Master Thesis

A Goal-Question-Metric-based Framework for Retrieval-Augmented Generation System Evaluation

Anton Komar

Supervisor
Prof. Dr. Steffen Herbold
Prof. Dr. Florian Lemmerich

July 28, 2025

University of Passau
Faculty of Computer Science and Mathematics
Chair of AI Engineering

Confirmation

I hereby declare that I have independently prepared the Master Thesis submitted on July 28, 2025 entitled

A Goal-Question-Metric-based Framework for
Retrieval-Augmented Generation System Evaluation

under the supervision of Prof. Dr. Steffen Herbold. I have completed and written the thesis independently and have properly cited all references.

Passau, July 28, 2025

Abstract

Retrieval-Augmented Generation (RAG) systems have become an effective way to enhance Large Language Models by allowing them to draw on external knowledge sources. However, evaluating these systems is still a difficult task because they involve multiple interconnected components and many different quality factors. This thesis presents an evaluation framework for RAG systems based on the Goal-Question-Metric (GQM) approach. This framework helps break down high-level evaluation goals into concrete, measurable questions and then connects these questions to specific metrics. Each part of a RAG system is assessed individually, as well as the system as a whole. This framework uses traditional metrics and LLM-based evaluations, which lets for more in-depth and nuanced analysis. The results suggests that this approach is better at diagnostic problems than existing methods and gives clear, useful insights that can be used to improve RAG systems. Experiments on a number of RAG implementations shows that the framework is effective at identifying weaknesses in specific components and supporting targeted optimizations.

Contents

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Problem Statement . . . 2
- 1.3 Research Questions . . . 3
- 1.4 Expected Contributions . . . 4
- 1.5 Thesis Structure . . . 4

2 Background and Foundations **5**
- 2.1 Retrieval-Augmented Generation Systems . . . 5
  - 2.1.1 RAG Architecture . . . 5
  - 2.1.2 Components and Information Flow . . . 6
  - 2.1.3 RAG Variants and Advanced Architectures . . . 7
  - 2.1.4 Common Challenges . . . 8
- 2.2 Goal-Question-Metric Methodology . . . 8
- 2.3 Evaluation Metrics . . . 10
  - 2.3.1 Semantic and Neural Metrics . . . 10
  - 2.3.2 Task-Specific Metrics for Information Retrieval and QA . . . 10
  - 2.3.3 LLM-based Evaluation . . . 11

3 Related Work **12**
- 3.1 RAG System Evaluation Approaches . . . 12
  - 3.1.1 Component-Level Evaluation . . . 12
  - 3.1.2 End-to-End Evaluation . . . 12
  - 3.1.3 Human Evaluation Studies . . . 13
- 3.2 Automated Evaluation Frameworks for RAG . . . 13
  - 3.2.1 RAGAS . . . 14
  - 3.2.2 DeepEval . . . 15
  - 3.2.3 Giskard: Open-Source RAG Testing Toolkit . . . 15
  - 3.2.4 ARES: Automated RAG Evaluation System . . . 15

4 The GQM-based RAG Evaluation Framework **17**
- 4.1 Goals Definition . . . 18
- 4.2 Metric Aggregation Strategies . . . 20

5 Practical Application and Experimental Features **21**
- 5.1 Framework Evaluation Workflow . . . 23
- 5.2 System Architecture . . . 23

5.2.1 Subsystem Overview . . . 23
5.3 Framework Core Components . . . 25
5.3.1 Core Evaluation Framework . . . 25
5.3.2 Metrics Subsystem . . . 25
5.3.3 Test Generation Subsystem . . . 26
5.3.4 External Systems Integration . . . 26
5.3.5 Dashboard User Interface . . . 27
5.3.6 File System Persistence . . . 27
5.4 Metric Implementations . . . 27
5.4.1 Retrieval Metrics . . . 27
5.4.2 Generation Metrics . . . 29
5.4.3 System-Level Metrics . . . 32

6 Results **33**
6.1 Experimental Setup . . . 33
6.2 Overall Framework Performance . . . 34
6.3 Goal-by-Goal Analysis . . . 34
6.4 Component-Level Performance Analysis . . . 36
6.5 Performance by Question Types . . . 37
6.6 Comparative Analysis Across Experiments . . . 39
6.7 Detailed Per-Query Results . . . 39

7 Discussion **41**
7.1 Key Findings . . . 41
7.2 Answering the Research Questions . . . 42
7.3 Framework Limitations . . . 44
7.4 Future Work . . . 46

8 Conclusion **49**

List of Acronyms **51**

List of Figures **52**

List of Tables **53**

Bibliography **54**

Chapter 1

Introduction

This thesis explores the application of the Goal-Question-Metric methodology, traditionally used in software engineering, to create a systematic evaluation framework for Retrieval-Augmented-Generation systems and shows how hierarchical goal decomposition enables more precise assessment than current evaluation approaches.

1.1 Motivation

The rapid development of Large Language Model (LLM)s had a big impact on artificial intelligence, introducing new capabilities to understand and generate natural language [58, 13]. These models, trained on extensive text data, demonstrate impressive performance across several tasks, from answering questions to writing creatively. But there is a major drawback: they cannot fetch information beyond their training data. Once trained, these models cannot learn new information or add to their current knowledge base, i.e., they become less helpful with time.

One solution to this issue is Retrieval-Augmented Generation (RAG) systems [28]. These provide the ability for LLMs to retrieve and utilize external information during response generation, allowing for real-time information retrieval. This marries what LLMs excel at (reasoning and understanding of language) with access to precise information from outside sources. Several industries have begun to utilize this method for developing customer support applications, research assistants, and more.

However, understanding whether these RAG systems work well is difficult, much more complex than evaluating traditional Natural Language Processing (NLP) systems [9]. The reason is that RAG systems have several parts that need to work together: a retrieval module responsible for identifying relevant documents, a generation module that formulates responses, and, in some cases, additional components such as query rewriters or rerankers. All of these elements can fail in various ways, and when they combine, their failures interact in a complicated way, making them even harder to diagnose and comprehend.

In addition, many different aspects of quality must be considered. It is not enough for a RAG system to find relevant information - it also has to generate responses that are accurate, complete, and make sense. The system should not hallucinate while maintaining a natural and fluent style. It must be capable of effectively handling both simple and complex queries. Most

current evaluation methods only look at one or two of these aspects, which fail to provide a comprehensive evaluation.

Today, RAG evaluation is comprised of different fragmented metrics and tools. Some frameworks, such as RAGAS, only test for faithfulness and relevance [14], while others, such as DeepEval, offer various metrics without describing how they integrate with one another [11]. While using this approach, it is not clear which areas should be optimized and how the optimization of one metric will impact others.

The Goal-Question-Metric (GQM) method has been helping software engineers improve their systems for decades [4, 5]. It structures evaluation by decomposing high-level objectives into detailed questions, then translating those into measurable metrics. This forms the sort of ordered framework that RAG evaluation requires. This thesis shows how we can adapt GQM for RAG systems, transforming the current scattered evaluation approach into a systematic and actionable approach that supports meaningful system improvement.

1.2 Problem Statement

Although RAG systems are widely used, they do not have a theoretically grounded evaluation framework that is understandable and easy to use, creating obstacles for both researchers and developers.

In the first place, there is no systematic structure in current evaluation frameworks. Available tools offer lists of metrics without definite relations among them and instructions for their proper application. Therefore, it is hard for developers to realize what metrics are applicable in their particular case.

Second, the multi-component nature of RAG systems is not properly handled in current evaluation approaches. These systems consist of interconnected components (query processing, retrieval, reranking, and generation) whose interactions significantly impact overall performance. When a RAG system produces incorrect or incomplete answers, existing tools cannot effectively diagnose whether the failure results from poor document retrieval, ineffective context utilisation, or generation errors. This diagnostic blindness forces developers into inefficient optimisation cycles, wasting resources and time.

Third, current metrics often fail to provide practical insights. A low faithfulness score shows that generated responses may contain hallucinations, but offers no guidance on whether this comes from poor retrieval quality, inadequate context integration, or generation model limitations. Similarly, relevance metrics may show poor performance without details on whether the issue lies in query formulation, document ranking, or answer generation.

Fourth, it is not clear how reliable the automated evaluation is itself. Recent studies have shown inconsistent correlation between automated metrics and human judgment, particularly in domain-specific applications [30]. While LLM-based evaluation methods show promise, they suffer from prompt sensitivity and potential biases inherited from training data. Organizations deploying RAG systems in production often rely on expensive human evaluation because they cannot trust automated metrics to capture the quality dimensions that matter to their users.

Last but not least, there is a lack of theoretical basis for designing and choosing evaluation

measures for RAG systems. Existing approaches take measures from similar domains such as information retrieval and text summarisation without modification for the specific aspects of RAG. Lack of theory contributes to blind spots in evaluation and makes it challenging to argue for the completeness of measures or determine missing dimensions in evaluation.

1.3 Research Questions

This thesis investigates three research questions about the core issues in RAG system evaluation:

RQ1: How can the Goal-Question-Metric methodology be effectively adapted for evaluating RAG systems?

This question looks into the theoretical basis of the approach. While GQM has worked well in software engineering contexts, it needs to be carefully adapted to work for evaluating AI systems, especially for complex multi-component systems like RAG. The question also investigates how to structure the hierarchical decomposition from abstract quality goals to concrete metrics, how to handle the unique characteristics of RAG systems such as component interactions and probabilistic outputs, and how to maintain the actionability that makes GQM valuable. Answering this question sets the theoretical context that forms the remaining work.

RQ2: What combination of metrics, measurement strategies, and test case generation methods indicates RAG system effectiveness across different use cases?

Building on the GQM framework, this question addresses the practical challenge of metric selection, design, and test case development. We investigate which existing metrics from NLP evaluation can be adapted for RAG systems, what new metrics are needed to capture RAG-specific quality dimensions, and how to measure component interactions that existing frameworks ignore. This includes exploring both traditional metrics and LLM-based evaluation approaches, as well as developing strategies for component isolation, interaction analysis, and systematic test case generation methodologies. The response to this question constitutes the technical core of the evaluation system.

RQ3: How can evaluation results be systematically translated into practical optimization recommendations?

The last question presents an obstacle to converting evaluation outputs into actionable improvements. Although performance data across a range of dimensions is given, it is usually hard to decipher these outcomes and understand precisely which changes will best improve our RAG systems. This question examines processes for putting in place clear diagnostic pathways from high-level performance metrics to concrete component failures. The research considers how the hierarchical GQM framework can support root cause analysis, regardless of whether problems lie in retrieval, generation, or their combination, and how to produce evidence-based suggestions that conform to system-specific objectives. Responding to this question guarantees that the evaluation framework is a practical manual for iterative improvement and not just a measurement instrument.

1.4 Expected Contributions

This thesis makes four primary contributions to the field of RAG system evaluation:

- Empirical analysis of factors that impact RAG system performance
- Adaptation and application of the GQM framework to RAG systems, providing a structured approach to their evaluation
- Automated evaluation framework based on the GQM approach with advanced metrics
- Comprehensive diagnostic mechanisms that translate evaluation results into actionable recommendations

1.5 Thesis Structure

This thesis is organized into nine chapters as follows:

Chapter 2 provides the necessary background knowledge for understanding the contributions. It describes RAG system architectures, the GQM software engineering methodology, and metrics for NLP evaluation. This chapter sets the technical background and vocabulary for the rest of the thesis.

Chapter 3 is an in-depth review of related work in RAG evaluation. It discusses current evaluation methods, automated toolsets, and metric development. It offers a thorough review of available tools and their drawbacks.

Chapter 4 presents the GQM-based RAG assessment framework, the primary theoretical contribution of this thesis. It explains the framework structure, the hierarchical goals, the process of operationalizing questions, and the metric choice strategy.

Chapter 5 covers the experimental and practical application. It explains the evaluation pipeline workflow, system design using UML component diagram, main framework components, detailed metric implementations, and test case generation methods. The chapter includes architectural diagrams and design decisions.

Chapter 6 gives complete experimental results. It breaks down framework performance over three varying RAG configurations, offers goal-by-goal analysis, component-level performance breakdowns, and question-type-wise performance analysis. The chapter illustrates the diagnostic abilities of the framework using detailed visualizations.

Chapter 7 deals with the implications of the results. It combines main findings, gives answers to every research question, examines framework limitations, and specifies directions for future research.

Chapter 8 concludes the thesis by summarizing the contributions, highlighting the framework's impact on RAG evaluation, and emphasizing its role in advancing systematic AI assessment.

Chapter 2

Background and Foundations

This chapter sets the theoretical and technical background for this thesis's contributions. We begin with the discussion of RAG systems, their architecture, and their intrinsic problems. We then cover the GQM methodology from software engineering and its applicability to systematic evaluation. Finally, we review evaluation metrics ranging from traditional NLP measures to modern neural and LLM-based approaches, establishing the technical background within which our framework operates.

2.1 Retrieval-Augmented Generation Systems

Retrieval-Augmented Generation represents a paradigm shift in how large language models access and utilize information. By augmenting the parametric knowledge of LLMs with non-parametric retrieval mechanisms, RAG systems address fundamental limitations of static language models while introducing new architectural complexities that demand sophisticated evaluation approaches [28, 18].

2.1.1 RAG Architecture

RAG refers to a class of Artificial Intelligence (AI) systems that enhance a generative language model by providing it with relevant external information retrieved from a knowledge source. In a standard RAG architecture, a user's query is first used to fetch supporting documents or passages from an external corpus, and those retrieved documents are then supplied to a Large Language Model (LLM), which generates a response conditioned on both the query and the retrieved content. At a high level, a RAG system can be viewed as a pipeline consisting of two primary modules: a retriever module that identifies relevant information from external knowledge sources (which could be a document database, vector index, or the web), and a generator module that is a neural language model that produces the final answer, using the retrieved evidence as additional context [26]. These elements operate together to enable the system to respond to queries with knowledge that the LLM has not memorized, in effect providing the model with a dynamic memory. Figure **??** presents the basic architecture of a RAG system, including the information flow from query to retrieval to generation.

In the indexing step, documents are mapped to dense vector representations by an encoder model, usually 768-dimensional or more vectors for models such as all-mpnet-base-v2 or 384-

dimensional for BGE-small-en-v1.5, representing semantic meaning beyond lexical similarity [48]. The encoded vectors are indexed in specialized vector databases such as Facebook AI Similarity Search (FAISS), Pinecone, or Weaviate that support fast approximate nearest neighbor search at scale [25].

At retrieval time, the user query is passed through the same encoding pipeline, yielding a query vector. The retriever conducts a similarity search, usually via cosine similarity or inner product, to retrieve the k most pertinent documents. These documents and the original query constitute the augmented input to the generator model. The generator, frequently a decoder-only transformer such as GPT or LLaMA variants, generates the final response conditioned on this augmented context [54].

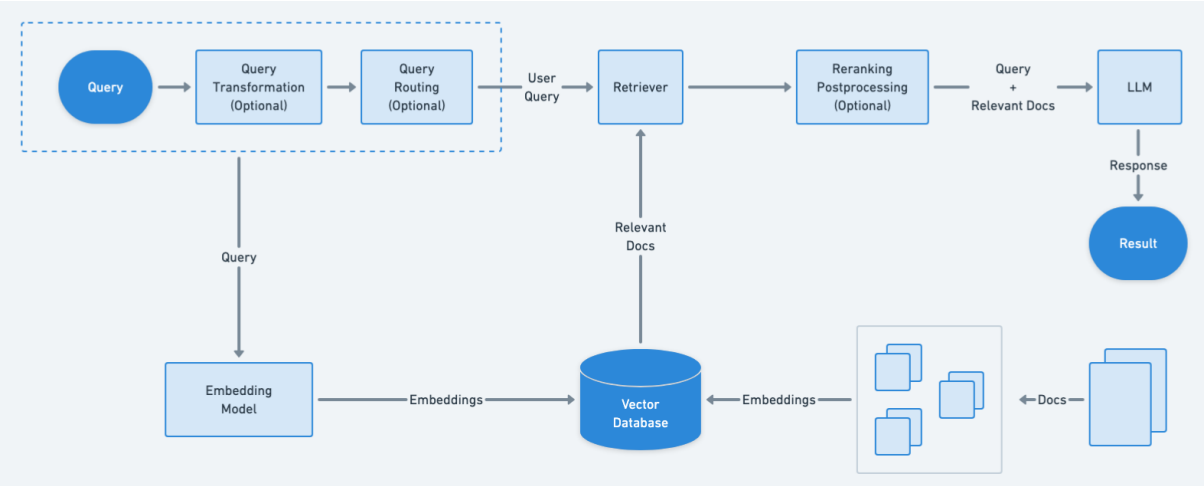

Figure 2.1: Basic RAG system architecture illustrating the flow from user query through retrieval and generation components.

2.1.2 Components and Information Flow

Interpreting RAG system analysis involves in-depth familiarity with component interactions and data transformations along the pipeline.

Prior to retrieval, queries are typically preprocessed to enhance retrieval effectiveness. This can involve query expansion, wherein the system produces related terms or reformulates the query to more closely resemble document terminology. Query categorization is used by some systems to route various query types to specialized retrieval methods.

The pipeline for document processing converts the raw text to searchable forms in multiple steps:

- **Chunking Strategies:** Texts are divided into retrieval units with strategies-fixed-size windows, sentence boundaries, or semantic segmentation. Chunking strategy has a major influence on the effectiveness of retrieval, with trade-offs between context preservation and retrieval accuracy [12].
- **Embedding Models:** The embedding model choice underlies retrieval quality. Some models, such as all-MiniLM-L6-v2, provide efficiency with acceptable quality, streaming text at high speed with 384-dimensional representations. More complex models, such as

BGE-small-en-v1.5 or voyage-2, give better semantic comprehension. Recent research shows that domain-specific fine-tuning of such models can greatly enhance retrieval performance [22].

- **Indexing Structures:** Vector databases utilize different indexing methods to trade off search quality and search speed. Hierarchical Navigable Small World (HNSW) graphs support high recall with acceptable query latency, and Inverted File Index (IVF) methods achieve higher memory efficiency in large-scale implementation [37].

The retrieval module uses techniques other than straightforward similarity search:

- **Dense Retrieval:** Current RAG systems rely mainly on dense retrieval, wherein both documents and queries are encoded into continuous vector spaces. Dense retrieval models capture semantic similarity that is not captured by lexical methods, thus allowing retrieval of conceptually similar but lexically different content [26].
- **Hybrid Retrieval:** The combination of dense and sparse retrieval techniques tends to be more effective. Hybrid methods take advantage of the semantic comprehension of dense models while preserving the accuracy of lexical matching for certain terms [36].
- **Reranking:** Multi-stage retrieval pipelines utilize reranking models for improving initial retrieval results. Cross-encoder architectures, although computationally costly, yield more precise relevance scores by processing query-document pairs jointly [43].

The generation module has special challenges for the use of recovered information:

- **Context Fusion:** Generators have to combine information from several, possibly conflicting sources. Transformer architectures with attention mechanisms inherently facilitate this, but successful fusion is still difficult, especially for complicated reasoning [55].
- **Attribution and Grounding:** Making generated responses grounded in the retrieved context is a key challenge. Recent research on controllable generation and citation mechanisms tries to enhance verifiability [40, 16].
- **Context Window Management:** Current LLMs allows for long context windows, but making use of lengthy contexts is still challenging. Positional bias, dilution of information, and computation limitations require us to curate contexts carefully [31].

2.1.3 RAG Variants and Advanced Architectures

As RAG systems have matured, several architectural variants have emerged to address specific limitations:

- **Iterative and Multi-hop RAG:** Iterative RAG systems perform multiple retrieval-generation cycles to handle complex queries requiring information synthesis across documents. Each iteration refines the query based on previously retrieved information, enabling exploration of interconnected topics [51]. Multi-hop reasoning architectures explicitly model reasoning paths through documents. Systems like HopRetriever and Baleen demonstrate improved performance on complex questions by maintaining reasoning chains [61, 27].

- **Adaptive RAG:** Dynamically adjusts retrieval strategies based on query characteristics. Simple factual queries may require single-document retrieval, while complex analytical questions benefit from comprehensive multi-document retrieval. This adaptation improves efficiency while maintaining quality [24].
- **Self-Reflective RAG:** Self-reflective architectures incorporate feedback loops to improve response quality. SELF-RAG introduces special reflection tokens that enable the model to assess and refine its outputs iteratively [2]. This approach shows particular promise for reducing hallucinations and improving factual accuracy.

2.1.4 Common Challenges

Despite architectural advances, RAG systems face persistent challenges that motivate our evaluation framework:

- **Semantic Gap:** Embedding models may fail to capture nuanced query intent, particularly for domain-specific or ambiguous queries. This semantic gap leads to the retrieval of similar but irrelevant documents [8].
- **Coverage Limitations:** Knowledge bases may lack comprehensive coverage of query topics. Even perfect retrieval cannot compensate for missing information, necessitating graceful handling of knowledge gaps [41].
- **Hallucination:** Despite grounding in retrieved documents, generators may produce unsupported claims. This occurs through various mechanisms-extrapolation beyond source material, conflation of multiple sources, or reliance on parametric knowledge [39].
- **Attribution Errors:** RAG systems may correctly extract information but attribute it incorrectly, undermining trustworthiness. Clear citation mechanisms and verification strategies are essential for deployment in high-stakes applications [6].
- **Context Integration Failures:** Generators may struggle to synthesize information from multiple documents coherently. This manifests as contradictory statements, incomplete coverage of retrieved information, or failure to identify relationships between sources [42].

2.2 Goal-Question-Metric Methodology

GQM methodology is a structured approach for defining and interpreting performance measures, originally developed in the field of software engineering to ensure that metrics are tied to explicit goals [4]. GQM was introduced by Victor Basili and colleagues as a way to formalize the process of metrics selection: rather than collecting data for its own sake, one should first specify what the goals are, derive the questions whose answers illuminate progress toward those goals, and then decide on the metrics that would quantitatively answer those questions (see Figure 2.2. This approach has been widely used in software project management and quality assurance to measure things like productivity, defect rates, and customer satisfaction in a methodical way [4, 56].

The methodology rests on four foundational principles. Firstly, every measurement activity must serve explicitly defined goals. GQM requires clear articulation of what is needed to

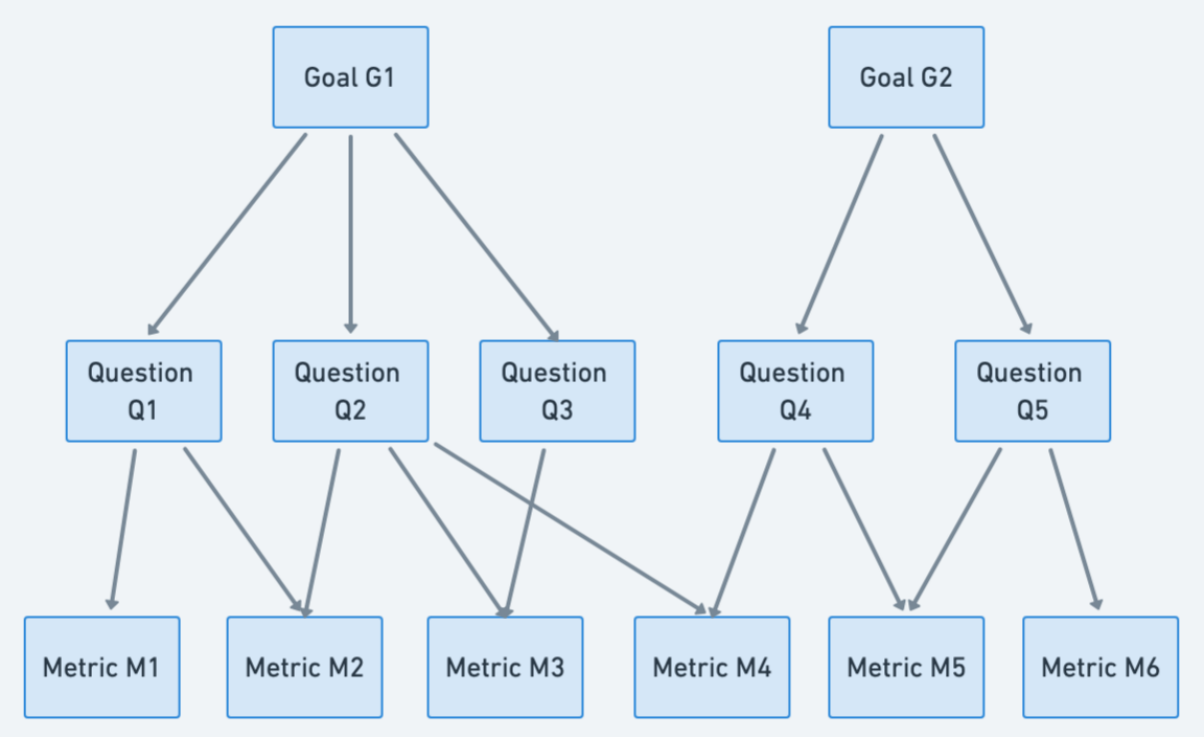

Figure 2.2: Example GQM hierarchy.

achieve or understand through measurement [4]. Second, abstract goals undergo refinement into operational questions, which map to quantifiable metrics. This approach ensures comprehensive coverage while maintaining traceability from high-level objectives to specific measurements [57]. Third, GQM recognizes that metrics are not universally applicable. The same goal may require different operationalization depending on organizational context, project characteristics, and needs. And finally, goals can be analyzed from various viewpoints, ensuring comprehensive coverage of quality concerns.

Goals undergo formalization using the GQM template:

- **Object:** What is being examined (process, product, resource)
- **Purpose:** Why examine it (characterize, evaluate, predict, improve)
- **Quality Focus:** Which attribute matters (reliability, efficiency, maintainability)
- **Viewpoint:** Whose perspective guides evaluation
- **Context:** Environmental factors affecting interpretation

Questions operationalize goals by identifying information needed to determine goal achievement. Metrics provide quantitative answers to questions. The mapping considers such metric properties as validity, reliability, efficiency, and practical constraints. Multiple metrics may answer a single question, while one metric may serve multiple questions [7].

2.3 Evaluation Metrics

The evolution of evaluation metrics from simple string matching to sophisticated neural approaches reflects growing understanding of language complexity. This section traces this evolution and examines modern metrics particularly relevant to RAG evaluation.

2.3.1 Semantic and Neural Metrics

In recent years, the evaluation of generated text has moved toward **semantic similarity** metrics that leverage neural network representations to go beyond exact string overlap. These metrics aim to capture whether the meaning of the generated text aligns with the meaning of a reference or target text. Two prominent approaches in this category are **BERTScore** and embedding-based cosine similarity metrics.

BERTScore is a metric that uses pre-trained contextual embeddings (from models like Bidirectional Encoder Representations from Transformers (BERT) or its variants) to compare texts [63]. Instead of looking for exact n-gram matches, BERTScore computes a similarity score based on token embeddings. The typical procedure is: for each token in the generated text, find the most similar token (cosine similarity in embedding space) in the reference text, and vice versa, and then aggregate these similarity scores (often using a weighted precision and recall, optionally with Inverse Document Frequency weighting to down-weight common words). The result is a score that reflects how well the candidate and reference cover each other's semantic content. BERTScore has been shown to correlate better with human judgments of quality in tasks like summarization and translation compared to Bilingual Evaluation Understudy (BLEU)/Recall-Oriented Understudy for Gisting Evaluation (ROUGE), especially when there is significant paraphrasing [63]. BERTScore can be tuned by choosing different underlying models; for instance, using RoBERTa-large or DeBERTa as the embedding model often yields higher correlation with human evaluations [33, 19].

Another approach to semantic evaluation is using **embedding-based similarity** at the sentence or paragraph level. Instead of matching individual token embeddings as BERTScore does, one can compute a single vector embedding for the whole generated answer and one for the reference (or for the query, or for relevant source text), and then calculate cosine similarity between these vectors.

However, not all embedding models are equal in capturing nuanced similarity. Models like **RoBERTa or DeBERTa fine-tuned for similarity tasks** (as used in BERTScore or a metric called BLEURT [50]) perform better than generic embeddings. **BLEURT**, for instance, is a learned metric where a BERT model was fine-tuned on a collection of human ratings to predict a quality score [50]. BLEURT can directly output a score that correlates with human judgment by considering semantic context. It essentially combines aspects of BERTScore with a learned weighting scheme. Using BLEURT or similar neural evaluators can provide a single score for how good an answer is, given a reference. These are often used in academic evaluations nowadays.

2.3.2 Task-Specific Metrics for Information Retrieval and QA

RAG systems inherit evaluation challenges from both retrieval and generation tasks, making Information Retrieval (IR) and Question Answering (QA) metrics particularly relevant:

- **Recall@k:** This measures what fraction of the relevant documents are present in the top k results retrieved [38].
- **Precision@k:** This is the fraction of the top k retrieved documents that are actually relevant to the query [38]. High precision means the retriever is not fetching a lot of irrelevant stuff that could confuse the generator.
- **Mean Average Precision (MAP):** If we consider the entire ranked list, MAP provides a single-number summary of ranking quality, giving higher weight to getting relevant documents ranked higher.
- **Mean Reciprocal Rank (MRR) (Mean Reciprocal Rank):** This metric looks at the rank position of the first relevant document in the results [60]. MRR is useful when typically one document (the one containing the answer) is what matters.
- **Normalized Discounted Cumulative Gain (NDCG):** Accounts for graded relevance and position bias, recognizing that users are less likely to examine lower-ranked results. NDCG's handling of non-binary relevance aligns well with RAG scenarios where documents may be partially relevant [23].

2.3.3 LLM-based Evaluation

An emerging trend in evaluating language generation (including RAG outputs) is the use of large language models themselves as evaluators or judges [32]. These **LLM-based evaluation** approaches leverage the reasoning ability of models like GPT-4 to assess the quality of an answer, often in contexts where automated metrics fall short or where human evaluation would be too costly. For RAG systems, LLM-based evaluation can be especially useful to judge factuality and coherence in a reference-free setting. LLMs can evaluate complex criteria like factual accuracy, coherence, and completeness that traditional metrics cannot capture. Prompt engineering in this case significantly impacts evaluation quality. [64].

To mitigate cost and consistency issues, researchers have developed specialized models or methods. One notable example is the **Hughes Hallucination Evaluation Model (HHEM)** and its improved version **HHEM v2.1** [59]. HHEM is a neural model specifically trained to detect factual inconsistencies (hallucinations) in LLM outputs with respect to the provided source text. In other words, given a pair: (source documents, LLM-generated answer), HHEM produces a score, typically between 0 and 1, indicating how likely it is that the answer is factual given the source. A score of 1 would mean the answer is fully consistent with the source (no hallucination), while a low score means the answer contains unsupported content. HHEM v2 has been reported as a state-of-the-art automated metric for hallucination detection and factuality checking in summarization and QA contexts [59]. Its appeal is that it can operate much faster than prompting a large model for each check (HHEM is a smaller model that can run locally), and yet it was trained to mimic a lot of the judgment capability of large models and humans using a large annotated dataset of factual consistency cases. This makes it an attractive tool in a RAG evaluation framework for automatically scoring the *faithfulness* of answers.

Another example is Anthropic's constitutional approach that uses AI systems to evaluate outputs against explicit principles [3]. This enables consistent evaluation of complex criteria like helpfulness, harmlessness, and honesty.

Chapter 3

Related Work

3.1 RAG System Evaluation Approaches

Evaluating retrieval-augmented generation (RAG) systems is a multi-faceted challenge because it involves assessing both the information retrieval component and the natural language generation component, as well as their combined performance [28, 18, 62]. Broadly, existing evaluation approaches can be categorized into *component-level evaluation*, *end-to-end evaluation*, and *human-centric evaluation* strategies. This section reviews these categories, laying the groundwork for understanding how our Goal-Question-Metric (GQM) approach relates to prior work.

3.1.1 Component-Level Evaluation

One common strategy is to evaluate the retrieval and generation components separately using traditional metrics. For the **retrieval module**, standard information retrieval metrics like precision, recall, mean average precision (MAP), or hit rates (Recall@k) are used to quantify how many of the top retrieved documents are relevant to the query [52, 10]. However, focusing only on high recall at k may encourage retrieving many possibly relevant documents, rather than the most useful ones, which might not reflect the true utility of the retrieval for generation [**adlakha2024evaluating**].

The generation component is typically evaluated with metrics borrowed from QA and text generation literature. If reference answers or texts are available, one can compute exact match or F1 for QA, or ROUGE-L and BLEU scores for longer outputs [45, 29]. These **reference-based metrics** measure how closely the generated answer matches a gold standard answer. They have been used in early RAG works (e.g., Lewis et al. [28] evaluated their RAG model on QA tasks by exact match accuracy against ground-truth answers). While such metrics are useful, they often fail to capture nuances like factual correctness and relevance when the output can be phrased in many valid ways or when no single reference answer is available.

3.1.2 End-to-End Evaluation

End-to-end evaluation treats the RAG system as a black box, judging the quality of its final answers in response to queries without separately scoring intermediate retrieval steps. This

approach is crucial because the ultimate goal is correct and helpful answers; even if each component performs adequately in isolation, their combination might fail [9].

End-to-end metrics often assess whether the output answers the question correctly and completely. If a benchmark provides ground-truth answers, the simplest measure is answer accuracy (for factual questions) or ROUGE-L for descriptive answers.

In many realistic settings, however, ground truth answers are unavailable (e.g., dynamic knowledge or proprietary documents), so end-to-end evaluation must rely on **reference-free** criteria. One approach is to use a *simulated evaluator model* to judge the answer. Recently, large language models (LLMs) themselves have been used as automatic judges, a method sometimes dubbed "LLM-as-a-judge" or *Generative evaluation* (G-Eval) [32, 15].

End-to-end evaluation captures the holistic performance, but its reliability depends on the evaluation method: automatic methods (like LLM judges or string overlap metrics) may not always align with true user satisfaction or factuality.

3.1.3 Human Evaluation Studies

Human evaluation remains a gold standard for evaluating RAG systems, especially for subjective criteria or when automatic metrics are insufficient. Humans can judge whether the answer is correct, well-supported by evidence, clear, and useful. Typical human evaluation for RAG involves rating outputs on dimensions such as relevance to the query, factual correctness (faithfulness to source), and fluency [47, 21].

In knowledge-intensive domains (e.g., biomedical or legal RAG applications), domain experts might manually verify if the system's answer is supported by the retrieved sources [44, 35]. Human judges can also detect subtle issues like logically correct but irrelevant answers, or hallucinations that evade automatic detectors.

The downside is that human evaluation is time-consuming, expensive, and not scalable for rapid development cycles. Furthermore, human judgments can be inconsistent. Hence, multiple annotators and well-defined guidelines are used to ensure reliability. Despite these costs, human evaluation is often used to validate automated metrics. Many RAG evaluations include a human-agreed set of scores or rankings as the ground truth to which automated evaluation frameworks are correlated [53].

3.2 Automated Evaluation Frameworks for RAG

In response to the challenges above, a number of automated evaluation frameworks have been proposed to assess RAG systems more efficiently and comprehensively. These frameworks aim to evaluate various quality dimensions of RAG pipelines (e.g., relevance, faithfulness, correctness) with minimal human intervention. Below, we discuss several prominent frameworks and tools: RAGAS, DeepEval, Giskard, ARES, and others. Each offers a different approach to RAG evaluation, ranging from LLM-based evaluators to synthetic data generation and specialized metrics.

3.2.1 RAGAS

RAGAS (Retrieval Augmented Generation Assessment Suite) is an open-source framework [14] to provide a reference-free, multi-faceted evaluation of RAG pipelines. The key idea in RAGAS is to evaluate a RAG system along several dimensions using an LLM as a judge, without requiring any ground-truth answers or manual annotations. Specifically, RAGAS defines a suite of metrics targeting the major quality aspects of a RAG system:

- **Answer Faithfulness**: The extent to which the generated answer is grounded in the retrieved context. An answer is considered faithful if all its factual claims can be *inferred from the provided documents*. RAGAS measures this by breaking the answer into atomic statements and using an LLM to verify each statement against the retrieved passages [14]. The fraction of statements supported by the context yields a faithfulness score. This metric directly targets the hallucination problem: a low faithfulness score indicates the answer contains information not supported by the sources.
- **Answer Relevance**: Whether the answer addresses the user's question appropriately (regardless of factual errors). RAGAS evaluates answer relevance by a clever embedding-based method: it prompts an LLM to generate a question that the given answer could be answering, then computes the semantic similarity between this generated question and the original user question [14]. A high cosine similarity indicates that the answer is on-topic and fully responsive to the query. This metric penalizes answers that are incomplete, off-topic, or contain irrelevant information, even if they may be factual.
- **Context Relevance**: How focused and pertinent the retrieved context is to the question. Even if an answer is correct, an irrelevant or overly broad context can indicate inefficiencies or risks (e.g., introducing distractors). RAGAS computes context relevance by asking an LLM to extract the subset of sentences from the context that are needed to answer the question. It then scores the context by the proportion of retrieved content that was deemed relevant [14]. A context containing a lot of unneeded information (or missing key information) would score low. This metric reflects the precision of retrieval: ideally, the retriever should fetch as narrowly and accurately as possible the information required.

These three metrics (faithfulness, answer relevance, context relevance) correspond closely to the main failure modes of RAG systems: hallucinating unsupported facts, not answering the question, or retrieving poorly. RAGAS relies entirely on prompt-driven LLM evaluations and embedding similarity, avoiding any ground-truth answer or document labels. This makes it applicable even when no labeled dataset is available. The framework integrates with popular RAG pipelines (LangChain, LlamaIndex) for ease of use.

However, a limitation of RAGAS is that it uses a fixed set of hand-crafted prompts for the LLM judge and static heuristics, which may not adapt optimally to different domains or tasks [46]. For instance, the same prompts are used whether evaluating a biomedical QA system or a general knowledge chatbot, potentially affecting accuracy. Indeed, subsequent research found that while RAGAS is useful for quick evaluation, its scores can diverge from human judgment in certain cases, motivating more adaptive approaches [49]. Nonetheless, RAGAS has become a baseline for automated RAG evaluation and has inspired many follow-up tools.

3.2.2 DeepEval

DeepEval is an automated RAG evaluation framework developed by Confident AI that emphasizes integration into the development pipeline and continuous evaluation (e.g., unit tests and CI/CD) for RAG systems [11].

DeepEval provides a set of metrics similar in spirit to RAGAS, often with slightly different terminology. For example, it includes: (1) Contextual Precision and Contextual Recall, which quantify how much of the retrieved context is relevant to the query and whether all necessary context was retrieved; (2) Contextual Relevance, an overall measure of retrieval relevance; (3) Answer Relevancy (analogous to answer relevance in RAGAS); and (4) Faithfulness, measuring answer grounding. Under the hood, DeepEval uses large models to compute many of these metrics, so an OpenAI key is required. It often leverages prompt templates for scoring.

DeepEval distinguishes itself by its developer-focused tooling: it integrates with various RAG pipelines and can be invoked programmatically to test a pipeline's performance on a set of queries in code. Moreover, Confident AI's platform provides dashboards for DeepEval, allowing visualization of metrics over time, error analysis, and comparisons of different prompts or model versions.

3.2.3 Giskard: Open-Source RAG Testing Toolkit

Giskard is an open-source AI testing framework that provides a dedicated toolkit for evaluating RAG systems, often stylized as the RAG Evaluation Toolkit (RAGET) in their documentation [17]. Unlike RAGAS and DeepEval, which focus on on-the-fly evaluation using LLMs, Giskard's approach emphasizes generating a reproducible **test set** and then evaluating the system's answers against reference answers using both automated metrics and human-defined criteria.

The typical workflow is: (1) Automatically generate a set of query-answer pairs (and ensure the answers can be found in the knowledge base), (2) Run the RAG system on these queries to get its answers, and (3) Compare the system's answers to the reference answers using an LLM-based judge for correctness. Giskard's LLM judge essentially checks if the system's answer is semantically equivalent to the ground-truth answer, or explains differences if not. A report is then produced detailing various metrics and breaking down errors.

One of Giskard's notable features is that it scores each **RAG component** on the test set, helping pinpoint where issues lie. For example, it can attribute failures to the retriever versus the generator. The toolkit defines question categories that map to components: e.g., "lookup" questions primarily test retrieval, whereas "reasoning" questions test the generator's ability to synthesize information. By aggregating performance by category, Giskard yields a score for the Retriever, Generator, and any intermediary, like a query rewriter or router. This is very useful in diagnosing bottlenecks in a complex RAG pipeline.

3.2.4 ARES: Automated RAG Evaluation System

ARES (Automated Retrieval-Enhanced Scoring) is a recent framework [49] that advances automated RAG evaluation by using *synthetic data generation and model fine-tuning* to create specialized evaluators. ARES is notable for introducing a learning-based approach: instead of

relying on fixed prompts to an LLM (as RAGAS does), it generates a tailored "LLM judge" for each evaluation dimension by fine-tuning smaller language models on synthetic Q&A data.

The framework focuses on three key dimensions similar to RAGAS: Context Relevance, Answer Faithfulness, and Answer Relevance. However, the way these are evaluated is different:

- **Synthetic Data Generation**: Given the domain's document corpus, ARES uses an LLM to automatically generate a large set of question–answer pairs from the corpus. Essentially, it asks the LLM to pretend to be a user and create questions answerable by the documents, along with the correct answers. This artificial dataset spans several facets of the domain knowledge. Also, some of the answers are perturbed or combined with irrelevant passages to create negative examples.
- **LLM Judge Training**: ARES then specifies three classification tasks: context relevance judging, answer faithfulness, and answer relevance. For each task, it fine-tunes a lightweight model on the synthetic data. The labels for training come either directly from how the data was constructed or from a limited number of human annotations for validation. Importantly, ARES introduces the use of **prediction-powered inference (PPI)** [1]: a technique where a small set of human-labeled examples (on the order of 100–200) is used to calibrate the model's predictions and estimate confidence intervals for the scores. This addresses a major gap in prior work, which often gave point estimates with no notion of uncertainty.
- **Automated Scoring**: Once the judges are trained, ARES evaluates new RAG systems by feeding the judges with the system's outputs: for each question in an evaluation set, the context judge scores the retrieved passages, the faithfulness judge checks the answer against those passages, and the relevance judge checks the answer against the question. Since the judges are attuned to the domain, they are more precise compared to a general prompt.

ARES's trade-offs include complexity and computational expense: it needs initial investment to create data and train judges for each new application or domain. If the knowledge underlying the task shifts substantially, the synthetic data creation and fine-tuning procedure might have to be duplicated.

Chapter 4

The GQM-based RAG Evaluation Framework

The framework follows established GQM principles [4] while adapting to RAG-specific requirements. The proposed solution presents a four-layer hierarchical architecture that translates abstract objectives into measurable criteria. The architectural layers consist of:

- **Goal Layer**: The uppermost layer establishes abstract quality goals for RAG systems. The goals are written in accordance with GQM templates, naming the object (components of RAG systems) and quality aspects. Weights are assigned to each goal according to application priorities, allowing for tailored evaluation approaches.
- **Question Layer**: Questions operationalize objectives into specific, measurable dimensions. Questions are framed to be answerable by empirical measurement and are of direct concern to system improvement.
- **Metric Layer**: Metrics give quantitative responses to questions. Questions are mapped to one or several metrics, where the mapping takes into account metric attributes such as validity, reliability, and computation cost. The framework incorporates various types of metrics, such as conventional NLP metrics, neural similarity metrics, and LLM-based scorings. Metrics are normalized to a shared scale for combination.
- **Aggregation Layer**: The lower layer aggregates metric outcomes to yield actionable information. Aggregation adheres to the hierarchical framework, first merging metrics to reply to questions, and then integrating question outcomes to evaluate goal attainment. Weighted aggregation is employed in the framework with configured weights, and it offers diagnostic information regarding which goals/questions/metrics signify issues.

There is clear traceability for every evaluation result up the hierarchy. When a metric performs poorly, it can be traced upward to determine which questions and goals are affected. When goals are not achieved, the framework traces downwards to determine which individual metrics need to improve.

4.1 Goals Definition

The goal layer forms the foundation of our GQM-based framework, defining what constitutes quality in RAG systems. This section details our systematic approach to goal definition and how goals map to the multifaceted nature of RAG quality. For RAG systems, we identify five primary goals that comprehensively cover quality dimensions:

Goal 1: The RAG system generator component returns precise and true answers

Motivation: The fundamental promise of RAG systems is to provide accurate information from reliable sources. Unlike pure LLMs that may hallucinate, RAG systems should generate only information supported by retrieved documents. This goal addresses the critical trust requirement for deployment in domains where accuracy is vital.

Question 1.1: Is the response provided by the generator component based on facts and ground truth?

- **Component**: Generator
- **Metrics**:
  - **Faithfulness** (primary): Directly measures whether generated content is grounded in retrieved documents by decomposing responses into claims and verifying each against context
  - **Factual Consistency** (secondary): Uses specialized models to detect hallucinations and unsupported claims
  - **Attribution Score** (supporting): Evaluates the quality of source attribution in responses

Question 1.2: Does the generator component hallucinate when the required information is unavailable?

- **Component**: Generator
- **Metrics**:
  - **Faithfulness** (primary): Specifically detects unsupported claims that indicate hallucination
  - **Factual Consistency** (primary): Employs hallucination detection models to identify fabricated content

Goal 2: The RAG system returns only relevant information

Motivation: A RAG system must not only provide accurate information but also ensure it addresses the user's actual query. This goal encloses both retrieval relevance (finding related documents) and response relevance (generating appropriate answers).

Question 2.1: Does the retrieval component return documents that are contextually relevant to the user's query?

- **Component**: Retriever

- **Metrics**:
  - **Context Precision** (primary): Measures retrieval accuracy and document relevance
  - **Context Relevance** (primary): Uses LLM-based assessment of document-query alignment
  - **Semantic Diversity Score** (secondary): Ensures retrieved documents provide diverse perspectives without redundancy

Question 2.2: Does the generation component sufficiently address the user's query and provide useful information?

- **Component**: Generator
- **Metrics**:
  - **Answer Relevance** (primary): Measures query-response alignment through reverse question generation
  - **Answer Completeness** (secondary): Ensures all query aspects are adequately addressed

Goal 3: The RAG system covers all edge cases

Motivation: Many queries, especially in professional contexts, require comprehensive responses that cover different aspects. A response might be accurate and relevant yet still incomplete, missing crucial aspects that users need for informed decisions.

Question 3.1: Does the retrieval component identify and return all possible information relevant to the query?

- **Component**: Retriever
- **Metrics**:
  - **Context Recall** (primary): Measures coverage of relevant information in retrieved documents
  - **Context Entities Recall** (secondary): Ensures important named entities are not missed

Question 3.2: Does the generator component provide all necessary details for a comprehensive answer?

- **Component**: Generator
- **Metrics**:
  - **Answer Completeness** (primary): Explicitly checks if all query aspects are addressed
  - **Context Utilization Rate** (secondary): Measures how effectively available information is used

Goal 4: The generator component produces understandable and coherent answers.

Motivation: Even accurate, relevant, and complete information fails if presented incoherently. This goal ensures that generated responses are not only correct but also well-structured, logically organized, and easy to understand.

Question 4.1: Are the responses provided by the generator component easy to understand and logically coherent?

- **Component**: Generator
- **Metrics**:
  - **Self-Consistency Score** (primary): Detects contradictions and logical inconsistencies
  - **BERTScore** (secondary): Assesses semantic coherence and fluency

Goal 5: The RAG system handles complex queries effectively.

Motivation: Real-world information needs often require more than simple fact retrieval. Users may need a synthesis across documents or handling of conflicting information. This goal ensures RAG systems can handle sophisticated queries.

Question 5.1: Can the RAG system generate satisfactory answers for complex and confusing queries?

- **Component**: System-wide (Retriever + Generator)
- **Metrics**:
  - **Multi-hop Reasoning Score** (primary): Specifically evaluates connecting information across documents
  - **Answer Correctness** (secondary): Provides overall quality assessment
  - **Context Utilization Rate** (supporting): Measures synthesis effectiveness

4.2 Metric Aggregation Strategies

Within-Question Aggregation

For questions with multiple metrics, the framework employs weighted aggregation respecting metric roles:

- Primary metrics receive the highest weights (0.8-1.0)
- Secondary metrics provide complementary assessment (0.4-0.8)
- Supporting metrics offer additional validation (0.1-0.4)

Question-to-Goal Aggregation

Questions contribute equally to goal scores unless domain-specific requirements dictate otherwise.

Chapter 5

Practical Application and Experimental Features

This chapter describes the practical implementation of the GQM-based RAG evaluation framework, detailing the transformation of theoretical concepts into a fully functional system through modular design and systematic engineering practices. The complete implementation workflow is illustrated through architectural diagrams, demonstrating how each component contributes to the framework's evaluation capabilities.

We begin by examining the evaluation pipeline's end-to-end workflow through detailed flowchart analysis (Figure 5.1), tracing data and control flows from initialization through result presentation. Next, we present the system's component-based architecture, as depicted in the UML diagram (Figure 5.2), explaining the five primary subsystems—Core Evaluation Framework, Metrics, Test Generation, External Systems, and Dashboard UI and the design rationale behind their modular organization. The chapter then delves into the framework's core components, including the YAML-based configuration system, automated test case generation with intelligent caching, and the hierarchical GQM evaluation engine that orchestrates metric computation.

Following the architectural overview, we detail the implementation of fifteen specialized metrics across three categories: retrieval metrics for document selection quality, generation metrics for response evaluation, and system-level metrics for emergent properties. The integration mechanisms for RAG systems are then explored through the LangChain adapter example, demonstrating the framework's extensibility. The chapter concludes with the user interface and reporting capabilities, showing how evaluation results are persisted, visualized through an interactive dashboard, and made accessible for both programmatic and visual analysis.

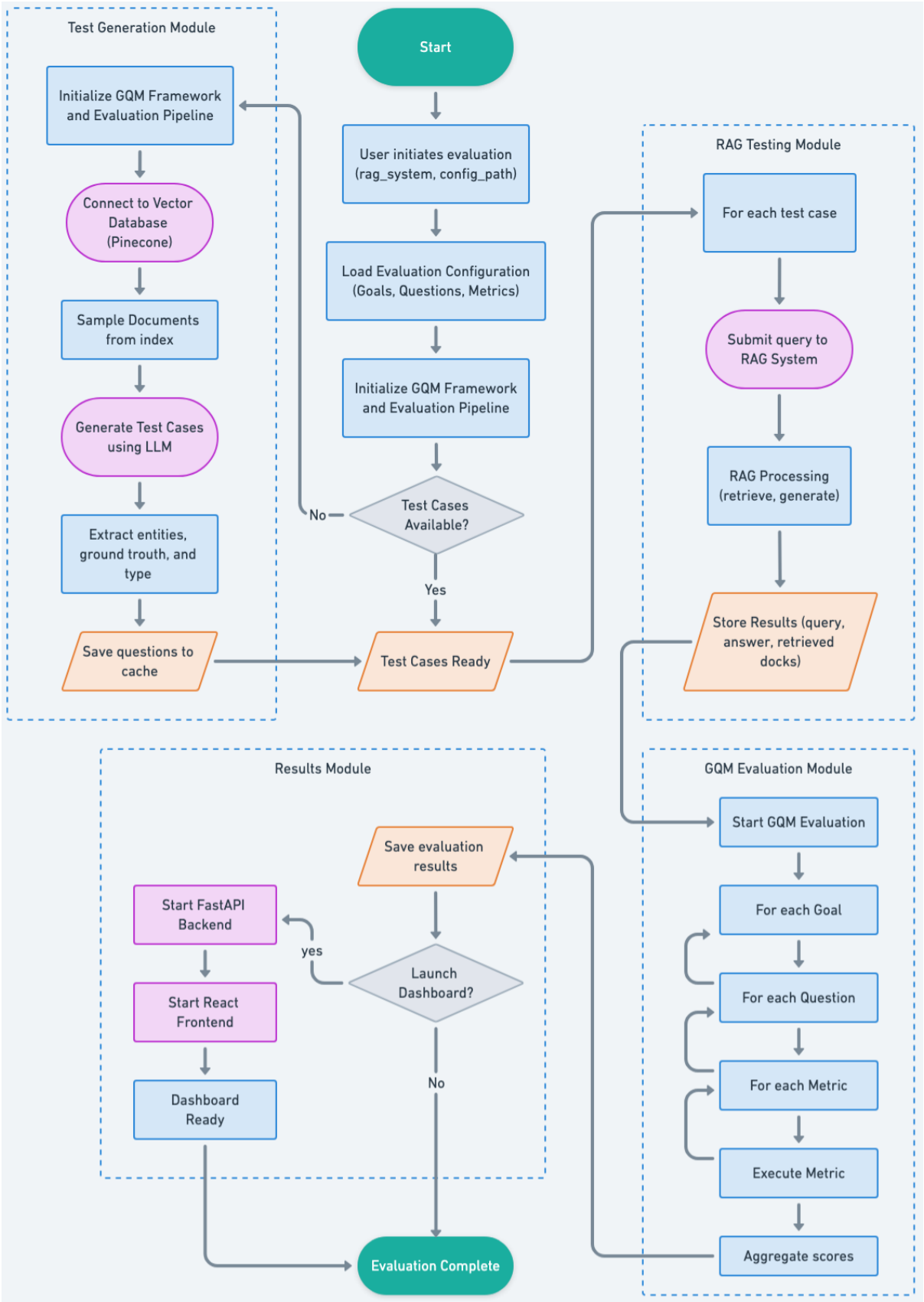

Figure 5.1: Flowchart diagram of the evaluation framework.

5.1 Framework Evaluation Workflow

This section explains the end-to-end evaluation process governing practical experimentation within the GQM-based RAG evaluation framework. It refers to the sequence of phases by which configurations, test cases, the system under test, and evaluation metrics are orchestrated to yield actionable results. Figure 5.1 presents a high-level flowchart of this process, illustrating its modular and reproducible structure.

1. **Initialization Phase:** At start, the framework reads configurations that specify system parameters, evaluation goals, and environment variables. Validation guarantees completeness and reproducibility across runs (see Subsection 5.3.4).
2. **Test Cases Generation:** Evaluation questions are either generated using LLM-driven sampling strategies, manually provided or drawn from a cached repository. This step guarantees comprehensive coverage across different query types (see Sec.5.3.3).
3. **RAG System Testing:** Each prepared test case is executed against the target RAG system. Both the generated answer and the underlying retrieval contexts are captured for later inspection.
4. **GQM Evaluation:** The GQM engine aggregates metric scores hierarchically - from metric-level values, through question-level scores, up to goal-level results - applying the weights defined in the configuration.
5. **Results Presentation:** Evaluation outcomes are persisted as structured JSON files for versioning and external analysis. Optionally, results are visualized via an interactive dashboard, enabling users to explore high-level summaries and fine-grained metric details.

5.2 System Architecture

The implementation of the GQM-based RAG evaluation framework follows a modular, component-based architecture designed for extensibility, maintainability, and ease of integration. Figure 5.2 illustrates the high-level architecture, decomposing the system into five primary subsystems: Core Evaluation Framework, Metrics Subsystem, Test Generation Subsystem, External Systems Integration, and the Dashboard User Interface.

This organization reflects software engineering best practices of high cohesion and loose coupling, allowing independent evolution and robust testing of each module.

5.2.1 Subsystem Overview

- **Core Evaluation Framework:** Orchestrates evaluation, manages configuration, and coordinates other subsystems via the `EvaluationPipeline`, `GQMFramework`, `MetricExecutor`, and `LangChainRAGAdapter`.
- **Metrics Subsystem:** Implements evaluation metrics as distinct modules for retrieval, generation, and holistic system-level assessment.
- **Test Generation Subsystem:** Automates creation and caching of diverse test cases via `QuestionGenerator` and `QuestionCache`.

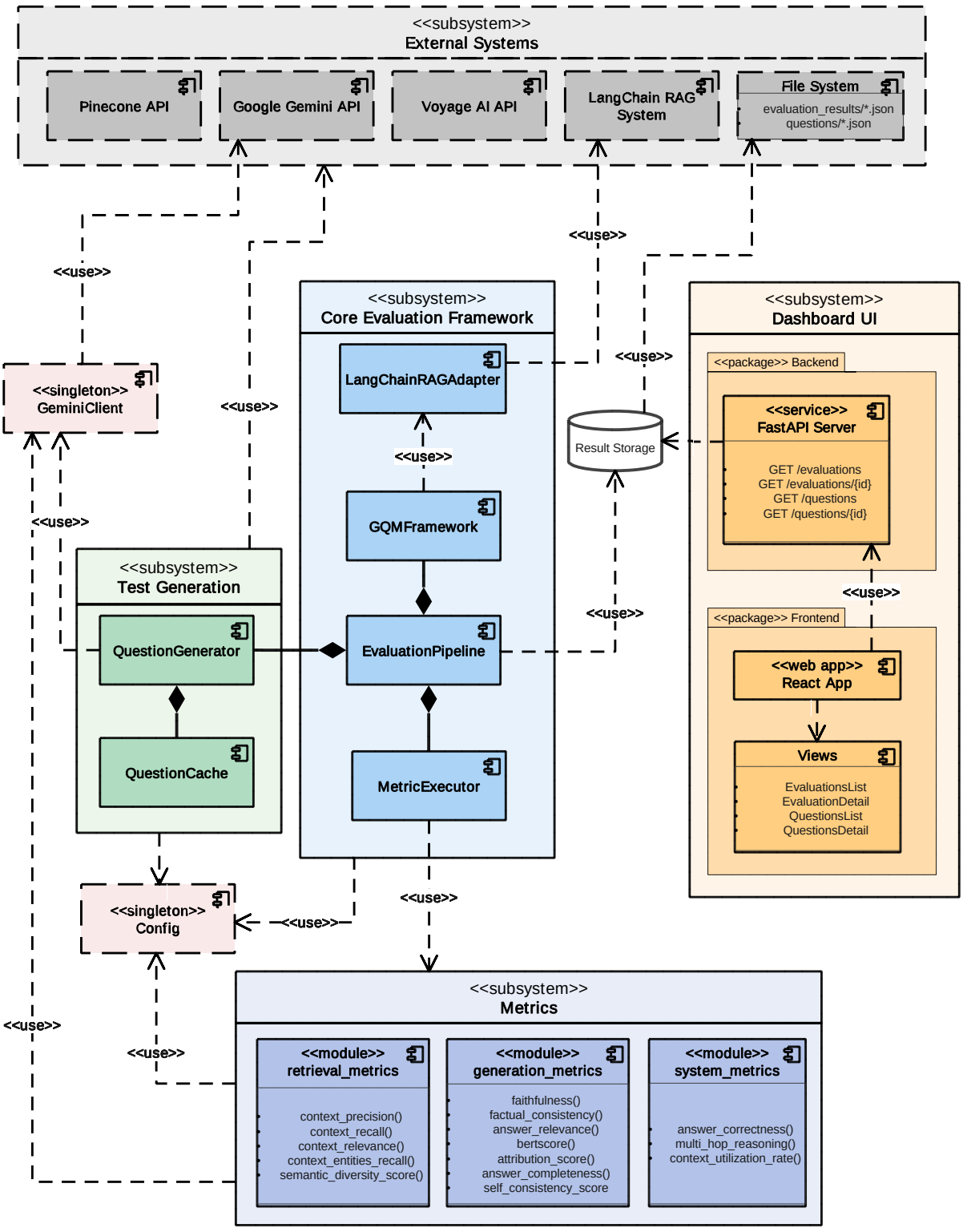

Figure 5.2: UML Component Diagram of the GQM-based RAG Evaluation Framework.

- **External Systems Integration:** Abstracts cloud service access (e.g., Google Gemini, Pinecone, Voyage AI) through singleton clients.
- **Dashboard User Interface:** Visualizes results using a modern web stack (FastAPI + React).

5.3 Framework Core Components

The following sections detail each subsystem's rationale, main components, and design patterns.

5.3.1 Core Evaluation Framework

At the heart of the system lies the Core Evaluation Framework, which orchestrates the entire evaluation process. This subsystem comprises four essential components:

- `EvaluationPipeline`: Serves as the primary entry point and coordinator, managing the evaluation workflow from configuration loading through result generation. It implements the Orchestrator pattern [20], coordinating multiple components and managing the complex multi-step evaluation process. The pipeline controls test case generation when needed, orchestrates evaluation execution, and manages result persistence.
- `GQMFramework`: Implements the hierarchical Goal-Question-Metric methodology [4], maintaining the structured relationships between goals, questions, and concrete metrics that are defined in structured configuration. It employs a Composite pattern to represent the tree structure of goals and questions, supporting recursive evaluation, flexible weighting, and aggregation of results. The framework maintains a metric cache to optimize repeated evaluations, implementing memoization to avoid redundant computations.
- `MetricExecutor`: Encapsulates the complexity of metric computation, providing a uniform interface for executing diverse metric types. It implements a function dispatch mechanism, using a lookup table to map metric identifiers to their corresponding function implementations. This component handles both traditional NLP metrics and LLM-based evaluations, abstracting the differences in their execution models and providing consistent error handling and result formatting across all metric types.
- `LangChainRAGAdapter`: Implements the Adapter pattern to provide a standardized interface for interacting with RAG systems. This abstraction layer enables the framework to evaluate different RAG implementations without modification to the core evaluation logic, promoting extensibility and reusability.

5.3.2 Metrics Subsystem

The Metrics subsystem organizes evaluation metrics into three specialized modules:

- `retrieval_metrics`: Implements metrics specific to the document retrieval component, including context precision, recall, relevance, and semantic diversity. These metrics leverage both traditional information retrieval measures and modern neural approaches, providing a comprehensive assessment of retrieval quality [52].

- `generation_metrics`: Focuses on evaluating the quality of generated responses, implementing metrics such as faithfulness, factual consistency, and answer relevance. These metrics employ advanced techniques including claim extraction and verification using specialized models like HHEM [59], as well as LLM-based evaluation approaches that have shown strong correlation with human judgment [32].
- `system_metrics`: Provides holistic evaluation capabilities that assess the integrated performance of retrieval and generation components. Metrics such as answer correctness, multi-hop reasoning ability, and context utilization rate capture emergent properties that cannot be evaluated through component-level metrics alone.

5.3.3 Test Generation Subsystem

The Test Generation subsystem automates the creation of evaluation test cases, addressing the challenge of comprehensive coverage across different query types and complexity levels:

- `QuestionGenerator`: Implements sophisticated test case generation strategies, utilizing LLMs to create diverse question types that target specific RAG components. It employs a sampling strategy to select representative documents from the vector database, ensuring that generated questions reflect the actual content distribution. The generator supports multiple question types - simple, complex, distracting, situational, double, and conversational - each designed to evaluate different aspects of RAG system capabilities (see Table 5.1).
- `QuestionCache`: implements a storage system for generated test cases. The cache uses a hash-based indexing scheme that considers both generation parameters and source documents.

Type	Components	Description
Simple	Retriever, Generator, Router	Basic factual queries
Complex	Generator	Multi-step reasoning queries
Distracting	Generator, Retriever, Rewriter	Includes misleading or irrelevant context
Situational	Generator	Context-dependent, scenario-based
Double	Retriever, Generator	Multiple sub-questions in one
Conversational	Rewriter	Queries in conversational context

Table 5.1: Supported Test Case Types in Automated Generation

5.3.4 External Systems Integration

The External Systems subsystem represents managed access to cloud-based services essential for evaluation operations.

- `GeminiClient`: The singleton component manages all interactions with Google's Gemini Application Programming Interface (API). The client maintains separate generation configurations for different use cases - metric evaluation requires high consistency (low temperature), while question generation benefits from controlled creativity.

- `Config`: The singleton component implements centralized configuration management, providing type-safe access to environment variables, test generation settings, evaluation goals, and parameters throughout the system. The configuration system leverages Pydantic for type-safe parameter management, automatically validating required fields and providing helpful error messages for misconfigurations. Environment variables store sensitive information such as API keys, following security best practices.
- *Other Services*: Pinecone vector database and Voyage AI embedding model follow analogous patterns by providing singleton components for managing all interactions with their APIs.

5.3.5 Dashboard User Interface

The Dashboard UI subsystem provides visual access to evaluation results, implementing a modern single-page application architecture.

- **Backend**: FastAPI-based REST API exposes endpoints for retrieving evaluation results, browsing test questions, and accessing detailed metrics.
- **Frontend**: React SPA implements responsive visualization of hierarchical evaluation results, enabling drill-down from overall scores to individual metric results.

5.3.6 File System Persistence

The framework utilizes local file system storage for evaluation results and cached test questions, implementing a simple but effective persistence strategy. Results are stored as timestamped JSON files, enabling easy versioning and comparison. The structured JSON format preserves the hierarchical nature of GQM results while remaining human-readable and processable by external tools.

5.4 Metric Implementations

5.4.1 Retrieval Metrics

- **Context Precision:** evaluates retrieval quality by measuring the ranking effectiveness of retrieved documents. The metric rewards systems that place relevant documents at higher positions.

  Formula

  $$\text{Context Precision} = \frac{\sum_{k=1}^{K} (\text{IsRelevant}_k \times \text{Precision@k})}{\text{Total Relevant Documents}} \tag{5.1}$$

  where:

  - K = number of retrieved documents (typically 5)
  - IsRelevant_k = 1 if document at position k is relevant, 0 otherwise
  - Precision@k = $\frac{\text{Number of relevant documents in top } k}{k}$

Implementation: Relevance assessed via LLM (Gemini) with binary classification, evaluates top-5 documents ($K = 5$).

- **Context Recall:** measures the completeness of retrieved documents by evaluating whether they contain all necessary information present in the ground truth answer. The metric decomposes the reference answer into atomic claims and verifies their presence in retrieved contexts.

  Formula

  $$\text{Context Recall} = \frac{\sum_{i=1}^{N} \text{Attribution}_i}{N} \tag{5.2}$$

  where:

  - N = total number of claims extracted from ground truth
  - Attribution_i = 1 if claim i can be attributed to retrieved contexts, 0 otherwise

  Implementation: Claims extracted from ground truth using LLM (Gemini), each claim verified against retrieved contexts via LLM-based attribution.

- **Context Relevance:** evaluates the quality of retrieved documents by extracting statements and measuring their relevance to the query. The metric identifies the proportion of meaningful information that directly addresses the user's question.

  Formula

  $$\text{Context Relevance} = \frac{\sum_{j=1}^{M} \text{IsRelevant}_j}{M} \tag{5.3}$$

  where:

  - M = total number of statements extracted from retrieved contexts
  - IsRelevant_j = 1 if statement j is relevant to the query, 0 otherwise

  Implementation: Statements extracted from retrieved contexts using LLM (Gemini), relevance classification performed via LLM for each statement, evaluates up to 5 retrieved documents.

- **Context Entities Recall:** measures the completeness of entity coverage in retrieved documents by comparing extracted entities against expected entities from the ground truth. The metric evaluates whether important named entities are successfully retrieved.

  Formula

  $$\text{Context Entities Recall} = \frac{|\mathcal{E}_{\text{retrieved}} \cap \mathcal{E}_{\text{expected}}|}{|\mathcal{E}_{\text{expected}}|} \tag{5.4}$$

  where:

  - $\mathcal{E}_{\text{expected}}$ = set of normalized entities from ground truth
  - $\mathcal{E}_{\text{retrieved}}$ = set of normalized entities extracted from retrieved documents

Implementation: Entity extraction using spaCy model (`en_core_web_sm` `en_core_web_trf`), entity normalization: lowercase, article removal ("the", "a", "an"), whitespace standardization, processes all retrieved documents combined.

- **Semantic Diversity Score:** evaluates the diversity of retrieved documents using Maximal Marginal Relevance (MMR) to balance relevance and non-redundancy. The metric ensures retrieved documents provide varied perspectives while maintaining query relevance.

  Formula

$$\text{Semantic Diversity} = \frac{1}{2}\left(\frac{1}{K}\sum_{i=1}^{K}\text{MMR}_i + 1\right) \tag{5.5}$$

  where MMR for document i is:

$$\text{MMR}_i = \lambda \cdot \text{sim}(d_i, q) - (1-\lambda) \cdot \max_{j \in S_{i-1}} \text{sim}(d_i, d_j) \tag{5.6}$$

  and:

  - K = number of evaluated documents (default 5)
  - λ = trade-off parameter between relevance and diversity (default 0.5)
  - $\text{sim}(d_i, q)$ = cosine similarity between document i and query
  - S_{i-1} = set of previously selected documents
  - Normalization maps MMR from $[-1, 1]$ to $[0, 1]$

  Implementation: Document embeddings via Voyage AI model (`voyage_3_large`), sequential document selection maximizing MMR.

5.4.2 Generation Metrics

- **Faithfulness:** measures whether generated responses are grounded in retrieved contexts by decomposing answers into atomic claims and verifying each against the source documents. The metric prevents hallucination by ensuring all statements are supported.

  Formula

$$\text{Faithfulness} = \sum_{i=1}^{C} w_i \cdot \text{HHEM}_i \tag{5.7}$$

  where:

  - C = number of claims extracted from generated answer
  - HHEM_i = hallucination score for claim i (0 to 1, higher is better)
  - $w_i = \frac{\text{length}(c_i)}{\sum_{j=1}^{C}\text{length}(c_j)}$ = normalized weight based on claim length
  - $\text{length}(c_i)$ = word count of claim i

Implementation: Claims extracted via LLM (Gemini) with structured prompting, HHEM model (`vectara/hallucination_evaluation_model`) scores each claim, longer claims receive higher weights reflecting information density.

- **Factual Consistency:** evaluates whether the entire generated response is factually consistent with retrieved contexts using hallucination detection. Unlike faithfulness, this metric assesses the answer holistically without decomposition.

  Formula

  $$\text{Factual Consistency} = \text{HHEM}(\text{answer}, \text{combined\_context}) \tag{5.8}$$

  where:

  - answer = complete generated response
  - combined_context = concatenation of all retrieved documents
  - HHEM = Hughes Hallucination Evaluation Model score (0 to 1)

  Implementation: Direct application of HHEM model to full answer-context pair, all retrieved documents concatenated with double newlines, single-pass evaluation without claim extraction.

- **Answer Relevance:** measures how well the generated answer addresses the user's query using reverse question generation. The metric generates questions from the answer and measures their similarity to the original query, based on the principle that relevant answers enable reconstruction of similar questions.

  Formula

  $$\text{Answer Relevance} = \begin{cases} 0.5 \cdot s_1 + 0.3 \cdot s_2 + 0.2 \cdot s_3 & \text{if } |Q_g| \geq 3 \\ \frac{1}{|Q_g|} \sum_{i=1}^{|Q_g|} s_i & \text{otherwise} \end{cases} \tag{5.9}$$

  where:

  - Q_g = set of questions generated from the answer
  - s_i = cosine similarity between original query and i-th generated question (sorted descending)
  - Weights (0.5, 0.3, 0.2) follow exponential decay to prioritize best matches

  Similarity Computation

  $$s_i = \frac{\mathbf{e}_q \cdot \mathbf{e}_{g_i}}{||\mathbf{e}_q|| \cdot ||\mathbf{e}_{g_i}||} \tag{5.10}$$

  where $\mathbf{e}_q$ and $\mathbf{e}_{g_i}$ are Voyage AI embeddings of original and generated questions.

  Implementation: Question generation via LLM (Gemini) with structured prompting, embeddings using Voyage AI (`voyage-3-large`) for semantic similarity, generates 3 questions capturing different answer aspects.

- **BERTScore:** evaluates semantic similarity between generated answers and retrieved contexts using contextual embeddings. The metric captures meaning preservation beyond surface-level matching by leveraging pre-trained language models [63].

Formula

$$\text{BERTScore} = \max_{d \in D} F_1(a, d) \tag{5.11}$$

where for each document d:

$$P = \frac{1}{|a|} \sum_{a_i \in a} \max_{d_j \in d} \text{sim}(\mathbf{a}_i, \mathbf{d}_j) \cdot \text{idf}(d_j) \tag{5.12}$$

$$R = \frac{1}{|d|} \sum_{d_j \in d} \max_{a_i \in a} \text{sim}(\mathbf{a}_i, \mathbf{d}_j) \cdot \text{idf}(d_j) \tag{5.13}$$

$$F_1 = 2 \cdot \frac{P \cdot R}{P + R} \tag{5.14}$$

and:

- a = tokens in generated answer, d = tokens in context document
- $\mathbf{a}_i$, $\mathbf{d}_j$ = contextual embeddings from pre-trained model
- sim = cosine similarity between embeddings
- idf = inverse document frequency weighting
- D = set of all retrieved documents

Implementation: Uses `roberta-large` model for contextual embeddings, computes F1 score between answer and each retrieved document, takes maximum score across documents (best semantic match).

- **Attribution Score:** evaluates how well generated answers attribute information to their sources. The metric assesses whether factual claims are properly cited and whether cited sources actually contain the referenced information [47, 6]. This metric is judged by an LLM evaluator that scores attribution quality on a 0-10 scale, which is then normalized to [0, 1].
- **Answer Completeness:** measures how comprehensively the generated answer addresses all aspects of the query. The metric identifies sub-questions within the query and evaluates whether each receives adequate treatment in the response. This metric is judged by an LLM evaluator that scores completeness on a 0-10 scale, which is then normalized to [0, 1].

5.4.3 System-Level Metrics

- **Answer Correctness:** evaluates the overall quality of generated answers by comparing them against ground truth references. The metric provides a holistic assessment considering factual accuracy, completeness, and consistency. This metric is judged by an LLM evaluator that scores correctness on a 0-10 scale, which is then normalized to [0, 1].
- **Multi-Hop Reasoning Score:** evaluates the system's ability to connect information across multiple documents to answer complex queries. The metric assesses synthesis quality when answering requires combining facts from different sources [34, 61]. This metric is judged by an LLM evaluator that scores reasoning quality on a 0-10 scale, which is then normalized to [0, 1]. Only queries with at least 2 retrieved documents are evaluated.
- **Context Utilization Rate:** measures how effectively the generation component leverages retrieved documents to construct answers. The metric evaluates whether available information is properly incorporated rather than ignored or underutilized. This metric is judged by an LLM evaluator that scores utilization effectiveness on a 0-10 scale, which is then normalized to [0, 1].

Chapter 6

Results

This chapter presents a comprehensive evaluation results of three distinct RAG system configurations using our GQM based framework. We report the results of three RAG experiments, each with a distinct model configuration, and analyze their performance across multiple evaluation goals, components, and query types. The hierarchical Goal-Question-Metric structure enables a fine-grained comparison of these systems, providing insights that would be difficult to obtain with traditional black-box metrics or single-score evaluations.

6.1 Experimental Setup

We evaluated three configurations of the RAG system (Experiment 1, Experiment 2, and Experiment 3), each employing a different underlying language model and embedding model. Table 6.1 summarizes the key characteristics of these experiments. Experiment 1 used a lightweight model *phi4-mini* for generation with the BAAI *bge-small-en-v1.5* encoder for document embeddings. Experiment 2 used a medium capacity LLM (denoted "llama3.2:3b") with the *sentence-transformers/all-MiniLM-L6-v2* embedding model. Experiment 3 used a larger 7B-parameter LLM (LLaMA 2 7B) together with a higher capacity embedder *all-MiniLM-L12-v2*. All other components (retriever, dataset, and evaluation process) were kept constant between experiments to ensure a fair comparison.

Experiment	LLM	Params	Embedding Model	Embed Dim
1	*phi4-mini*	3.8B	BAAI/bge-small-en-v1.5	384
2	*llama3.2:3b*	3B	all-MiniLM-L6-v2	384
3	*llama2:7b*	7B	all-MiniLM-L12-v2	384

Table 6.1: RAG Experiments and Model Configurations

Each configuration maintained consistent retrieval parameters ($k = 3$ documents) and generation temperature ($T = 0.1$). The embedding models were selected to represent different trade-offs between computational efficiency and semantic representation quality.

Each RAG system was evaluated on the same set of previously generated test cases. Automated test generation produced 30 diverse test cases distributed equally across six types of questions. This diversity ensures that the evaluation probes different aspects of the RAG

systems: from handling simple questions to maintaining performance on complex or noisy queries.

All three experiments followed an identical evaluation procedure based on the GQM framework described in Chapter 4. The evaluation defined a consistent set of five high-level goals, each broken down into specific questions with associated metrics.

The GQM framework computed not only individual metric values for each query but also aggregated them into goal-level scores and an overall framework score for each experiment. In the following sections, we present the results in terms of overall performance, breakdown by goals, component-level analysis, question type analysis, comparison of experiments.

6.2 Overall Framework Performance

We begin by examining the top-level outcome of the evaluation for each experiment. Overall Score is the single aggregate metric produced by the framework, reflecting the weighted achievement across all goals (with each goal here weighted equally). Table 6.2 presents the overall scores of the three experiments. Experiment 1 (phi4-mini) achieved an overall score of only **55.9%**, indicating that the baseline configuration struggled to meet many of the evaluation goals. In contrast, Experiment 2 (3B LLaMA-based model) achieved a much higher overall score of **72.5%**. This marks a substantial improvement of over 16 percentage points, highlighting the impact of using a more capable LLM and a stronger embedding model. Experiment 3 (7B LLaMA) reached an overall score of **70.5%**. Notably, the 7B model did not outperform the 3B model in our evaluation.

	Exp.1 (phi4-mini)	**Exp.2 (LLaMA3B)**	**Exp.3 (LLaMA7B)**
Overall Score (%)	55.9	72.5	70.5

Table 6.2: Overall Score (aggregated performance) in each experiment

The general performance trend is clear: the RAG system configuration with a more powerful generative model and a better retriever embedding (Exp.2) yields a significantly higher overall quality than the lightweight baseline (Exp.1). However, simply increasing the LLM size from 3B to 7B (Exp.3) did not proportionally increase the score. This suggests that other factors (such as retrieval quality or possibly the specific tuning of the models) play an important role in the RAG system's effectiveness, and bigger is not always strictly better in our evaluation setting. The overall scores demonstrate that our GQM-based framework can differentiate the systems at a high level, confirming that the evaluation setup is sensitive to improvements in model and retriever components. The next sections break down these overall results by goals, components, and query types to provide deeper insight into where each system excels or falls short.

6.3 Goal-by-Goal Analysis

To understand the strengths and weaknesses of each experiment, we analyze the performance on each evaluation goal defined by the GQM framework. Figure 6.1 presents a radar graph that compares the goal achievement scores of Experiments 1 to 3 on the five high-level goals.

Each axis of the radar corresponds to one goal, and a higher value indicates better fulfillment of that goal. The quantitative goal scores for each experiment are also listed in Figure 6.2.

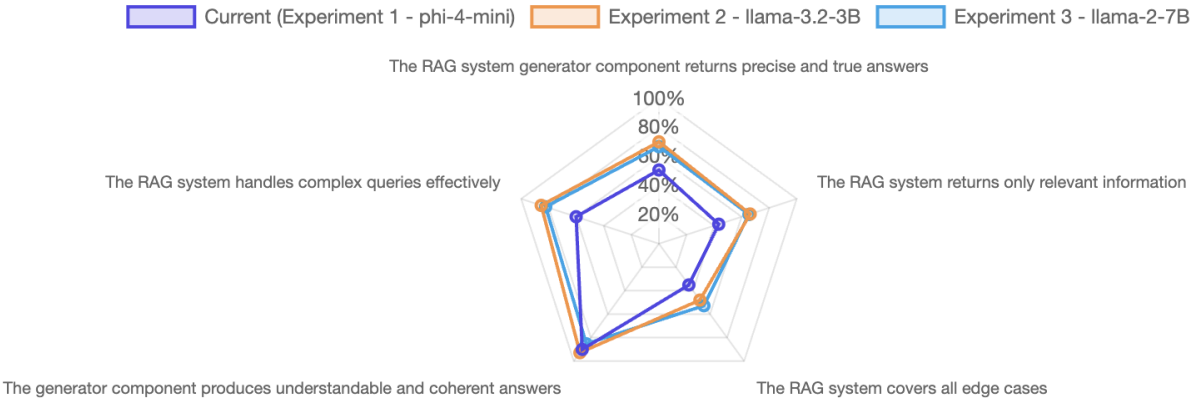

Figure 6.1: Goal attainment profile for the three experiments, shown as a radar chart.

Metric	Current (Experiment 1 - phi-4-mini)	Experiment 2 - llama-3.2-3B	Experiment 3 - llama-2-7B
Overall Score	55.9%	72.5%	70.5%
The RAG system generator component returns precise and true answers	50.8%	70.2%	66.9%
The RAG system returns only relevant information	43.4%	66.2%	65.3%
The RAG system covers all edge cases	35.1%	48.1%	52.8%
The generator component produces understandable and coherent answers	90.1%	92.8%	85.3%
The RAG system handles complex queries effectively	60.1%	85.4%	82.2%

Figure 6.2: Performance by Evaluation Goal for each Experiment (goal scores in %).

Several important observations can be made from these goal-level results. First, Experiments 2 and 3 outperform Experiment 1 on almost every goal, often by a wide margin. The radar chart (Figure 6.1) shows that Exp.1 is much smaller, especially along the axes for G1, G2, and G5. This indicates that the phi4-mini model struggled to answer questions correctly and retrieve relevant information, particularly for complex queries. In numerical terms, Exp.1 achieved only about 50–60% of the target on those goals, whereas Exp.2 and Exp.3 scored in the mid-60s to mid-80s, reflecting far more successful performance.

Second, the largest performance gains from Exp.1 to Exp.2 were seen in the goals related to answer precision/truthfulness and handling complex queries. For Goal 1, the score jumped from **50.8%** in Exp.1 to **70.2%** in Exp.2, an improvement of nearly 20 percentage points. A similar improvement is observed for Goal 5, rising from **60.1%** to **85.4%**. These results suggest that the more advanced LLM in Exp.2 was markedly better at providing factually correct answers and coping with challenging queries that likely require reasoning or multi-step retrieval. The embedding upgrade in Exp.2 also likely contributed to this by retrieving more useful context for complex questions.

Third, the goal analysis reveals some areas where models struggled. Goal 3 received the lowest scores across the board: only **35.1%** in Exp.1, improving to **48.1%** in Exp.2 and **52.8%** in Exp.3. Although the larger models did better at covering edge cases than the smaller model, a score around 50% indicates that roughly half of the edge-case aspects were missed by all systems. This goal, which involves ensuring that the system can handle unusual or particularly difficult scenarios, remains a challenge.

Interestingly, Goal 4 was a strong point for all models. Exp.1 already scored **90.1%**, and Exp.2 even reached **92.8%**. Exp.3 scored slightly lower at **85.3%**, but still relatively high. The near-maximal scores for Exp.1 and Exp.2 indicate that even the smallest model was able to produce answers that are fluent, well-structured, and self-consistent. This is not surprising: coherence and grammaticality are attributes that even smaller modern language models handle well, as they are core strengths of pretrained transformers [58].

In summary, the goal-by-goal analysis confirms that upgrading the RAG system (from Exp.1 to Exp.2/3) led to improvements, especially in answer correctness, relevance, and tackling complexity, while all systems performed strongly on answer coherence and all struggled to fully address edge cases. This level of analysis, enabled by the GQM framework, pinpoints specific capability gaps (e.g., edge case handling) that can direct future system improvements.

6.4 Component-Level Performance Analysis

Beyond high-level goals, our framework allows us to isolate performance at the level of individual system components – principally the retrieval versus generation stages – as well as combined system-level metrics. By aggregating metrics into retrieval-related, generation-related, and system-level categories, we can examine how each part of the RAG pipeline contributed to the overall performance in each experiment. Figure 6.3 provides a comparison of average retrieval component, generation component, and system scores for the three experiments. Table 6.4 enumerates component metrics averages.

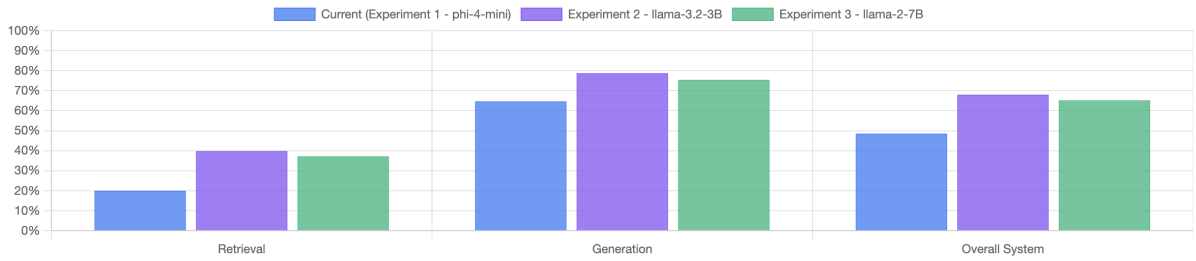

Figure 6.3: Component-level performance comparison for Experiments 1, 2, and 3. Each group of bars shows the average score for Retrieval metrics, Generation metrics, and Overall System metrics outcomes.

In summary, component-level analysis underscores that Exp.1 was retrieval-limited, whereas both Exp.2 and Exp.3 achieved a better balance, yet still with relatively weaker retrieval than generation. Improving retrieval (precision and recall of relevant context) appears to be a key lever for boosting overall RAG performance further, since the generation component – especially in the larger models – is already performing at a high level given whatever information it has. The GQM framework's ability to break out these component metrics provides this insight clearly: for instance, we can trace low edge-case coverage (Goal 3) to low context recall

Component Metric Category	Exp.1	Exp.2	Exp.3
Retrieval	20.0	39.8	37.2
Generation	64.7	78.7	75.4
Overall System	48.6	68.0	65.1

Table 6.3: Average Component Performance Metrics by Experiment (%)

and entity recall in retrieval, even when generation completeness was high. This is an advantage over end-to-end metrics alone, as it identifies whether deficiencies lie in the retriever, the generator, or the interplay of both.

6.5 Performance by Question Types

We next analyze how the RAG systems performed on different categories of questions. Each test query was classified as Simple, Complex, Distracting, Situational, Double, or Conversational (as defined in the Experimental Setup). By examining performance on each type, we can see which kinds of queries were easier or harder for the systems, and how the model upgrades impacted each category. Figure 6.4 provides a heatmap of metric scores for each question type in each experiment.

The heatmap (Figure 6.4) is especially informative: each row corresponds to a question type and each column to a particular evaluation metric (from faithfulness and factual consistency, through various metrics, to answer correctness). We observed clear differences in the difficulty of question types. Simple questions were handled best by all systems. In Experiment 1, simple queries still had mediocre scores (e.g., only 40% correctness), but in Experiments 2 and 3, simple queries yielded among the highest scores across metrics: for Exp.3, the simple questions achieved 96% answer relevance, 84% answer completeness, and 80% answer correctness, indicating that the larger models correctly answered straightforward questions with high precision. Similarly, Exp.2 got 80% correctness on simple questions. The improvements from 40% (Exp.1) to 80% (Exp.2 and Exp.3) on simple queries demonstrate how additional model capacity and better retrieval turn trivial questions from partial failures to near-perfect successes.

Complex questions benefited from the advanced models as well. Exp.1 struggled with the complex queries (only 68% answer correctness and 33% factual consistency).Exp.2 produced a mostly correct answer that still missed some aspect (hence high faithfulness but not full correctness). In contrast, Exp.3 likely provided a more complete answer to the complex query, earning a higher correctness score.

For distracting questions (which include irrelevant or misleading information), we found that Exp.1 and Exp.2 had mixed results, whereas Exp.3 showed improvement in some metrics. Exp.3 achieved higher factual consistency (62%) on distracting queries and maintained high context precision (82%). Both larger experiments had reasonably good answer correctness on distracting questions (Exp.2: 96%, Exp.3: 70%), whereas Exp.1 was lower (46%). These data suggest that distracting questions remained challenging, but the models improved their ability to ignore irrelevant information (high precision) and still answer correctly.

In summary, examining performance by question type reveals that Simple and Situational

Figure 6.4: Heatmap of metric performance by question type for Experiment 1-3.

queries were easiest, with near-perfect success by the top systems, while Complex and Distracting queries improved greatly with model size but still showed some variability. Double questions and Conversational queries highlighted interesting cases where the largest model did not always outperform the smaller one. Overall, the trend is that the advanced RAG configurations (Exp.2 and Exp.3) markedly improve performance on all question types compared to the baseline Exp.1, but certain challenging query forms can still pose difficulties. This type-oriented analysis demonstrates the utility of diverse test cases in the evaluation framework: it ensures that improvements are validated across different user scenarios, and it helps identify specific query types where further model or retrieval enhancements are needed.

6.6 Comparative Analysis Across Experiments

Experiment 2 configuration emerged as the top performer in all tests, slightly edging Experiment 3 in overall performance. Both substantially outperformed the Experiment 1 baseline. The trend suggests that intelligent configuration choices (and perhaps fine-tuning) can sometimes outweigh brute-force scaling. The GQM evaluation framework proved critical in reaching this conclusion, as it revealed the multi-dimensional performance landscape underlying a single overall score: for instance, it uncovered that Exp.3's slight dip was tied to specific goals and query types rather than a uniform performance decrease. Such insights validate the framework's ability to guide iterative system development by pinpointing areas of improvement or regression in each new experiment.

6.7 Detailed Per-Query Results

Beyond the aggregated results and analyses presented above, our evaluation framework also provides detailed, per-query results for each test case. For every query posed to the RAG system, we have recorded the full set of metric scores. This granular data allows for case-by-case inspection of system behavior. This section briefly highlights how such detailed results can be used to gain additional insights, and mentions an example failure case from Experiment 1 to illustrate.

Each test query result includes the query itself, the system's generated answer, the question type label, and all metric evaluations for that single interaction. By reviewing these, one can perform error analysis to see exactly where a system failed on a given query. For instance, one of the simple queries in our set asked: "What is the email address for the office contact of Professor Tomas Sauer?" In Experiment 1, the system's answer was "I don't know." - a failure to provide the requested info. The detailed metrics for this case showed extremely low scores: 0% for answer correctness and answer completeness, 0% context recall (it retrieved no relevant document about Prof. Sauer), and near-zero factual consistency, despite a high attribution score (100%) because the model did not hallucinate anything beyond what it knew (it correctly "attributed" its lack of answer to not having info). In contrast, Experiment 2 output for the same query was "I don't know, there is no information about Professor Tomas Sauer in the provided context." – which, while still not giving the email, was a more informative failure. The metrics reflected a slight improvement: answer relevance jumped to 85% (since the answer addressed the query by explaining the info isn't available), though context precision was still 0% (no relevant document found). Experiment 3 attempted a very elaborate answer for this query, discussing privacy and suggesting how to contact the office, which was not

necessary. Its metrics showed moderate faithfulness (40%) and completeness (0% because it never provided the email), illustrating how it tried to compensate for the lack of information with general statements. This example demonstrates that the per-query metric breakdown can explain why a query was answered incorrectly: in this case, it was a retrieval failure (no relevant contact info was retrieved from the knowledge base), as evidenced by the context recall/utilization metrics, rather than a generation language issue. By similarly examining other individual cases, we can identify specific patterns.

Chapter 7

Discussion

This chapter combines the findings from our implementation and experimental evaluation of the GQM-based RAG evaluation framework. We reflect on the key insights gained, examine how the research questions have been answered, analyze the framework's advantages and limitations, discuss practical implications, and outline directions for future research.

7.1 Key Findings

Applying the GQM-based evaluation framework to various RAG systems revealed several significant insights:

- The framework effectively decomposed the complex evaluation process into clear goals, questions, and metrics, resulting in a detailed performance profile. This approach uncovered specific weaknesses often concealed by aggregated scores from existing alternative tools.
- The framework exhibited strong diagnostic capabilities by clearly linking low-level metrics to high-level system performance goals.
- Comprehensive and balanced metrics enabled a holistic evaluation of RAG systems, integrating traditional IR metrics, LLM-based semantic metrics, and custom measures. This approach aligns with human evaluation studies [30], emphasizing the need to assess correctness, completeness, and grounding in sources.
- The framework consistently demonstrated effectiveness across diverse RAG architectures tested, regardless of LLM size or retriever type. Radar charts of goal-level scores clearly illustrated trade-offs and provided distinct performance profiles, enhancing the capability to discriminate and characterize system performance.

7.2 Answering the Research Questions

With the empirical results and insights in mind, we now revisit the research questions and provide answers to each.

RQ1: How can the Goal-Question-Metric methodology be effectively adapted for evaluating RAG systems?

Our work demonstrates that the Goal-Question-Metric methodology can be effectively adapted to RAG system evaluation through several key innovations:

- **Hierarchical Goal Structure:** We developed a five-goal hierarchy that comprehensively covers RAG quality dimensions: precision and truthfulness, contextual grounding, edge case handling, robustness, and complex query support. The goal weights enable context-specific customization, acknowledging that different applications may prioritize different quality aspects.
- **Question Operationalization:** The framework formulates specific questions under each goal, targeting the aspects that need to be evaluated to determine if the goal is met.
- **Metric Mapping Strategy:** Adapting GQM effectively meant choosing metrics appropriate to RAG's characteristics: some metrics come from information retrieval (for retrieval-focused questions), others from natural language generation evaluation, and some were developed specifically for RAG contexts. Each such metric was carefully aligned to its question, preserving the GQM principle that every metric's purpose is traceable to a higher-level concern. We established a principled approach for mapping metrics to questions, categorizing metrics as primary, secondary, or supporting based on their relevance and diagnostic value. This weighted mapping ensures that evaluation results accurately reflect the relative importance of different quality aspects.

Additionally, our adaptation leveraged LLM-based evaluation methods within the GQM framework, a combination of a classical methodology with cutting-edge LLM evaluators. For instance, some questions (like checking answer factuality or coherence) are difficult to answer with static heuristics, so we integrated an LLM-as-a-judge approach: the LLM (Gemini, in our case) was prompted to score or classify the answer with respect to the question. This is in line with recent advancements in reference-free evaluation [14, 11], but our contribution was to embed these methods into a GQM-driven process. We carefully adapted prompt designs to ensure consistency and bias mitigation (e.g., having the LLM explicitly compare the answer to provided sources for factuality checks). This adaptation required rigorous validation (we cross-validated some LLM-judged scores with, see Chapter 6 results) to ensure the GQM conclusions remain trustworthy.

The result is a theoretical framework that mirrors the RAG system's architecture and concerns, ensuring that evaluation is systematic and tightly aligned with the overall quality objectives of RAG.

RQ2: What combination of metrics, measurement strategies, and test case generation methods indicates RAG system effectiveness across different use cases?

We identified and implemented a comprehensive suite of metrics for capturing RAG component performance and interactions:

Retrieval Metrics: We demonstrated that combining traditional IR metrics (precision, recall) with modern neural approaches (semantic similarity, diversity) provides comprehensive retrieval evaluation. The addition of RAG-specific metrics like context coverage and entity recall addresses unique retrieval challenges in knowledge-intensive tasks.

Generation Metrics: Our implementation shows that effective generation evaluation requires multiple perspectives. Faithfulness metrics using claim extraction and entailment models capture grounding quality. Relevance metrics assess query-answer alignment. Completeness metrics ensure comprehensive responses. The combination provides a detailed generation quality assessment.

System-Level Metrics: We developed metrics for component interactions, including context utilization rate (measuring how effectively generation uses retrieved information) and multi-hop reasoning scores (assessing complex inference across documents). These metrics capture higher system properties that component-level metrics miss.

Test Case Generation: Our automated test generation strategy produces diverse question types targeting different system capabilities. The approach combines template-based generation for systematic coverage with LLM-based generation for natural variation. Caching prevents redundant generation while maintaining test diversity. The six question categories ensure comprehensive evaluation across varying complexity levels.

Measurement strategies: Our key strategy involved LLM-as-judge scoring for nuanced metrics. We crafted specific prompts for each metric and logged the LLM's rationales to help developers understand failures.

RQ3: How can evaluation results be systematically translated into practical optimization recommendations?

The GQM-based framework addresses this question through three key mechanisms that transform evaluation outputs into actionable improvement strategies:

Hierarchical Traceability and Diagnostic Pathways: The framework's four-layer architecture establishes bidirectional traceability between metrics and goals, enabling systematic diagnosis of performance issues. When metrics indicate poor performance, it can be traced upward to understand affected goals, then traced downward to identify specific components requiring optimization. For instance, in our experiments (see Chapter 6), Experiment 1's low Goal 3 score (35.1%) was attributed to inadequate edge case handling, particularly in scenarios where requested information was not retrieved while it was present in the knowledge base. This deficiency stems from limitations in the retrieval component, suggesting that enhancements to the retrieval system would address this specific weakness.

Visualisations: React UI provides analytics (radar charts, scatter plots, heatmaps) with actionable insight based on evaluation results. A goal-level **radar chart** gives an at-a-glance

profile of system performance across objectives, highlighting weakest areas. Drill-down **metric distributions** expose variability and outliers, revealing if issues occur consistently or only on certain queries. A strong correlation (visualized in a metric correlation heatmap) between retrieval metrics and answer metrics would signal that improving retrieval will yield better answers.

Component-Specific Metric Mapping: The framework explicitly links metrics to RAG system components, providing targeted optimization guidance:

- **Retrieval component metrics** (context precision/recall) - Document selection optimization
- **Generation metrics** (faithfulness/consistency) - Response synthesis improvements
- **System metrics** (context utilization rate) - Effectiveness of system component interactions enhancements

This mapping proved effective in practice: Experiment 1's low context precision scores indicated retrieval ranking issues, leading to targeted retrieval optimizations that helped to improve overall performance from 55.9% to 72.5% in Experiment 2.

Question Type Analysis for Targeted Improvements: The framework's six question categories enable precise identification of system weaknesses and help to guide granular analysis for specific improvements:

- Simple query failures - Fundamental retrieval issues
- Complex query failures - Multi-hop reasoning deficiencies
- Distracting query failures - Poor noise filtering

Practical Implementation Workflow: The framework supports a systematic improvement process: (1) identify failing goals through visualizations or provided goal/metric scores, (2) trace to specific underperforming questions, (3) analyze contributing metric patterns, (4) examine concrete test case failures, (5) implement component-specific optimizations, and (6) validate improvements through re-evaluation.

7.3 Framework Limitations

While the proposed framework has many benefits, it is important to critically examine its limitations. Some limitations arise from our methodological choices and experimental setup, while others are inherent challenges in RAG evaluation that our framework does not fully solve. We enumerate these limitations below, along with an analysis of their implications:

Limited Diversity and Potential Bias in Test Case Generation: One notable limitation of our evaluation approach is the use of the same large language model (Google's Gemini LLM, in our case) for both generating test cases and scoring the outputs (LLM-as-a-judge). Relying on a single model for both roles can introduce biases and reduce the diversity of the evaluation. Since the Gemini model was used to formulate questions, those questions may reflect the model's own "worldview" or areas of strength. For example, the LLM might generate questions that are phrased in a particular style or focus on certain facts it finds salient in

the documents, potentially overlooking other types of questions that users might ask. Consequently, our test set might not fully represent the spectrum of real user queries - it could be easier or harder in systematic ways.

Moreover, when the same model judges the answers, it may be overly sympathetic to answers that resemble its way of answering. In other words, the evaluation could be biased to favor the style or reasoning patterns of the Gemini model. If the RAG system uses a different model (say Llama-2 or GPT-4) as the generator, there's a risk that the Gemini judge doesn't fairly evaluate it - perhaps penalizing correct answers that are phrased differently or not catching errors that it would make. This shared-model bias is a concern for validity.

We tried to mitigate this by prompt engineering (instructing the judge to be critical and focus on factual consistency) and by manual spot-checking of some evaluations with human judgment. Indeed, we found generally that the LLM judge's scores correlated well with human assessments for obvious errors. However, subtle biases might remain; for instance, Gemini might be less knowledgeable in a niche domain, so it could fail to notice a factual error in that domain in the RAG output, giving a high score. Or it might rate an answer as incomplete because it expected a different detail, even if a user would be satisfied. This limitation means that our evaluation results, while useful, should not be taken as ground-truth in an absolute sense - they are filtered through the lens of a particular model. In future deployments, one should consider using multiple models or human review to counter this bias.

Additionally, the diversity and volume of test cases could be a limiting factor. Although we generated a substantial number of questions, the coverage of possible queries is inevitably partial. We focused on the domains present in our knowledge sources and on reasonably complex questions, but there may be whole categories of questions (e.g., extremely open-ended questions, or ones requiring logical puzzles) that were absent. If the system were used in a different context, our evaluation might not generalize. The test cases were also mostly academic or factoid in nature; user queries in the wild can be messy or contain irrelevant information, which we did not extensively test.

Moreover, the quality of LLM-generated questions, while generally high, might sometimes produce ambiguous or misleading questions. We observed a few instances where the question could be interpreted in multiple ways, and the "expected" answer from the LLM's perspective might differ from what a human might consider correct. If such a test case wasn't manually filtered out, it could lead to unfairly penalizing the RAG system or confusing results. We attempted to manually review and refine the test set for clarity and answer unambiguity. This limitation suggests that human oversight in curating or at least validating the test cases is important; fully automated test generation, as done here, is efficient but not infallible.

Dependence on LLM-Based Metrics and their Reliability: Our framework leans heavily on LLM-based evaluation metrics (e.g., for faithfulness, completeness, relevance). These metrics, being model-driven, have their limitations. One issue is consistency and reproducibility: LLMs can produce slightly different judgments if there's any randomness in the process or if the prompt is altered. We used fixed prompts and deterministic settings where possible, but minor prompt changes could potentially shift the scores. This means our evaluation might not be fully stable over time or across different LLM evaluators. We partially addressed this by calibrating the LLM outputs (for example, mapping qualitative judgments to numeric scores consistently), but some variance is inherent.

Another subtle limitation in metrics is incompleteness of metric coverage. Despite our broad set of metrics, there are still aspects of RAG performance we did not measure quantitatively. One such gap is explainability: while we measured if answers were grounded in sources, we did not fully evaluate how well the system explained or justified the answers to the user. For instance, does the system provide source citations in a user-friendly manner? Does it indicate its confidence or reasoning steps? These are important for user trust but are hard to capture in a simple metric. Our attribution metric touched on it (checking if sources are cited), but not the quality of explanations.

Similarly, user-centric measures such as user satisfaction, clarity, or helpfulness were not explicitly measured. We assumed that if an answer is correct, complete, and coherent, it will be helpful, but that might not always hold. These qualitative aspects often require human evaluation. Our focus was more on technical correctness and completeness, so there is a risk that a system might score well on our metrics but still not delight users.

Use of the Same Model in Multiple Roles: Expanding on the Gemini model issue, using the same LLM as both test generator and judge raises the possibility of evaluation leakage or circularity. For instance, Gemini might generate a question and already "know" the answer (since it formulated it). When judging the RAG system's answer, it might implicitly compare to the answer it had in mind. This is somewhat mitigated when the RAG system uses different knowledge or retrieval, but consider if Gemini itself was used as the RAG generator. Then we had the situation of a model judging its own answers (through the test it also created) - a highly self-referential scenario. This could lead to an overly optimistic evaluation. We tried to avoid same-model comparisons, but the risk of systemic bias remains.

Experimental Constraints and Pipeline Bottlenecks: On the experimental side, our evaluation was constrained by computation and cost considerations. Running LLM-based metrics for every test case can be slow. We had to limit the number of test cases and the number of variations for practical reasons. This means some statistical confidence is sacrificed. In a sense, we traded breadth of metrics for depth of test cases per metric. A more extensive study might run thousands of queries and average out more noise.

Focus on Technical Metrics over End-User Impact: A limitation to acknowledge is that our framework, in its current state, focuses on technical quality metrics rather than end-user impact metrics. We do not measure things like user satisfaction rating, task success rate, or other Human-Computer Interaction (HCI)-oriented outcomes. Those often require user studies. While our metrics are proxies for quality, they are not a substitute for real user feedback. Therefore, one limitation is that the framework should be seen as a tool for developers and evaluators, not as a direct measure of user happiness.

7.4 Future Work

While this thesis has laid a foundation for GQM-based RAG evaluation, there remain many exciting avenues to extend and improve the framework. Future work can address the current limitations, enhance automation and usability, and expand the evaluation of RAG system capabilities. We outline several concrete directions below:

Enhancing LLM-as-a-Judge Robustness

To address the limitation of relying on a single LLM evaluator, future work should explore ensemble approaches and human feedback integration for the judging process. An ensemble of LLM judges could involve using multiple models to evaluate each answer and then aggregating their scores. The rationale is that different models have different knowledge and biases, so agreement among them is more trustworthy, and major disagreements could flag uncertainty. For instance, if two judges say an answer is correct and one says it's not, the framework could mark that case as needing review or take the majority but note the dissent. This would improve reliability and perhaps reduce the influence of any one model's blind spots. Another approach is to incorporate human-in-the-loop at strategic points.

Improving Human-Readable Reporting and Visualization

While we provided some basic visualizations in this work, there is large room for more human-friendly reporting of evaluation results. Future efforts can develop a summary generator that translates raw metric findings into natural language insights. This kind of summary could be produced by an LLM or prompted on the metrics-to-text task.

Additionally, future work can add rich visual analytics. For example, interactive plots that allow filtering test cases (like "show me all queries where faithfulness < 0.5").

All these enhancements in reporting would make the framework more actionable and accessible to a broader audience, and possibly reveal patterns not obvious from raw data.

Full Automation of the Evaluation Pipeline

One future improvement is achieving full automation across all stages of the evaluation process. In our current setup, some steps need manual configuration or periodic intervention. For example, curating the initial test questions or interpreting the results to decide next actions. A fully automated pipeline would involve automatically updating the test suite and metrics as the system or its use cases evolve.

One approach is to incorporate a dynamic test case generation mechanism that continuously learns from real interactions: for instance, log user queries that the system struggled with, then automatically convert those into new test cases to add to the evaluation suite. An advanced pipeline could use anomaly detection on logs to flag unusual queries and then use an LLM to generate variations of those as tests.

Moreover, automation could extend to suggesting fixes: given the GQM diagnostic output, a future system might recommend specific improvements (like "increase vector dimension"). On the reporting side, full automation means that after every evaluation run, the framework could automatically produce a summary report or even trigger actions. Achieving this level of automation will likely require more sophisticated orchestration and robust handling of changes.

Incorporation Dialog-Level Evaluations

Extending the framework to handle multi-turn conversations is a natural next step, given that many RAG applications are conversational assistants. Our current evaluation treats queries

independently, but in a dialog, earlier interactions set context for later ones. Future work should develop metrics and test cases for dialog coherence, context retention, and adaptation. For example, one could introduce a goal like “Multi-Turn Consistency” with questions such as “Does the system remember facts provided by the user earlier in the conversation?”. Metrics to answer these could involve checking later answers against prior answers for contradictions or tracking whether entities introduced remain consistent.

Chapter 8

Conclusion

This thesis presented a comprehensive evaluation framework for Retrieval-Augmented Generation (RAG) systems based on the Goal-Question-Metric (GQM) methodology. By adapting established software engineering principles to AI system assessment, we addressed the critical gap in RAG evaluation, transforming metrics into a structured, actionable framework.

Our primary contribution is the successful adaptation of GQM methodology to RAG evaluation through a four-layer hierarchical architecture. We developed five high-level goals covering precision and truthfulness, relevance, edge case handling, coherence, and complex query support, each decomposed into specific questions and mapped to concrete metrics. The framework implements fifteen specialized metrics across retrieval, generation, and system-level categories, combining traditional NLP measures with modern LLM-based evaluation approaches.

The practical implementation in Python demonstrates the framework's viability, featuring automated test case generation across six query types, modular metric computation, and comprehensive visualization through an interactive dashboard. Our experiments on three different RAG configurations validated the framework's effectiveness, revealing that it successfully discriminates between systems and provides actionable diagnostic information.

The framework's diagnostic capabilities extend beyond simple scoring. Through hierarchical traceability, developers can navigate from high-level performance indicators to specific component failures. The component-specific metric mapping enables targeted optimization strategies, while comprehensive visualizations, including radar charts and heatmaps, make complex evaluation results accessible and actionable.

This work provides a systematic methodology that can be extended to other complex AI systems. It offers immediate value through diagnostic capabilities that guide optimization efforts based on quantitative evidence. By providing clearly defined goals and metrics, the framework contributes to standardization efforts in RAG assessment, enabling meaningful comparisons between systems and accelerating progress in the field.

While our framework advances RAG evaluation significantly, multiple areas require further investigation. Future work should address the potential bias from using a single LLM for both test generation and evaluation by implementing ensemble approaches with multiple models. Extending the framework to handle multi-turn conversations would capture dialog-level dynamics crucial for conversational AI applications. Additionally, incorporating direct

user satisfaction metrics would strengthen the connection between technical evaluation and real-world effectiveness.

The framework's modular architecture positions it well for evolution alongside RAG technology. As new retrieval methods, language models, and architectural patterns emerge, the GQM structure provides a stable foundation for incorporating new metrics and evaluation strategies while maintaining systematic rigor.

As RAG systems become integral to production AI applications, systematic evaluation becomes essential for trustworthy deployment. This thesis demonstrates that principled evaluation is achievable through careful adaptation of proven methodologies. The GQM-based framework provides the RAG community with a foundation for comprehensive assessment, enabling quantitative comparison and targeted improvement of these critical AI systems. By establishing clear relationships between high-level quality goals and low-level metrics, we enable developers to build more reliable and effective retrieval-augmented generation systems, advancing the field toward more trustworthy AI deployment.

List of Acronyms

AI Artificial Intelligence. 5, 11, 49–51

API Application Programming Interface. 26, 27, 51

BERT Bidirectional Encoder Representations from Transformers. 10, 51

BLEU Bilingual Evaluation Understudy. 10, 51

FAISS Facebook AI Similarity Search. 6, 51

GQM Goal-Question-Metric. 2–5, 8, 9, 17, 21, 23, 24, 27, 33, 34, 36, 39, 41–43, 46, 47, 49–51, 53

HCI Human-Computer Interaction. 46, 51

HHEM Hughes Hallucination Evaluation Model. 11, 26, 30, 51

HNSW Hierarchical Navigable Small World. 7, 51

IR Information Retrieval. 10, 41, 43, 51

IVF Inverted File Index. 7, 51

LLM Large Language Model. 1–3, 5, 7, 11, 19, 23, 25, 26, 28, 30–35, 41–47, 49, 51

MAP Mean Average Precision. 11, 51

MMR Maximal Marginal Relevance. 29, 51

MRR Mean Reciprocal Rank. 11, 51

NDCG Normalized Discounted Cumulative Gain. 11, 51

NLP Natural Language Processing. 1, 3–5, 25, 49, 51

QA Question Answering. 10, 11, 51

RAG Retrieval-Augmented Generation. 1–8, 10, 11, 17, 20, 21, 23–26, 33, 34, 36, 37, 39, 41–47, 49–51, 53, 54

ROUGE Recall-Oriented Understudy for Gisting Evaluation. 10, 51

List of Figures

2.1 Basic RAG system architecture illustrating the flow from user query through retrieval and generation components. 6
2.2 Example GQM hierarchy. 9

5.1 Flowchart diagram of the evaluation framework. 22
5.2 UML Component Diagram of the GQM-based RAG Evaluation Framework. . . 24

6.1 Goal attainment profile for the three experiments, shown as a radar chart. . . 35
6.2 Performance by Evaluation Goal for each Experiment (goal scores in %). . . . 35
6.3 Component-level performance comparison for Experiments 1, 2, and 3. Each group of bars shows the average score for Retrieval metrics, Generation metrics, and Overall System metrics outcomes. 36
6.4 Heatmap of metric performance by question type for Experiment 1-3. 38

List of Tables

5.1 Supported Test Case Types in Automated Generation 26

6.1 RAG Experiments and Model Configurations 33
6.2 Overall Score (aggregated performance) in each experiment 34
6.3 Average Component Performance Metrics by Experiment (%) 37

Bibliography

[1] Anastasios N Angelopoulos et al. "Prediction-powered inference". In: *arXiv preprint arXiv:2301.09633* (2023).

[2] Akari Asai et al. "Self-rag: Learning to retrieve, generate, and critique through self-reflection". In: *arXiv preprint arXiv:2310.11511* (2023).

[3] Yuntao Bai et al. "Constitutional ai: Harmlessness from ai feedback". In: *arXiv preprint arXiv:2212.08073* (2022).

[4] Victor Basili, Gianluigi Caldiera, and H Dieter Rombach. "Goal question metric paradigm". In: *Encyclopedia of software engineering*. Vol. 1. 1994, pp. 528–532.

[5] Victor R Basili, Gianluigi Caldiera, and H Dieter Rombach. "Software modeling and measurement: the Goal/Question/Metric paradigm". In: *Technical Report CS-TR-2956* (1992).

[6] Bernd Bohnet et al. "Attributed question answering: Evaluation and modeling for attributed large language models". In: *arXiv preprint arXiv:2212.08037* (2022).

[7] Lionel Briand, Christiane M Differding, and H Dieter Rombach. "Using the goal/question/metric paradigm to integrate measurement into object-oriented software development environments". In: *Information and Software Technology* 38.6 (1996), pp. 407–423.

[8] Hung-Ting Chen and Eunsol Chen. "Understanding retrieval augmentation for long-form question answering". In: *arXiv preprint arXiv:2211.14502* (2022).

[9] Jiawei Chen et al. "Benchmarking large language models in retrieval-augmented generation". In: *arXiv preprint arXiv:2309.01431* (2024).

[10] Wenhu Chen et al. "MuRAG: Multimodal retrieval-augmented generator for open question answering over images and text". In: *arXiv preprint arXiv:2210.02928* (2022).

[11] Confident AI. *DeepEval: The evaluation framework for LLMs*. `https://github.com/confident-ai/deepeval`. 2024.

[12] Zhuyun Dai et al. "Promptagator: Few-shot dense retrieval from 8 examples". In: *arXiv preprint arXiv:2209.11755* (2022).

[13] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. 2019, pp. 4171–4186.

[14] Shahul Es et al. "Ragas: Automated evaluation of retrieval augmented generation". In: *arXiv preprint arXiv:2309.15217* (2023).

[15] Jinlan Fu et al. “GPTScore: Evaluate as you desire”. In: *arXiv preprint arXiv:2302.04166* (2023).

[16] Tianyu Gao et al. “Enabling large language models to generate text with citations”. In: *arXiv preprint arXiv:2305.14627* (2023).

[17] Giskard AI. *Giskard RAG evaluation toolkit*. `https://docs.giskard.ai/en/latest/open_source/testset_generation/index.html`. 2024.

[18] Kelvin Guu et al. “Retrieval augmented language model pre-training”. In: *arXiv preprint arXiv:2002.08909* (2020).

[19] Pengcheng He et al. “DeBERTa: Decoding-enhanced BERT with Disentangled Attention”. In: *International Conference on Learning Representations*. 2021.

[20] Gregor Hohpe and Bobby Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Boston, MA: Addison-Wesley Professional, 2003.

[21] Or Honovich et al. “TRUE: Re-evaluating factual consistency evaluation”. In: *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 2022, pp. 3905–3920.

[22] Gautier Izacard et al. “Unsupervised dense information retrieval with contrastive learning”. In: *arXiv preprint arXiv:2112.09118* (2022).

[23] Kalervo Järvelin and Jaana Kekäläinen. “Cumulated gain-based evaluation of IR techniques”. In: *ACM Transactions on Information Systems (TOIS)* 20.4 (2002), pp. 422–446.

[24] Soyeong Jeong et al. “Adaptive-rag: Learning to adapt retrieval-augmented large language models through question complexity”. In: *arXiv preprint arXiv:2403.14403* (2024).

[25] Jeff Johnson, Matthijs Douze, and Hervé Jégou. “Billion-scale similarity search with gpus”. In: *IEEE Transactions on Big Data* 7.3 (2019), pp. 535–547.

[26] Vladimir 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)*. 2020, pp. 6769–6781.

[27] Omar Khattab, Christopher Potts, and Matei Zaharia. “Baleen: Robust multi-hop reasoning at scale via condensed retrieval”. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 27670–27682.

[28] Patrick Lewis et al. “Retrieval-augmented generation for knowledge-intensive nlp tasks”. In: *Advances in Neural Information Processing Systems*. Vol. 33. 2020, pp. 9459–9474.

[29] Chin-Yew Lin. “Rouge: A package for automatic evaluation of summaries”. In: *Text summarization branches out*. 2004, pp. 74–81.

[30] Derek Liu et al. “Evaluating the factual consistency of large language models through news summarization”. In: *arXiv preprint arXiv:2211.08412* (2023).

[31] Nelson F Liu et al. “Lost in the middle: How language models use long contexts”. In: *Transactions of the Association for Computational Linguistics* 12 (2024), pp. 157–173.

[32] Yang Liu et al. “G-eval: Nlg evaluation using gpt-4 with better human alignment”. In: *arXiv preprint arXiv:2303.16634* (2023).

[33] Yinhan Liu et al. “RoBERTa: A Robustly Optimized BERT Pretraining Approach”. In: *arXiv preprint arXiv:1907.11692* (2019).

[34] Yixuan Liu et al. “MultiHop-RAG: Benchmarking retrieval-augmented generation for multi-hop queries”. In: *arXiv preprint arXiv:2401.15391* (2024).

[35] Antoine Louis, Gerasimos van Dijck, and Gerasimos Spanakis. “Interpretable long-form legal question answering with retrieval-augmented large language models”. In: *arXiv preprint arXiv:2309.17050* (2024).

[36] Xueguang Ma et al. “Fine-tuning llama for multi-stage text retrieval”. In: *arXiv preprint arXiv:2310.08319* (2023).

[37] Yu A Malkov and Dmitry A Yashunin. “Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs”. In: *IEEE transactions on pattern analysis and machine intelligence* 42.4 (2018), pp. 824–836.

[38] Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to information retrieval*. Cambridge university press, 2008.

[39] Joshua Maynez et al. “On faithfulness and factuality in abstractive summarization”. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 2020, pp. 1906–1919.

[40] Jacob Menick et al. “Teaching language models to support answers with verified quotes”. In: *arXiv preprint arXiv:2203.11147* (2022).

[41] Sewon Min et al. “Factscore: Fine-grained atomic evaluation of factual precision in long form text generation”. In: *arXiv preprint arXiv:2305.14251* (2023).

[42] Sewon Min et al. “Silo language models: Isolating legal risk in a nonparametric datastore”. In: *arXiv preprint arXiv:2308.04430* (2023).

[43] Rodrigo Nogueira and Kyunghyun Cho. “Passage re-ranking with bert”. In: *arXiv preprint arXiv:1901.04085*. 2019.

[44] Baolin Peng et al. “Check your facts and try again: Improving large language models with external knowledge and automated feedback”. In: *arXiv preprint arXiv:2302.12813* (2023).

[45] Pranav Rajpurkar et al. “Squad: 100,000+ questions for machine comprehension of text”. In: *arXiv preprint arXiv:1606.05250* (2016).

[46] Ravi Raju et al. “Constructing domain-specific evaluation sets for LLM-as-a-judge”. In: *arXiv preprint arXiv:2408.08808* (2024).

[47] Hannah Rashkin et al. “Measuring attribution in natural language generation models”. In: *arXiv preprint arXiv:2112.12870* (2021).

[48] Nils Reimers and Iryna Gurevych. “Sentence-bert: Sentence embeddings using siamese bert-networks”. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. 2019, pp. 3982–3992.

[49] Jon Saad-Falcon et al. “ARES: An automated evaluation framework for retrieval-augmented generation systems”. In: *arXiv preprint arXiv:2311.09476* (2023).

[50] Thibault Sellam, Dipanjan Das, and Ankur P Parikh. “BLEURT: Learning Robust Metrics for Text Generation”. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 2020, pp. 7881–7892.

[51] Zhihong Shao et al. “Enhancing retrieval-augmented large language models with iterative retrieval-generation synergy”. In: *arXiv preprint arXiv:2305.15294* (2023).

[52] Nandan Thakur et al. “BEIR: A heterogeneous benchmark for zero-shot evaluation of information retrieval models”. In: *arXiv preprint arXiv:2104.08663* (2021).

[53] Romal Thoppilan et al. “LaMDA: Language models for dialog applications”. In: *arXiv preprint arXiv:2201.08239* (2022).

[54] Hugo Touvron et al. “Llama 2: Open foundation and fine-tuned chat models”. In: *arXiv preprint arXiv:2307.09288* (2023).

[55] Harsh Trivedi et al. “Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions”. In: *arXiv preprint arXiv:2212.10509* (2022).

[56] Axel Van Lamsweerde. *Goal-Oriented Requirements Engineering: A Guided Tour*. IEEE Computer Society, 2001.

[57] Rini Van Solingen and Egon Berghout. “The goal/question/metric method: a practical guide for quality improvement of software development”. In: (1999).

[58] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems*. Vol. 30. 2017.

[59] Vectara. *Hughes Hallucination Evaluation Model v2.1: Technical Report*. Tech. rep. Vectara Inc., 2024. URL: https://www.vectara.com/blog/hhem-2-1-a-better-hallucination-detection-model.

[60] Ellen M Voorhees. “The TREC-8 question answering track report”. In: *In Proceedings of TREC* 99 (1999), pp. 77–82.

[61] Shubham Yadav, Sarthak Ramesh, and Manish Joshi. “Hop-retriever: Retrieve hops over wikipedia to answer complex questions”. In: *arXiv preprint arXiv:2401.13472* (2024).

[62] Hao Yu et al. “Evaluation of retrieval-augmented generation: A survey”. In: *arXiv preprint arXiv:2405.07437* (2024).

[63] Tianyi Zhang et al. “Bertscore: Evaluating text generation with BERT”. In: *arXiv preprint arXiv:1904.09675* (2019).

[64] Lianmin Zheng et al. “Judging llm-as-a-judge with mt-bench and chatbot arena”. In: *arXiv preprint arXiv:2306.05685* (2023).