponding attributes.

Several approaches have been proposed to address these challenges. Early table-aware models such as TaBERT [26] and TAPAS [12] are explicitly designed to incorporate table structure into the encoding process by jointly modeling cell content, column headers, and row or column positions. More recent approaches, such as TableRAG [16], adopt a different paradigm based on structured query execution. Rather than embedding tables directly, these methods rely on a large language model to generate SQL queries conditioned on the extracted table schema, which are then executed against the underlying database to retrieve relevant results.

However, such approaches rely heavily on the capabilities of the underlying models, and in the case of SQL-based methods, on accurate schema understanding and query generation. An alternative strategy explored in prior work is table serialization, where tabular rows are converted into structured textual representations before being passed to a standard text embedding model. This paradigm has been used in table understanding and retrieval settings to enable language models to process structured data via linearized text representations, as demonstrated in TabFact [27] and more recently evaluated in the context of generative pipelines by TARGET [28]. Formally, each row is transformed into a linearized sequence that explicitly encodes attribute–value relationships, for example:

`Material Name: S355; Yield Strength: 355 MPa; Density: 7850 kg/m`$^3$`; ...;.`

By embedding attribute names alongside their corresponding values in a key–value-style representation, serialization allows pre-trained language models to leverage their natural

language understanding while preserving essential table semantics. This enables tabular data to be embedded into the same vector space as unstructured documents, facilitating embedding-based retrieval for downstream generative tasks [27, 28].

This strategy enables tabular data and unstructured documents to be embedded into a shared vector space, facilitating unified retrieval and fusion in RAG systems. However, the effectiveness of serialization-based representations depends critically on the chosen format, the treatment of numerical values, and the consistency of attribute naming across tables. These considerations directly influence retrieval quality and motivate the design choices explored later in this thesis.

## 3.2 Dense Retrieval

Dense retrieval represents the shift from lexical matching toward semantic matching by encoding queries and documents as dense vectors in a continuous embedding space. Unlike classical retrieval models, which rely on exact term overlap and sparse representations, dense retrieval systems make use of learned embeddings to capture semantic similarity between texts. Thereby, relevance is determined by computing similarity scores between vector representations, allowing retrieval even when queries and relevant documents share minimal to no lexical overlap. This paradigm addresses the several limitations of classical retrieval and is the core component of modern RAG systems.

### 3.2.1 Dense Representation for Retrieval

In dense retrieval, both queries and documents are represented as dense vectors in a shared embedding space. Let $x_q$ denote a query and $x_d$ a document. Using an embedding model $f_\theta$ as defined in Section 3.1, the corresponding vector representations are given by:

$$q = f_\theta(x_q), \quad d = f_\theta(x_d),$$

where $q, d \in \mathbb{R}^D$ and $D$ denotes the output dimensionality of the embedding model. Retrieval is formulated as a similarity search problem in the embedding space, where relevance is computed using a similarity function $\texttt{sim}(\cdot, \cdot)$ as introduced in Section 3.1.1:

$$s(q, d) = \texttt{sim}(q, d).$$

Documents are ranked according to this score, and the top-$k$ most similar embeddings are retrieved.

### 3.2.2 Bi-Encoder Architecture

Dense retrieval systems commonly adopt a bi-encoder architecture, where queries and documents are encoded independently into a shared embedding space introduced in the previous subsection. Under this architecture, document embeddings are computed and stored either locally or in a cloud-based storage, while query embeddings are computed at retrieval time. During retrieval, the relevance score is determined by comparing the query embedding against the stored document embeddings using a similarity function.

The main benefit of this design is its computational efficiency. Since the document embeddings are independent of the query, they can be precomputed and indexed, enabling scalable similarity search over large collections. Formally, let $\mathcal{D} = \{d_1, d_2, \ldots, d_N\} \subset \mathbb{R}^k$

be the embedding index where each document's embedding is stored. Then given a query, the similarity search is computed as

$$s(q, d_i) = \texttt{sim}(q, d_i), \quad d_i \in \mathcal{D},$$

and retrieval corresponds to selecting the $k$ documents with the highest similarity scores:

$$\mathcal{R} = \arg\operatorname{top}_k\{s(q, d_i) \mid d_i \in \mathcal{D}\}.$$

Figure 3.1 illustrates the bi-encoder retrieval pipeline, where document embeddings are indexed offline and `top-`$k$ retrieval is performed by ranking $d_i \in \mathcal{D}$ according to $s(q, d_i)$.

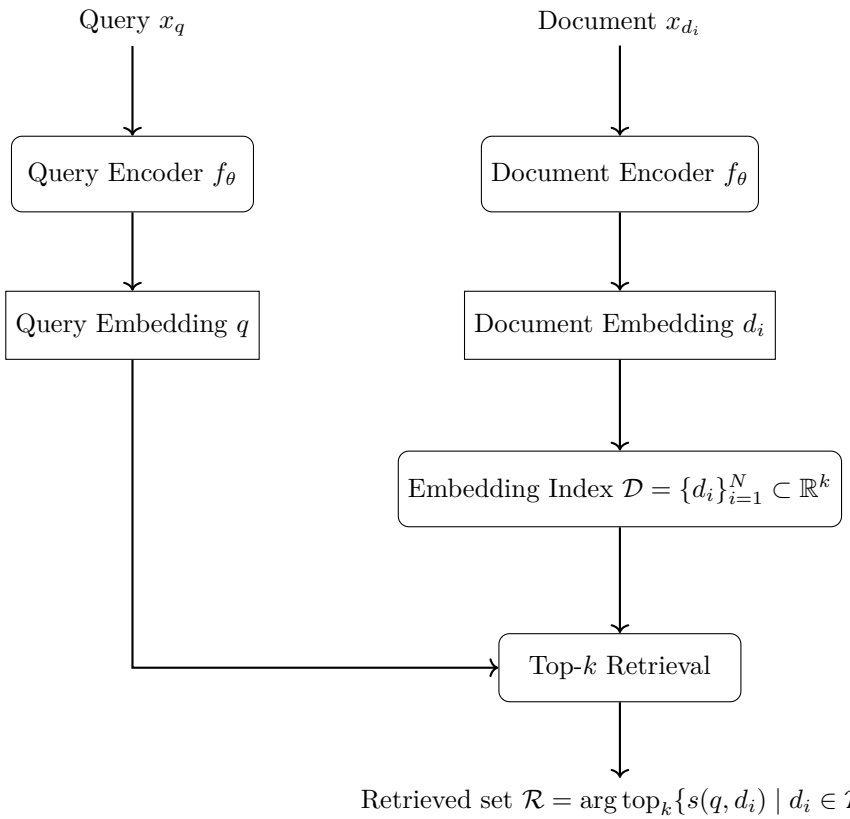



Figure 3.1: Bi-encoder architecture for dense retrieval. Document embeddings $d_i$ are precomputed and stored in an embedding index $\mathcal{D}$. At query time, the query embedding $q$ is used to retrieve the top-$k$ documents by maximizing similarity scores $s(q, d_i) = \texttt{sim}(q, d_i)$.

### 3.2.3 Approximate Nearest Neighbour Search

In dense retrieval, relevance is determined by nearest-neighbour search in a high-dimensional embedding space. This is due to performing exact similarity search over vast collections of embeddings is computationally expensive, as it requires comparing the query embedding against all the stored embeddings in the index. To address this, practical dense retrieval systems rely on approximate nearest neighbour (ANN) search methods, which trade exactness for substantial improvements in retrieval efficiency.

ANN techniques construct index structures that enable sublinear-time retrieval while preserving high recall of the most relevant candidates. Modern vector stores employ ANN search internally such as Qdrant [29] and ChromaDB [30], allowing dense retrieval to scale to large document collections and making RAG feasible in practice.

### 3.2.4 Limitations of Dense Retrieval

Despite its ability to capture semantic similarity beyond lexical overlap, dense retrieval exhibits several inherent limitations that arise from the properties of learned embedding spaces and their training objectives.

The primary limitation of dense retrieval is the use of fixed-dimensional vector representations to encode complex content. Compressing long documents or multi-topic passages into a single embedding can result in information loss, which leads to retrieval of documents that are semantically related but only partially relevant to the query. This limitation has been discussed in the context of dense passage retrieval, where splitting the document into passages directly influence retrieval quality [31].

In addition, dense retrieval performance is strongly dependent on the data distribution and supervision signals used during training. Embedding models trained on specific corpora or task formulations may generalize poorly to new domains, query styles or specialized terminology, which results in degraded retrieval effectiveness under changing domains. This sensitivity to training data has been observed in empirical evaluations of pretrained dense retrievers [32].

Furthermore, scalability in dense retrieval relies on approximate nearest neighbour (ANN) search, which introduces an additional source of approximation. Although state-of-the-art ANN algorithms achieve high recall across a wide range of settings, they do not guarantee exact nearest-neighbour retrieval. As a result, relevant document embeddings may occasionally be excluded from the top-$k$ results, particularly in large-scale or high-dimensional embedding spaces [33].

To address these limitations, hybrid retrieval strategies and reranking mechanisms are commonly employed. Further explored in Section 3.3, these approaches combine the semantic generalization capabilities of dense retrieval with the precision and interpretability of complementary methods.

## 3.3 Retrieval-Augmented Generation

RAG extends standard language modeling by incorporating external information retrieved from a document collection at inference time. Given a user query, the RAG model first retrieves top-$k$ relevant documents or passages by commonly using approximate nearest neighbour search in the embedding space. Then, the model provides the additional context to a generative language model. This formulation separates the knowledge storage from the generative model and enables the system to ground its outputs on external evidence.

Formally, let $x$ denote an input query and $\mathcal{D}$ a document collection. Dense retrieval is used to obtain a set of relevant documents:

$$\mathcal{R} = \{d_1, \ldots, d_k\} \subset \mathcal{D},$$

where the selection is based on similarity in the embedding space, as described in Section 3.2. The generative language model produces thereafter an output conditioned on both the query and the retrieved context:

$$y \sim p(y \mid x, \mathcal{R}).$$

Note that $y$ denotes the textual output generated by the language model. By conditioning generation on externally retrieved documents, RAG reduces the underlying reliance on the model's internal parametric knowledge. This grounding mechanism mitigates hallucinations by constraining the generation process to information that is explicitly in the retrieved context.

### 3.3.1 Document Chunking and Context Representation

In RAG systems, documents are typically segmented into smaller units, also referred to as chunks or passages, prior to embedding and indexing. This design choice reflects the fact that long documents often contain multiple independent pieces of information and that dense embedding models operate on fixed-length inputs, motivating passage-level retrieval [31].

Formally, let $d \in \mathcal{D}$ denote a document in the corpus, and let a deterministic text-extraction function $g$ map a document from its original format into an ordered sequence of textual units:

$$U = g(d) = (u_1, u_2, \ldots, u_M),$$

where $u_j$ corresponds to a block-level extraction (e.g. a page or text block), and the ordering reflects the document's original reading order. In the case $M = 1$, the document is treated as a single textual unit.

Then a deterministic segmentation operator $C$ partitions $U$ into a sequence of chunks:

$$\mathcal{C}(d) = C(U; \Phi) = (c_1, c_2, \ldots, c_N),$$

where $\Phi$ denotes fixed chunking parameters required by the $C$ operator, such as maximum token length $L$, overlap $o$ and other separator rules. Each chunk $c_j$ is formed by grouping one or more consecutive extracted units into a single retrieval unit, preserving their original order:

$$c_j = u_a \oplus u_{a+1} \oplus \cdots \oplus u_b,$$

where $1 \leq a \leq b \leq M$ and $\oplus$ corresponding to the string concatenation with a predefined separator, such as newline between the textual units. Determinism is then ensured by using a fixed $\Phi$ and a deterministic ordering of $U$, which implies that for a given document $d$, the resulting chunk sequence $\mathcal{C}(d)$ is unique.

If the chunks are constrained by a maximum length $L \in \Phi$ in tokens under a tokenizer $\tau$, then each chunk satisfies the condition:

$$|\tau(c_j)| \leq L,$$

with $|\cdot|$ corresponding to the length of the sequence. Each pair of consecutive chunks may also overlap by a fixed amount of $o \in \Phi$ tokens:

$$|\tau(c_j) \cap \tau(c_{j+1})| = o.$$

This overlap ensures continuity between adjacent chunks, reducing the risk that relevant contextual information is lost at chunk boundaries.

Each chunk $c_j$ is thereafter stored as a retrieval unit together with metadata $\mu_j$, which forms a context item:

$$z_j = (c_j, \mu_j),$$

where $\mu_j$ usually includes identifiers such as document ID, page range, source document name and chunk index:

$$\mu_j = (\texttt{id}(d), j, a, b, \dots).$$

The embedding model $f_\theta$ maps then each chunk into a dense vector

$$d_j = f_\theta(c_j) \in \mathbb{R}^k,$$

where $k$ denotes the embedding dimensionality. The indexed collection for retrieval is thus defined as:

$$\mathcal{D} = \{(d_j, \mu_j)\}_{j=1}^{N}.$$

During retrieval, the system finds the top-$k$ chunks whose embeddings $d_j$ exhibit the highest similarity to the query embedding $q$. The associated metadata $\mu_j$ is included to ensure traceability of the retrieved content to its source documents and to provide extra information during the generation process.

### 3.3.2 Fusion Retrieval from Multiple Sources

For RAG systems to retrieve from several domains, methods exist to fuse retrieved components from multiple indices together. The three methods covered here are relative score fusion, distribution-based score fusion and reciprocal rank fusion, all are based on rearranging the initial rankings using simple mathematical formulations. Let $\mathcal{D}^{(1)}$ and $\mathcal{D}^{(2)}$ denote two retrieval indices, in which each index containing embeddings derived from distinct document collections. Given a query embedding $q$, retrieval is performed independently over each index, which produces ranked candidate sets:

$$\mathcal{R}^{(m)} = \{(d_i^{(m)}, s_i^{(m)})\}_{i=1}^{K}, \quad m \in \{1, 2\},$$

with $d_i^{(m)}$ denoting a retrieved chunk from index $m$ and $s_i^{(m)}$ is its corresponding similarity score. Fusion retrieval then aims to combine these ranked results into a unified ranking set:

$$\mathcal{R} = \mathcal{F}(\mathcal{R}^{(1)}, \mathcal{R}^{(2)}),$$

where $F$ denotes the fusion function. This fusion strategy allows each retriever to be retrieving independently, enabling a more equitable retrieval for a RAG system with multiple data sources.

Relative score fusion normalizes the similarity scores within each retriever's result set to make them comparable across indices before aggregation [34]. Formally, for retriever $m$, scores are normalized the following way:

$$\tilde{s}_i^{(m)} = \frac{s_i^{(m)} - \texttt{min}_j s_j^{(m)}}{\texttt{max}_j s_j^{(m)} - \texttt{min}_j s_j^{(m)}},$$

where the score $s_i^{(m)}$ is mapped to $\tilde{s}_i^{(m)} \in [0,1]$, preserving relative ordering within each retriever. The fused score for chunk $d_i$ is then computed as:

$$s_i = \sum_{m=1}^{M} a_m \tilde{s}_i^{(m)},$$

with $a_m \geq 0$ being fusion weights satisfying $\sum_m a_m = 1$.

Distribution-based score fusion is a variant of relative score fusion, where min-max normalization based on observed score extremes is replaced with normalization derived from the score distribution. The reason for this adjustment is to reduce sensitivity to outliers and unstable score ranges that may arise when individual retrievers produce skewed or heavy-tailed similarity distributions [35].

Formally, let $\mathcal{R}^{(m)} = \{(d_i, s_i^{(m)})\}_{i=1}^{K}$ denote the retrieval results of the retriever $m$. Instead of computing normalization bounds from the minimum and maximum scores, distribution-based fusion estimates them from the score distribution:

$$\mu^{(m)} = \frac{1}{K}\sum_{i=1}^{K} s_i^{(m)}, \quad \sigma^{(m)} = \sqrt{\frac{1}{K}\sum_{i=1}^{K}\left(s_i^{(m)} - \mu^{(m)}\right)^2}.$$

The normalization bounds are then defined as:

$$a^{(m)} = \mu^{(m)} - 3\sigma^{(m)}, \quad b^{(m)} = \mu^{(m)} + 3\sigma^{(m)},$$

where it corresponds to a three-standard-deviation range under the assumption of approximately normal score distributions. Each score is then normalized as:

$$\tilde{s_i}^{(m)} = \frac{s_i^{(m)} - a^{(m)}}{b^{(m)} - a^{(m)}},$$

and fused across retrievers using weighted aggregation, identical to relative score fusion. By relying on distributional statistics rather than extreme values, distribution-based score fusion results in more stable normalization and improved robustness when fusing heterogeneous retrieval sources [35].

Reciprocal Rank Fusion (RRF) [36] is a standard fusion function used in RAG systems that does not make use of normalization. Its a rank-based fusion combines retrieval results using rank positions rather than raw scores. Given a ranked list $\mathcal{R}^{(m)}$, each chunk $d_i$ is assigned a rank $r_i^{(m)}$. The fused score is defined as:

$$s_i = \sum_{m=1}^{M} \frac{1}{k + r_i^{(m)}},$$

where $k > 0$ is a smoothing constant that reduces the influence of results that are low-ranked.

Unlike relative score fusion and distribution-based score fusion, RRF does not require any score normalization and is robust to variations in score scales across retrievers. By emphasizing the ranking across retrievers, RRF prioritizes candidates that are consistently ranked highly, even when the absolute score values differ.

### 3.3.3 Reranking Models

Reranking models further refine the output of an initial retrieval stage by re-approximating relevance scores over a limited candidate set using a more expressive scoring function. In a standard two-stage retrieval pipeline, the initial retriever is optimized to generate a scalable set of candidates, while reranking is applied as a second-stage procedure to further improve precision by performing deeper query-document interaction modeling with the limited set of retrieved candidates [31].

Formally, let $q$ denote a query and $\mathcal{D}$ the indexed collection of chunks. A standard first-stage retriever produces a candidate set restricted by its top-$k$ parameter:

$$R_k = \arg \texttt{top}_k\{s(q, d) \mid d \in \mathcal{D}\},$$

where $s$ denotes the retrieval score and $k$ is chosen to limit the number of candidates passed to the reranking model, ensuring computational feasibility [31].

A reranker model is then used to compute a refined relevance score for each candidate chunk $d \in \mathcal{R}_k(q)$ using a scoring function $f_\phi$:

$$r(d \mid q) = f_\phi(q, d),$$

where $\phi$ denotes the reranking model's learnable parameters. The final reranked list is thus obtained by sorting candidates by $r(d \mid q)$:

$$R_j^{rerank} = \arg \texttt{top}_j\{r(d \mid q) \mid d \in \mathcal{R}_k(q)\}, \quad j \leq k.$$

This formulation enables more expressive relevance modeling to be applied only to a reduced candidate set.

A common reranking architecture is the cross-encoder, which encodes the query and candidate chunk in conjunction to enable token-level interactions. Formally, the cross-encoder takes the paired input $[q; d]$ and produces a scalar relevance score:

$$r(d \mid q) = f_\phi([q; d]).$$

By jointly encoding each pair, cross-encoders can model fine-grained alignment between query terms and the candidate contents, which has shown to improve ranking effectiveness compared to independent encoders in passage reranking systems [37].

However, one disadvantage in cross-encoder reranking models is its computational complexity. Unlike bi-encoders, where document embeddings can be precomputed, cross-encoders require evaluating $f_\phi(q, d)$ for each candidate $d$. This motivates their use as a second-stage component in the system operating on small $k$ [31].

To summarize its application to RAG systems, reranking improves the quality of the context provided to the language model by prioritizing candidates that are most relevant under stronger interaction modeling. If the generative model predicts an output $y$ conditioned on the query and retrieved context, reranking effectively improves the conditioning set:

$$y \sim p\left(y \mid q, \mathcal{R}_k^{rerank}(q)\right).$$

Due to the factual grounding of the generator being dependent on the relevance of the supplied evidence, improved ranking precision can reduce the inclusion of weakly relevant or misleading chunks, improving factual consistency in generation [6].

## 3.4 Evaluation Metrics

Various metrics exist to evaluate different aspects of RAG systems. These metrics are typically designed to assess either retrieval performance or generative performance, and in some cases an end-to-end combination of both. Retrieval-oriented metrics focus on evaluating the relevance and ranking quality of the retrieved context, independent of the downstream generation process [38]. The metrics covered here are hit rate, recall, precision, F1-score and Mean Reciprocal Rank (MRR)

Hit rate is a foundational retrieval metric that indicates whether any relevant chunk is retrieved for a given query. Each test instance is assigned a value of 1 if at least one relevant chunk appears among the top-$k$ retrieved results, and 0 otherwise. Formally, hit rate@k is defined as:

$$\text{Hit rate@}k = \begin{cases} 1, & |\mathcal{R}_k \cap \mathcal{G}| \geq 1, \\ 0, & \text{otherwise}, \end{cases} \qquad \begin{array}{l} \mathcal{R}_k : \text{ set of top-}k\text{ retrieved chunks}, \\ \mathcal{G} : \text{ ground-truth set of relevant chunks}. \end{array}$$

When aggregated over a test set, hit rate@k is computed as the average across all test instances and therefore lies in the range $[0, 1]$. While hit rate provides a coarse indication of retrieval success, it does not account for ranking order or the number of relevant chunks retrieved [38].

Recall measures the proportion of all relevant chunks that are successfully retrieved within the top-$k$ results. It is defined as:

$$\text{Recall@}k = \frac{\text{TP}}{\text{TP} + \text{FN}} \in [0, 1],$$

where True Positives (TP) denote relevant chunks that are retrieved, and False Negatives (FN) denote relevant chunks that are not retrieved [38].

Recall emphasizes coverage of relevant information. In cases where a test instance contains only a single relevant chunk, recall@k becomes equivalent to hit rate@k. recall is commonly averaged per test instance so that each query contributes equally to the final score.

Precision measures the proportion of retrieved chunks that are relevant. Formally, it is defined as:

$$\text{Precision@}k = \frac{\text{TP}}{\text{TP} + \text{FP}} \in [0, 1],$$

where False Positives (FP) denote retrieved chunks that are not relevant [38].

Precision reflects the amount of irrelevant information included among the retrieved results. In retrieval systems that return a fixed number of chunks, precision may be low when only a small number of relevant chunks exist. Consequently, precision is typically interpreted in conjunction with recall and ranking-based metrics [38].

The F1-score represents both precision and recall in one measure as the harmonic mean of their values. It follows the formula:

$$\mathrm{F1} = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}} = \frac{2 \cdot \mathrm{TP}}{2 \cdot \mathrm{TP} + \mathrm{FP} + \mathrm{FN}} \in [0, 1],$$

Reciprocal Rank (RR) evaluates how highly the most relevant chunk is ranked by considering the position of the first relevant chunk in the ranked retrieval output. For a single query, RR is defined as the reciprocal of the rank at which the first relevant chunk appears. If no relevant chunk is retrieved within the top-$k$ results, the reciprocal rank is defined as zero.

When averaged across multiple queries, the metric yields the Mean Reciprocal Rank (MRR). Formally, MRR@k is defined as:

$$\mathrm{MRR@}k = \frac{1}{|\mathcal{Q}|} \sum_{q \in \mathcal{Q}} \mathrm{RR}_q@k,$$

where $\mathcal{Q}$ denotes the set of test queries.

MRR is sensitive to ranking order and is particularly informative in settings where highly ranked results are prioritized, as it emphasizes the position of the earliest relevant chunk in the ranking [36].

# 4

# Methodology

The research methodology is structured into six areas: Data, User Study, Test Set Generation, RAG Pipeline, Evaluation and Experimental Setup. The Data section describes the data domains as well as the data preparation and preprocessing executed to enable retrieval. This is covered both in terms of general RAG development and in terms of the data which is practically available in the setting of this thesis. The second section describes the setup of the user study conducted on the *personas*. The third section explains the properties of available data and covers how test sets were created, partially dependent on the user studies, as well as the constraints it brings to this research study. The RAG pipeline section methodologically explains the RAG components, and the selection of both components and techniques used in the present work is motivated and connected to theory. The evaluation section is dedicated to metrics and the process chosen to analyze the behavior of the system. Lastly the schema of experiments is covered, focusing on the three main topics; tabular representation, fusion methods, and routing. This schema is visualized in Figure 4.1.

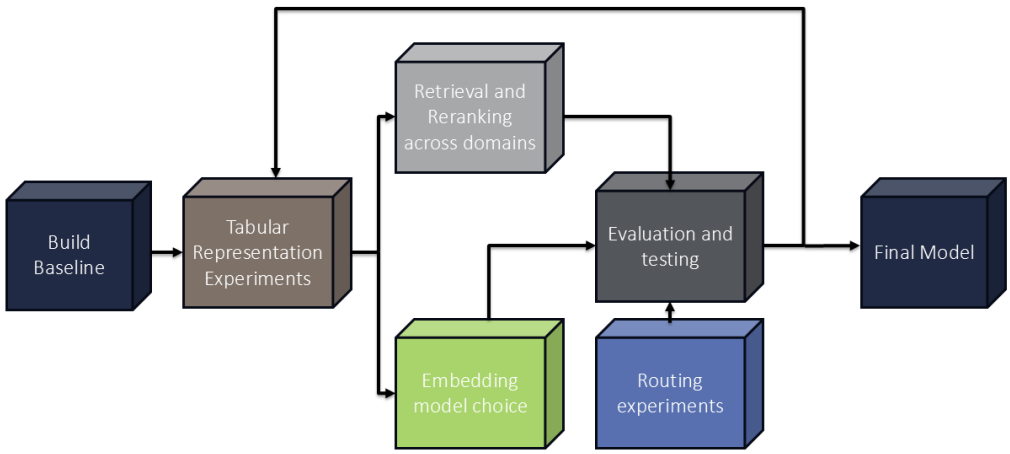



Figure 4.1: Schema of the experimental setup

## 4.1 Data

The heterogeneous industrial data consists of multiple sources and formats and was therefore categorized into coherent domains which were processed independently. This section summarizes the available data and its categorization. It then describes how raw files were systematically partitioned into datasets to enable testing and evaluation of the RAG system. Lastly, it outlines the preprocessing steps tailored to each data category, including loading, parsing, chunking, embedding, and building of the vector store. Although this pipeline shares components which are used at inference time by the RAG system, such as the embedding model, it is described here as preprocessing since it is handled prior to the inference phase.

### 4.1.1 Data Domains and Sources

The data sources related to materials vary in formats and level of structuredness, thus the first level of categorization distinguishes between unstructured and structured data, and named as the textual and the tabular data domain, respectively. The overview is shown in Figure 4.2. Documents in the textual domain may contain tables within them, however these are still considered as belonging to the unstructured domain as they are processed within the textual document parser. A second level of categorization was introduced between internal Volvo Group data and external supplier data. This was done mainly to keep a representative balance between the document types at evaluation and test set generation.

The textual data domain contained mainly documents in pdf format. The single largest collection of textual documents was the internal Volvo Group source Corporate Standards Database (CSD) and the second largest was the external collection of Environmental Product Declarations (EPDs) collected from the Environdec database [39]. While CSD documents are typically in two-column style with one English and one Swedish version of the content in parallel, EPDs are typically regular style documents in English.

The tabular data domain incorporates all data stored in structured formats. Tabular data is mostly used to store information on physical material properties, and this data is mainly hosted in the internal Common Materials Database (CMD). The database comprises information on both actual physical properties of materials given by suppliers or lab tests, and of desired properties specified in standards documents.

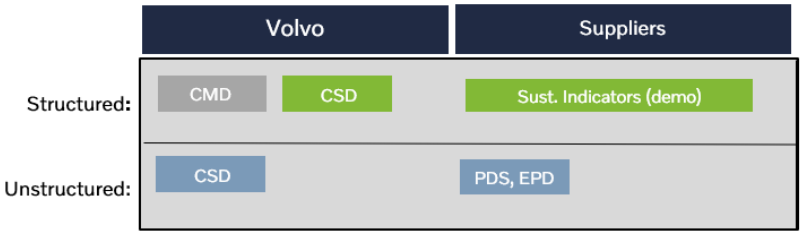
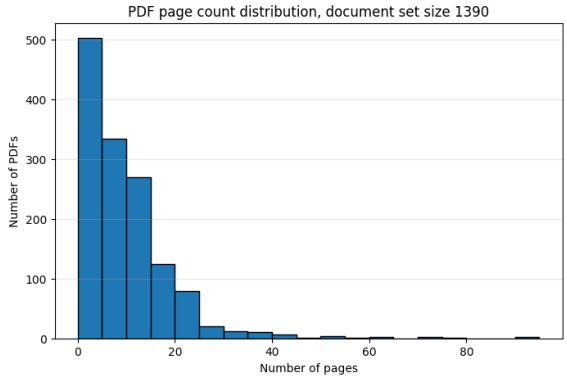



Figure 4.2: Categorization of included data sources

The full corpus of 1,390 documents had a mean length of 9.5 pages and a median of 7 pages per document. Across all subsets, the mean page count ranged from 7.6 to 9.5 pages, and the median from 7 to 8 pages. The page-length distribution for the full corpus is presented using bins of size 5 in Figure 4.3a. The full tabular corpus consisted of 5 tables, with cell

counts (rows times columns) varying in the range 3,900 to 43,000, with proportions of empty cells (null rates) as shown in Figure 4.3b. The global null rate was 68.85%.




(a) Page number distribution over the full set of CSD and EPD documents

(b) Material data table sizes and null rates

| Table ID | Rows | Columns | Null rate |
|---|---|---|---|
| 0 | 648 | 66 | 0.877 |
| 1 | 463 | 16 | 0.290 |
| 2 | 402 | 32 | 0.621 |
| 3 | 205 | 19 | 0.599 |
| 4 | 707 | 8 | 0.00 |
| | | | **0.6885** (global) |

Figure 4.3: Properties of unstructured and structured datasets

### 4.1.2 Data Preparation

Preparing the raw data files for preprocessing and test set generation, all files were systematically loaded into subsets with the intent to test retrieval performance on various sizes of vector stores. Due to imbalance in amount of available data the subset sizes varied between the tabular and textual domains. From the tabular data domain, all 5 available tables were used in every subset. From the textual domain, the largest subset included all 1,390 documents. Of these, 950 were CSD documents, 402 were EPDs from the Environdec database and 38 were supplier-provided EPDs.

In addition to the full textual domain dataset, 4 smaller subsets were constructed by repeatedly sampling 50% of the documents randomly, consequently achieving subsets of the file count sizes 695, 347, 174, 88. By sampling iteratively, it was ensured that all files present in one subset were also present in all the larger subsets, and thus the data preparation adheres to the testing logic explained in Section 4.5.2. The sampling procedure was constructed to maintain the relative proportion between CSD and EPD document counts to ensure consistency, these counts are shown below in Table 4.1.

Table 4.1: Document collection size in relation to CSD and EPD counts

| Document collection Size | CSDs | EPDs |
|---|---|---|
| 1,390 | 950 | 440 |
| 695 | 475 | 220 |
| 347 | 237 | 110 |
| 174 | 119 | 55 |
| 88 | 60 | 28 |

Tabular files were exported from the internal PostgreSQL database into Parquet [40] files. For 3 of the tables a high use of technical abbreviations was present among column headings, which serves a practical purpose in the database, however it was expected to hinder LLM understanding of the material properties. For this reason, a set of dictionaries was created to map these technical labels into natural language expressions of the properties they

represent. The dictionaries for the 3 specific tables that were heading-processed are shown in Figure B.1 in Appendix B.

### 4.1.3 Data Preprocessing

The preprocessing stage transformed raw tabular and textual data into a format ready for indexing, the chunks, which are the unit of retrieval used by the model. Regardless of input modality, each data unit was processed into its final format, and was then associated with its filename as metadata, as well as given an embedding vector and unique chunk ID.

Textual data files were read sequentially from their subset folders. Each textual document was parsed using the MuPDF4LLM parser, converting PDFs and DOCX files into Markdown. Functionally, it extracts raw text content while preserving structural elements such as paragraphs, headings, and page boundaries, as well as storing metadata such as file name and page locations. Each processed text was then chunked with a fixed max-length and overlap specified in configuration files.

For the tabular data files, the innate structure was exploited as a row-based retrieval strategy was chosen, thus each table was parsed row by row and no further chunking strategy was needed. CSV files and SQL exports were serialized into a deterministic *key-value* string with standard field names, unified units, and validated types. Each row was thus represented by key-value pairs, keys being the column name and value being the corresponding value to that column, being a numerical or categorical property or identifiers for a material. In order to evaluate the efficacy of different table representation formats, 9 different table serializer functions were built and applied to the initial key-value pair representation. Every table was serialized into the formats listed, the string outputs are displayed as examples in Figure C.1 in Appendix C.

1. **JSON** – Concise key-value-based format.
2. **YAML** – Concise key-value-based format with indentation to convey data structure.
3. **HTML** – Verbose format conveying structure through semantic tags.
4. **Markdown Table** – Lightweight format denoting table structure by pipe and hyphen symbols for column and row separation, respectively.
5. **Markdown Transposed** – Transposed Markdown table where keys are listed vertically and values horizontally.
6. **Markdown Key-Values** – Markdown format excluding pipes and hyphens. Each key-value pair is written as a separate Markdown list item on a new line.
7. **Semantic** – Each key-value pair is converted into a simple natural-language statement using specific linking words, following the transformation pattern: Key: Value → 'Key is Value'. This approach adds linguistic coherence while preserving factual content.
8. **Rich semantic** – Each key-value pair is converted into a natural-language statement using specific linking words, according to the transformation: Key: Value → 'Item Key is Value and', additionally each row is augmented with the introductory string: 'Item in {tablename} is: ' This approach adds linguistic clarity while preserving factual content.

As the textual output from each serializer varies in length and content, the final assignment of a chunk ID for each table row was based on the original key–value pairs. This simplified evaluation, as test questions could be generated from the original table rows and reused across all serializers

## 4.2 User Study

A user study on the *personas*, conducted through structured testing sessions, was performed to evaluate the capabilities of the RAG system and to collect feedback based on the *personas* interactions. It was designed to explore and understand realistic data needs, user tasks and what type of queries the selected *personas* formulate when selecting materials, as well as to investigate how well the model was adapted to them. One round of interviews was performed with four employees from different departments. To make the interviews relevant to the intended usage, participants were selected so that they represented different *personas*. Each session lasted approximately 30 minutes and consisted of an initial demonstration followed by hands-on querying the model by the participant and a structured feedback discussion. A prepared set of questions guided the sessions, focusing on realistic work tasks, perceived data coverage, and how the system behaved in scenarios of both succeeding and failing to answer a given query. Additionally, a few initial questions were directed to the users to understand their current usage and experience in using AI tools for their working tasks. The question template is provided in Appendix A. Feedback from the sessions was aggregated, condensed and categorized into four focus areas: user tasks, RAG system behavior, data, and usability.

## 4.3 Test Set Generation Pipeline

Building on insights from the user study and *persona* analysis, and as the company-provided data lacked ground-truth relevance annotations, a synthetic evaluation set was constructed. This utilized an LLM-based question generation pipeline, for which the development process was guided by these insights to reflect realistic information needs and terminology. The objective was to create controlled retrieval queries with unique and identifiable ground-truth chunks, and to evaluate the model based on the success in finding these. Question generation was mainly improved by refining the prompt template and reviewing generated questions in accordance with insights from the user study. Manual expert annotation was considered infeasible within the time frame of the thesis.

While several benchmark datasets exist for question-answering evaluation [41, 42], these datasets primarily target unstructured text-based question answering, excluding tabular corpora. Existing benchmark QA datasets were thus not suitable, as they did not reflect the mixed document–tabular retrieval setting considered in this work. To achieve such a mixed evaluation, two distinct test set generation pipelines were constructed for the 2 domains. These are referred to as $Doc_q$ and $Tab_q$, respectively, and their union was denoted $Combo_q$.

### 4.3.1 Document-Based Question Generation

The document-based question generation pipeline utilizes two LLM sub-tasks as is illustrated in Figure 4.4. For each document, chunks were grouped in order to provide a contextual window for the LLM, from which questions were generated. Subsequently, the LLM was utilized to associate each generated question with a single chunk selected from the original group of chunks from which the question was generated. This design choice was intended to avoid overly constraining the question generation process to a single chunk, which would potentially bias generated questions towards maximizing semantic similarity with that specific chunk.

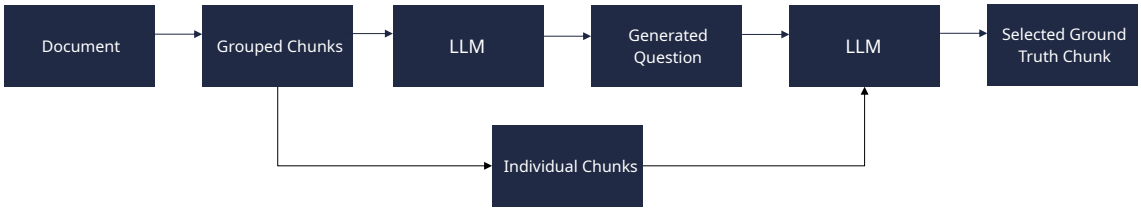



Figure 4.4: Test set generation pipeline for document data, where each generated question is mapped to a single chunk using an LLM as a judge.

To generate questions from semantically coherent contexts, a grouping process of chunks was made using a fixed sliding window of size $n$. This ensured that each group consisted of $n$ consecutive chunks from a document, as illustrated in Figure 4.5. Let $\hat{c}_i^{(n)} \in \hat{C}^{(n)}$ denote a grouped set of chunks extracted from an arbitrary document $d$, such that

$$\hat{c}_i^{(n)} = \{c_i, c_{i+1}, \ldots, c_{i+n-1}\}, \qquad \hat{C}^{(n)} = \left\{\hat{c}_i^{(n)}\right\}_{i=1}^{M-n+1},$$

where $c_i$ denotes the $i$-th chunk in the sequential ordering of document $d$ and the complete collection of grouped chunks is $\hat{C}^{(n)}$ where $M$ denotes the total number of chunks in document $d$. Given $\hat{C}^{(n)}$ the objective was to identify the top-$k$ chunk groups with maximum internal semantic coherence. A maximum chunk length of 256 tokens and a window size of $n = 5$ were selected during initial experimentation, resulting in approximately 1,280 tokens per group.

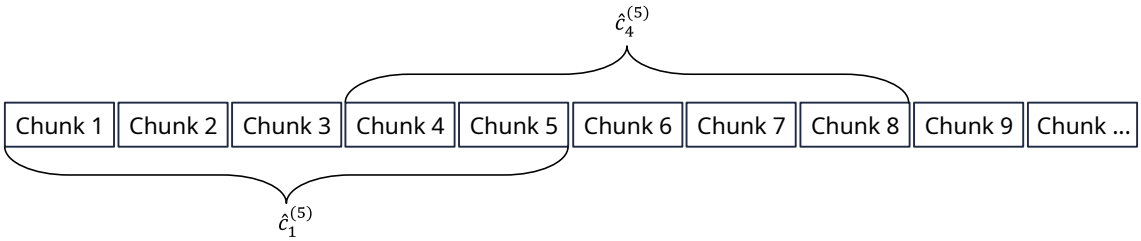



Figure 4.5: Illustration of the grouping process for a window size $n = 5$

For each grouped chunk $\hat{c}_i^{(n)}$, an average cross-similarity score was computed by evaluating pairwise semantic similarities between all chunks within the group. These pairwise similarities formed a similarity matrix, which was then aggregated into a single scalar score representing the overall semantic coherence of the group. Formally, the average cross-similarity score was computed by first embedding each chunk into a semantic vector space. Let

$$\mathbf{e}_{i,a} = f(c_{i+a}) \in \mathbb{R}^D, \qquad a \in \{0, 1, \ldots, n-1\},$$

where $f(\cdot)$ denotes the embedding function and $D$ is the embedding dimension. A pairwise similarity matrix $\mathbf{S}_i \in \mathbb{R}^{n \times n}$ was then constructed with entries

$$[\mathbf{S}_i]_{ab} = s(\mathbf{e}_{i,a}, \mathbf{e}_{i,b}), \qquad a, b \in \{0, 1, \ldots, n-1\},$$

where $s(\cdot, \cdot)$ denotes a similarity function. In this work, cosine similarity was used:

$$s(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u}^\top \mathbf{v}}{\|\mathbf{u}\|_2 \, \|\mathbf{v}\|_2}.$$

Since cosine similarity is symmetric, $\mathbf{S}_i$ is symmetric, i.e., $[\mathbf{S}_i]_{ab} = [\mathbf{S}_i]_{ba}$. The diagonal entries satisfy $[\mathbf{S}_i]_{aa} = 1$ and were therefore excluded when aggregating similarity scores. The group-level coherence score $\sigma_i$ was defined as the mean of the off-diagonal similarities:

$$\sigma_i = \frac{1}{n(n-1)} \sum_{a=0}^{n-1} \sum_{\substack{b=0 \\ b \neq a}}^{n-1} [\mathbf{S}_i]_{ab} \,.$$

Equivalently, due to symmetry, $\sigma_i$ can be computed using only the upper-triangular entries:

$$\sigma_i = \frac{2}{n(n-1)} \sum_{a=0}^{n-2} \sum_{b=a+1}^{n-1} [\mathbf{S}_i]_{ab} \,.$$

Finally, the top-$k$ chunk groups were selected by ranking $\{\hat{c}_i^{(n)}\}$ according to $\sigma_i$ in descending order. During selection, an additional non-overlap constraint was enforced, which ensured that no two selected groups shared any identical chunks. This prevented duplicate or partially overlapping groups from being included in the final selection.

Prior to question generation, top-$k$ chunk groups were extracted from a subset of the document collection and used as reference contexts. In total, 88 documents were selected for test generation. For documents chunked with a maximum length of 256 tokens, a window size of $n = 5$ and $k = 3$ groups per document were used. For documents chunked with a maximum length of 512 tokens, a smaller window size of $n = 2$ was selected to ensure a comparable total token length per group.

Each selected chunk group $\hat{c}_i^{(n)} = \{c_i, c_{i+1}, \ldots, c_{i+n-1}\}$ was concatenated into a single textual context, which was then provided to the LLM for question generation. Let

$$t = c_i \cup c_{i+1} \cup \cdots \cup c_{i+n-1}$$

denote the aggregated context. The generation of a question $y$ was modeled as

$$y \sim p(y \mid T, t),$$

where $T$ denotes the system prompt conditioning the LLM and $y$ represents the generated question.

The system prompt was designed to instruct the LLM to produce a single, specific, and realistic question that could be answered solely using the information contained in the provided context. The prompt explicitly restricted the use of external knowledge and constrained the output format to ensure consistency across the generated test set. Details of the system prompt and generation procedure are provided in Figure D.1 in Appendix D.

Each generated question was paired with the chunk group from which it was derived, and processed by a separate LLM to identify the individual chunk best answering that question. Rather than operating on the concatenated group context, the LLM was provided with the generated question together with the individual chunks $\{c_i, c_{i+1}, \ldots, c_{i+n-1}\}$ as separate inputs, and tasked with selecting the single chunk that best answered the question, as shown on the right-hand side of Figure 4.6.

Each chunk was assigned a deterministic node identifier using UUID version 5 with the DNS namespace [43]. The node identifier corresponding to the selected chunk was used as the ground-truth reference for the question and later served as the basis for retrieval evaluation.

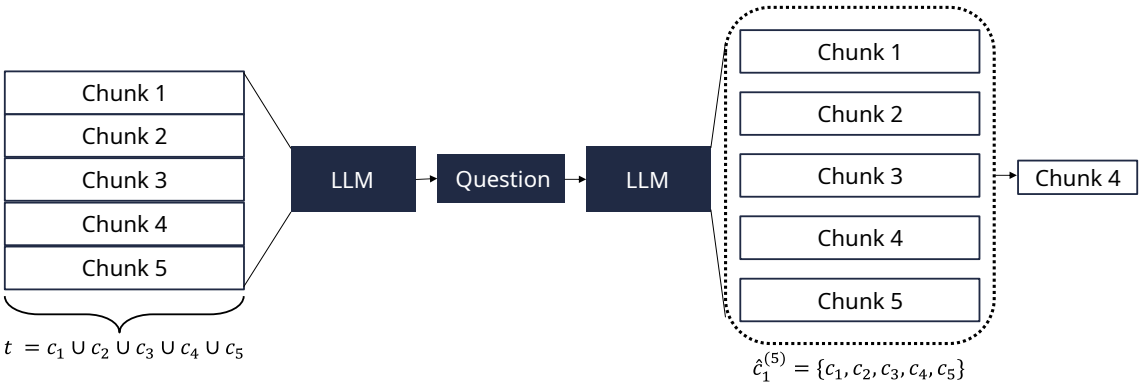



Figure 4.6: The question generation component of the pipeline. An LLM is conditioned on the aggregated context $t$ to generate a question. A second LLM then assigns the generated question to one individual chunk in the group $\hat{c}_i^{(5)}$ that most likely contains the answer.

Finally, question generation was treated as a qualitative process. Generated questions were manually inspected to verify compliance with the system prompt constraints, which was tuned after the user study described in Section 4.2. Questions that were less specific, ambiguous, or unverifiable using the provided context were discarded from the final test set.

### 4.3.2 Table-Based Question Generation

The table-based question generation began by uniformly sampling from the available tables, and then following the process outlined in Figure 4.7. Each table was stored in Parquet [40] format and converted into a Python list. To limit the input size, the list was truncated to 100 rows and provided as context to an LLM which was tasked with generating a natural language question together with a corresponding SQL query. The LLM additionally was supplied with schema name, table name, and other relevant metadata required to construct a valid SQL query for the selected table. The system prompt for this process is displayed in Figure D.2 and Figure D.3 in Appendix D

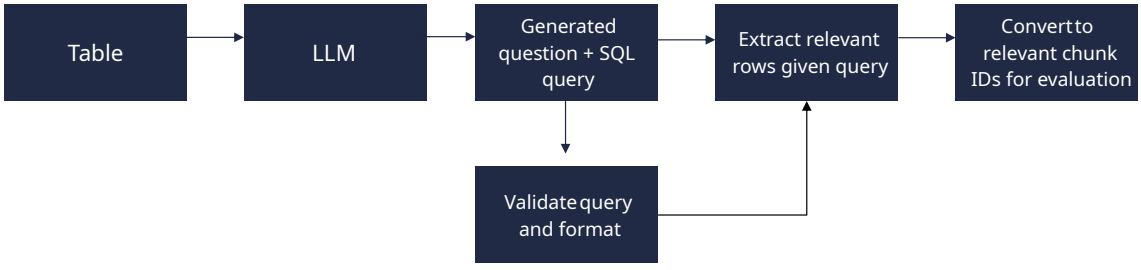



Figure 4.7: Illustration of the table-based question generation pipeline.

The prompts instructed the model to produce outputs in JSON Lines format. These outputs were parsed into a list of dictionaries, each containing the generated question and its associated SQL query. Malformed outputs or hallucinations were detected via checks during JSON parsing and by validating the generated SQL queries. Failed tests re-invoked the LLM until a valid output satisfying both criteria was obtained. Accepted dictionaries were used to query the source table, and node IDs for the resulting rows were then appended as a list of ground truths to each dictionary, finalizing the test set generation.

## 4.4 Retrieval-Augmented Generation Pipeline

This section describes the inference-time RAG pipeline taking user queries as input, retrieving chunks over pre-indexed source data, and outputting natural-language responses together with source citations. The pipeline as described here allows for internal variations, but the essential components of the chosen implementation are depicted in Figure 4.8. The first section gives an overview of the pipeline, and following sections break down the core components in more detail in the order in which data flows through the pipeline.

### 4.4.1 Pipeline Overview

Initially, a user-provided query is received and condensed using a large language model to produce a more focused representation. Second, the condensed query is embedded into a vector using the same embedding model that was employed during index construction. This ensures that query and chunk embeddings are comparable in the same vector space. The embedded query vector is then used in the retrieval stage, where similarity search is performed over the pre-built vector indices. This process returns the top-$k$ highest-scoring chunks. Retrieved chunks and the original user query are passed on to a reranking module which reorders the candidates. Lastly, the top-$n$ reranked chunks are concatenated with the original query and supplied as contextual input to a generator LLM. The generator produces a natural-language response constrained by a system prompt that enforces citation of the utilized sources, which is passed back to the user.

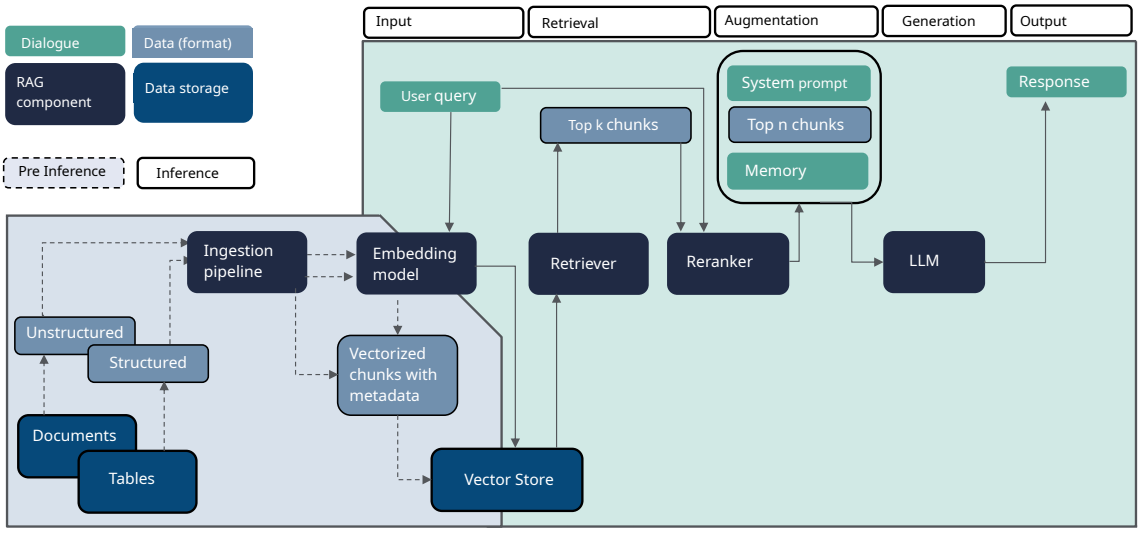



Figure 4.8: RAG pipeline partitioned into pre-inference and inference phases. The same embedding model is used both to embed chunks during ingestion and to embed user queries at inference. Vector stores are built during the pre-inference phase and later accessed as vector indices.

### 4.4.2 Model and Retrieval Parameter Configuration

The RAG pipeline contains numerous parameters affecting the properties of the final model, mainly associated with the chunking process and retrieval steps. Parameters were initialized to common default values, and the most relevant ones were varied experimentally as outlined in Section 4.6.

In the chunking process, tabular data chunking is straightforward, as each table row corresponds to a single chunk, whereas document chunking admits multiple design choices. When adopting a fixed-length chunking strategy for documents, two parameters are defined: the maximum chunk length and the chunk overlap, set respectively to 256 and 20 as default.

Within the retrieval pipeline, the retriever, fusion mechanism, and reranker each apply a threshold on the number of chunks selected. The parameters controlling retrieval depth for both the retriever and the fusion stage are denoted as top-$k$. These values are set identically to ensure that the same number of chunks is forwarded to the reranker, regardless of whether retrieval is performed using a single vector index or multiple indices. For multi-index retrieval methods, each individual index must be queried with its own retrieval limit. In these cases, the same top-$k$ value is used for each index, resulting in an initial retrieval of twice the desired number of chunks, which are subsequently filtered down to the target top-$k$ through the fusion mechanism. Finally, the reranker applies a threshold denoted as $top_n$, which was fixed to 10 across all experiments.

### 4.4.3 Embedding Model

The choice of embedding model directly affects the quality and semantic fidelity of the vector representations used for retrieval. The primary embedding model employed in this work is *BAAI/BGE-M3* [44] which is a multilingual and multi-granular model. Its ability to handle heterogeneous inputs makes it well suited for embedding serialized tabular data and unstructured document chunks within a unified vector space, while also supporting both Swedish and English queries and content.

### 4.4.4 Query Processing

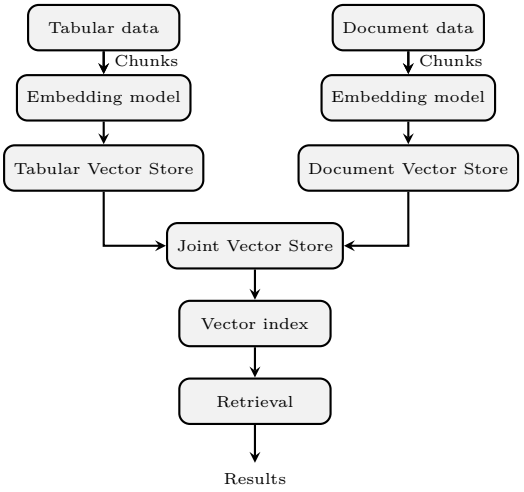
Due to variability in the quality, specificity, and conversational nature of user-generated queries, an additional preprocessing step is applied prior to embedding. Specifically, each incoming query is first condensed using a large language model (LLM), which incorporates the current query together with relevant previous conversational context to produce a coherent, standalone question suitable for embedding. This query condensation step improves alignment between query embeddings and the representations of indexed chunks, thereby increasing retrieval robustness across both document and tabular domains.

### 4.4.5 Retrieval Methods

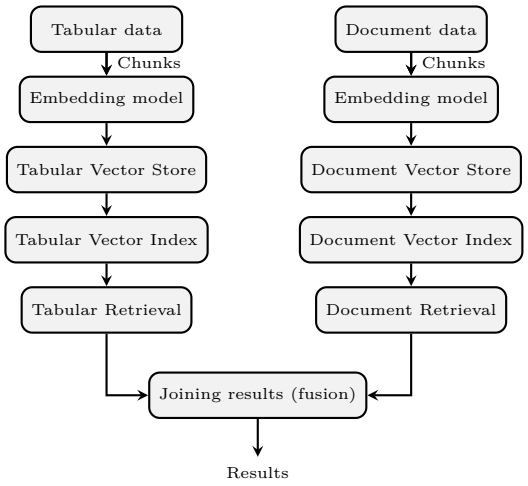
We implement one baseline method and three additional retrieval strategies. The primary differences between retrieval methodologies lie in the number of vector indices employed, and in the mechanisms used to filter and select the final set of retrieved chunks. While the baseline methodology relies on a single vector index, the three alternative methods employ different strategies that combine retrieval results from multiple vector indices through different fusion approaches.

The Baseline retriever (BL) illustrated in Figure 4.9a was constructed using a single embedding model and merging all ingested data into one unified vector store prior to index construction. Although tabular and document data are ingested separately, they are stored in a unified representation at their respective collections in the vector store. For the baseline, these were merged into a single combined collection, on which one vector index was built and used for retrieval.

The Simple Fusion retriever (SF) presented in Figure 4.9b was constructed as an advancement from the Baseline retriever, where tabular and document data remain in separate vector indices. Both indices were queried with the same embedded query, returning top-$k$ chunks each. Utilizing the Llamaindex class QueryFusionRetriever for fusing the retrieval results, a set of top-$k$ chunks out of the total 2·top-$k$ were outputted. The ordering and subsequent filtering was determined by the built-in simple fusion algorithm, which orders results based on solely the similarity scores assigned at retrieval from the respective indices.




(a) Pipeline of Baseline model, utilizing a tabular and document data joint vector storage




(b) Pipeline of Simple Fusion and Late Fusion (identical) model using separate vector indices for tabular and document data and a fusion algorithm

Figure 4.9: Comparison of Baseline and Simple Fusion retrieval architectures

The Late Fusion retriever (LF) was constructed as an extension of the Simple Fusion pipeline and differs from it in the choice of fusion algorithm. The LF pipeline is thus structurally identical to the SF in accordance with Figure 4.9b. We evaluate Relative Score Fusion [34], Distribution-Based Fusion [35] and Reciprocal Rank Fusion (RRF) [36], introduced in Section 3.3.2. The two former methods combine and rank retrieved chunks across indices, while RRF rely solely on the rank positions of the retrieved chunks. The incorporation of these fusion strategies was made to assess whether lightweight statistical fusion approaches can improve the integration of retrieval results from the two distinct data domains.

The Weighted Fusion retriever (WF) illustrated in Figure 4.10 extends on the SF and LF retrievers by weighting the domains according to their estimated relevance for the query. For weight determination, the multilingual cross-encoder *jina-reranker-v2-base-multilingual* was used as a reranker scoring query–chunk relevance. For each domain, the retrieved chunks were passed independently to the reranker together with the query, producing a relevance score for each individual chunk. The mean value of scores within each domain was then computed, yielding a single scalar reflecting the overall relevance per domain for the given query. The scores was scaled by a temperature parameter to control strength of the resulting weighting distribution, and then normalized using a softmax function. Scores were ultimately applied as a scaling factor to each retrieved chunk per originating domain. Rescaled chunks were then sorted in descending order with top-$k$ chunks being selected and passed on the the reranking stage.

The following formulation describes the domain-weighted fusion procedure before final

reranking. Formally, let $\mathcal{R} = \{r_1, r_2\}$ denote the set of retrieval domains (tabular and document), and let $\mathcal{C}_{r_i} = \{c_{i1}, \ldots, c_{iK}\}$ be the set of chunks retrieved from domain $r_i$ for a given query $q$. The reranker assigns a relevance score $s(q, c_{ij}) \in \mathbb{R}$ to each retrieved chunk. A domain-level relevance score is computed as the mean reranker score over all retrieved chunks from a given domain:

$$\bar{s}_{r_i} = \frac{1}{|\mathcal{C}_{r_i}|} \sum_{c \in \mathcal{C}_{r_i}} s(q, c).$$

The mean scores are scaled by a temperature parameter $\alpha$ and passed through a softmax function to normalized domain weights:

$$w_{r_i} = \frac{\exp(\bar{s}_{r_i}/\alpha)}{\sum_{r_j \in \mathcal{R}} \exp\left(\bar{s}_{r_j}/\alpha\right)}.$$

Each retrieved chunk is then assigned a weighted relevance score based on its originating domain:

$$\tilde{s}(q, c_{ij}) = w_{r_i} \cdot s(q, c_{ij}),$$

Finally, all weighted chunks across domains are merged, sorted in descending order according to $\tilde{s}(q, c)$, and truncated to retain the top-$k$ chunks, which are passed to the final reranking stage.

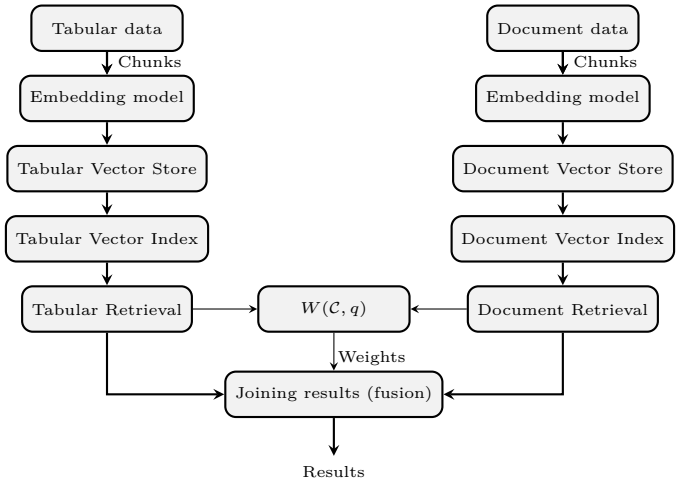



Figure 4.10: Pipeline of weighted retriever model, utilizing a cross-encoder for weight scoring. W(C, q) utilizes an LLM assigning a score to each of the two retrieval domains, which normalized are used as weights for the fusion algorithm

### 4.4.6 Reranking

The reranker was constructed using a cross-encoder model operating on each retrieved chunk together with the single user query and chunk as pairs. All pairs were assigned scores and ordered accordingly, upon which the $top_n$ were selected. We implemented a LlamaIndex FlagEmbeddingReranker [45] class for this purpose, and chose the BAAI BGE-reranking-v2-m3 [44] reranking model as it is suitable for multilingual data.

### 4.4.7 Response Generation

Response generation was set up to be handled by a CondesePlusContextChatEngine class from Llamaindex [45]. It was configured with one of the four retrievers, the reranker, an LLM and a system prompt. We use Deepseek-r1-70b as it was deemed to have best competitive performance among the available options. The system prompt, shown in Figure D.4 in Appendix D enforce concise answering and source citation. Additionally, chat memory was enabled and set to 4000 tokens of memory which was intended to allow the model in processing new queries that build on previous interactions.

## 4.5 Evaluation

This section first describes and justifies the metrics used to evaluate model performance, building on their theoretical introduction in Section 3.4. It then outlines the implementation of the evaluation pipeline, including how metric calculations were adapted to the generated test sets. The evaluation metrics and pipeline underpin the testing schema defined in Section 4.6.

### 4.5.1 Evaluation Metrics

Common retrieval metrics for RAG systems include hit rate, MRR, and NDCG [46]. While MRR and NDCG are sensitive to the ranking order of retrieved results, hit rate primarily assess the coverage of relevant chunks. Hit rate and MRR were chosen as core metrics and are used in all tests. Additionally, to achieve higher granularity of retrieval coverage for the test sets including multiple ground truths, recall and precision were included, as well as the F1-score. In total, five metrics were thus used: *hit rate, F1, precision, recall* and *MRR*.

Evaluation was performed on test sets containing multiple test instances, each composed of a query and an associated set of ground-truth chunks, referred to as relevant documents. Relevance was treated as a binary property: a retrieved chunk is either relevant or non-relevant based solely on whether it is included in the ground-truth set. For a given query, the retrieval system produces a ranked list of chunks, from which the top-$k$ results are used to compute the evaluation metrics.

For test instances with multiple ground truths, the relevant chunks were represented as sets. Each test instance was given equal weight, and metric scores were averaged over instances. This enabled joint evaluation on combined test sets containing both tabular and document questions. For document questions with a single ground truth, recall becomes binary (0 or 1) and is equivalent to hit rate. However, with this weighting scheme recall for tabular and document questions can be reported as a combined measure.

### 4.5.2 Evaluation Pipeline

A separate pipeline was built for evaluation. It provided the RAG model with access to the vector indices and test sets, ran inference, and finally passed results to the metric functions computing performance. This was accomplished by a configuration-driven Python script using YAML files. Test runs were executed sequentially as the vector store client implementation only allowed one client access simultaneously. Before evaluation, the datasets were ingested and used to build the vector indices. Multiple indices were built in order to test the parameters affecting the ingestion, such as chunk size and embedding model, and the scaling capacity across the 5 datasets sizes described in Section 4.1.2.

During inference, queries are processed as outlined in Section 4.4.1, eventually returning a set of $top_n$ chunks from the reranker, identically for each retrieval method. At this step, ground truth node IDs of the test instances together with IDs of RAG retrieval results were sent to functions calculating the score of each metric. In addition to checking the correctness of each retrieved chunk, the assigned scores and ranks were used. Results were lastly aggregated per test set.

## 4.6 Experimental Setup

This section describes the six testing stages conducted to evaluate and refine the RAG pipeline. The stages were executed sequentially, with each step building upon insights gained in the previous one. The first stage investigated the impact of tabular versus textual representation and served to determine an appropriate serializer function. The second stage focused on performance as a function of vector index size. This acted as a sanity check of the implemented pipeline, ensuring correct similarity search behavior as the number of indexed nodes increased. Stages three to five explored core architectural parameters of the RAG pipeline. Finally, the sixth stage compared the four retrieval approaches BL, SF, LF, and WF, using the parameter settings identified as beneficial in earlier stages. The six testing stages are summarised below:

1. Comparison of tabular serializers
2. Performance vs vector index size
3. Performance vs chunk size and top-$k$
4. Comparison of fusion modes
5. Comparison of embedding models
6. Comparison of retrieval methods

After Test 1, the best-performing serializer was chosen and used for the remaining test stages. For all tests, evaluation was run on the largest vector index, containing embeddings for all five tables and the set of 1,390 documents. The only exception was the specific test on vector index scale, which used all five vector indices.

### 4.6.1 Tabular Serializer Performance

The set of nine table serializer functions was used to create nine unique instances of vector indices containing the table data. This was accomplished by merging each unique table vector store with the fixed document data vector store produced by the largest document set, before building vector indices. This order of operations follows the Baseline approach, which requires concatenating the two vector stores.

The performance on the nine constructed joint indices were evaluated on the two test sets $Doc_q$, $Tab_q$. The intent of comparison was to select a best performing serializer to enable proper search for tabular data, which is inherently at a disadvantage. At the same time it was deemed important to sanity check that improvements in tabular retrieval do not come at the cost of a deterioration in document retrieval. Both hit rate and MRR were measured and a judgement on the performance on both these metrics were set to serve as the selection criterion for the serializer.

### 4.6.2 Vector Index Scaling

The retrieval performance was tested on the five subsets. All three question sets $Doc_q$, $Tab_q$ and $Combo_q$ were used, the combination set essentially producing an average performance of the prior two. Chunk size and overlap were kept at 256 and 20 for each test.

The test sets were run for the Baseline and Simple Fusion methods to include both a joint-index and a separate-index method in the scaling test. With this setup, both methods relied on the same embedding model, so similarity scores for each chunk, both document and tabular, were directly comparable between methods. Baseline retrieved the top-$k = 20$ chunks from the single joint index. Simple Fusion retrieved up to 20 chunks from each separate index, applied its fusion algorithm, and then outputted the top-$k = 20$. Due to the identical scoring, the final top 20 was the same, regardless of whether the best chunks come from one index or are split in some proportion across both of them.

### 4.6.3 Chunk Size and top-k Influence

The effect of chunk size and number of retrieved chunks were evaluated in a joint test, using chunk size of 256 and 512 tokens and top-$k$ values of 10-80 at 10-intervals, resulting in a total of 16 tests. This was done for two methods, BL and LF, to include both a joint-index and a separate-index approach. Late Fusion was chosen over Simple Fusion because Simple Fusion behaves identically to the Baseline in this setup, whereas LF allows us to differentiate between the two architectures.

As chunk size affects the balance of conciseness and noise or variation within single retrieval units, and the top-k parameter affects the total amount of retrieved tokens per query, these were evaluated simultaneously. Chunk size only affects the embeddings of document data, as table data is parsed row-wise, while top-$k$ affects retrieval of both. As the embeddings of document chunks may still affect the similarity search with respect to tabular queries, all three test sets $Doc_q$, $Tab_q$ and $Combo_q$ were used.

### 4.6.4 Fusion Modes

For the Late Fusion method, we tested the impact of the fusion algorithm by running all three test sets with three fusion strategies: Distribution-Based Fusion, Reciprocal Rank Fusion, and Relative Score Fusion. These experiments were conducted only with chunk size=512.

### 4.6.5 Weighted retriever

The primary experiments conducted on the weighted retriever include a hyperparameter ablation study focusing on the number of chunks top-$j$ retained per domain, as well as the temperature parameter $\alpha$, which controls the sensitivity of the softmax function used to determine the weight distribution. These experiments were conducted to identify hyperparameter configurations that were well-suited to the characteristics of the available data.

### 4.6.6 Retrieval Methods

Comparing the four implemented retrieval methods, each method employed a different combination of components, as described in Section 4.4.5. An overview of the evaluated configurations is provided in Table 4.2, where the number of embedding models and

retrievers is indicated numerically, and the presence or absence of a reranker and fusion component is denoted by O and X, respectively.

Each method was tested on all three test tests. The best performance of each method was reported with its parameter settings and compared, to select an overall best configuration.

Table 4.2: Overview of evaluated retrieval approaches and their architectural components.

| **Approach** | **Embedding model(s)** | **Retriever(s)** | **Reranker** | **Fusion** |
|---|---|---|---|---|
| Baseline | 1 | 1 | O | X |
| Simple Fusion | 1 | 2 | O | O |
| Late Fusion | 1 | 2 | O | O |
| Weighted Fusion | 1 | 2 | O | O |

# 5

# Results

The proposed RAG approaches are primarily assessed through ablation-style experiments, where individual components and hyperparameters are varied to analyze their contributions to the RAG performance. Following this structure, the majority of results are presented at the level of individual components and design choices.

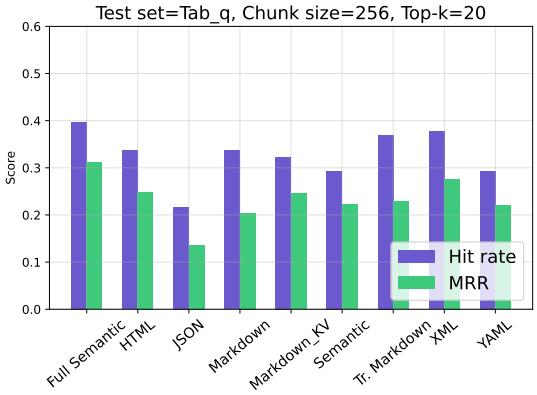
The evaluated components include tabular data formatting strategies, fusion-based retrieval methods, and domain weighting mechanisms. Formatting of tabular data is assessed first and the preferred tabular representation is fixed for all subsequent experiments. Fusion algorithms are then assessed using the previously selected hyperparameter settings, in order to choose a default retrieval configuration for the remaining experiments. Finally, domain weighting mechanisms are evaluated, using the best-performing retrieval pipeline configurations identified in the preceding evaluations. This assessment focuses on how different weighting schemes affect cross-domain retrieval behavior and overall retrieval effectiveness.

Alongside ablation experiments, and chronologically before the final evaluation of the RAG system, a qualitative user study was conducted to contextualize the system performance as well as aid in its development. While the experiments measure retrieval performance under controlled settings, the user study evaluate practical limitations and use-case focused aspects like data coverage and usability.

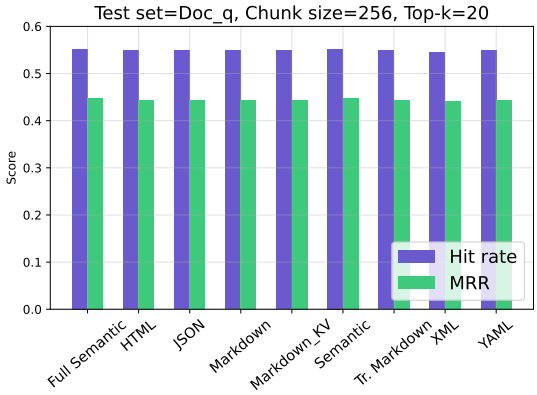
## 5.1 Tabular Serializer Performance

RAG performance comparison between the nine serializer functions indicate minimal impact on the $Doc_q$ test set scores and substantial impact on the $Tab_q$ test set. Figure 5.1 visualizes the hit rate@10 and MRR scores for the respective test set. On tabular queries, the *Rich semantic* serializer performs best on both metrics, scoring a 39.7% hit rate and 31.1% MRR, outperforming the second best serializer scores, both being set by the XML serializer, with 2.0 and 3.5 percentage points on the respective metrics. Scores for each serializer on each test set and metric is shown in Table 5.1.

On document queries, performance is nearly unaffected by the choice of table serializer. While most serializers produced identical hit rate@10 and MRR scores, the XML serializer suffered a 0.4 and 0.2 percentage point decrease, respectively, and the two semantic serializers achieved 0.4 percentage point higher scores on each metric.




(a) MRR and hit rate performance on tabular test queries ($Tab_q$)




(b) MRR and hit rate performance on document test queries ($Doc_q$)

Figure 5.1: Comparison of performance for nine serializers, exhibiting clear differences among on tabular test queries in the span 0.21-0.39 (hit rate) and 0.14-0.32 (MRR). Differences on documents queries are negligible (0.544 ± 0.004 on hit rate and 0.444 ± 0.004 on MRR)

Table 5.1: Hit rate@10 and MRR for nine serializers on $Tab_q$ and $Doc_q$ test sets. Hyperparameters are top-$k = 20$, top-$n = 10$, chunk overlap = 20, chunk size = 256.

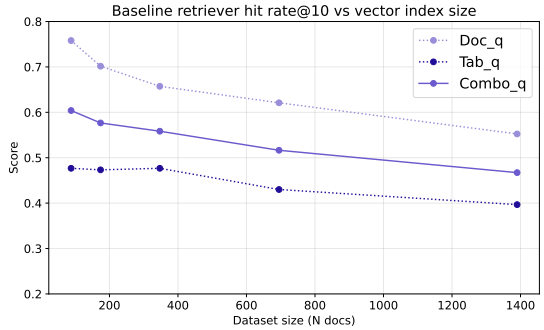
| Serializer | $Tab_q$ | | $Doc_q$ | |
|---|---|---|---|---|
| | Hit rate | MRR | Hit rate | MRR |
| Full Semantic | **0.3967** | **0.3114** | **0.5524** | **0.4478** |
| HTML | 0.3367 | 0.2483 | 0.5484 | 0.4438 |
| JSON | 0.2167 | 0.1365 | 0.5484 | 0.4438 |
| Markdown | 0.3367 | 0.2046 | 0.5484 | 0.4438 |
| Markdown KV | 0.3233 | 0.2457 | 0.5484 | 0.4438 |
| Semantic | 0.2933 | 0.2219 | **0.5524** | **0.4478** |
| Transposed Markdown | 0.3700 | 0.2289 | 0.5484 | 0.4438 |
| XML | 0.3767 | 0.2763 | 0.5444 | 0.4418 |
| YAML | 0.2933 | 0.2197 | 0.5484 | 0.4438 |

## 5.2 Vector Index Scaling

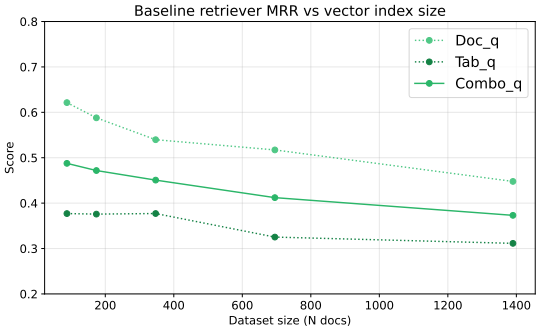
Evaluation on five sizes of vector indices shows deteriorating performance as more documents are ingested and indices consequently grow in size. For Baseline and Simple Fusion retriever all test sets $Doc_q$, $Tab_q$ and $Combo_q$ were run on each of the 5 vector index sizes. Figure 5.2 shows hit rate and MRR scores for the 5 indices on each test sets, displayed in separate figures for the two retriever methods. Results are shown for only the tests on the largest index size in Table 5.2. The full table including results in every index size in shown in Figure E.1 in Appendix E.

Across Baseline and Simple Fusion, performance is identical which is expected according to their configuration as explained in Chapter 4.6.2. All metrics show decreasing trends as index size grows. Document queries result in a higher performance of around 15-20 percentage points in hit rate compared to tabular queries. This gap increases for the lower vector indices. For MRR, the same pattern occurs, with a slightly smaller gap, 13 percentage points for the larger vector index size. Overall, retrieval hit rates span from

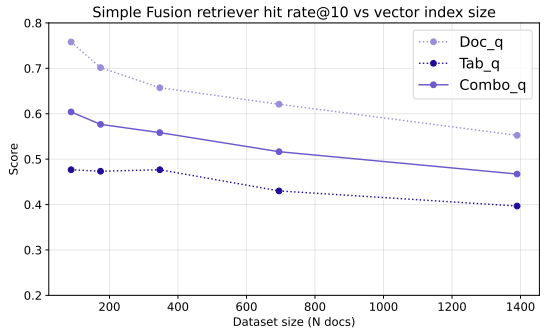
60.4-46.7% from smallest to largest vector index on the $Combo_q$ test set. MRR scores spans 48.8-37.3% using the same comparison. While hit rate performance shows that relevant chunks are found for almost 50% of the queries on the largest index, MRR score informs that on average, these relevant chunks are found on second to third place among the ordered outputs.



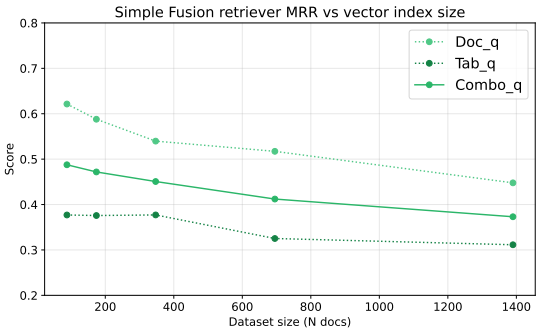

(a) Baseline retriever hit rate




(b) Baseline retriever MRR




(c) Simple Fusion retriever hit rate




(d) Simple Fusion retriever MRR

Figure 5.2: Hit rate and MRR scores vs vector index size for three test sets $Doc_q$, $Tab_q$ and $Combo_q$, shown separately for the Baseline (upper) and Simple Fusion (lower) retriever methods. Hyperparameters used: Top-$k$ = 20, Top-$n$ = 10, chunk overlap = 20, chunk size = 256

Table 5.2: Scores for Baseline and Simple Fusion methods for index size 1,390, the largest size.

| Method | Test set | Hit rate@10 | Precision@10 | F1@10 | Recall@10 | MRR |
|---|---|---|---|---|---|---|
| Baseline | $Doc_q$ | **0.5524** | 0.1004 | 0.0552 | 0.5524 | **0.4478** |
| Baseline | $Tab_q$ | 0.3967 | 0.1597 | 0.1410 | 0.1841 | 0.3114 |
| Baseline | $Combo_q$ | 0.4672 | 0.1583 | 0.1022 | 0.3508 | 0.3731 |
| Simple Fusion | $Doc_q$ | **0.5524** | 0.1004 | 0.0552 | 0.5524 | **0.4478** |
| Simple Fusion | $Tab_q$ | 0.3967 | 0.1597 | 0.1410 | 0.1841 | 0.3114 |
| Simple Fusion | $Combo_q$ | 0.4672 | 0.1583 | 0.1022 | 0.3508 | 0.3731 |

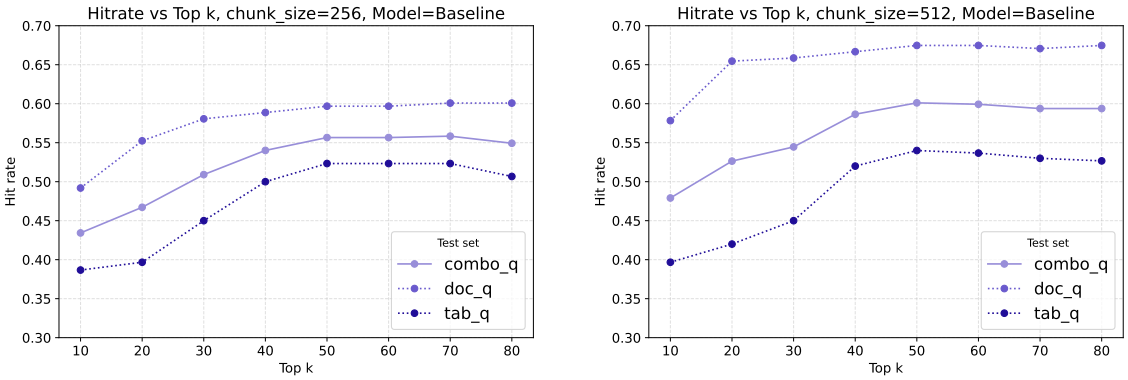
## 5.3 Chunk Size and Top-k Influence

Given the identical performance of Baseline and Simple Fusion shown in Figure 5.2, the experiments investigating impact of chunk size and top-$k$ were conducted using the Late

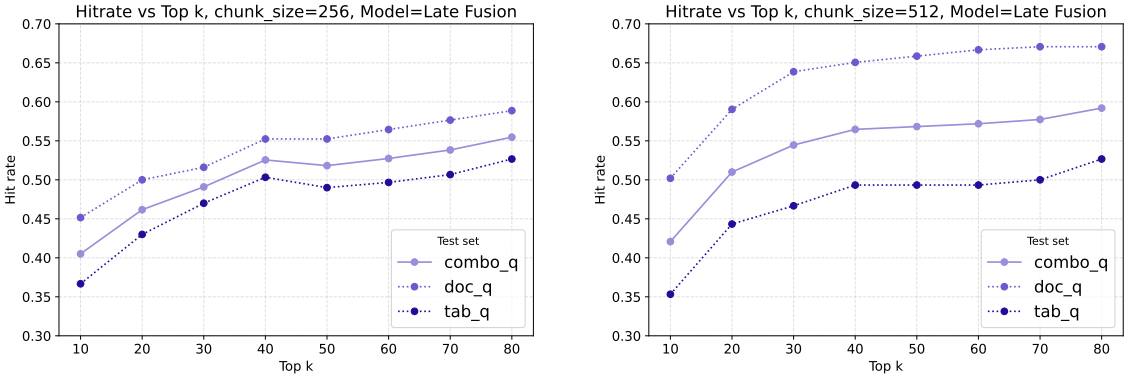
Fusion retriever using a relative score fusion strategy. The chunk size and top-$k$ parameters are varied over $256, 512$ and $10 - 80$, respectively. Figure 5.3 illustrates how hit rate is affected by changes in top-$k$, while Figure 5.4 presents the corresponding changes in MRR. A table containing the full results is given in Table E.2 in Appendix E

Comparing hit rate for chunk size 256 vs 512, the larger chunk size consistently outperforms the smaller, the single exception being for tabular queries at top-$k$=10 for the Late Fusion method. In general scores increase sharply when increasing from a low top-$k$, however these improvements stagnate after top-$k$=40. For the Late Fusion method, however, scores increase to give the best performance at top-$k$=80. Comparing Baseline with Late Fusion, the Baseline method performs better around top-$k$ values of 40 and 50.

For MRR, the performance shows similar patterns as for hit rate, albeit with less pronounced differences. MRR scores nearly stagnates after top-$k$=40 at chunk size 512, while for the 256 chunk size some improvement is still gained for the highest values of top-$k$.




(a) Baseline, CS=256 (left) and 512 (right)

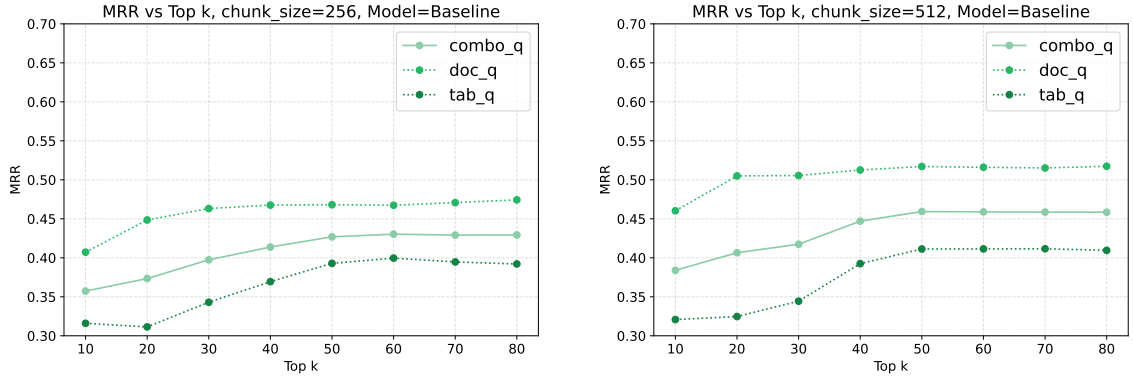



(b) Late Fusion, CS=256 (left) and 512 (right)

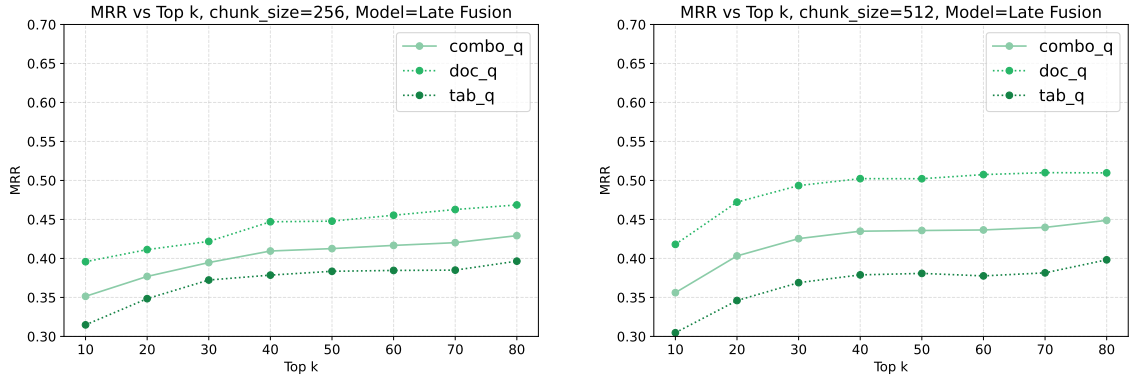
Figure 5.3: Hit rate scores for Baseline and Late Fusion methods with chunk sizes 256 and 512, evaluated on three data sets $Doc_q$, $Tab_q$ and $Combo_q$.

## 5.4 Fusion Modes

For the Late Fusion method, three fusion algorithms were evaluated across the three test sets. Specifically, the evaluated fusion algorithms were reciprocal rank fusion, relative score fusion, and distribution-based score fusion. The resulting MRR and hit rate values are




(a) Baseline, CS=256 (left) and 512 (right)




(b) Simple Fusion, CS=256 (left) and 512 (right)

Figure 5.4: MRR scores for Baseline and Simple Fusion methods with chunk sizes 256 and 512, evaluated on three data sets $Doc_q$, $Tab_q$ and $Combo_q$.

shown in Figure 5.5.

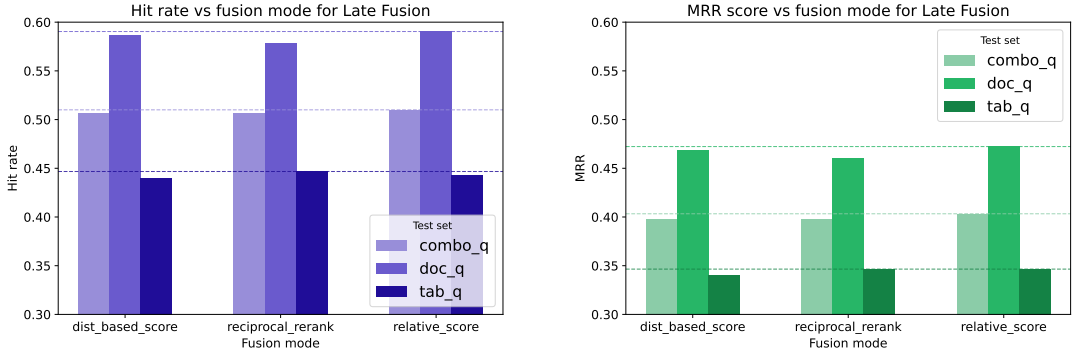
Across the evaluated datasets, the observed performance differences between fusion algorithms are small, with relative score fusion achieving marginally higher scores in several cases. This holds for both hit rate and MRR.

Table 5.3: MRR and HitRate@10 performance for three fusion modes on the test sets $Doc_q$, $Tab_q$, and $Combo_q$. Vector index size: 1,390.

| | Reciprocal rerank | | Relative score | | Dist-based score | |
|---|---|---|---|---|---|---|
| Test set | HitRate@10 | MRR | HitRate@10 | MRR | HitRate@10 | MRR |
| $Doc_q$ | 0.5783 | 0.4604 | **0.5904** | **0.4723** | 0.5863 | 0.4682 |
| $Tab_q$ | **0.4467** | **0.3465** | 0.4433 | 0.3460 | 0.4400 | 0.3398 |
| $Combo_q$ | 0.5064 | 0.3982 | **0.5100** | **0.4033** | 0.5064 | 0.3980 |

## 5.5 Weighted Retriever

This section presents the results obtained with the weighted retriever, evaluating the effects of the truncation parameter top-$j$ and the temperature parameter $\alpha$. All experiments were conducted using a chunk size of 512 and a fixed retrieval depth of top-$k = 40$, taken from



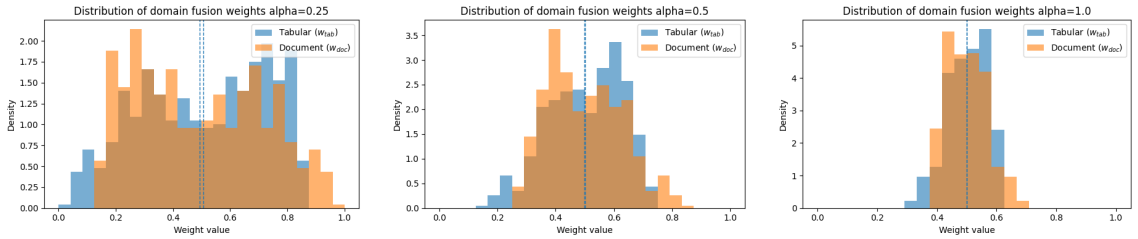

(a) Hit rate scores

(b) MRR scores

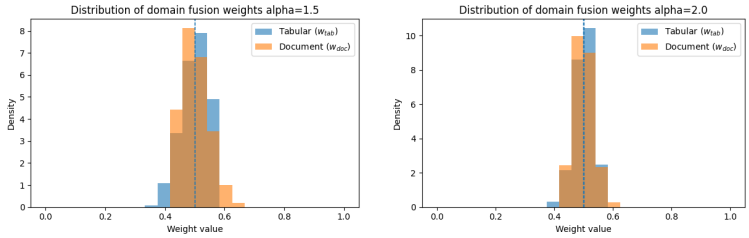
Figure 5.5: MRR and hit rate for 3 fusion modes evaluated on the three test sets $Doc_q$, $Tab_q$ and $Combo_q$. Hyperparameters are : Top-$k$ = 20, Top-$n$ = 10, chunk overlap = 20, chunk size = 512. Vector index size 1,390 (largest)

the previous experiments. Table 5.4 reports retrieval performance for different values of top-$j$. Across all reported metrics, performance increases with larger values of top-$j$, with the highest scores observed at $top_j = 40$, indicating that using all the retrieved chunks performs the best. The corresponding runtime measurements are shown in Table 5.5. Retrieval time increases with larger top-$j$, while reranking time remains constant across configurations, resulting in an increase in total runtime.

Table 5.6 presents retrieval performance for different values of the weighting parameter $\alpha$. The highest overall metric values are observed at $\alpha = 1.0$, with similar performance for $\alpha = 1.5$ and $\alpha = 2.0$ across most metrics. Figure 5.6 illustrates the distribution of retrieval weights for different $\alpha$ values. As $\alpha$ increases, the weight distributions become progressively concentrated towards 0.5, with the strongest concentration observed at $\alpha = 2.0$.




(a) Weights distribution, $\alpha = 0.25$. (b) Weights distribution, $\alpha = 0.5$. (c) Weights distribution, $\alpha = 1.0$.




(d) Weights distribution, $\alpha = 1.5$. (e) Weights distribution, $\alpha = 2.0$.

Figure 5.6: Comparison of weight distribution across $\alpha = \{0.25, 0.5, 1.0, 1.5, 2.0\}$ in weighted retrieval. A higher alpha restricts weight values closer to 0.5 (balanced weight)

Table 5.4: Weighted Retrieval performance metrics for different values of top truncated chunks top-$j$. Chunk size = 512, top-$k$ = 40

| top-$j$ | HitRate@$k$ | F1@$k$ | Precision@$k$ | Recall@$k$ | MRR |
|---|---|---|---|---|---|
| 10 | 0.5829 | 0.1911 | 0.1213 | 0.4495 | 0.4488 |
| 20 | 0.5847 | 0.1920 | 0.1220 | 0.4498 | 0.4536 |
| 40 | **0.5883** | **0.1937** | **0.1235** | **0.4490** | **0.4550** |

Table 5.5: Weighted Retrieval runtime breakdown for different values of top-$k$. GPU: Tesla V100-DGXS-32GB

| top-$j$ | Retrieval Time (s) | Rerank Time (s) | Total Time (s) |
|---|---|---|---|
| 10 | 0.74 | 2.14 | 2.88 |
| 20 | 1.00 | 2.14 | 3.14 |
| 40 | 1.47 | 2.14 | 3.61 |

## 5.6 User studies

The overall qualitative feedback from user tests covers a mix of strengths and insufficiencies of the system. The condensed list of received responses is listed per the four topics user tasks, system behavior, data and usability in Figure A.2 in Appendix A. The main points are stated here.

On the topic of user tasks, the participant *personas* identified several high-value tasks where a RAG assistant could reduce manual effort. They point to main tasks like report writing on materials and selecting new or replacement materials for components. Concrete subtasks include locating specific material data points, tables, or whole documents relating to certain materials or use-cases.

In the tested state of the system, participants repeatedly encountered lack of key data sources. Multiple suggestions for internal sources and external databases to add were given. In the majority of these cases, it could be confirmed that the particular data had not been provided to the RAG system at all, due to limitations outside of the influence of the thesis. Other comments noted that lack of data was generally handled well by the system, however some occasions of mistakes and an inability of the system to identify or explain that data was missing were seen.

Usability feedback included suggestions of integration with existing systems, traceability and synonym handling. Among existing strengths at the current state was a perceived good query understanding, which was observed when data existed. Then, the system was able to compare material options as well as allowing for iterative refinement of queries, e.g. follow-up question to learn more about certain aspects.

Table 5.6: Weighted Retrieval performance for different alpha values with top-$j = 40$.

| $\alpha$ | HitRate@$k$ | F1@$k$ | Precision@$k$ | Recall@$k$ | MRR |
|---|---|---|---|---|---|
| 0.25 | 0.5865 | 0.1940 | 0.1239 | 0.4477 | 0.4532 |
| 0.50 | 0.5883 | 0.1937 | 0.1235 | 0.4490 | 0.4550 |
| **1.00** | **0.5938** | **0.1950** | **0.1242** | **0.4527** | **0.4587** |
| 1.50 | 0.5920 | 0.1945 | 0.1239 | 0.4526 | 0.4564 |
| 2.00 | 0.5920 | 0.1947 | 0.1240 | **0.4527** | 0.4559 |

# 6

# Discussion

This chapter presents insights from the experimental results reported in Chapter 5, following a similar outline. The findings from each set of experiments are interpreted with respect to the research objectives of the thesis, with particular emphasis on how the evaluated design choices influence retrieval performance in multi-domain RAG systems. Based on these interpretations, the chapter discusses the broader implications of the results and identifies limitations encountered during the study. This is followed by a discussion of potential directions for future work, outlining opportunities for extending and improving the proposed methods beyond the scope of this thesis. Finally, the key contributions of this thesis are presented in a conclusion, highlighting the overall outcomes and their relevance to the intended domain of application.

## 6.1 Tabular Serializer Performance

The Fully Semantic serializer achieved the strongest overall performance. It outperformed the remaining eight serializer variants on both hit rate and MRR, without degrading in terms of document-based query performance. It was therefore selected as the default serializer for all subsequent experiments.

These findings suggest that the pre-trained embedding model, which was primarily trained on unstructured natural language text, benefits from a more semantic representation of tabular data. Serializing tabular content as coherent natural language statements appears to align more closely with the embedding model's training distribution than compact structured formats like YAML or JSON. While structured formats preserve syntactic clarity, they may fail to capture the semantic relationships between keys and values as effectively as a fully semantic representation. Consequently, more semantically rich tabular representations can lead to higher preservation of content after embedding, leading to improved retrieval performance.

An interesting result was the strong performance of the Transposed Markdown format. This format, which commonly appears in unstructured documents after conversion to Markdown, achieved competitive results on the tabular test set. This suggests that the pre-trained embedding model is already familiar with this representation and is, to some extent, able to reliably capture tabular information embedded within unstructured documents. These results indicate that tabular data expressed in simple Markdown table structures may be effectively handled by general-purpose embedding models, provided that tables are not overly complex or deeply nested within surrounding text. This finding highlights the potential for leveraging existing document parsing pipelines to extract useful tabular signals without requiring fully structured representations in all cases.

## 6.2 Vector Index Scaling

The vector index scaling experiments showed a reduction in retrieval performance as the index size increased. This effect was more pronounced for document queries, although their overall performance in terms of MRR and Hit Rate remained higher than that of tabular queries. Since only the document portion of the embedded data grows while the tabular component remains constant, the tabular similarity search remains unaffected by competition from more similar chunks, and a larger performance decline for document queries is to be expected. A smaller decline for tabular queries is also reasonable, given that the index is unified and increases in size overall.

Although the largest index evaluated in this study remains modest from a RAG perspective and does not approach system scalability limits, the observed performance degradation at moderate index sizes are within acceptable bounds and indicates that the baseline setup continues to function reliably. Retrieval performance remains stable enough to support both tabular and document queries without fundamental deterioration in either domain.

## 6.3 Chunk Size and top-$k$ Influence

Across both the baseline and late fusion retrieval methods, the results show that larger chunk sizes and higher top-$k$ values generally lead to improved retrieval performance. Increasing the chunk size consistently improved performance, with the effect being more pronounced for the late fusion approach. Similarly, higher top-$k$ values tended to yield stronger results across most configurations, indicating that retrieving a larger candidate set increases the likelihood of identifying relevant chunks.

These findings highlight the importance of providing the reranking stage with a sufficiently large and informative candidate pool. At the same time, Table E.2 in Appendix E illustrates how increasing chunk size and retrieval depth introduces additional computational cost, as more chunks must be retrieved and reranked. This emphasizes the inherent trade-off between retrieval effectiveness and computational efficiency that must be considered when configuring RAG systems in practice.

## 6.4 Fusion Modes

As a core feature of the LF method, fusion modes were compared, with low differentiation between available options as the outcome. With only subtle differences for both hit rate and MRR, relative score mode still produced the highest scores for both metrics, even though tabular hit rate was higher for reciprocal rank. A more defined difference could have been expected, however as the fusion technique is intended to fuse results from both BM-25 and dense retrieval, as well as between different embedding models, it is plausible that the current test case with same embedding model and only dense embeddings would produce a very similar output, albeit using the different modes.

## 6.5 Weighted Retriever

The baseline retrieval model, using its optimized hyperparameters, achieved a hit rate of 0.5865 and an MRR of 0.447, whereas the weighted retriever, also using optimized hyperparameters, achieved a higher hit rate of 0.5938 and an MRR of 0.4587. These results show a consistent improvement in retrieval performance when domain weighting is applied.

Overall, the weighted retriever provided more effective ranking of relevant chunks compared to the baseline approach, indicating that incorporating domain-aware weighting can improve retrieval quality in multi-domain RAG settings without requiring hard routing decisions. The results further show that hit rate improves slightly as the truncation parameter top-$j$ increases, up to the full retrieved set. In this configuration, each model retrieved 40 chunks, and the maximum value of top-$j$ was therefore also 40.

The primary trade-off associated with weighted retrieval is the additional computational overhead introduced by the weighting mechanism, as reflected in the runtime results. Although the cross-encoder used to compute weighting scores is lightweight, it nevertheless increases inference time compared to the baseline system. A direct runtime comparison on identical hardware was not possible, as the experiments were conducted on different GPUs. However, given that the NVIDIA H100 significantly outperforms the Tesla V100-DGXS-32GB, and the weighted retriever exhibited an increase in total runtime of approximately two seconds despite running on the weaker GPU, the computational overhead of the weighted retriever remains moderate.

## 6.6 Usability

The results of user studies indicate that the system could contribute value to the workflows; however, multiple factors are currently limiting the system from reaching a satisfactory level of performance. The identification of possible tasks aligned with what a RAG system is intended to do, and helped in understanding what users need, as well as how synthetic test set questions should be generated. However, an additional more in-depth exploration and classification of user task would be beneficial to both the development and evaluation of the RAG.

Data coverage was found to be low, which was reasonable given the limited set included in the designated material data corpora. This factor limited testing of the model; however, all the desired types of data domains, particularly both tabular and textual, were represented and the balance in retrieval between these could be tested. Both types were found to return decent retrieval and provision of relevant responses in cases where the desired data existed.

On a general performance level, the system was found to handle the breadth of task expected by the users. The occasions of failed tasks were pinpointed by the test users to be detrimental to user trust, independent of whether they were produced in the retrieval step by retrieving incorrect information, or in the generation step by failing to appropriately present and cite sources of the data. An additional weakness can be revealed by user queries for which up-to-date data sources coexist with older versions of the same data, and the older is still presented to the user, leading to the model using outdated sources. This was equally undesirable and *personas* expressed the need for an extremely high success rate, as even a single mistake may cause user distrust towards the RAG system.

Given the strengths experienced and described by the user, it is indicated that the implemented model is suitable for the use case. However, the errors found require further improvements before the system could be deployed to production. To further improve the system and adapt it to the use case, a more extensive dataset would be highly beneficial, as it would allow for a more detailed assessment of how the model acts when data is certain to be present in the system's knowledge base.

## 6.7 Future Work

Improvements to tabular query performance remain an important direction for future work, as tabular retrieval currently represents the primary performance bottleneck in terms of both hit rate and MRR. In this work, tabular data was embedded using pre-trained embedding models applied to serialized table representations. An alternative approach would be to explicitly identify lookup-style queries and employ an LLM to generate corresponding SQL queries, an approach which would align with the TableRAG framework [16] which makes use of LLMs for SQL generation to directly retrieve relevant rows from structured databases. Additionally, a performance improvement of the remaining serializers could be explored by fine-tuning of the tabular embedding model to better recognize the patterns introduced by different serialization formats.

Another direction for future work is the evaluation of alternative embedding models. A key advantage of the proposed architecture in this thesis which separates retrieval across different domains, is the flexibility to use domain-specific embedding models tailored to the characteristics of each data source. For instance, embedding models optimized for natural language may be more suitable for unstructured documents, while specialized models designed for tabular or numerical data could better capture the semantics of structured databases. Such strategy has the potential to improve retrieval performance by aligning representation learning with the inherent structure of each domain. However, due to time constraints, a systematic exploration and benchmarking of different embedding models was not feasible within the scope of this project.

While the previous direction focuses on improving model performance, another important aspect of future work concerns the evaluation methodology itself. To better align with the intended use-case, further work on heterogeneous industrial data should attempt to achieve a manually constructed test set of realistic queries and ground truths both ID based and content based. As opposed to a synthetic generated test set, this would qualitatively improve evaluation on both retrieval and generative aspects, thus enabling a more effective testing of developed systems by means of higher quality evaluation.

With the main focus of the current work being retrieval techniques, it is still motivated to expand the evaluation schema to cover generative aspects to examine how well the RAG system as a whole utilizes the retrieval chunks. This is particularly of interest when retrieval results to a large degree consist of tabular data, which, having successfully been retrieved and prioritized by help of the embedding models and cross-encoder, still needs to be processed successfully by the LLM producing a final response.

## 6.8 Conclusion

Investigating how Retrieval-Augmented Generation can be implemented to support industrial material decision-making across heterogeneous sources, this thesis developed and evaluated a modular multi-domain RAG system. Focusing on the challenge of retrieving information from both the textual and tabular domain, the outcomes are mainly related to the representation of tabular data and the fusion of the textual and tabular domains.

A semantically enriched table row representation was found to outperform alternatives while not degrading textual retrieval. Beyond tabular representation, retrieval effectiveness was strongly influenced by standard RAG hyperparameters. Larger chunk sizes and higher top-$k$ generally improved hit rate and MRR for both baseline and late fusion, at increased

computational cost, while performance declined as the unified vector index grew, with document queries consistently outperforming tabular queries. Alternate fusion modes within the late fusion method showed only minor performance differences, likely due to the homogeneous embedding space due to using only one embedding model. The weighted retriever method still provided an improvement over the baseline, demonstrating the benefit of soft domain weighting in multi-domain RAG.

These findings indicate that it is feasible to implement a multi-domain RAG system that exceeds a baseline configuration in retrieval performance for heterogeneous corpora in the industrial domain. The practical implication is that, in industrial RAG settings where structured and unstructured sources must be queried jointly, meaningful gains can be achieved by aligning structured representations with the linguistic assumptions of the embedding model and by introducing lightweight domain-aware ranking mechanisms.

Limitations to the study that affect the outcome include that evaluation relied on synthetic test data due to the absence of established benchmarks and the limited feasibility of expert annotation within the project scope, which may not fully capture the nuance and distribution of real industrial information needs. The qualitative user study included four participants representing different personas, providing valuable, albeit small scale, feedback. In addition, this work emphasized retrieval quality rather than end-to-end generation, meaning that response factuality, citation behavior, and failure transparency were not systematically quantified. User feedback also highlighted practical deployment risks that are only partially reflected in retrieval metrics, including insufficient data coverage, occasional mistakes that can undermine trust, and challenges when outdated and up-to-date sources coexist.

Future work should prioritize improving tabular retrieval, which currently underperforms compared to textual retrieval. Promising directions include fine-tuning embedding models for non-semantic serializers, systematically extracting and structuring tables from unstructured documents, evaluating the generative stage to assess end-to-end RAG quality, and transitioning from synthetic to real industrial data with labeled ground truth. For further efforts in multi-domain industrial RAG applications, it is encouraged also to increase the inclusion of and evaluation on real industrial data and expert-labeled test sets, as well as to evaluate the generative stage and end-to-end performance. Overall, this thesis demonstrates that multi-domain RAG for industrial material data can be feasible and beneficial for material decision support. It conveys hyperparameter tradeoffs and retrieval design choices, and highlights key features needed to develop RAG systems from prototype performance towards deployment-ready and trustworthy digital assistants.

# Bibliography


[1] K. Hummel and D. Jobst, "An overview of corporate sustainability reporting legislation in the european union," *Accounting in Europe*, vol. 21, no. 3, pp. 320–355, 2024. DOI: 10.1080/17449480.2024.2312145.

[2] X. Zhao, X. Zhou, and G. Li, "Chat2data: An interactive data analysis system with rag, vector databases and llms," *Proceedings of the VLDB Endowment*, vol. 17, pp. 4481–4484, 12 Aug. 2024, ISSN: 21508097. DOI: 10.14778/3685800.3685905. [Online]. Available: https://dl.acm.org/doi/10.14778/3685800.3685905.

[3] Y. Gao et al., "Retrieval-augmented generation for large language models: A survey," *Proceedings - 2024 Conference on AI, Science, Engineering, and Technology, AIxSET 2024*, pp. 166–169, Dec. 2023. DOI: 10.1109/AIxSET62544.2024.00030. [Online]. Available: https://arxiv.org/abs/2312.10997v5.

[4] J. Chen, H. Lin, X. Han, and L. Sun, "Benchmarking large language models in retrieval-augmented generation," Tech. Rep., 2024. [Online]. Available: https://arxiv.org/abs/2309.01431.

[5] Y. Zhou et al., "Trustworthiness in retrieval-augmented generation systems: A survey," Sep. 2024. [Online]. Available: http://arxiv.org/abs/2409.10102.

[6] P. Lewis et al., *Retrieval-augmented generation for knowledge-intensive nlp tasks*, 2021. arXiv: 2005.11401 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2005.11401.

[7] K. Guu, K. Lee, Z. Tung, P. Pasupat, and M.-W. Chang, *Realm: Retrieval-augmented language model pre-training*, 2020. arXiv: 2002.08909 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2002.08909.

[8] S. Borgeaud et al., *Improving language models by retrieving from trillions of tokens*, 2022. arXiv: 2112.04426 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2112.04426.

[9] G. Izacard et al., *Atlas: Few-shot learning with retrieval augmented language models*, 2022. arXiv: 2208.03299 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2208.03299.

[10] A. Asai, Z. Wu, Y. Wang, A. Sil, and H. Hajishirzi, *Self-rag: Learning to retrieve, generate, and critique through self-reflection*, 2023. arXiv: 2310.11511 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2310.11511.

[11] S. Yao et al., *React: Synergizing reasoning and acting in language models*, 2023. arXiv: 2210.03629 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2210.03629.

[12] J. Herzig, P. K. Nowak, T. Müller, F. Piccinno, and J. Eisenschlos, "Tapas: Weakly supervised table parsing via pre-training," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Association for Computational Linguistics, 2020. DOI: 10.18653/v1/2020.acl-main.398. [Online]. Available: http://dx.doi.org/10.18653/v1/2020.acl-main.398.

[13] B. Wang, R. Shin, X. Liu, O. Polozov, and M. Richardson, *Rat-sql: Relation-aware schema encoding and linking for text-to-sql parsers*, 2021. arXiv: 1911.04942 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1911.04942.

[14] Z. Li et al., *Structrag: Boosting knowledge intensive reasoning of llms via inference-time hybrid information structurization*, 2024. arXiv: 2410.08815 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2410.08815.

[15] S. Xue et al., *Db-gpt: Empowering database interactions with private large language models*, 2024. arXiv: 2312.17449 [cs.DB]. [Online]. Available: https://arxiv.org/abs/2312.17449.

[16] X. Yu, P. Jian, and C. Chen, "Tablerag: A retrieval augmented generation framework for heterogeneous document reasoning," Tech. Rep., Sep. 2025. [Online]. Available: https://github.com/yxh-y/TableRAG.

[17] W. Chen, H. Zha, Z. Chen, W. Xiong, H. Wang, and W. Wang, *Hybridqa: A dataset of multi-hop question answering over tabular and textual data*, 2021. arXiv: 2004.07347 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2004.07347.

[18] C. Zhang and Q. Chen, "Hd-rag: Retrieval-augmented generation for hybrid documents containing text and hierarchical tables," Apr. 2025. [Online]. Available: https://arxiv.org/abs/2504.09554v1.

[19] W. Yeo, K. Kim, S. Jeong, J. Baek, and S. J. Hwang, "Universalrag: Retrieval-augmented generation over corpora of diverse modalities and granularities," Tech. Rep., May 2025. [Online]. Available: https://universalrag.github.io..

[20] T. Schick et al., *Toolformer: Language models can teach themselves to use tools*, 2023. arXiv: 2302.04761 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2302.04761.

[21] E. Tyndall, C. Gayheart, A. Some, J. Genz, T. Wagner, and B. Langhals, "Impact of retrieval augmented generation and large language model complexity on undergraduate exams created and taken by ai agents," *Data and Policy*, vol. 7, Aug. 2025, ISSN: 26323249. DOI: 10.1017/dap.2025.10024.

[22] M. Alkhalaf, P. Yu, M. Yin, and C. Deng, "Applying generative ai with retrieval augmented generation to summarize and extract key clinical information from electronic health records," *Journal of Biomedical Informatics*, vol. 156, p. 104 662, Aug. 2024, ISSN: 1532-0464. DOI: 10.1016/J.JBI.2024.104662.

[23] P. D. Turney and P. Pantel, "From frequency to meaning: Vector space models of semantics," *Journal of Artificial Intelligence Research*, vol. 37, pp. 141–188, Feb. 2010, ISSN: 1076-9757. DOI: 10.1613/jair.2934. [Online]. Available: http://dx.doi.org/10.1613/jair.2934.

[24] A. Vaswani et al., *Attention is all you need*, 2023. arXiv: 1706.03762 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1706.03762.

[25] N. Reimers and I. Gurevych, *Sentence-bert: Sentence embeddings using siamese bert-networks*, 2019. arXiv: 1908.10084 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1908.10084.

[26] P. Yin, G. Neubig, W.-t. Yih, and S. Riedel, *Tabert: Pretraining for joint understanding of textual and tabular data*, 2020. arXiv: 2005.08314 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2005.08314.

[27] W. Chen et al., *Tabfact: A large-scale dataset for table-based fact verification*, 2020. arXiv: 1909.02164 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1909.02164.

[28] X. Ji, P. Glenn, A. G. Parameswaran, and M. Hulsebos, *Target: Benchmarking table retrieval for generative tasks*, 2025. arXiv: 2505.11545 [cs.IR]. [Online]. Available: https://arxiv.org/abs/2505.11545.

[29] Qdrant, *Qdrant vector database*, High-performance vector similarity search engine. Accessed: 2026-01-02, 2023. [Online]. Available: https://github.com/qdrant/qdrant.

[30] ChromaDB, *Chroma*, Open-source embedding database. Accessed: 2026-01-02, 2023. [Online]. Available: https://github.com/chroma-core/chroma.

[31] V. Karpukhin et al., "Dense passage retrieval for open-domain question answering," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, B. Webber, T. Cohn, Y. He, and Y. Liu, Eds., Online: Association for Computational Linguistics, Nov. 2020, pp. 6769–6781. DOI: 10.18653/v1/2020.emnlp-main.550. [Online]. Available: https://aclanthology.org/2020.emnlp-main.550/.

[32] J. Lin, R. Nogueira, and A. Yates, *Pretrained transformers for text ranking: Bert and beyond*, 2021. arXiv: 2010.06467 [cs.IR]. [Online]. Available: https://arxiv.org/abs/2010.06467.

[33] M. Wang, X. Xu, Q. Yue, and Y. Wang, *A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search*, 2021. arXiv: 2101.12631 [cs.IR]. [Online]. Available: https://arxiv.org/abs/2101.12631.

[34] E. Dilocker et al., *Weaviate*, Open-source vector database and search engine, 2024. [Online]. Available: https://github.com/weaviate/weaviate.

[35] M. Mazzeschi, *Distribution-based score fusion (dbsf): A new approach to vector search ranking*, https://medium.com/plain-simple-software/distribution-based-score-fusion-dbsf-a-new-approach-to-vector-search-ranking-f87c37488b18, Accessed: 2026-01-07, 2023.

[36] G. V. Cormack, C. L. A. Clarke, and S. Buettcher, "Reciprocal rank fusion outperforms condorcet and individual rank learning methods," in *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '09, Boston, MA, USA: Association for Computing Machinery, 2009, pp. 758–759, ISBN: 9781605584836. DOI: 10.1145/1571941.1572114. [Online]. Available: https://doi.org/10.1145/1571941.1572114.

[37] R. Nogueira and K. Cho, *Passage re-ranking with bert*, 2020. arXiv: 1901.04085 [cs.IR]. [Online]. Available: https://arxiv.org/abs/1901.04085.

[38] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[39] EPD International AB, *Environdec: Epd library*, https://environdec.com/library, Accessed: 2024-01-15, 2024.

[40] Apache Software Foundation, *Apache parquet*, https://parquet.apache.org/, Accessed: 2026-01-15, 2013.

[41] M. Joshi, E. Choi, D. S. Weld, and L. Zettlemoyer, *Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension*, 2017. arXiv: 1705.03551 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1705.03551.

[42] Z. Yang et al., *Hotpotqa: A dataset for diverse, explainable multi-hop question answering*, 2018. arXiv: 1809.09600 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1809.09600.

[43] K. R. Davis, B. Peabody, and P. Leach, *Universally Unique IDentifiers (UUIDs)*, RFC 9562, May 2024. DOI: 10.17487/RFC9562. [Online]. Available: https://www.rfc-editor.org/info/rfc9562.

[44] J. Chen, S. Xiao, P. Zhang, K. Luo, D. Lian, and Z. Liu, *Bge m3-embedding: Multilingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation*, 2024. arXiv: 2402.03216 [cs.CL].

[45] J. Liu, *LlamaIndex*, Nov. 2022. DOI: 10.5281/zenodo.1234. [Online]. Available: https://github.com/jerryjliu/llama_index.

[46] K. Zhu et al., "Rageval: Scenario specific rag evaluation dataset generation framework," in *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Association for Computational Linguistics, 2025, pp. 8520–8544. DOI: 10.18653/v1/2025.acl-long.418.

# A

# Appendix - User studies

## A.1 User study questions

During each demo session, the following questions were asked to the personas, focusing on the RAG agent's behavior. Some of these questions may have overlapping answers, and some were not possible to answer due to insufficient underlying data to test the model's full capabilities. The responses from the session have been condensed and categorized into 4 focus areas: User tasks, the RAG system's behavior, Data, and Usability. The main questions were:

- To what extent does the system seem to understand the queries?
- Were the responses accurate and relevant to answer your queries?
- At what level of complexity does the model fail to grasp the task?
- Does failed responses give a clue to why the question was not answered properly?
- How well can the model come around difficulties by clarification of queries?
- To what extent does it seem like the data you expect was present in the dataset?
- Does the system provide source references?
- What aspects of the system's responses contributed to confidence in it?
- What aspects of the system's responses contributed to distrust towards it?
- How can this system fit into your workflow?
- What is your overall impression of the system at the current state (1-5)?

Figure A.1: Questions put to test users before and during RAG test sessions.

## A.2 User study responses

**User tasks**

- Support for CMD report writing, particularly linking CSD document material specifications to specific materials, which is currently handled manually across many files.
- Comparing components and materials, including identifying replacement options.
- Locating specific standards or tables needed for requirements and engineering work.
- Purchasing use cases such as creating future purchasing plans using internal standards and sustainability documents.

**System behavior (perceived reasoning capabilities and handling of both success and failure in providing relevant responses)**

- The system was generally perceived to understand queries and attempt meaningful comparisons when relevant data existed.
- Failures were sometimes well-motivated, often with lack of data as the reason, but sometimes the RAG system did not explain why it failed or where answers could become misleading.
- Observed cases where the agent referenced older standards when newer versions might exist, which is a risk for correctness and trust.

**Data (coverage and missing sources)**

- Need for CMD tabular information for plastics (and metals to some extent).
- Suggested additions: Sphera, EcoInvent, CO2AI, CAMPUS-plastics, and potentially CPI or other indices for virgin material cost baselines.
- Purchasing-specific needs included QCDF (Quality, Cost, Delivery, Flexibility) risk information.

**Usability (features and performance relevant for real work tasks)**

- Strong need for table-centric lookup, filtering, and comparison capabilities.
- Requests to include digitalized standard, such as ESPR (Ecodesign for Sustainable Products Regulation).
- Need for persistent chat history and a personal track record to support traceability and backtracking decisions.
- Handling of domain-specific nomenclature and strong synonym understanding across areas where the same terms may be used differently.
- Desire for clearer transparency about what sources are available (e.g., a document describing the current content coverage).
- Value in iterative querying and refining searches by adding more criteria.
- Preference to unify tools within the organization rather than introducing another isolated system.

Figure A.2: Condensed responses from users test, categorized into the four topics user tasks, system behavior, data and usability. Aggregated from tests with four separate *personas*.

# B

# Tabular conversion dictionaries

Table B.1: Excerpt from dictionary used for table: chemical_composition

| **Original column heading** | **Mapped name** |
|---|---|
| STD Material Name | Material Name |
| c_min | Minimum Carbon (%) |
| c_max | Maximum Carbon (%) |
| si_min | Minimum Silicon (%) |
| w_min | Minimum Tungsten (%) |
| material_id | Material ID |

Table B.2: Excerpt from dictionary used for table material_composition

| **Original column heading** | **Mapped name** |
|---|---|
| std_material_name | material_name |
| material_name | Material Name |
| dimension | Dimension (mm) |
| density | Density (kg/m$^3$) |
| material_id | Material ID |

Table B.3: Excerpt from dictionary used for table materials_column

| **Original column heading** | **Mapped name** |
|---|---|
| materialid | material_id |
| std_material_name | material_name |
| from_std | from_volvo_standard |
| mfg_process | manufacturing_process |
| material_id | Material ID |
| material_name | Material Name |
| from_volvo_standard | From Volvo Standard |

Figure B.1: Dictionaries mapping technical table column headings into natural language names for materials and properties.

# C

# Appendix - Table serialization

### Serializer: JSON

```
{
  "Metal properties": {
    "Material": "Steel A",
    "Melting point": 1340,
    "Yield strength": "240 MPa"
  }
}
{
  "Metal properties": {
    "Material": "Steel B",
    "Melting point": 1400,
    "Yield strength": "Null"
  }
}
```

### Serializer: YAML

```
Metal properties:
  Material: Steel A
  Melting point: 1340
  Yield strength: 240 MPa

Metal properties:
  Material: Steel B
  Melting point: 1400
  Yield strength: 'Null'
```

### Serializer: Markdown

```
## Metal properties
|:---------------|:--------|
| Material       | Steel A |
| Melting point  | 1340    |
| Yield strength | 240 MPa |

## Metal properties
|:---------------|:--------|
| Material       | Steel B |
| Melting point  | 1400    |
| Yield strength | Null    |
```

### Serializer: Transposed Markdown

```
## Metal properties
| Material   |   Melting point | Yield strength |
|:-----------|----------------:|:---------------|
| Steel A    |            1340 | 240 MPa        |

## Metal properties
| Material   |   Melting point | Yield strength |
|:-----------|----------------:|:---------------|
| Steel B    |            1400 | Null           |
```

### Serializer: Markdown_Key-Values

```
## Metal properties
Material: Steel A
```
Melting point: 1340
```
Yield strength: 240 MPa
```

## Metal properties
Material: Steel B
```
Melting point: 1400
```
Yield strength: Null
```
```

**Serializer: XML**

```
<?xml version="1.0" ?>
<root>
  <Metal_properties>
    <Material>Steel A</Material>
    <Melting_point>1340</Melting_point>
    <Yield_strength>240 MPa</Yield_strength>
  </Metal_properties>
</root>

<?xml version="1.0" ?>
<root>
  <Metal_properties>
    <Material>Steel B</Material>
    <Melting_point>1400</Melting_point>
    <Yield_strength>Null</Yield_strength>
  </Metal_properties>
</root>
```

**Serializer: HTML**

```
<header>Metal properties<header>
<ul><li>Material: Steel A</li>
<li>Melting point: 1340</li>
<li>Yield strength: 240 MPa</li></ul>

<header>Metal properties<header>
<ul><li>Material: Steel B</li>
<li>Melting point: 1400</li>
<li>Yield strength: Null</li></ul>
```

**Serializer: Semantic**

```
Item in Metal properties:, Material is Steel A,
Melting point is 1340, Yield strength is 240 MPa.

Item in Metal properties:, Material is Steel B,
Melting point is 1400, Yield strength is Null.
```

**Serializer: Full semantic**

```
Item in Metal properties where Material is Steel A
and Melting point is 1340 and Yield strength is 240 MPa.

Item in Metal properties where Material is Steel B
and Melting point is 1400 and Yield strength is Null.
```

Figure C.1: Serializer outputs using an example table containing 2 items with 2 properties each

# D

# Appendix - System prompts

## D.1 Document-question generation

You are an expert human annotator for a machine learning dataset.

**Your task:**

- You will receive a chunk of text extracted from a larger document.
- Write a single, clear, and natural question that can be answered solely and completely using the information in this chunk.
- The question must be specific to the details, facts, or data present in this chunk, and should not require information from outside this chunk.
- If the question relates to a specific product, material, standard or other entity, include the name of that entity whenever possible.
- The question must be self-contained and understandable without any external context.
- Focus questions on information relevant to human employees working with material data.
- Avoid general or vague questions; focus on details, relationships, or facts that are explicitly stated or directly inferable from the chunk.
- Do **NOT** mention or reference "the document", "the text", "the chunk", the format, or the location of the text.
- The question should sound as if it was asked by a real human, not generated by a prompt or summarization tool.

**Output instructions:**

- Respond with **only** the question as a plain sentence.
- Do **not** include explanations, reasoning, instructions, or any additional text.
- Do **not** use markdown, formatting, or quotation marks.

**Example outputs:**

- What is the maximum allowed pressure for the hydraulic system described in the limitations section of the standard 030-444?
- Which cold-forged aluminium alloys have a yield strength above 400 MPa and granularity below 0.01%?

Figure D.1: System prompt for document-question generation

## D.2 Table-question generation

You are an evaluation data generator for a Retrieval-Augmented Generation (RAG) system.

Your task is to generate synthetic evaluation examples from a Parquet table that has been converted to a Python list of rows (pylist). The table contains structured material data.

For each example, you must:

- Generate a realistic, specific natural-language question that can be answered using **only** the table data.
- Generate the corresponding SQL query that correctly answers the question.

**Language requirements**

- Generate exactly `|n|` examples in total.
- The language distribution of the questions should be roughly 50% Swedish and 50% English.
- Each question must be entirely in one language (no mixing within a single question).

**Query type requirements**

- Approximately 50% of the examples must ask for a **list** of materials that satisfy a condition (e.g., "Which materials have X > Y?", "List all materials where...", "Vilka material har...?").
- The remaining examples can be single-material lookups or other simple filters.
- Do not focus repeatedly on one specific material name; vary the filters and the returned sets.

**Strict output format requirements**

- Output must be valid JSON.
- The top-level JSON object must contain exactly one key: `"examples"`.
- `"examples"` must be a list.
- Each item in `"examples"` must be a JSON object with exactly these two keys:
  - `"question"`: string
  - `"sql_query"`: string
- Do not include explanations.
- Do not include comments.
- Do not include Markdown.
- Do not include extra text before or after the JSON.
- Do not include trailing commas.

Figure D.2: System prompt for table-question generation (Part 1)

**SQL requirements**

- SQL must be valid PostgreSQL syntax.
- Every `"sql_query"` must start with `SELECT *`.
- The schema name is |schema|.
- The table name is |table name|.
- The available column names are:
  |columns|
- All identifiers (schema, table, column names) must use double quotes (").
  - Never use backticks (|‘|).
- All queries must explicitly reference the table in this exact form: |"schema"."table name"|.
- Use only the column names listed above.
- Use only WHERE clauses with comparison operators: `=`, `>`, `<`, `>=`, `<=`, and `BETWEEN`.
- Do not hallucinate columns.
- Do not hallucinate tables.
- Do not use JOIN.

**Question quality requirements**

- The questions must not have any exact column names in the question.
- Questions must be realistic and relevant to material engineering or material properties.
- Questions must map directly and unambiguously to the generated SQL query.
- The answer must be fully derivable from the SQL result alone.
- Vary questions by filtering on different columns across examples.
- Prefer conditions that likely return multiple rows for “list” questions (avoid overly specific equality on a single material name).

Figure D.3: System prompt for table-question generation (Part 2)

## D.3 Response generation

You are a precise assistant. Use **only** the retrieved context.

- Do not mention retrieval, chunks, or tools.
- Cite sources with the table name or document name.

Figure D.4: System prompt for LLM response generation

# E

# Appendix - Results

## E.1 Vector Index Scaling Results

| Test set | Index size | Baseline | | Simple Fusion | |
|---|---|---|---|---|---|
| | | **HitRate@10** | **MRR** | **HitRate@10** | **MRR** |
| $Doc_q$ | 88 | **0.7581** | **0.6214** | **0.7581** | **0.6214** |
| | 174 | 0.7016 | 0.5879 | 0.7016 | 0.5879 |
| | 374 | 0.6573 | 0.5397 | 0.6573 | 0.5397 |
| | 695 | 0.6210 | 0.5172 | 0.6210 | 0.5172 |
| | 1390 | 0.5524 | 0.4478 | 0.5524 | 0.4478 |
| $Tab_q$ | 88 | **0.4767** | 0.3770 | **0.4767** | 0.3770 |
| | 174 | 0.4733 | 0.3758 | 0.4733 | 0.3758 |
| | 374 | **0.4767** | **0.3771** | **0.4767** | **0.3771** |
| | 695 | 0.4300 | 0.3252 | 0.4300 | 0.3252 |
| | 1390 | 0.3967 | 0.3114 | 0.3967 | 0.3114 |
| $Combo_q$ | 88 | **0.6040** | **0.4876** | **0.6040** | **0.4876** |
| | 174 | 0.5766 | 0.4718 | 0.5766 | 0.4718 |
| | 374 | 0.5584 | 0.4507 | 0.5584 | 0.4507 |
| | 695 | 0.5164 | 0.4121 | 0.5164 | 0.4121 |
| | 1390 | 0.4672 | 0.3731 | 0.4672 | 0.3731 |

Table E.1: Retrieval performance for Baseline and Simple Fusion across the three test sets $Doc_q$, $Tab_q$ and $Combo_q$ and five index sizes (88, 174, 347, 695 and 1390). Hit rate and MRR scores are highest for the smallest index, except for tabular queries where performance is the same and marginally above the smallest (88) index also for the medium size index (347). Performance is identical between Baseline and Simple Fusion retreiver methods.

## E.2 Chunk Size and Top-k experiment results

| **Retrieval:** | | **Baseline** | | | | **Late Fusion** | | | |
|---|---|---|---|---|---|---|---|---|---|
| **Chunk size:** | | **256** | | **512** | | **256** | | **512** | |
| **Test set** | **Top-k** | **HR** | **MRR** | **HR** | **MRR** | **HR** | **MRR** | **HR** | **MRR** |
| $Doc_q$ | 10 | 0.4919 | 0.4072 | 0.5783 | 0.4603 | 0.4516 | 0.3958 | 0.5020 | 0.4180 |
| | 20 | 0.5524 | 0.4485 | 0.6546 | 0.5050 | 0.5000 | 0.4113 | 0.5904 | 0.4722 |
| | 30 | 0.5806 | 0.4632 | 0.6586 | 0.5055 | 0.5161 | 0.4218 | 0.6386 | 0.4934 |
| | 40 | 0.5887 | 0.4676 | 0.6667 | 0.5126 | 0.5524 | 0.4470 | 0.6506 | 0.5023 |
| | 50 | 0.5968 | 0.4681 | **0.6747** | 0.5171 | 0.5524 | 0.4478 | 0.6586 | 0.5022 |
| | 60 | 0.5968 | 0.4674 | **0.6747** | 0.5161 | 0.5645 | 0.4554 | 0.6667 | 0.5075 |
| | 70 | **0.6008** | 0.4708 | 0.6707 | 0.5152 | 0.5766 | 0.4627 | **0.6707** | **0.5100** |
| | 80 | **0.6008** | **0.4743** | **0.6747** | **0.5174** | **0.5887** | **0.4686** | **0.6707** | 0.5097 |
| $Tab_q$ | 10 | 0.3867 | 0.3160 | 0.3967 | 0.3208 | 0.3667 | 0.3148 | 0.3533 | 0.3047 |
| | 20 | 0.3967 | 0.3114 | 0.4200 | 0.3247 | 0.4300 | 0.3484 | 0.4433 | 0.3460 |
| | 30 | 0.4500 | 0.3430 | 0.4500 | 0.3444 | 0.4700 | 0.3723 | 0.4667 | 0.3689 |
| | 40 | 0.5000 | 0.3694 | 0.5200 | 0.3925 | 0.5033 | 0.3786 | 0.4933 | 0.3789 |
| | 50 | **0.5233** | 0.3929 | **0.5400** | 0.4113 | 0.4900 | 0.3835 | 0.4933 | 0.3808 |
| | 60 | **0.5233** | **0.3996** | 0.5367 | 0.4114 | 0.4967 | 0.3846 | 0.4933 | 0.3776 |
| | 70 | **0.5233** | 0.3947 | 0.5300 | **0.4116** | 0.5067 | 0.3850 | 0.5000 | 0.3815 |
| | 80 | 0.5067 | 0.3921 | 0.5267 | 0.4096 | **0.5267** | **0.3965** | **0.5267** | **0.3982** |
| $Combo_q$ | 10 | 0.4343 | 0.3573 | 0.4791 | 0.3840 | 0.4051 | 0.3514 | 0.4208 | 0.3561 |
| | 20 | 0.4672 | 0.3734 | 0.5264 | 0.4065 | 0.4617 | 0.3769 | 0.5100 | 0.4032 |
| | 30 | 0.5091 | 0.3974 | 0.5446 | 0.4174 | 0.4909 | 0.3947 | 0.5446 | 0.4254 |
| | 40 | 0.5401 | 0.4138 | 0.5865 | 0.4470 | 0.5255 | 0.4095 | 0.5647 | 0.4349 |
| | 50 | 0.5566 | 0.4269 | **0.6011** | **0.4593** | 0.5182 | 0.4126 | 0.5683 | 0.4358 |
| | 60 | 0.5566 | **0.4303** | 0.5993 | 0.4589 | 0.5274 | 0.4167 | 0.5719 | 0.4365 |
| | 70 | **0.5584** | 0.4291 | 0.5938 | 0.4586 | 0.5383 | 0.4202 | 0.5774 | 0.4398 |
| | 80 | 0.5493 | 0.4293 | 0.5938 | 0.4585 | **0.5547** | **0.4292** | **0.5920** | **0.4488** |

Table E.2: Retrieval performance for Baseline retriever and Late Fusion across chunk sizes (256, 512) and top-k (10-80). Three test sets $Doc_q$, $Tab_q$ and $Combo_q$ were used and ran on the largest vector index (size 1390), MRR and hit rate@10 was measured. HR conveys hit rate@10