Universiteit Leiden

Evaluating RAG Frameworks for Industrial Applications: A Comparative Study of Performance and Applicability

Phi Pham (s3609987)

Supervisors:
Zhaochun Ren, Jujia Zhao & Mark Musters

Bachelor Thesis– Informatica & Economie

Leiden Institute of Advanced Computer Science (LIACS)
liacs.leidenuniv.nl

July 4, 2026

Abstract

Dutch housing corporations operate within a highly regulated environment, in which regulations are numerous and frequently changing. Consultants who support these organizations spend substantial time locating and verifying relevant regulatory passages to substantiate their advice. Retrieval-Augmented Generation (RAG) offers a potential solution to the problem of hallucinations by retrieving relevant documents and generating answers based on those documents. However, RAG systems can be implemented using paradigms of varying complexity, and it remains unclear which paradigm is most suitable for the housing corporation domain. In this thesis, three RAG paradigms will be compared: Naive, Advanced, and Modular. To ensure a fair comparison, all paradigms use the same embedding model, generator, and prompting strategy. As a result, only the RAG architecture differs. The selection of components is informed by a systematic literature review. Each paradigm is evaluated along four dimensions: answer quality, retrieval quality, latency, and implementation complexity.

The evaluation is conducted in two stages. First, the paradigms are assessed on the English-language RAGBench benchmark. Second, they are evaluated on a Dutch corpus containing documents relevant to the housing corporation sector. Answer quality is measured using RAGAS, with both 8B and 70B large language models (LLMs) as judges. A sample of answers is reviewed by a domain expert as a small sanity check, with mixed and inconclusive results.

In the implemented configurations and on the evaluated datasets, the three paradigms produce answers of comparable quality. The only significant differences appear in context precision, where the reranker in the Advanced paradigm selects a smaller and more relevant set of chunks. This higher precision does not result in a better final answer. Compared with the Naive pipeline, the Modular paradigm is about nine times slower on the Dutch corpus and fifteen times slower on RAGBench, with almost no gain in answer quality. The choice of generator has a larger effect on answer quality than the choice of paradigm. For this domain, the simpler pipeline suffices and is more user-friendly in terms of implementation. The added components of the more complex paradigms increase latency and complexity without a clear improvement in quality.

Contents

1 Introduction

This section introduces the problem addressed in this thesis.

1.1 The situation

Dutch housing corporations operate in a highly regulated environment, in which the rules are numerous and frequently change. Laws and regulations, such as the Woningwet, the Aedes processenboek, the Baseline Informatiebeveiliging Corporaties (BIC), the CORA/VERA reference architecture and NIS2, affect their day-to-day operations. These documents have a high volume, can change rapidly, and reference each other. The corporations themselves, along with the consultants who support them, spend considerable time locating the relevant passages in multiple documents to substantiate their operations and advice.

VVA is an IT consultancy that operates in this sector, and they encounter this information access problem on a daily basis. Their consultants handle questions and projects about policy, governance and information security. They require answers that comply with the law and regulations, and that can be traced back to them. Manual lookup is slow, as the passages are fragmented across multiple sources and depend on experts to verify them. A system that retrieves the relevant passages and generates an answer would reduce the workload and improve consistency across the consultants and the corporation.

1.2 Retrieval-Augmented Generation

Retrieval-Augmented Generation (RAG) combines a retrieval component, which selects passages from a collection of documents, with a generative model that generates an answer based on those passages [10]. This system allows the large language model (LLM) to gather information from relevant passages in the document collection that are not included in the LLM's training data, which is essential when working with domain-specific data.

An LLM could be applied to this domain in other ways. Fine-tuning the model on the regulatory corpus embeds the knowledge in the parameters. However, it requires retraining when a law or policy changes. Loading all relevant documents in a context window avoids retrieval altogether [3]. However, volume becomes an issue when combining the Woningwet, the BIC, and related documents. It exceeds the context limits, and this approach does not provide a link that can be traced back from the answer to the passage it is based on. RAG is preferred in this case as it keeps the knowledge base external and updatable.

RAG reduces but does not eliminate hallucination. It can still generate incorrect answers [20]. Retrieval quality is sensitive to the choice of chunking strategy and query formulation [10]. Additionally, a misleading passage may hurt performance more than an absent one [29]. Latency can also become a bottleneck in production settings, as real-time retrieval adds time to every query [3]. Evaluating RAG systems is itself an open problem, as reference-free metrics rely on LLMs as judges and inherit their unreliability [7, 20].

These limitations make the choice between different RAG paradigms non-trivial. A Naive pipeline may be enough for some queries, but fail on others, while more elaborate paradigms add more

components that might help or lead to additional failures. RAG systems are commonly grouped into paradigms of increasing complexity [10, 22]. This leads to the research question of this thesis:

> *How do different RAG paradigms compare in terms of performance and applicability for industry-specific use?*

Two terms in this question are defined here. Performance refers to the quality and speed of the answers a paradigm produces. It covers three aspects: answer quality, retrieval quality, and latency. Answer quality is measured through faithfulness, answer relevance, and answer correctness. Retrieval quality is measured through context precision and context recall. Latency is the end-to-end time taken to answer a query. Applicability refers to the effort needed to deploy and maintain a paradigm in an industrial setting. It is captured through implementation complexity, which includes the required hardware and maintenance effort. In this thesis, the paradigms on these dimensions are evaluated, as shown in Section 4.1.

To answer this, the thesis compares three paradigms: Naive, Advanced, and Modular, under controlled conditions.

1.3 Thesis overview

The thesis is structured as follows. Section 2 provides the background and discusses the related work. It is organized around benchmarks, architectural components, and evaluation methods. Section 3 describes the systematic literature review, on which design choices are based. Section 4 presents the experimental setup, the experiments, and their results. Section 5 includes a conclusion and outlines directions for further research.

This work is conducted as a bachelor thesis at the Leiden Institute of Advanced Computer Science (LIACS), Leiden University, under the supervision of Zhaochun Ren, Jujia Zhao, and Mark Musters.

2 Background and Related Work

2.1 From Large Language Models to Retrieval-Augmented Generation

A large language model (LLM) is a neural network trained on large amounts of text to predict the next token in a sequence [17]. During the training phase, the model stores patterns and facts from its training data in its parameters. Therefore, the model is able to answer questions and produce text based solely on its knowledge. However, this way of storing knowledge has two drawbacks that are relevant to this thesis. Firstly, the knowledge is fixed after the training period, so when a regulation or policy changes, the model cannot incorporate the change and must be retrained. Secondly, the model generates its answers from its parameters without a link to the source. As a result, it is difficult to verify the answers, and it is likely to hallucinate. The model can produce claims that sound plausible, but are not supported [10].

Retrieval-Augmented Generation (RAG) addresses these limitations by keeping the knowledge base separate from the model [15]. The system generates an answer from the retrieved passages

instead of relying on its training data [10]. The knowledge base stays external and can thus be updated without retraining, and the answer can be traced back to the passages it used. These two properties, updatability and traceability, are why RAG is suitable for a domain like the Dutch housing corporation sector, where the documents have a high volume, change over time, and require answers that can be substantiated.

2.2 The RAG pipeline

There are two phases to a RAG pipeline. The first is indexing, which occurs only once at the beginning of a process. During indexing, the documents are loaded and then split into smaller pieces called chunks. Without splitting the documents, they are likely too long for the embedding model or the model's context window. The text can be split in multiple ways, for example on a fixed size or on sentence boundaries. The way the text is split will impact the information that is retrieved [10]. Each chunk is then turned into a dense vector by an embedding model, which places texts with similar meaning close to each other in the vector space [12]. The embedding model in this thesis is BGE-M3, a multilingual model that also supports Dutch [4]. The resulting vectors are stored in a vector store that supports nearest-neighbor search. The second phase is querying, and this happens for every question. The query is embedded with the same model and compared to the stored vectors. Afterwards, the top-k most similar chunks are returned. This is called dense retrieval. A different approach is sparse retrieval, where a method such as BM25 ranks chunks based on the exact overlapping of words instead of on their meanings [19]. The two can be combined into a hybrid setup, where the separate rankings are merged with a fusion method, such as Reciprocal Rank Fusion (RRF) [6]. The retrieved chunks can also be reordered by a reranker, which gives a score to each chunk against the query more precisely than the embedding model and keeps only the best matched ones.

In the last step, the retrieved chunks are combined with the query in a prompt. Next, the LLM uses this to generate the answer. The chunker, the embedding model, the retriever, the reranker, and the generator, form the building blocks of a RAG system. The paradigms described in the next section differ by how each component is used and arranged.

2.3 The RAG paradigms

RAG systems are usually described as a set of paradigms that increase in complexity [10]. The simplest is Naive RAG. It follows the basic process from the previous section. The query is embedded, and the most similar chunks are retrieved using dense retrieval. These chunks are then passed to the LLM to generate the answer. It uses the least amount of components and serves as the baseline. Advanced RAG improves on this by adding a post-retrieval component called the reranker. The retrieved chunks are reranked so that the LLM receives a smaller and more relevant portion of the documents. Modular RAG goes a step further by introducing new module types, such as query transformation and hybrid retrieval. Examples would be using dense and sparse retrieval merged through a fusion step, adding a routing step that selects a retrieval path per query, or inserting a query transformation such as HyDE [9]. A fourth paradigm is Agentic RAG, where an LLM agent controls the process by deciding when and what to retrieve, and can repeat the retrieval over several steps [22]. This adds the most flexibility, but also the most components and

the highest latency. This thesis will focus on comparing the first three paradigms, Naive, Advanced, and Modular, under a unified setup, while the Agentic paradigm is left for future work, as discussed in Section 5.

2.4 Evaluating RAG

Evaluating a standalone LLM is mainly a question of generation quality. For example, if the output is correct and fluent. Evaluating a RAG paradigm is more complicated, because the quality of the answer depends on both the retrieval and the generation [7]. The system consists of two phases. The first phase is retrieval. The question here is whether the system found the relevant passages and filtered out the irrelevant ones. The second phase is generation. Here it is about whether the answer is grounded in the retrieved passages and free of hallucination. These are two distinct phases of the system. A paradigm can retrieve the right chunks, but it can still generate a wrong answer or a correct answer from chunks that are not relevant. An evaluation therefore has to consider both parts, not only the final output. General LLM benchmarks, such as MMLU [11], do not capture this. They test the knowledge stored in the model, but not whether a retrieval stage has supplied the right context or whether the answer has stayed faithful to it. RAG evaluation, therefore, relies on a set of metrics that assess retrieval quality and generation quality separately, rather than a single score [7]. Another difficulty is the reference. There are metrics that need a ground-truth answer to compare against, while others can be computed without one. The first one is called reference-based, and the second is reference-free [7, 20]. The choice depends on whether labeled answers are available.

The most common metrics target three aspects: faithfulness, answer relevance, and context relevance [7]. Faithfulness measures how much of the answer is supported by the retrieved chunks, so a low score indicates hallucination. Context relevance measures whether the retrieved chunks are relevant to the query, and answer relevance determines whether the answer addresses the query. For the retrieval specifically, context precision and context recall measure how much of the retrieved context is relevant and how much of the relevant information was actually retrieved [25]. Answer correctness is included as a metric, a reference-based metric that combines a factual-correctness component and a semantic-similarity component in a 0.75/0.25 weighting [24, 26]. This metric is deprecated in the RAGAS version used (0.4.3) and is scheduled for removal in v1.0. The metric is retained as it is the only measure of the final answer's correctness against the ground truth. Faithfulness, answer relevance, and context relevance can be computed reference-free, whereas context precision, context recall, and answer correctness require a ground-truth answer and are thus reference-based.

Many of these metrics are computed with an LLM as a judge, where a separate model scores the answer against the query and the context. RAGAS is a widely used framework that follows this approach [7]. A drawback is that the scores inherit the biases and limitations of the judge model, which makes the choice of judge an open problem. How this affects the present study and which judge is used are discussed in Section 4.1.

2.5 Related Work

In this section, the existing body of literature will be analyzed. A Systematic Literature Review (described in Section 3) resulted in twelve core papers that form the basis of this comparative study.

The themes of the papers are: (1) benchmarks for RAG evaluation, (2) models and architectural components, and (3) evaluation methods. Each theme is further explained below.

2.5.1 Benchmarks for RAG Evaluation

The literature presents an overview of RAG benchmarks. It ranges from general purpose questions to domain-specific corpora. Table 13 summarizes the benchmarks, the domains, and the metrics on which they were evaluated.

There are benchmarks that target the enterprise context. RAGBench [8] offers 100,000 examples from five industry domains sourced from real world corpora. The corpora includes documents such as user manuals. HERB [5] simulates an enterprise environment in various stages. The documents are from the planning, development, and deployment stages. RARE [30] includes financial, economic, and policy-related documents. These benchmarks are relevant to the study's scope as their domains are similar to the housing corporation sector.

The other benchmarks evaluate different settings. GraphRAG-bench [28] explores when graph structure is beneficial to retrieval, while TempRAGEval [23] tests time-sensitive question answering. A correct answer depends on a particular time frame, and retrieval depends on temporal reasoning. RAGuard [29] evaluates robustness against misleading retrievals. It shows that when the retrieved content is misleading, the RAG systems can perform worse than zero-shot baselines.

2.5.2 Models and Architectural Components

Several open-source models appear repeatedly across the reviewed papers. Tables 14 and 15 provide an overview of the generator and retrieval components identified.

For the text generation, Llama 3.x [2, 23], Mistral [2] and Qwen 2.5 [21] are the most frequently used models in the analyzed literature. Cao et al. [2] evaluated eleven LLMs ranging from 1B to 123B parameters and found that small models could be equally robust to retrieval noise as the larger ones. Nonetheless, the larger models tended to achieve higher overall accuracy.

For retrieval, BM25 serves as a sparse baseline, while the BGE-family provides dense encoders for performing semantic search. The trend across the literature is to use a hybrid approach by combining both in the setup [5], which often outperforms the components if they are used separately. SR-RAG [27] shows that 7B class models trained to choose between external and internal knowledge can reduce retrieval calls by 21-40%, while improving accuracy by 2.1-8.5% across three different LLMs. This shows that selective retrieval is a viable option for more efficient RAG pipelines. A complete overview of generator models and retrieval components found in the literature is provided in Appendix A.

2.5.3 Evaluation Methodologies

RAG evaluation is continuously growing and being actively researched. RAGAS is one of the most widely used automated evaluation frameworks [7]. It uses an LLM to score faithfulness, context relevance, and answer relevance. Across the reviewed literature, there is no standard methodology, and most authors introduce their own benchmark-specific metrics. The authors of RAGBench [8] find that a finetuned DeBERTa-large classifier outperforms LLM-based judges on the same

task, suggesting that smaller models can be more accurate. Sardana [20] additionally benchmarks five reference-free hallucination detectors and finds that detection performance is strongly domain dependent. The choice of evaluation tool is therefore non-trivial and is discussed further in Section 4.1.

2.5.4 Synthesis and Identified Gaps

Three gaps are identified based on the reviewed literature. Firstly, while industrial benchmarks exist [8, 5], none exist for the Dutch housing corporation domain. The documents within this domain use legal terminology and contain policy updates. Secondly, the RAG paradigms identified by Gao et al. [10] and Singh et al. [22] are rarely compared head-to-head within a consistent framework. Lyu et al. [16] suggest that a well-configured minimal pipeline can match or exceed Agentic systems, while HERB [5] identifies retrieval as the primary bottleneck regardless of the paradigm. These conflicting findings indicate the need for a systematic comparison. This thesis compares three of these paradigms (Naive, Advanced, and Modular) under a unified framework. The Agentic paradigm is left to future work (Section 5).

Thirdly, implementation complexity is rarely considered an evaluation dimension in the literature. Faithfulness and relevance dominate [7], while latency is only addressed by a few papers [27, 3]. For organizations that deploy RAG, the effort required to implement and maintain each paradigm matters as much as answer quality and latency. This thesis addresses the gap by treating implementation complexity as a fourth evaluation dimension alongside faithfulness, relevance, and latency.

3 Systematic Literature Review

This section describes the Systematic Literature Review (SLR) on which the design choices are based. The review was conducted following the guidelines of Kitchenham and Charters [13], and it identifies the relevant literature on RAG framework evaluation. The resulting components and configurations are specified in the experimental setup (Section 4.1).

3.1 Search Strategy

The literature was obtained from arXiv, and it was confirmed as a suitable database during a supervisor review. It was due to the fast-growing nature of RAG research, where relevant work is often posted as preprints before formal publication. Literature was searched on arXiv on 16 March 2026, using a query for RAG evaluation, benchmarking, and industrial applications. The full query is provided in Appendix B. The search resulted in 165 papers that were screened against the criteria below.

3.2 Inclusion and Exclusion Criteria

The following criteria were applied to the literature to ensure that only relevant papers were included. A paper was included if all the requirements were met:

- **I1:** The paper is related to Retrieval-Augmented Generation or its evaluation.

- **I2:** The paper addresses at least one of the four paradigms: Naive, Advanced, Modular or Agentic RAG.
- **I3:** The paper addresses evaluation methods, benchmarks or experimental results.
- **I4:** The paper is published in 2020 or later.
- **I5:** The paper is written in English.

A paper was excluded if any of the following criteria were applied:

- **E1:** The paper is not related to RAG or large language model retrieval.
- **E2:** The paper focuses exclusively on multimodal RAG with no text-based evaluation.
- **E3:** The paper is a duplicate of an already included study.
- **E4:** The paper does not describe its methodology in enough detail to evaluate the validity of its findings.
- **E5:** The paper is not fully accessible.

3.3 Screening Process

The screening was conducted in three stages. In the first stage, the titles were reviewed to assess the relevance of the paper against the inclusion and exclusion criteria. The second stage consisted of a more in-depth screening, where the abstract was assessed using the criteria. When the abstract was insufficient to conclude an answer, the introduction was consulted in combination with a global scan of the paper. Full-text screening was not required, as confirmed during a supervisory review. In addition to the search on arXiv, backward snowballing was performed on the included papers. Their reference lists were scanned for relevant work that the query might have missed. This step was selective rather than exhaustive, and it did not result in new literature. The papers were either already captured by the search or did not meet the inclusion criteria, so the final included set is entirely from the database search. In the last stage, the remaining papers were reviewed and a final set of twelve core papers was selected based on relevance to the research question and coverage of the four paradigms. The screening process is visualized in Figure 1.

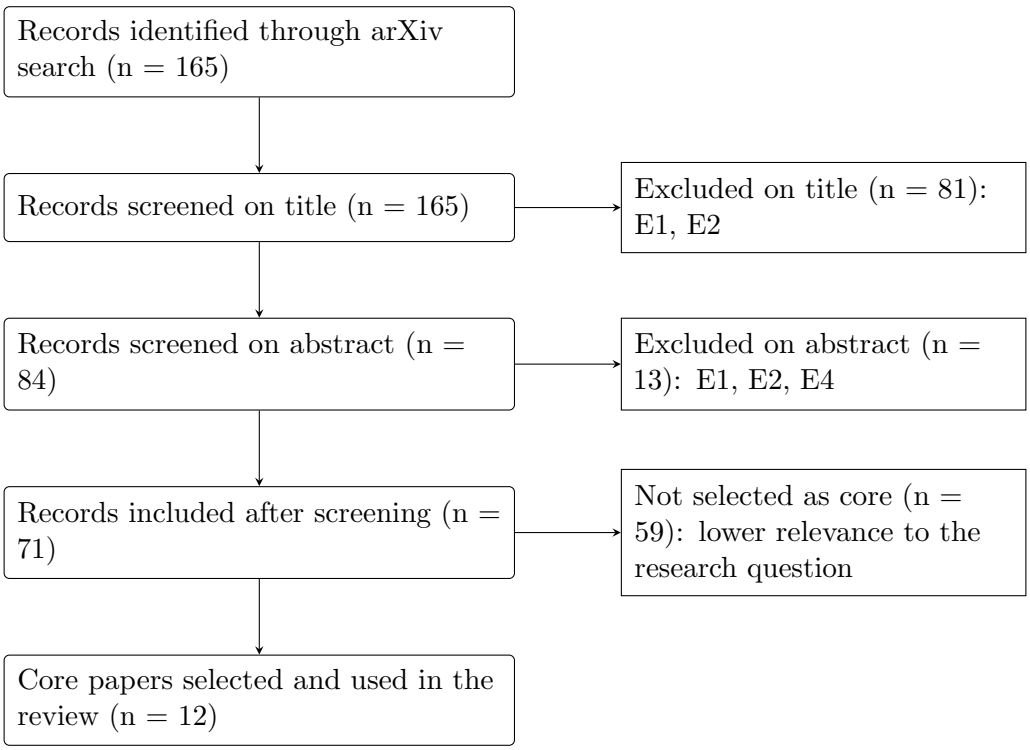

Figure 1: PRISMA flow diagram of the systematic literature review process.

3.4 Citation scope

Beyond the twelve core papers, this thesis cites a number of references that were not retrieved by the systematic literature review and are therefore not part of the PRISMA chart. Various papers are cited to define the concepts, methods and paradigms which form the foundation of this study. These papers fall outside the criteria defined in Section 3.2, but are necessary to support the literature and reproducibility. The original RAG formulation [15], dense passage retrieval [12], BM25 [19], reciprocal rank fusion [6], HyDE [9], the surveys that establish the RAG paradigms [10, 22], and the question taxonomy used to categorize the Dutch evaluation set [14]. The following citations document the models and software used in the experiment. The BGE-M3 embedding model [4] and the RAGAS evaluation framework [7]. The review methodology follows the guidelines of Kitchenham and Charters [13].

4 Experiments

In this thesis, three RAG paradigms of increasing complexity are implemented and compared in a single controlled setup. The paradigms Naive, Advanced, and Modular share several components. The embedding model, generator, and prompt are consistent across the three paradigms, so that any difference in the output can be traced back to the paradigm rather than the components. This section presents the experiments. It first describes the experimental setup: the shared infrastructure, the paradigm implementations, and the evaluation process. This is followed by the results of the two evaluation phases, first on the English RAGBench benchmark (Section 4.2) and then on the

Dutch corpus (Section 4.3), and finally by a comparison between the paradigms across all evaluation dimensions (Section 4.6).

4.1 Experimental Setup

The systematic review served as the basis for the component selection. It is based on which embedding models, retrievers, and rerankers are repeated in the reviewed literature. The papers that originally introduce these components are then cited to define them. These papers were not part of the screening itself, as several did not adhere to the criteria defined in Section 3.2. Each component, model, configuration, and dataset is selected, because it was the most commonly used or best performing option in the reviewed papers. Furthermore, it also did not violate the constraints of this study: open-source availability, multilingual for Dutch, and compatibility with local or organizational hardware.

4.1.1 System Overview

All three paradigms share the same infrastructure and differ only in their retrieval and orchestration logic. Figure 2 shows the overall design of the system. The pipeline runs in two phases. In the indexing phase, the documents are loaded, split into chunks, embedded with BGE-M3, and stored in a Qdrant vector store. This phase runs once in the beginning and is identical for every paradigm. In the querying phase, the query is processed in a paradigm-specific way. Naive retrieves the top 5 chunks through dense retrieval. Advanced retrieves the top 10 chunks in the same manner, but reranks them to a top 3 before generation. Modular rewrites the query with HyDE, does hybrid retrieval through dense and sparse methods, fuses the two rankings with reciprocal rank fusion (RRF), and reranks to the top 3. All three paradigms use the same BGE-M3 embedding model and the same Llama 3.1 8B generator. This way, any differences in the output can be traced back to the paradigm rather than the components. The individual components are described below, and the configuration of each paradigm is summarized in Table 1.

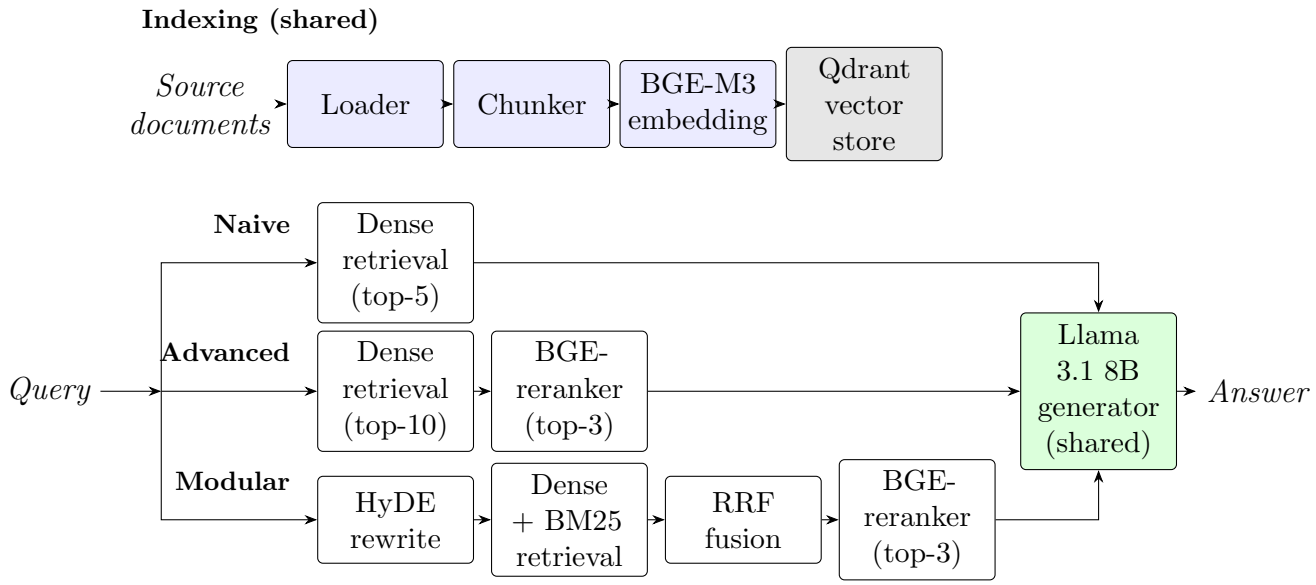

Figure 2: System overview of the shared infrastructure and the three RAG paradigm pipelines.

4.1.2 Infrastructure

Firstly, a shared infrastructure is build to ensure that all paradigms run on the same infrastructure, in order to make a fair comparison. The infrastructure consists of:

- **Corpus loader:** loads the pre-processed source documents. Word documents are converted to Markdown with markitdown beforehand, and the loader reads the resulting Markdown and plain-text files.
- **Chunker:** splits documents into chunks on sentence boundaries with a fixed size.
- **BGE-M3 embedding model:** encodes both chunks and queries into dense vectors. BGE-M3 is multilingual, supports Dutch, and is shared across all three paradigms.
- **Qdrant vector store:** stores the embedded chunks and supports nearest-neighbor search. Qdrant is chosen because it is open-source and locally hostable.
- **RAGAS harness:** an automated evaluation system using RAGAS that scores faithfulness, context relevance and answer relevance based on an LLM as judge. The complete metric set per phase is described in Section 4.1.7.
- **Latency timer:** measures end-to-end and per-stage timing for each query.
- **Llama 3.1 8B Instruct:** is the main generation model for all three paradigms. The Llama family is the most frequently used open-source LLM in the reviewed papers. The 8B variant is chosen as it satisfies the available hardware while supporting Dutch text generation.

The Naive paradigm is implemented first, after which the remaining two paradigms reuse the same infrastructure with paradigm-specific components.

4.1.3 Paradigm Implementations

The three paradigms are implemented as separate pipelines that share the same LLM and infrastructure but differ in their retrieval and orchestration logic. All three are build in LlamaIndex. Table 1 summarizes the configurations. The full set of hyperparameters is listed in Appendix C.

Table 1: Component configuration per RAG paradigm.

Paradigm	Configuration
Naive	BGE-M3 multilingual embedding for retrieval, top-5 passages passed to Llama 3.1 8B for generation. Establishes the baseline.
Advanced	Adds the BGE-reranker-v2-m3 cross-encoder, which re-ranks the top-10 retrieved passages down to the top-3 before generation.
Modular	Adds HyDE query rewriting before retrieval, and combines dense (BGE-M3) and sparse (BM25) retrieval through Reciprocal Rank Fusion (RRF), followed by the BGE-reranker.

The choices of components reflect the SLR findings. BGE-family (embedding and reranker) are chosen because they are open-source and support Dutch through their multilingual training. This sets them apart from most encoders in the reviewed papers. BM25 remains the standard sparse baseline. The Agentic paradigm is not implemented and is left to future work (Section 5).

4.1.4 Phase 1 Validation on RAGBench

Before evaluating the paradigms on the Dutch corpus, all three pipelines are first validated on RAGBench [8]. It is an English benchmark of 100,000 examples sourced from documents, such as user manuals across five industry domains. This is publicly available and provides reference answers. This allows RAGAS to score the full metric set including context precision, context recall and answer correctness. The Dutch evaluation set also provides references, but those are generated through DRAGON (Section 4.1.5) and not taken from a benchmark or labeled by humans. Validating on RAGBench first will ensure that each pipeline produces the correct output before introducing domain-specific data.

4.1.5 Phase 2 Dutch Corpus Construction

A Dutch evaluation set is constructed from Dutch housing corporation documents. The corpus combines policy and regulatory documents, including the Woningwet, Aedes processenboek, Baseline Informatiebeveiliging Corporaties (BIC 4.0), NIS2 guidelines and CORA/VERA reference architectures. Most of these are publicly available. Additionally along these documents a small number of VVA specific documents (where access permits.) are included in the corpus.

The evaluation corpus is generated using the data generation method from DRAGON [21], which is an automated data generation method for domain-specific retrieval. The generation method is used on the Dutch corpus and the accompanying benchmark DRAGONBench is not included in this study. The data generation pipeline is automated and generates question-answer pairs together with

sentence-level provenance, so that each output can be traced back to the specific source sentence on which the answer based. A reproducible test set is generated without any manual annotation. The generated test set and the evaluation process together form the Phase 2 benchmark. Because the questions and answers are synthetic, they are validated by a domain expert afterwards (Section 4.1.8).

Every query in the generated set is based on a single document, and each answer maps to one provenance passage from that source. No answer therefore combines information from more than one document. This follows from the single-hop generation approach and the extractive nature of the set. When the supporting passage can be retrieved directly, the additional retrieval components of Advanced and Modular have little room to improve the answer (Section 5.1). The single passage generation also limits how far the findings transfer over to questions that require reasoning across multiple sources (Section 5.2).

The queries are further categorized by type, following the taxonomy of Kolomiyets and Moens [14]. In this corpus, four types were found. In the taxonomy reason or purpose is mapped to the causal category and yes/no to confirmation. Factoid and definition keep their names. The taxonomy also defines list, procedural and other types that do not occur in the Dutch set. Each query is classified by hand. Table 2 reports the distribution of the generated queries. Factoid questions dominate the set and make up just over half of the queries.

Table 2: Distribution of the four query types over the 55 Dutch queries, with a short description and an example of each.

Type	**Description and example**	**n**	**%**
Factoid	Asks for a single fact such as a name, number, date, or retention period. Example: 'Hoe lang worden weekstaten van werknemers bewaard?'	30	54.5
Reason or purpose	Asks why something is the case or what its goal is. Example: "Wat is het doel van een calamiteitenplan?"	11	20.0
Yes/no	A question answered by confirming or denying a statement. Example: "Moeten voedingskabels en kabels voor het versturen van gegevens of die informatiediensten ondersteunen worden beschermd?"	8	14.5
Definition	Asks for the meaning of a term or concept. Example: "Wat is Escrow?"	6	10.9

4.1.6 Phase 2 Runs

All the paradigms are run on the Phase 2 using the same prompt template, embedding model, and generator. The difference between them lies in the retrieval (including retrieval depth) and orchestration logic, as summarized in Table 1. Keeping the components constant ensures that performance differences can be traced back to the paradigm design rather than to differences in configuration.

4.1.7 Automated Evaluation

Each paradigm is evaluated on four dimensions introduced in Section 2.5.4: faithfulness, relevance, latency, and implementation complexity. Three dimensions are automatically measured (faithfulness, relevance) using RAGAS for quality and a Python-based timer for performance (latency). The fourth metric measured is implementation complexity, which is measured separately and listed in Section 4.1.9. Faithfulness and relevance are reference-free and are computed in every run. When ground-truth answers and provenance are available, RAGAS also reports the reference-based metrics. These metrics are context precision, context recall and answer correctness. This applies to the Phase 1 RAGBench runs, where reference answers are available. It also applies to Phase 2 runs, where the data allows.

- **Faithfulness:** the proportion of claims in the generated answer that are supported by the retrieved context. A low score indicates hallucination.
- **Relevance:** a combined assessment of whether the retrieved context is relevant to the query (context relevance) and whether the generated answer addresses the query (answer relevance).
- **Context precision and context recall:** retrieval quality metrics that measure, respectively, how much of the retrieved context is relevant and how much of the relevant information was retrieved. Both require ground-truth provenance.
- **Answer correctness:** the agreement between the generated answer and a ground-truth reference answer. Reported only where reference answers are available.
- **Latency:** measured per query as the median (p50) and the 95th percentile (p95). The p95 captures worst-case time. An important point for production feasibility.

4.1.8 Expert Evaluation

The automated metrics are validated by human experts. VVA acts as an expert and carries out the validation. Two raters from VVA were asked to take part in the validation, so an inter-rater validation could be reported alongside the automated validation. This design does carry a conflict of interest as both raters work at the organization that would deploy this system. Hence, the validation should be seen as expert validation, rather than as an independent benchmark.

A sample of six queries is drawn from the Phase 2 set. The six are selected to cover results from both ends of the spectrum. Questions that were rated high and thus successful, and rated low, meaning failures, so the complete behavior can be validated by the experts. Each query is assessed for all paradigms, giving a total of 18 items. The items are in a randomized order, with the paradigm hidden, and are presented to the raters through a survey. A separate key map was made so the item could be traced back to the paradigm for analysis. Each item is scored on three dimensions: groundedness, correctness, and relevance on an ordinal scale.

4.1.9 Implementation Complexity

A fourth evaluation dimension is introduced to fill the gap that is identified in Section 2.5.4: implementation complexity. For an organization deciding whether to deploy a paradigm, complexity includes technical as well as operation concerns. Examples of the latter are how much effort it

takes to build, who can build it, and what it costs to keep running. Complexity therefore cannot be measured in a single objective number, and as a result it is documented per paradigm through technical and practical proxies.

Technical proxies:

- **Lines of code:** the lines of code in each paradigm as a rough proxy for development effort.
- **Number of components:** the amount of unique paradigm operations (retriever, reranker, fusion, query transformation, generator), indicating the technical and maintenance level.
- **Hardware requirements:** the models that must be loaded and the device they run on.
- **Tunable parameters:** the amount of configuration settings. For example retrieval depth or rerank cut-off.

Practical proxies (low, medium, high):

- **Estimated implementation time:** amount of time it takes to build and implement the paradigm on top of the shared infrastructure.
- **Required technical expertise:** the technical expertise required to understand, implement, and run the paradigm.
- **Maintenance burden:** the difficulty of keeping the paradigm running as the corpus and dependencies change.

The technical proxies are counted directly from the three pipelines. The practical proxies and the estimated implementation time is judged by the author. The required expertise and the maintenance burden are scored by a domain expert at VVA through an interview. Each level is defined in advance to ensure that a rating does not depend on a single overall impression. The expert assessed the paradigms from the perspective of feasibility and transferability to colleagues, rather than as a technical expert. The expert is employed at the organization and operates within the branch that would deploy the system. The resulting limitations are discussed alongside the results in Section 4.5.

4.1.10 Cross-Paradigm Comparison

The individual paradigm scores from the automated evaluation (Section 4.1.7), the expert evaluation (Section 4.1.8) and the implementation-complexity assessment (Section 4.1.9) are combined into a comparison table per paradigm. The trade-offs with the four dimensions are analyzed. Where a more complex paradigm improves quality, where it decreases, and what the cost is in terms of latency and implementation complexity.

Differences between the paradigms are tested per metric within each subset. The scores are paired by query, as every paradigm is run on the same queries. The paired t-test is used as the primary test. The result of the metrics are between [0,1] and often cluster at the extremes, so the per-query differences are not exactly normally distributed. With 55 to 100 paired observations, the test is robust to this through the central limit theorem. Alongside each test, the raw mean difference between the paradigm is reported to show the size of the differences on the scale of the metric.

Table 3: Baseline levels for the two expert-rated practical proxies, required expertise and maintenance burden.

Proxy	Low	Medium	High
Required expertise	A general developer can build and run the paradigm from the documentation.	The developer must understand RAG principles and which components it contains, but not the underlying algorithms.	The developer must know in detail how the pipeline works, which algorithms it uses, and why.
Maintenance burden	The pipeline runs as delivered and needs only an occasional fix when something breaks.	The pipeline needs periodic model and dependency updates, and the maintainer must know which component to update.	The pipeline must be checked regularly to confirm that every component still works and is current, as components evolve and older ones are phased out.

4.2 Phase 1: Validation on RAGBench

This section reports the Phase 1 validation results. RAGBench is used to confirm that the three pipelines run correctly on an established benchmark before the domain-specific evaluation. Since RAGBench provides reference answers, all RAGAS metric set is computed, including the reference-based metrics (context precision, context recall, and answer correctness).

4.2.1 Overall scores

Each paradigm is run on the five RAGBench subsets with 100 queries each, totaling 500 queries per paradigm. The generated answers are scored twice: once with the 8B judge, which is the same model as the pipeline generator, and once with the 70B judge. Both judges score the identical generated answers, so any difference between the two judges can be attributed to the judge model and not the pipeline. Table 4 reports the mean scores per paradigm, averaged over the five subsets. The full per-subset scores are given in Appendix D.1.

The three paradigms are close to each other on every metric. Within a single judge, the differences are mostly in the third decimal, for example, a faithfulness of 0.639, 0.633, and 0.643 for Naive, Advanced, and Modular under the 8B judge. The judge changes the scores much more than the paradigm does. This is clearest for context recall. For the same answers, it drops from around 0.80 under the 8B judge to around 0.55 under the 70B judge. This judge effect is examined separately in Section 4.2.4.

Table 4: Mean RAGAS scores per paradigm on RAGBench, averaged over the five subsets (100 queries each), under the 8B and 70B judges.

Judge	Paradigm	Faithf.	Ans. rel.	Ctx. prec.	Ctx. rec.	Ans. corr.
Llama 3.1 8B	Naive	0.639	0.405	0.722	0.823	0.459
	Advanced	0.633	0.388	0.754	0.794	0.462
	Modular	0.643	0.409	0.756	0.811	0.463
Llama 3.1 70B	Naive	0.659	0.452	0.701	0.593	0.379
	Advanced	0.662	0.448	0.737	0.528	0.375
	Modular	0.709	0.462	0.746	0.544	0.387

4.2.2 Effect of reranking

The difference between Naive and Advanced is the reranker. The two pipelines share the same generator, embedder, and prompt, therefore the effect of the reranker is isolated. Naive passes the model the top 5 chunks, while Advanced retrieves the top 10 and reranks it down to the top 3. This means that the retrieval metrics are computed over a different number of chunks. Part of any shift is therefore the smaller context, not the reranker itself.

The reranker raises context precision and lowers context recall under both judges (8B and 70B). Each paradigm has ten sets, five under each judge. Context precision goes up in eight out of ten times. It rises from 0.722 to 0.754 under the 8B judge and from 0.701 to 0.737 under the 70B judge. Context recall goes down in nine out of the ten times. It falls from 0.823 to 0.794 under the 8B judge and from 0.593 to 0.528 under the 70B judge. TechQA is the exception. There, the reranker lowers both precision and recall, which does not follow the pattern of the other subsets. The full per-subset numbers are in Appendix D.1.

The end-to-end answer quality does not follow this shift. Faithfulness, answer relevancy and answer correctness stay in the same range from Naive to Advanced. For example a faithfulness of 0.639 against 0.633 under the 8B judge and 0.659 against 0.662 under the 70B judge. The reranker changes which of the retrieved chunks reaches the generator, but it does not make the generated answer measurably better. Section 4.2.6 shows latency remains practically unchanged, so on this benchmark the reranker brings no measurable benefit.

4.2.3 Isolating the reranker

The comparison between Naive and Advanced changes two things at once. Advanced adds the reranker, but it also passes fewer chunks to the generator. Naive sends the top 5, while Advanced sends the top 3 that come from the reranking. The similar results between the paradigms could therefore mean that the reranker adds no benefit. It could also mean that the reranker does help, but the smaller context it passes cancels that out. To ensure of the difference, the Advanced pipeline was rerun with an adjusted top 5 rather than the top 3. This way, it passes the same number of chunks as Naive. Naive at top 5 and Advanced at top 5 then only differ in the reranker.

At equal context size, the reranker does not influence the answer, as no difference is discovered. Naive and Advanced at top 5 reach an almost equal answer relevancy of 0.452 against 0.447, and

Table 5: Isolating the reranker at equal context size, averaged over the five RAGBench subsets under the 70B judge. Naive and Advanced at top 5 pass the same number of chunks and differ only in the reranker.

Configuration	Faithf.	Ans. rel.	Ctx. prec.	Ctx. rec.	Ans. corr.
Naive (top 5)	0.659	0.452	0.701	0.594	0.379
Advanced (top 3)	0.662	0.448	0.737	0.528	0.376
Advanced (top 5)	0.667	0.447	0.724	0.594	0.370

answer correctness of 0.379 against 0.370. At constant context size, the reranker neither positively nor negatively impacts the answer. Comparing the other metrics against Advanced top 3 and Advanced top 5 raises the recall from 0.528 to 0.594, as expected. Answer relevancy is practically unchanged at 0.448 against 0.447. The quality of Naive and Advanced in the main results is similar. This is not due to the smaller context after reranking, as the reranker does not improve the answer, but due to the extra context. The full per-subset scores for Advanced at top 5 are in Table 24 in Appendix D.

4.2.4 Judge reliability

The results above report each metric on the 8B and the 70B judge (Table 4). Both judges score the generated identical answers, meaning that the difference is solely the effect of the judge model as the pipeline is fixed. That difference is larger than the difference between the paradigms. The change is not in the same direction for every metric: faithfulness and answer relevancy increase under the 70B judge, while context recall and answer correctness decrease.

Context precision stays consistent across the judges, ranging from 0.722, 0.754 and 0.756 under the 8B judge to 0.701, 0.737 and 0.746 under the 70B judge for Naive, Advanced and Modular. Faithfulness and answer relevance both increase under the 70B judge as seen in Table 4. The metric answer relevancy is staying around 0.40 under 8B judge against around 0.45 under 70B judge.

Context recall and answer correctness show the most differentiation per judge, and both drop in numbers under the 70B judge. Context recall drops from 0.823, 0.794 and 0.811, 0.593, 0.528 to 0.544, a drop of about 0.25 for every paradigm. Answer correctness declines from around 0.46 to around 0.38. Moving to a larger judge does not mean that all scores go up or down. It decreases in recall and correctness, while it increases in the other metrics.

4.2.5 Effect of generator size

The judge is not the only model that impacts the scores. The generator also influences the results. To check this, all three pipelines were rerun on RAGBench with a 70B generator instead of the 8B generator, while all the other components remained constant. Both runs are scored by the same 70B judge, so the only change is the generator. Table 6 reports the metrics, faithfulness, answer relevancy, and answer correctness over the five subsets. The retrieval metrics are not included as the retrieval did not change.

The larger generator increases the score of every metric, where answer relevancy increases the most.

Answer relevancy goes up by about 0.24 to 0.25 for each paradigm, from 0.45 to 0.70. Faithfulness and answer correctness also increase, but by about 0.10. The size of this improvement is larger than any difference between paradigms. Within a single generator, the paradigms differ by less than 0.03 on answer relevancy. Moving from 8B to 70B generator moves the same metric by 0.25.

The paradigm order is not constant across the two generators. Under the 8B generator, Modular has the highest faithfulness, while under the 70B, Naive has the highest. The paradigms differ by less than 0.03 on answer relevancy within a single generator, which is small enough that the ordering does not survive a change of generator. The same answer relevancy differences were not significant on the Dutch corpus (Section 4.6). For this kind of task, the choice of generator has a larger effect on answer quality than the choice of retrieval architecture.

Table 6: Average RAGBench scores on the 70B judge for the 8B and 70B generators. Only the generation dependent metrics are presented, as retrieval is unchanged.

Generator	Paradigm	Faithf.	Ans. rel.	Ans. corr.
Llama 3.1 8B	Naive	0.659	0.452	0.379
	Advanced	0.662	0.448	0.375
	Modular	0.709	0.462	0.387
Llama 3.1 70B	Naive	0.774	0.691	0.490
	Advanced	0.748	0.702	0.483
	Modular	0.755	0.715	0.487

This comparison should be read with care, because the two generators differ in more than size. The 70B is quantized to AWQ-INT4, while the 8B model runs in bfloat16. The 8B generator samples without a fixed seed with a repetition penalty, while the 70B runs through vLLM with a fixed seed with no repetition penalty. The two also run on a different inference stack, HuggingFace for the 8B model and vLLM for the 70B model. Furthermore, since the 70B model generator is the same model as the judge, the judge may score its own answers more favorably than those of the 8B generator. The direction of the effect is consistent across all three metrics and paradigms, but the exact size should be read as an upper bound rather than a precise measurement. The full per-subset scores for the 70B generator run are in Appendix D.3.

4.2.6 Latency

Latency is measured end to end per query, from the question to the generated answer. It does not include the RAGAS step. Table 7 shows the median (p50), the 95th percentile (p95) and the mean per paradigm over the 500 queries. All three paradigms use the same 8B generator on the same L4 GPU.

Naive and Advanced are in the same range of results, showing a difference of at most 1 second in the 95th percentile. Naive has a median of 1.85s and Advanced 1.59s, with a p95 of 11.38 against 10.26. Adding the reranker shows that it does not necessarily mean that the pipeline will be slower, even though it is an extra step. The results of Modular are in a different bracket altogether. A median of 28.28s, roughly fifteen times that of Naive and a p95 of 37.72s.

Table 7: End-to-end query latency per paradigm on RAGBench (seconds)

Paradigm	Median (p50)	p95	Mean
Naive	1.85	11.38	3.22
Advanced	1.59	10.26	2.90
Modular	28.28	37.72	26.29

If the results above are combined with the quality results, it shows that the three paradigms barely differ. Advanced has shown little benefit over Naive and Modular pays about fifteen times the latency for the same quality. For a production setting, the p95 is a significant number to keep in mind. It would be around 11s for Naive and Advanced and 38 for Modular. The full trade-off across the dimensions is further discussed in Section 4.6. The per-subset breakdown is given in Appendix D.2.

4.3 Phase 2: Evaluation on the Dutch corpus

All three paradigms were run on the same 55-query set that was generated through DRAGON, as described in Section 4.1.5. Only the retrieval and orchestration logic differs across the paradigms, meaning any differences in scores can be traced back to the paradigm rather than the configuration.

Only the 70B judge is used in this phase. Section 4.2.4 showed the 8B judge returns unreliable scores on RAGBench. This holds especially true for faithfulness, where it both fails to return a score and assigns a zero to answers that are taken directly from the text. The 8B judge is therefore not used in the Dutch corpora evaluation. On this corpus the 70B judge was far more stable, returning a score on all but two of the 825 metric evaluations (55 queries, three paradigms, and five metrics). The missing metrics are on Naive: faithfulness and answer correctness.

One limitation of the evaluation should be kept in mind. The question-answer pairs were generated with provenance (Section 4.1.5), so the answers are strongly extractive and can usually be grounded in the retrieved context. Faithfulness therefore has little room to vary on the corpus and this is reflected by the scores. Therefore, answer relevancy is the more informative quality metric here.

4.3.1 Overall scores

Table 8 reports the mean RAGAS score per paradigm, with the median in parentheses, under the 70B judge.

Table 8: Mean RAGAS scores per paradigm on the Dutch corpus (55 queries), under the 70B judge. Mean (median).

Paradigm	Faithf.	Ans. rel.	Ctx. prec.	Ctx. rec.	Ans. corr.
Naive	0.931 (1.000)	0.686 (0.719)	0.853 (1.000)	0.964 (1.000)	0.702 (0.831)
Advanced	0.939 (1.000)	0.658 (0.663)	0.906 (1.000)	0.945 (1.000)	0.716 (0.845)
Modular	0.881 (1.000)	0.677 (0.768)	0.948 (1.000)	0.945 (1.000)	0.712 (0.781)

The three paradigms are close again on the metrics, which is also seen on RAGBench in Section 4.2. Faithfulness is high for all three, between 0.881 and 0.939 with a median of 1.000, which is due to the extractive nature of the synthetic set instead of the paradigm. Answer relevancy has more variation, between 0.658 and 0.686. Answer correctness is around 0.71 across all paradigms.

The difference between Naive and Advanced is the reranker, where Naive passes five chunks and Advanced three. The reranker raised context precision from 0.853 to 0.906, but it lowered the context recall from 0.964 to 0.945. This was the same case on RAGBench. The change of paradigm and components did not lead to a better answer. Faithfulness was almost unchanged, from 0.931 to 0.939, and answer relevancy was slightly lower, from 0.686 to 0.658. The Modular paradigm increased it a bit further to 0.948, but returned the lowest faithfulness of the three, 0.881.

At the level of individual queries, the reranker can remove the relevant chunk that supports the answer. On two queries Advanced paradigm scored a faithfulness of 0.000, where Naive scored 1.000. In both cases, the reranker dropped the relevant chunks from the top-3. On 'Hoe lang worden incasso-instructies bewaard?' is one of them. Naive retrieved that passage and answered correctly, while Advanced removed it during reranking. The answer was no longer grounded in the context and RAGAS scored its faithfulness as 0.000. The mean faithfulness of Advanced is slightly higher than that of Naive, so these two failures do not show in the summary. This is evidence that the reranker is not a free upgrade. Its effect depend on the query and some queries removes the passage that the answer needs.

4.3.2 Latency

Latency is measured in the same way as in Phase 1, end to end per query. From the question to the generated answer, without the validation on RAGAS. Table 9 shows the median (p50), the 95th percentile (p95) and the mean per paradigm from the 55 queries. All three paradigms use Llama-3.1-8B generator.

Table 9: End-to-end query latency per paradigm on the Dutch corpus (seconds).

Paradigm	Median (p50)	p95	Mean
Naive	3.49	7.80	4.61
Advanced	2.89	6.20	3.54
Modular	32.93	38.44	31.63

Naive and Advanced are in the same range. Advanced is a little faster, with a median of 2.89s against 3.49s. The p95 of Advanced is 6.20 against 7.80 of Naive. A likely reason is that Advanced passes three chunks to the generator, while Naive passes five. The shorter context makes it up for the extra time from the reranking step. The reranker does not slow down the pipeline, which is also seen on the RAGBench data. The Modular pipeline is in a different bracket with a median of 32.93s, about nine times slower than Naive. Modular has a p95 of 38.44s, against the Naive 7.80s. The extra time cost comes from the HyDE step and the hybrid retrieval, which add an LLM call and a second retrieval before reranking.

These scores match the scores of Phase 1. Advanced adds almost nothing over Naive in either quality or latency, while Modular pays about nine times the latency for no gain in answer quality.

4.4 Expert evaluation

The expert evaluation follows the design in section 4.1.8. Two raters were asked, but only one has completed the survey. This means that there is no inter-rater reliability for this study and the single-rater limitation is discussed in Section 5.1. The expert scored 18 items: six queries, answered by all three paradigms. The dimensions groundedness, correctness, and relevance map to the RAGAS metrics faithfulness, answer correctness, and answer relevancy. The full scores per item are listed in Appendix D.4.

Two patterns arise here. Firstly, RAGAS does not separate the paradigms on these items, but the expert does. The RAGAS faithfulness score 15 out of the 18 items a 1.000. The expert groundedness scores range from 1 to 5, so where RAGAS scores almost every answer fully grounded, the expert still separates strong from weak. Secondly, they differ at the level of single items. On the Naive item Dutch_0051, RAGAS gives a faithfulness of 0.000, while the expert rates groundedness 4 out of 5. RAGAS marks the answer as completely unsupported, where the expert judges it well grounded.

Secondly, the expert and RAGAS differ on which paradigm is best for relevance on this subset. Naive gets the highest score with a mean of 4.33 from the expert, followed by Advanced at 4.00, then Modular with 3.67. RAGAS answer relevancy ranks the paradigms oppositely. Naive as the lowest with a score of 0.398, followed by Advanced with 0.532, then Modular with 0.540. However, this should not be interpreted as final ranking. The queries were selected to capture the maximum variation rather than to be representative, and the scores differ from the full corpus scores in Section 4.3.1. A ranking from six queries with a single rater would not be reliable.

Altogether, the expert and RAGAS show different results. RAGAS faithfulness is uninformative on these items, while the expert separates the answers. On relevance, the two rankings are opposite. The expert scores offer weak and mixed support, instead of validation. While these results are inconclusive, they do show a broader picture.

4.5 Implementation Complexity

This section reports the implementation complexity of the three paradigms alongside the technical and practical proxies that are defined in Section 4.1.9. The technical proxies are counted directly from the three pipeline implementations. The practical proxies are qualitative. Implementation time is estimated by the author, while required expertise and maintenance burden are rated by a domain expert in Table 3. Table 10 summarizes the results.

The technical proxies show clear differences between the paradigms, especially for component count. This starts from two with Naive, three on Advanced and six on Modular. The tunable parameters count follows the same pattern. The lines of code is barely differ between the paradigms. Naive is at 119, Advanced up to 135, followed by Modular with 136. The three paradigms have the same indexing and loading structure, and LlamaIndex reduces each retrieval component to a few lines. The lines of code is therefore underestimating the difference in complexity. The expert rated the component count as the most informative technical proxy. The reason is that a higher component count requires more knowledge, as there are more components to maintain.

The tunable parameter count should be read with similar caution. The three additional parameter toggles are on/off switches, rather than a continuous setting. The HyDE and fusion module also

Table 10: Implementation complexity per paradigm. The hardware row lists the components loaded on top of the shared base of BGE-M3 and Llama 3.1 8B.

Proxy	Naive	Advanced	Modular
Technical			
Lines of code	119	135	136
Components	2	3	6
Tunable parameters	3	4	7
Hardware	BGE-M3 + 8B	+ reranker	+ reranker, BM25 (CPU)
Practical (low / medium / high)			
Implementation time	Low	Low–Medium	High
Required expertise	Medium	Medium	High
Maintenance burden	Medium	Medium	High

leave little room to tune. On the proxy how much actually can be configured, Modular is in reality closer to the other paradigms than the count of seven suggests.

For the practical proxies, the expert rated the required expertise as medium for Naive and Advanced, and high for Modular. The maintenance burden received the same rating. The estimated implementation time is low for Naive, medium for Advanced, and high for Modular.

The expert placed Naive and Advanced at the same level of expertise. Both paradigms received medium rating, because the underlying technique required solid understanding of the pipeline. Advanced differs from Naive only in the reranker, and thus it did not raise the required expertise in the eyes of the expert. The expert did not change the rating after viewing the codebase from both paradigms. Modular was rated high on required expertise, because of the additional components. HyDE, hybrid retrieval over BM25 and BGE-M3, and reciprocal rank fusion complicate the paradigm and require substantial insight of its components and pipeline. For maintenance, the expert scored the paradigms the same as the required expertise. Keeping a paradigm running involves handling updates, and debugging, which calls for at least medium expertise.

These ratings carry the same limitations as the expert evaluation in Section 4.4. They reflect a single rater, who asses the paradigm from the perspective of feasibility and transferability to colleagues rather than as technical expert. The rater works at VVA, the organization that would deploy the system, and therefore the assessment has a conflict of interest. There is no second rater, and thus no inter-rater agreement can be reported. The ratings are therefore indicative and should be read alongside the counted technical proxies.

4.6 Cross-paradigm comparison

This section combines the four evaluation dimensions and tests whether the differences between paradigms are meaningful. Quality is compared with a paired t-test, since every paradigm is run on the same 55 queries. The test is applied to each metric separately on the Dutch corpus and the mean difference is reported so that the size of a difference can be read on the scale of the metric. The results are in Table 11.

The difference in quality is minimal. For each metric, the average difference is below 0.10. There are no significant differences between the answer quality metrics, faithfulness, answer relevancy, and answer correctness. Faithfulness scores are at the ceiling on this corpus, so the test is uninformative. The only metric on which a difference is significant is context precision, where both Advanced and Modular score higher than Naive ($p = 0.038$ and $p = 0.007$).

Table 12 summarizes the four dimensions per paradigm. Answer quality does not differ significantly across the three paradigms. Latency and implementation complexity both grow from Naive to Modular (Section 4.3.2 and 4.5). These trade-offs are interpreted in Section 5.1.

Table 11: Paired t-test of the RAGAS metrics between paradigms on the Dutch corpus. Δ is the raw mean difference (first minus second). N, A, M are Naive, Advanced, Modular. Significant values ($p < 0.05$) are in bold. Comparisons involving Naive faithfulness or answer correctness use 54 pairs, the rest 55.

	N − A		N − M		A − M	
Metric	Δ	p	Δ	p	Δ	p
Faithfulness	−0.006	0.867	+0.052	0.286	+0.058	0.061
Answer relevancy	+0.028	0.413	+0.009	0.796	−0.019	0.389
Context precision	−0.053	**0.038**	−0.095	**0.007**	−0.042	0.123
Context recall	+0.018	0.569	+0.018	0.659	0.000	1.000
Answer correctness	−0.015	0.677	−0.020	0.633	+0.004	0.914

Table 12: Cross-paradigm comparison across the four evaluation dimensions on the Dutch corpus.

Dimension	Naive	Advanced	Modular
Answer quality (ans. rel.)	0.686	0.658	0.677
Retrieval quality (ctx. prec.)	0.853	0.906	0.948
Latency (p50 / p95, s)	3.49 / 7.80	2.89 / 6.20	32.93 / 38.44
Implementation complexity	2 comp., Medium	3 comp., Medium	6 comp., High

5 Conclusions and Further Research

In this section, the limitations of this study are discussed and the research question is answered. Additionally, directions for future research are provided.

5.1 Discussion

This study compares three RAG paradigms of increasing complexity in the Dutch housing corporation domain. The central result is that in the implemented configurations and on the evaluated datasets, the added complexity does not improve answer quality. Under a controlled setup, where only retrieval and orchestration logic change, the three paradigms produce answers of comparable quality on the Dutch corpus. The answer quality metrics, faithfulness, answer relevancy, and answer

correctness do not differ significantly between the paradigms (Table 11). This pattern was already visible on RAGBench in Section 4.2, where the three paradigms stayed within a few hundredths of each other on every metric. Adding a reranker, HyDE, or hybrid retrieval does not make the final answer better on this domain. This outcome is consistent with a part of prior work. Lyu et al. [16] report that a well-configured minimal pipeline can match or exceed more complex systems. HERB [5] identifies retrieval, not the paradigm as the primary bottleneck. This pattern is therefore not unique to this study.

The only significant differences are in context precision, where both Advanced ($p = 0.038$) and Modular ($p = 0.007$) score higher than Naive. However, these are not quality differences as they follow from the reranker. Naive sends the top 5 chunks to the generator, while Advanced and Modular send the top 3 provided by the reranker. A smaller and more selective set raises the share of the relevant chunks, so the context precision increases by construction. Section 4.2.3 confirms this. At an equal context budget, the reranker no longer separates the paradigms. The two significant results are therefore a predicted effect of the retrieval depth, not evidence that the more complex paradigms retrieve better context for the answer.

The reranker is not simply a free upgrade. On the Dutch corpus, it raised context precision, but this was not reflected in the final answer. The difference in answer relevancy between Naive and Advanced was small, 0.028, and not significant ($p = 0.413$, Table 11). The higher precision did not produce a better answer. At an equal context budget, the reranker added nothing to the answer (Section 4.2.3). It can even remove useful passages. On two queries, it dropped the chunk that supports the answer, after which the model was no longer grounded in the retrieved context, and RAGAS scored faithfulness as 0.000 (Section 4.3.1). The reranker therefore changes the context that reaches the generator without improving answer quality, and on some queries it removes the passage the answer was grounded in.

Modular does not simply repeat the retrieval of Advanced. On the Dutch corpus, about 23 percent of the final top-3 chunks selected by Modular do not appear in the Advanced selection, and on the median query, this rises to about 33 percent. Modular retrieves partly different context through HyDE and hybrid retrieval, yet the answer quality is the same as Naive and Advanced. This rules out one possible explanation for the lack of difference. The paradigms do not score the same simply because they retrieve the same chunks. A likely reason for this pattern lies in the nature of the evaluation set. The Dutch questions were generated with sentence-level provenance and are strongly extractive (Section 4.1.5), so the supporting passage can usually be retrieved by the dense retrieval used in the Naive pipeline. When the answer can already be directly retrieved through the simplest paradigm, the additional retrieval components of Advanced and Modular have little room to improve the final answer. On more abstract questions, such as those the consultants might receive, the retrieval components could add more value.

As the answers have comparable quality, the remaining dimensions determine the trade-off. Latency separates the paradigms clearly. Naive and Advanced both return an answer within four seconds, while Modular takes about nine times as long on the Dutch corpus (Section 4.3.2) and about fifteen times longer on RAGBench (Section 4.2.6). Implementation complexity increase in the same pattern (Section 4.1.9). Advanced matches Naive on both quality and latency, and Modular pays the largest latency and complexity cost for a quality that is not significantly above the baseline. For this domain, the simplest paradigm is enough. The added components of the more complex

paradigms are not justified by a gain in answer quality.

The choice of generator matters more than the choice of paradigm. Moving the generator from 8B to 70B raised answer relevancy from around 0.45 to around 0.70 for each paradigm. The difference per paradigm is less than 0.03 on the same metric within a single generator (Section 4.2.5). The generator effect is far larger than any paradigm effect. This comparison should be read with caution, as the two generator pipelines differ in more than size, yet the direction is clearly shown. A bigger generator will affect the answer quality more than a more complex paradigm.

Part of the differences between paradigms sits within the noise of the judge. Two scoring runs of the same Advanced answers, with no change to the pipeline, differed by 0.033 on context precision. This is the same order of difference between the paradigms on that metric. The 0.033 gap can be read as an indicative noise floor for the RAGAS scores in this study. A difference between paradigms that is smaller than this, or that is not significant in Table 11, should be treated as inconclusive. This is the reason that the analysis above rests on the significant context precision results and the large latency gap. The small answer quality differences are not treated as meaningful.

5.2 Limitations

The expert validation is the weakest part of this study. Two raters were asked, but only one completed the survey. This means there is no inter-rater reliability (Section 4.4). The rater works at VVA, the organization that would deploy the system, which is a conflict of interest. The result of the expert is a validation rather than an independent benchmark. The six queries were selected to cover the maximum range of successes and failures, and thus are not representative of the corpus. Hence, the expert scores cannot be read as a ranking of the paradigms.

The phase 2 test set is synthetic and extractive. It was generated with DRAGON with sentence level provenance (Section 4.1.5), so the answers can almost always be grounded in the retrieved passages. This pushed faithfulness to the ceiling and left it with little room to vary. Answer relevancy plays a larger role on this corpus as it carried more information due to this reason. Questions from real consultants are likely to be more abstract. This is where faithfulness would separate the paradigms more than how it is now.

The corpus does not cover the whole domain, just a small part. It is built from IT governance and information security documents such as the Woningwet, the BIC, NIS2 and CORA. The findings are for regulatory and governance documents and cannot be transferred to other types of documents, such as financial or documents concerning tenants.

The hyperparameters are fixed defaults rather than tuned values. Chunk size, overlap, retrieval depth, and rerank cut-off were held constant across the paradigms, thus the parameters were not optimized. This ensured that the comparison was executed in a controlled setup. A tuned configuration could shift the balance between paradigms. Hyperparameter optimization is left to future work.

The quality scores depend on an LLM as a judge and thereby inherits its limitations. The 8B judge was unreliable on faithfulness, returning no score on some answers and an incorrect 0.000 in answers that were taken directly from the context (Section 4.2.4). For this reason, only the 70B judge was used on the Dutch corpus. The 70B was more stable, but was not free of noise. Two scoring runs

of the same answers differed by 0.033 on context precision (Section 5.1), which is the same order as the differences between paradigms. On RAGBench the generator and the judge were tested on the same 70B to see the difference between generator size. This could have led to the answers being scored more favorably. The quality scores should therefore be read as indicative rather than exact.

5.3 Conclusion

This thesis compared three RAG paradigms: Naive, Advanced, and Modular, in terms of performance and applicability for industry-specific use. Performance covers answer quality, retrieval quality, and latency. Applicability covers implementation complexity, which includes hardware requirements and maintainability. The comparison was conducted in a controlled setup in which all paradigms shared the same embeddings model, generator, and prompt. It consisted of two stages: the English RAGBench benchmark for validating the pipeline and the Dutch corpus from the housing corporation domain for comparison.

On the performance side, the three paradigms produced an answer of comparable quality with the implemented configurations and on the evaluated datasets. The answer quality did not differ significantly between the paradigms on the Dutch corpus, and the same pattern was seen on RAGBench. The only significant differences were in context precision. This can be explained, because the reranker in Advanced and Modular selected a smaller and more relevant set of chunks. This increase in precision did not lead to a better final answer. Latency, on the other hand, is a clear separator between the paradigms. Naive and Advanced both produced an answer within a few seconds, whereas Modular was about nine times slower on the Dutch corpus and about fifteen times slower on RAGBench.

For the applicability aspect, the order for implementation complexity started from Naive and grew to Modular. This also applies to latency. The added components added more latency and complexity, while it did not improve answer quality. The conclusion is limited to this setting, but it is clear. These additional components do not justify their cost in latency and complexity. For these specific documents, questions, and configurations, the simple RAG pipeline is sufficient. In the housing corporation domain, the Naive pipeline is a suitable solution. The Advanced reranker can be added for some extra latency as compensation when a more selective context is needed. Beyond the paradigm comparison, the choice of generator affected answer quality more than the choice of paradigm. This suggests that the generator has more influence on the answer quality than the retrieval architecture for this kind of question.

This work makes three contributions. It provides a controlled, head-to-head comparison on three RAG paradigms on a domain not covered by existing benchmarks. It delivers a reproducible Dutch evaluation set for the housing corporation sector. It considers implementation complexity as a separate evaluation dimension, along with answer quality, retrieval quality, and latency.

5.4 Future Work

Several research directions follow from this study. The first is the Agentic paradigm, which was left out of the scope here because of the hardware constraints. An agent that decides when and what to retrieve and that can retrieve over several steps may behave differently compared to the

three paradigms from this study. A controlled comparison that includes the Agentic paradigm would complete the paradigm taxonomy. Another possible research is hyperparameter optimization. The chunk size, overlap, retrieval depth, and rerank cut-off were kept constant as fixed defaults to control the experiment. Meaning that the paradigms were compared in an untuned setting. A tuned configuration could shift the balance between them.

The evaluation process can be improved in two ways. The Dutch test set was synthetic and highly extractive, which left faithfulness little room to vary and made the task solvable through simple retrieval. A test set built from real questions that the consultants receive, including questions that require reasoning across multiple documents would test the paradigms on harder retrieval. This is where the more complex pipelines could show their value. Additionally, the expert evaluation was completed by a single rater from the organization that would deploy the system, which may lead to a conflict of interest. A second independent rater would allow for inter-rater reliability to be reported and would turn the expert step into a stronger evaluation.

For the findings, there are two ways that could help to understand how far these results would generalize. The corpus covered IT governance and information security documents, so the results only hold for these kinds of documents and not for other areas such as finance or documents concerning tenants. Extending the corpus to broader areas would show whether the pattern seen in this thesis between paradigms applies to only this material. The comparison of generator sizes also pointed to a large effect, but the two generator pipelines differed more than just size. Quantization, sampling and inference stack were not constant in both pipelines. A controlled study that varies in only the generator would measure that effect more precisely.

References

[1] Mohammad Mahdi Abootorabi, Amirhosein Zobeiri, Mahdi Dehghani, Mohammadali Mohammadkhani, Bardia Mohammadi, Omid Ghahroodi, Mahdieh Soleymani Baghshah, and Ehsaneddin Asgari. Ask in any modality: A comprehensive survey on multimodal retrieval-augmented generation, 2025.

[2] Shuyang Cao, Karthik Radhakrishnan, David Rosenberg, Steven Lu, Pengxiang Cheng, Lu Wang, and Shiyue Zhang. Evaluating the retrieval robustness of large language models, 2025.

[3] Brian J. Chan, Chao-Ting Chen, Jui-Hung Cheng, and Hen-Hsen Huang. Don't do rag: When cache-augmented generation is all you need for knowledge tasks. In *Companion Proceedings of the ACM on Web Conference 2025*, WWW '25, page 893–897. ACM, May 2025.

[4] Jianlv Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. M3-embedding: Multi-linguality, multi-functionality, multi-granularity text embeddings through self-knowledge distillation, 2025.

[5] Prafulla Kumar Choubey, Xiangyu Peng, Shilpa Bhagavath, Kung-Hsiang Huang, Caiming Xiong, and Chien-Sheng Wu. Benchmarking deep search over heterogeneous enterprise data, 2025.

[6] Gordon V. Cormack, Charles L A Clarke, and Stefan Buettcher. Reciprocal rank fusion outperforms condorcet and individual rank learning methods. In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '09, page 758–759, New York, NY, USA, 2009. Association for Computing Machinery.

[7] Shahul Es, Jithin James, Luis Espinosa-Anke, and Steven Schockaert. Ragas: Automated evaluation of retrieval augmented generation, 2025.

[8] Robert Friel, Masha Belyi, and Atindriyo Sanyal. Ragbench: Explainable benchmark for retrieval-augmented generation systems, 2025.

[9] Luyu Gao, Xueguang Ma, Jimmy Lin, and Jamie Callan. Precise zero-shot dense retrieval without relevance labels, 2022.

[10] Yunfan Gao, Yun Xiong, Xinyu Gao, et al. Retrieval-augmented generation for large language models: A survey, 2024.

[11] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding, 2021.

[12] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 conference on empirical methods in natural language processing (EMNLP)*, pages 6769–6781, 2020.

[13] Barbara Kitchenham and Stuart Charters. Guidelines for performing systematic literature reviews in software engineering. Technical Report EBSE 2007-001, Keele University and Durham University, 01 2007.

[14] Oleksandr Kolomiyets and Marie-Francine Moens. A survey on question answering technology from an information retrieval perspective. *Information Sciences*, 181(24):5412–5434, 2011.

[15] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474, 2020.

[16] Xinxi Lyu, Michael Duan, Rulin Shao, Pang Wei Koh, and Sewon Min. Frustratingly simple retrieval improves challenging, reasoning-intensive benchmarks, 2025.

[17] Humza Naveed, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usman, Naveed Akhtar, Nick Barnes, and Ajmal Mian. A comprehensive overview of large language models. *ACM Transactions on Intelligent Systems and Technology*, 16(5):1–72, 2025.

[18] Matan Orbach, Ohad Eytan, Benjamin Sznajder, Ariel Gera, Odellia Boni, Yoav Kantor, Gal Bloch, Omri Levy, Hadas Abraham, Nitzan Barzilay, Eyal Shnarch, Michael E. Factor, Shila Ofek-Koifman, Paula Ta-Shma, and Assaf Toledo. An analysis of hyper-parameter optimization methods for retrieval augmented generation, 2025.

[19] Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends in Information Retrieval*, 3:333–389, 09 2009.

[20] Ashish Sardana. Real-time evaluation models for rag: Who detects hallucinations best?, 2025.

[21] Haiyang Shen, Hang Yan, Zhongshi Xing, Mugeng Liu, Yue Li, Zhiyang Chen, Yuxiang Wang, Jiuzheng Wang, and Yun Ma. Dragon: Domain-specific robust automatic data generation for rag optimization, 2026.

[22] Aditi Singh et al. Agentic retrieval-augmented generation: A survey, 2025.

[23] Zhang Siyue, Xue Yuxiang, Zhang Yiming, Wu Xiaobao, Luu Anh Tuan, and Zhao Chen. Mrag: A modular retrieval framework for time-sensitive question answering, 2025.

[24] Vibrant Labs. Factual correctness. https://docs.ragas.io/en/latest/concepts/metrics/available_metrics/factual_correctness/, 2025. Accessed 8 June 2026.

[25] Vibrant Labs. List of available metrics. https://docs.ragas.io/en/latest/concepts/metrics/available_metrics/, 2025. Accessed 8 June 2026.

[26] Vibrant Labs. Semantic similarity. https://docs.ragas.io/en/latest/concepts/metrics/available_metrics/semantic_similarity/, 2025. Accessed 8 June 2026.

[27] Di Wu, Jia-Chen Gu, Kai-Wei Chang, and Nanyun Peng. Self-routing rag: Binding selective retrieval with knowledge verbalization, 2026.

[28] Zhishang Xiang, Chuanjie Wu, Qinggang Zhang, Shengyuan Chen, Zijin Hong, Xiao Huang, and Jinsong Su. When to use graphs in rag: A comprehensive analysis for graph retrieval-augmented generation, 2026.

[29] Linda Zeng, Rithwik Gupta, Divij Motwani, Yi Zhang, and Diji Yang. Worse than zero-shot? a fact-checking dataset for evaluating the robustness of rag against misleading retrievals, 2026.

[30] Yixiao Zeng, Tianyu Cao, Danqing Wang, Xinran Zhao, Zimeng Qiu, Morteza Ziyadi, Tongshuang Wu, and Lei Li. Rare: Retrieval-aware robustness evaluation for retrieval-augmented generation systems, 2025.

[31] Penghao Zhao, Hailin Zhang, Qinhan Yu, Zhengren Wang, Yunteng Geng, Fangcheng Fu, Ling Yang, Wentao Zhang, Jie Jiang, and Bin Cui. Retrieval-augmented generation for ai-generated content: A survey, 2024.

A Supporting Tables

Table 13: Text-based benchmarks identified in the systematic literature review.

Benchmark	Domain	Scale	Primary metrics	Ref.
RAGBench	5 industry domains, user manuals	100,000 examples	TRACe metrics	[8]
HERB	Enterprise software lifecycle	39,190 artifacts, 1,514 queries	Performance score	[5]
RARE-Set	Finance, economics, policy	527 docs, 48,295 questions	RARE-Met (robustness)	[30]
TempRAGEval	Temporal QA	Existing + perturbations	AR@k, ER@k, EM, F1	[23]
RAGuard	Political fact-checking	2,648 claims, 16,331 docs	Accuracy per evidence type	[29]
GraphRAG-Bench	Novel, medical	4,076 questions	ACC, ROUGE-L, Evidence Coverage, Faithfulness	[28]
DRAGONBench	8 collections, 4 domains	Variable	Faithfulness, answer relevance and context relevance	[21]
HPO datasets	5 domains incl. product docs	Variable	Lexical-AC, Lexical-FF, LLMaaJ-AC	[18]
Robustness set	NQ, HotpotQA, ASQA	1,500 QA pairs	No-degradation rate, Retrieval size robustness, Retrieval order robustness	[2]
Hallucination set	Finance, biomedical, general	6 applications	Precision, Recall, AUROC	[20]
SR-RAG sets	Knowledge-intensive QA	4 benchmarks	Accuracy, retrieval rate	[27]

Table 14: Open-source generator models recurring across the reviewed papers.

Model family	Sizes evaluated	Ref.
Llama 3.x	1B, 3B, 8B, 70B	[2, 23, 18]
Mistral	12B, 123B	[2]
Qwen 2.5	72B	[21]
Command	R, R+	[2]

Table 15: Retrieval components recurring across the reviewed papers.

Component	Specific model / variant	Ref.
Sparse retriever	BM25	[2, 23]
Dense embedder	BGE family	[2, 1, 18, 28]
Dense embedding pool	MTEB top-6	[21]
Hybrid retrieval	BM25 + dense	[5]
Query transformation	HyDE	[9, 10, 31]

B Search Query

The following query was executed on arXiv on 16 March 2026 and returned 165 results. It shows the parameters of the arXiv Advanced search interface for reproducibility. The query has been split across multiple lines for readability without changing its parameters.

```
order: -announced_date_first
size: 50
date_range: from 2020-01-01 to 2026-12-31
include_cross_list: True
terms: AND all="retrieval-augmented generation"
       AND (all=evaluation OR all=assessment OR all=benchmark)
       AND (all=framework OR all=comparison OR all=survey)
```

C Paradigm Configuration

Table 16 lists the concrete configuration used across the paradigms, for reproducibility.

Table 16: Implementation configuration.

Setting	Value
Chunker	SentenceSplitter, chunk size 1024, chunk overlap 100
Embedding model	BAAI/bge-m3
Generator	meta-llama/Llama-3.1-8B-Instruct (bfloat16)
Context window	8192 tokens
Max new tokens	512
Temperature	0.1
Repetition penalty	1.15
Reranker (Advanced, Modular)	BAAI/bge-reranker-v2-m3 (fp16)
Retrieval depth (Naive)	top-5, no reranking
Retrieval depth (Advanced & Modular)	top-10 retrieved, top-3 after reranking
Hybrid fusion (Modular)	dense (BGE-M3) + sparse (BM25), reciprocal rank fusion
Query transformation (Modular)	HyDE, original query retained

D Detailed Evaluation Results

D.1 RAGBench per-subset scores

The tables below give the per-subset RAGAS scores on RAGBench for all three paradigms, under both judges. Each cell in Tables 17 and 18 reports the mean with the median in parentheses, over 100 queries per subset. Tables 19 and 20 report the number of queries (out of 100) for which the judge returned no score.

Table 17: RAGBench per-subset scores under the 8B judge, mean (median).

Paradigm	Subset	Faithf.	Ans. rel.	Ctx. prec.	Ctx. rec.	Ans. corr.
Naive	CUAD	0.685 (0.750)	0.457 (0.501)	0.554 (0.533)	0.716 (1.000)	0.391 (0.380)
	FinQA	0.569 (0.500)	0.390 (0.382)	0.590 (0.725)	0.762 (1.000)	0.440 (0.422)
	HotpotQA	0.759 (1.000)	0.394 (0.378)	0.765 (1.000)	0.933 (1.000)	0.577 (0.569)
	PubMedQA	0.675 (1.000)	0.432 (0.392)	0.895 (1.000)	0.912 (1.000)	0.428 (0.411)
	TechQA	0.505 (0.500)	0.354 (0.431)	0.806 (1.000)	0.791 (1.000)	0.457 (0.435)
Advanced	CUAD	0.726 (0.817)	0.471 (0.487)	0.597 (0.708)	0.694 (0.800)	0.431 (0.424)
	FinQA	0.610 (0.667)	0.435 (0.416)	0.681 (1.000)	0.684 (1.000)	0.439 (0.404)
	HotpotQA	0.745 (1.000)	0.370 (0.382)	0.787 (1.000)	0.918 (1.000)	0.597 (0.565)
	PubMedQA	0.649 (0.800)	0.390 (0.372)	0.922 (1.000)	0.892 (1.000)	0.423 (0.400)
	TechQA	0.436 (0.417)	0.275 (0.000)	0.781 (1.000)	0.782 (1.000)	0.420 (0.403)
Modular	CUAD	0.700 (0.750)	0.443 (0.479)	0.588 (0.833)	0.707 (0.750)	0.414 (0.400)
	FinQA	0.572 (0.586)	0.477 (0.414)	0.698 (1.000)	0.715 (1.000)	0.445 (0.421)
	HotpotQA	0.791 (1.000)	0.387 (0.386)	0.788 (1.000)	0.925 (1.000)	0.595 (0.540)
	PubMedQA	0.659 (0.750)	0.403 (0.383)	0.927 (1.000)	0.904 (1.000)	0.424 (0.397)
	TechQA	0.494 (0.500)	0.335 (0.399)	0.781 (1.000)	0.803 (1.000)	0.439 (0.420)

Table 18: RAGBench per-subset scores under the 70B judge, mean (median).

Paradigm	Subset	Faithf.	Ans. rel.	Ctx. prec.	Ctx. rec.	Ans. corr.
Naive	CUAD	0.669 (0.800)	0.543 (0.567)	0.446 (0.464)	0.235 (0.000)	0.209 (0.151)
	FinQA	0.552 (0.750)	0.419 (0.438)	0.701 (1.000)	0.604 (0.707)	0.345 (0.153)
	HotpotQA	0.844 (1.000)	0.417 (0.421)	0.798 (1.000)	0.880 (1.000)	0.626 (0.640)
	PubMedQA	0.777 (1.000)	0.495 (0.430)	0.895 (1.000)	0.813 (1.000)	0.406 (0.406)
	TechQA	0.453 (0.292)	0.386 (0.463)	0.666 (0.867)	0.436 (0.400)	0.310 (0.171)
Advanced	CUAD	0.675 (0.857)	0.543 (0.568)	0.472 (0.500)	0.127 (0.000)	0.227 (0.158)
	FinQA	0.615 (0.750)	0.475 (0.458)	0.791 (1.000)	0.608 (0.700)	0.395 (0.347)
	HotpotQA	0.841 (1.000)	0.417 (0.423)	0.837 (1.000)	0.835 (1.000)	0.628 (0.648)
	PubMedQA	0.751 (1.000)	0.462 (0.399)	0.951 (1.000)	0.699 (0.750)	0.354 (0.352)
	TechQA	0.430 (0.333)	0.342 (0.376)	0.634 (1.000)	0.373 (0.333)	0.274 (0.186)
Modular	CUAD	0.783 (1.000)	0.551 (0.537)	0.489 (0.500)	0.115 (0.000)	0.215 (0.152)
	FinQA	0.658 (0.845)	0.506 (0.494)	0.826 (1.000)	0.643 (0.714)	0.412 (0.365)
	HotpotQA	0.844 (1.000)	0.422 (0.421)	0.839 (1.000)	0.850 (1.000)	0.644 (0.655)
	PubMedQA	0.778 (1.000)	0.457 (0.401)	0.949 (1.000)	0.720 (0.750)	0.376 (0.357)
	TechQA	0.481 (0.500)	0.376 (0.451)	0.628 (0.917)	0.394 (0.333)	0.288 (0.188)

Table 19: RAGBench per-subset NaN counts under the 8B judge (queries out of 100 with no returned score).

Paradigm	Subset	Faithf.	Ans. rel.	Ctx. prec.	Ctx. rec.	Ans. corr.
Naive	CUAD	5	0	0	3	14
	FinQA	1	0	6	1	3
	HotpotQA	0	0	1	1	0
	PubMedQA	0	0	0	2	4
	TechQA	1	0	3	1	9
Advanced	CUAD	0	0	0	2	9
	FinQA	2	0	4	0	7
	HotpotQA	0	0	0	1	0
	PubMedQA	0	0	0	0	3
	TechQA	0	0	1	4	10
Modular	CUAD	1	0	0	0	8
	FinQA	2	0	3	0	7
	HotpotQA	0	0	0	0	0
	PubMedQA	0	0	0	0	2
	TechQA	1	0	1	4	7

Table 20: RAGBench per-subset NaN counts under the 70B judge (queries out of 100 with no returned score).

Paradigm	Subset	Faithf.	Ans. rel.	Ctx. prec.	Ctx. rec.	Ans. corr.
Naive	CUAD	7	0	0	0	9
	FinQA	1	0	0	0	1
	HotpotQA	0	0	0	0	0
	PubMedQA	0	0	0	0	1
	TechQA	0	0	0	0	5
Advanced	CUAD	1	0	0	0	1
	FinQA	1	0	0	0	2
	HotpotQA	0	0	0	0	0
	PubMedQA	0	0	0	0	1
	TechQA	1	0	0	0	4
Modular	CUAD	1	0	0	0	4
	FinQA	0	0	0	0	0
	HotpotQA	0	0	0	0	0
	PubMedQA	0	0	0	0	1
	TechQA	0	0	0	0	3

D.2 RAGBench per-subset latency

Table 21 gives the per-subset query latency in seconds for all three paradigms, over 100 queries per subset. The per-paradigm values in Table 7 are pooled over these five subsets.

Table 21: RAGBench per-subset end-to-end query latency, in seconds (100 queries per subset).

Paradigm	Subset	p50	p95	Mean
Naive	CUAD	4.59	15.22	5.54
	FinQA	1.58	11.96	3.38
	HotpotQA	0.62	2.44	0.91
	PubMedQA	1.68	6.59	2.37
	TechQA	2.49	10.37	3.92
Advanced	CUAD	3.19	9.72	3.88
	FinQA	1.61	15.05	4.27
	HotpotQA	0.69	3.20	1.16
	PubMedQA	1.40	4.79	1.82
	TechQA	2.06	11.11	3.37
Modular	CUAD	28.09	38.10	27.71
	FinQA	23.81	40.03	24.35
	HotpotQA	13.97	28.11	15.78
	PubMedQA	32.98	36.33	31.90
	TechQA	33.32	38.40	31.68

D.3 RAGBench 70B-generator run

Table 22: RAGBench per-subset scores for the 70B-generator run under the 70B judge, mean (median). CUAD and TechQA were re-scored with a larger judge output budget (3072 tokens) to remove output truncation on some faithfulness and answer correctness items. Remaining missing scores are in Table 23.

Paradigm	Subset	Faithf.	Ans. rel.	Ctx. prec.	Ctx. rec.	Ans. corr.
Naive	CUAD	0.846 (1.000)	0.827 (0.885)	0.448 (0.450)	0.206 (0.000)	0.256 (0.177)
	FinQA	0.715 (1.000)	0.571 (0.495)	0.701 (1.000)	0.604 (0.707)	0.441 (0.391)
	HotpotQA	0.927 (1.000)	0.528 (0.506)	0.798 (1.000)	0.880 (1.000)	0.695 (0.669)
	PubMedQA	0.829 (1.000)	0.816 (0.913)	0.898 (1.000)	0.819 (1.000)	0.575 (0.594)
	TechQA	0.555 (0.569)	0.714 (0.793)	0.659 (0.850)	0.437 (0.400)	0.483 (0.472)
Advanced	CUAD	0.837 (1.000)	0.782 (0.803)	0.468 (0.500)	0.124 (0.000)	0.270 (0.171)
	FinQA	0.721 (1.000)	0.661 (0.860)	0.789 (1.000)	0.604 (0.683)	0.486 (0.492)
	HotpotQA	0.910 (1.000)	0.532 (0.511)	0.838 (1.000)	0.835 (1.000)	0.688 (0.661)
	PubMedQA	0.739 (1.000)	0.797 (0.908)	0.951 (1.000)	0.694 (0.750)	0.493 (0.479)
	TechQA	0.532 (0.569)	0.740 (0.795)	0.624 (1.000)	0.371 (0.333)	0.480 (0.464)
Modular	CUAD	0.823 (1.000)	0.835 (0.853)	0.487 (0.500)	0.117 (0.000)	0.258 (0.172)
	FinQA	0.744 (1.000)	0.691 (0.845)	0.822 (1.000)	0.642 (0.707)	0.496 (0.488)
	HotpotQA	0.925 (1.000)	0.521 (0.491)	0.849 (1.000)	0.850 (1.000)	0.689 (0.661)
	PubMedQA	0.755 (1.000)	0.781 (0.909)	0.949 (1.000)	0.715 (0.750)	0.502 (0.483)
	TechQA	0.527 (0.528)	0.747 (0.801)	0.630 (1.000)	0.398 (0.333)	0.489 (0.468)

Table 23: RAGBench per-subset NaN counts for the 70B-generator run under the 70B judge (queries out of 100 with no returned score). CUAD and TechQA were re-scored with a larger judge output budget, which resolved all previously truncated items.

Paradigm	Subset	Faithf.	Ans. rel.	Ctx. prec.	Ctx. rec.	Ans. corr.
Naive	CUAD	0	0	0	0	0
	FinQA	0	0	0	0	1
	HotpotQA	0	0	0	0	0
	PubMedQA	1	0	0	0	5
	TechQA	0	0	0	0	0
Advanced	CUAD	0	0	0	0	0
	FinQA	0	0	0	0	1
	HotpotQA	0	0	0	0	0
	PubMedQA	3	0	0	0	5
	TechQA	0	0	0	0	0
Modular	CUAD	0	0	0	0	0
	FinQA	0	0	0	0	1
	HotpotQA	0	0	0	0	0
	PubMedQA	3	0	0	0	5
	TechQA	0	0	0	0	0

Table 24: Per-subset scores for the Advanced pipeline at top 5 under the 70B judge, mean over 100 queries per subset. All NaN counts are zero.

Subset	Faithf.	Ans. rel.	Ctx. prec.	Ctx. rec.	Ans. corr.
CUAD	0.747	0.537	0.476	0.177	0.184
FinQA	0.488	0.411	0.753	0.608	0.366
HotpotQA	0.875	0.437	0.837	0.875	0.626
PubMedQA	0.764	0.506	0.936	0.873	0.393
TechQA	0.459	0.343	0.619	0.440	0.283

D.4 Expert evaluation item scores

Table 25 lists the expert scores and the matching RAGAS scores for all 18 items, grouped by query. The expert columns G, C, and R are groundedness, correctness, and relevance on a 1 to 5 scale. The RAGAS columns are on a 0 to 1 scale. One Naive answer correctness value (Dutch_0032) was not returned by the judge and is marked with a dash.

Table 25: Expert scores and RAGAS scores per item on the six-query subset. Expert: groundedness (G), correctness (C), relevance (R), each 1 to 5. RAGAS: faithfulness, answer correctness, answer relevancy, each 0 to 1.

		Expert			RAGAS		
Query	Paradigm	G	C	R	Faith.	Corr.	Rel.
Dutch_0005	Naive	3	3	4	1.000	0.683	0.421
	Advanced	2	1	3	1.000	0.683	0.449
	Modular	2	1	3	1.000	0.683	0.421
Dutch_0030	Naive	1	1	5	1.000	0.108	0.000
	Advanced	5	5	5	1.000	0.410	0.768
	Modular	5	5	5	1.000	0.931	0.768
Dutch_0032	Naive	5	5	5	1.000	–	0.707
	Advanced	4	4	4	1.000	0.671	0.842
	Modular	4	4	4	1.000	0.172	0.842
Dutch_0033	Naive	4	4	4	1.000	0.116	0.295
	Advanced	3	4	4	1.000	0.107	0.000
	Modular	2	2	2	0.500	0.123	0.347
Dutch_0043	Naive	4	4	4	1.000	0.908	0.230
	Advanced	3	2	4	1.000	0.138	0.355
	Modular	1	1	3	1.000	0.138	0.355
Dutch_0051	Naive	4	4	4	0.000	0.105	0.735
	Advanced	3	2	4	0.667	0.118	0.775
	Modular	5	5	5	1.000	0.990	0.507