School of Engineering

Centre for Artificial Intelligence

Master Thesis

Master of Science in Engineering (MSE)

Refusal-Aware Benchmarking of a Domain-Specific RAG Assistant with Validated LLM Judges

---

Submitted by: Samuel Truniger
Matr.-Nr.: 20638359

Supervised by: Dr. Ricardo Chavarriaga

Co-supervised by: Oliver Forster

Submission date: 27 August 2026

Abstract

Retrieval-augmented generation (RAG) allows conversational assistants to ground their answers in external non-parametric knowledge. However, the modular architecture of RAG precludes a one-size-fits-all configuration. The most effective setup must therefore be determined through systematic benchmarking. Benchmarks require domain-specific datasets and often rely on unvalidated LLM-as-a-judge metrics. This thesis addresses these gaps for the RAI Copilot, a self-hosted RAG assistant for responsible AI developed across three preceding projects.

Following the design science research paradigm, this thesis implements and validates a system-agnostic, refusal-aware evaluation toolbox. This toolbox generates synthetic Q&A datasets from a PDF corpus through a generate-then-validate approach and scores external RAG systems across twelve metrics, gated by a refusal classifier. The thesis further improves the RAI Copilot and applies the toolbox to identify a deployable configuration. Finally, it assesses the performance and reliability of the improved Copilot under different concurrency levels.

A manual screening and a human annotation study with five expert annotators show that the toolbox's artifacts are usable but not uniformly trustworthy. Two thirds of the screened dataset items required no correction, and judge reliability ranges from perfect agreement for the refusal classifier to a defective citation accuracy judge. The toolbox evaluated 20 pipeline configurations of the Copilot and identified a champion configuration (mistral-24b, hybrid retrieval with a bge-m3 reranker, $k_{\text{final}} = 4$) that meets the seven acceptance criteria. A load test confirms that the deployed system handles the expected 30 concurrent users with a 95th-percentile time to first token of 4.96 s. Under overload, the system degrades through latency rather than failures (time to first token p95 21.76 s).

Future work should focus on validating the LLMs employed in the Q&A generation pipeline. The evaluation pipeline should be extended with a judge-based answer correctness metric and a redesigned citation accuracy metric. The Copilot could further be improved with document cleaning and dedicated guardrails to harden the refusal behavior.

Contents

I Foundations **1**

1 Introduction **2**
1.1 Problem Statement . . . 2
1.2 Objectives . . . 3
1.3 Use of Generative AI . . . 3

2 Methodology Overview **4**
2.1 Design Science Research Process . . . 4
2.2 Preceding Projects . . . 5
2.3 Project Activities . . . 6
2.3.1 Design and Implementation . . . 6
2.3.2 Evaluation . . . 6

3 Background and Related Work **9**
3.1 Retrieval-Augmented Generation . . . 9
3.1.1 Fundamentals of a RAG Pipeline . . . 9
3.1.2 Limitations of RAG . . . 11
3.1.3 Advanced RAG Techniques . . . 12
3.1.4 Responsible AI in RAG Systems . . . 13
3.2 Evaluation of RAG Systems . . . 13
3.2.1 Evaluation Datasets . . . 14
3.2.2 Metrics . . . 16
3.2.3 LLM-as-a-Judge Evaluation . . . 17
3.2.4 Evaluation Frameworks . . . 18

II RAG Evaluation Toolbox **20**

4 Toolbox Design **21**
4.1 Framework Analysis . . . 21
4.2 Q&A Generation Pipeline . . . 22
4.2.1 Question Taxonomy . . . 23

4.2.2 Pipeline Architecture . . . 24
4.3 Evaluation Pipeline . . . 27
4.3.1 Metric Taxonomy . . . 28
4.3.2 Pipeline Architecture . . . 34
4.4 Packaging and CLI . . . 35
4.5 Limitations . . . 36
4.5.1 Scope Conditions . . . 36
4.5.2 Q&A Generation Pipeline . . . 37
4.5.3 Evaluation Pipeline . . . 37

5 Toolbox Validation **39**
5.1 Screening of the Benchmark Dataset . . . 39
5.2 Human Annotation Study . . . 41
5.2.1 Study Design . . . 41
5.2.2 Judge Reliability . . . 43
5.3 Recommendations for Human Annotation Studies . . . 45
5.4 Summary and Limitations . . . 46

III RAI Copilot **48**

6 RAI Copilot **49**
6.1 System Architecture . . . 49
6.1.1 Interaction of System Components . . . 49
6.1.2 Deployment Topology . . . 51
6.2 Document and Query Pipelines . . . 51
6.2.1 Document Pipeline . . . 52
6.2.2 Query Pipeline . . . 54
6.3 Key Features . . . 55
6.3.1 Feature Map . . . 55
6.3.2 Collaboration and Governance . . . 56
6.3.3 Privacy . . . 57
6.3.4 Citation Highlighting . . . 57
6.3.5 Metadata Management . . . 59
6.4 Limitations . . . 60

7 RAG Pipeline Benchmark **62**
7.1 Experimental Setup . . . 62
7.1.1 Benchmark Dataset . . . 62
7.1.2 Configuration Space and Execution Environment . . . 63
7.1.3 Answer Collection . . . 65
7.1.4 Metrics and Judge Configuration . . . 67
7.1.5 Configuration Selection . . . 68
7.2 Results and Analysis . . . 70

7.2.1 Acceptance Criteria and Champion Selection . . . 71
7.2.2 Refusal Behavior . . . 72
7.2.3 Effect of Retrieval Depth . . . 73
7.2.4 Citation Defect . . . 74
7.2.5 Traditional Metrics . . . 75
7.3 Summary and Limitations . . . 77

8 System Evaluation **79**
8.1 Automated Test Suite . . . 79
8.1.1 Backend Tests . . . 79
8.1.2 End-to-End Tests . . . 80
8.2 Load Testing . . . 81
8.2.1 Load Test Design . . . 81
8.2.2 Measured Timing Metrics . . . 82
8.2.3 Load Test Execution . . . 83
8.3 Results and Analysis . . . 84
8.3.1 Automated Test Suite . . . 84
8.3.2 Load Testing . . . 84
8.4 Limitations . . . 85

IV Synthesis **87**

9 Conclusion and Future Work **88**
9.1 Conclusion . . . 88
9.2 Key Contributions . . . 89
9.3 Threats to Validity . . . 89
9.4 Future Work . . . 90

Bibliography **92**

Appendices **105**

A Evaluation Toolbox: Technical Details **105**
A.1 Q&A Generation Pipeline . . . 105
A.2 Evaluation Pipeline . . . 107

B Toolbox Validation: Study Materials **110**
B.1 Annotation Tool . . . 110
B.2 Selected Configurations . . . 110
B.3 Verdict Distributions . . . 111

C RAI Copilot: Technical Details **112**
C.1 User Interface . . . 112

C.2 Document Metadata Schema 115
C.3 Configuration 116
C.4 Prompt Templates 117

D RAG Pipeline Benchmark: Configurations and Results 120
D.1 Benchmark Configuration 120
D.2 Benchmark Results 122

E System Evaluation: Load Tests 128
E.1 Load Test Results 128

List of Figures

4.1 Q&A Generation Pipeline 22
4.2 Evaluation Pipeline 28

6.1 C4 Container Diagram of the RAI Copilot 50
6.2 RAI Copilot RAG Pipeline Architecture 52

7.1 Answer Collection Pipeline 65
7.2 Acceptance Criteria Outcome 71
7.3 Sensitivity of the Eligible Set to each Threshold 72
7.4 Refusal Behavior of the Sampled Configurations 73
7.5 Retrieval Metrics against the Size of the Retrieved Context 74
7.6 Citation Coverage against Citation Accuracy 75
7.7 Answer Length Effect 76
7.8 Generator Answer Length Distribution 76

B.1 Interface of the Annotation Study Tool 110

C.1 Login View 112
C.2 Chat Interface 113
C.3 Citation Highlighting in the RAI Copilot 113
C.4 Document Overview 114
C.5 Metadata Management 114
C.6 Document Upload Interface 115
C.7 User Management Interface 115

D.1 Benchmark Latency Decomposition 127
D.2 Benchmark Indexing Latency 127

List of Tables

2.1 Mapping of DSR Activities to the Thesis 4
2.2 Overview of Empirical Studies 8

4.1 RAG Evaluation Framework Analysis 22
4.2 Evaluation Pipeline: VT2 vs. This Thesis 27
4.3 Metric Overview 29
4.4 Refusal Classification Outcomes 29

5.1 Outcome of the Benchmark Dataset Screening 40
5.2 Annotation Tasks 42
5.3 Judge–Author Agreement 43
5.4 Inter-rater Agreement 44

6.1 Key Features of the RAI Copilot 56

7.1 Composition of the Benchmark Dataset 63
7.2 Benchmarking Hyperparameter Search Space 64
7.3 Metrics Computed in the Benchmark 68
7.4 Benchmark Acceptance Criteria 69

8.1 Backend Test Suite Coverage 80
8.2 Load Test: VT2 vs. This Thesis 82
8.3 Load Test Measured Timing Metrics 83
8.4 Load Test Levels 83
8.5 Load Test Results 84

B.1 Configurations of the Human Annotation Study 111
B.2 Joint Distribution of Judge's and Author's Verdicts 111

C.1 Configuration Environment Variables 116

D.1 Models and Licensing 122
D.2 Benchmark Configuration Index 123
D.3 Refusal Classification per Configuration 124
D.4 Metric Scores per Configuration 125
D.5 Response Time per Configuration 126

E.1 Per-run Load Test Results 128

List of Abbreviations

AI artificial intelligence

API application programming interface

CAI Centre for Artificial Intelligence

DOI digital object identifier

DSR Design Science Research

E2E end-to-end

EVA1 Supplementary Event 1

FRR false refusal rate

LLM large language model

MMR maximal marginal relevance

MT Master Thesis

NDCG normalized discounted cumulative gain

NLI natural language inference

RAG retrieval-augmented generation

RAI responsible AI

RNG random number generator

RRF reciprocal rank fusion

SoE School of Engineering

SUT system under test

TRR true refusal rate

TTFT time to first token

UI user interface

VM virtual machine

VPN virtual private network

VT1 Specialization Project 1

VT2 Specialization Project 2

ZHAW Zurich University of Applied Sciences

Part I

Foundations

Chapter 1

Introduction

1.1 Problem Statement

Organizations that develop or adopt systems based on artificial intelligence (AI) face a growing body of regulations, standards, and guidelines on responsible AI [1], [2]. Navigating this fragmented landscape requires domain expertise that many development teams lack. Conversational assistants built on retrieval-augmented generation (RAG) address this need. They ground the responses of a large language model (LLM) in external context relevant to a user query. This approach mitigates hallucinations and outdated knowledge [3], [4]. The responsible AI (RAI) Copilot, developed by the author across three preceding student projects (Section 2.2), is such an assistant. The Copilot is a self-hosted RAG system that answers questions about responsible AI based on a curated corpus of regulations, standards, and research papers.

An assistant that advises on responsible AI must itself be reliable and trustworthy. This requirement exposes two evaluation gaps. First, whether a RAG system retrieves the right evidence, grounds its answers in it, and abstains from questions it should not answer cannot be established without a systematic benchmark. Existing evaluation frameworks cover this need only partially. To the best of the author's knowledge, none combines domain-specific dataset generation, refusal-aware evaluation, and a configuration-based workflow (Section 4.1). Second, such benchmarks increasingly rely on LLM-as-a-judge metrics [5], [6], whose verdicts cannot be assumed reliable. Judges should be validated against task-specific human judgments before their scores are trusted [7].

This thesis addresses both gaps. It contributes a reusable evaluation toolbox for domain-specific RAG systems (Chapter 4). It further validates the majority of the toolbox's judge-based metrics against human expert annotations (Chapter 5). It applies the toolbox to identify a deployable configuration of an improved RAI Copilot (Chapter 7). Finally, it validates whether the improved Copilot can handle the expected load of concurrent users in its intended deployment environment (Chapter 8).

1.2 Objectives

From the project assignment and the gaps stated above, four objectives were derived.

O1: Design and implement an evaluation toolbox for domain-specific RAG systems that (a) generates a synthetic Q&A dataset from an arbitrary PDF corpus, (b) evaluates answers of an external system under test (SUT) across refusal, retrieval, and generation metrics, and (c) is system-agnostic and fully configurable without code changes.

O2: Validate the toolbox by screening its generated dataset and comparing its judge verdicts against human expert annotations. Recommendations for conducting such annotation studies should further be derived from the findings.

O3: Implement an improved version of the RAI Copilot whose features operationalize responsible AI principles, including collaborative knowledge-base governance, metadata management, and verifiable source attribution.

O4: Evaluate the improved Copilot by benchmarking different RAG pipeline configurations against explicit acceptance criteria using the implemented evaluation toolbox. Its technical reliability should further be validated through automated tests and load tests at different concurrency levels.

The thesis is structured in four parts. Part I maps the underlying Design Science Research (DSR) activities to the chapters and lays the theoretical foundation for the subsequent parts. Part II addresses O1 and O2, and Part III addresses O3 and O4. Finally, Part IV synthesizes the findings.

1.3 Use of Generative AI

GitHub Copilot and Claude Code were used to identify errors and code improvements for both the evaluation toolbox and the RAI Copilot. Claude Code was further employed to design and improve plots. During the writing process, Claude was used to enhance the quality of the text. All generated output was reviewed by the author, and the design decisions, experiments, and conclusions are the author's own.

Chapter 2

Methodology Overview

This thesis contributes two artifacts: an evaluation toolbox for generating synthetic datasets and evaluating RAG systems, and an improved version of the RAI Copilot. Both artifacts were designed, implemented, and evaluated following the DSR paradigm, which is described in Section 2.1. Section 2.2 places the thesis within the preceding specialization projects. Finally, Section 2.3 gives an overview of the activities carried out for each artifact. Concrete design decisions, settings, and results are reported in the chapters referenced in the following sections.

2.1 Design Science Research Process

DSR produces artifacts that either address problems for which no satisfactory solution exists or improve existing solutions in terms of effectiveness or efficiency [8]. A widely cited framework for executing DSR was proposed by Peffers et al. [9]. Their process model structures projects into six activities: problem identification and motivation, objectives of a solution, design and development, demonstration, evaluation, and communication.

Peffers et al. [9] note that a project does not need to begin with the first activity. This thesis enters the process at the second activity. The problem was already identified in the preceding projects, and the objectives of both artifacts were derived from the defects and gaps observed in those phases. Table 2.1 maps each activity to the part of this thesis in which it is carried out for each artifact. The first activity is listed for completeness.

DSR activity	**Evaluation Toolbox**	**RAI Copilot**
Problem identification and motivation	Gaps in existing RAG evaluation frameworks (Section 4.1); defects of the VT2 pipeline	Defects and user feedback from VT2 version (Section 2.2); advanced RAG techniques (Section 3.1.3)
Objectives of a solution	Table 4.1, Table 4.2	Table 6.1
Design and development	Chapter 4	Chapter 6
Demonstration	Chapter 7	Chapter 7
Evaluation	Chapter 5	Chapter 7, Chapter 8
Communication	This thesis	This thesis

Table 2.1: Mapping of the six DSR process model activities [9] to the chapters of this thesis.

The benchmark reported in Chapter 7 uses both artifacts at once, but in different roles. For the RAI Copilot, it is both a demonstration and an evaluation. It is a demonstration because it

shows the system in use to identify an optimal hyperparameter configuration. It is an evaluation because the resulting scores quantify how well each configuration satisfies the objectives defined for the system. For the toolbox, the same benchmark constitutes a demonstration only. It applies the artifact to a realistic system without assessing whether its measurements are reliable. The validity of the toolbox itself is then evaluated in Chapter 5.

2.2 Preceding Projects

This thesis builds upon three preceding projects, which were all carried out by the author of this thesis. This section briefly describes the outcome of each project.

Supplementary Event 1

The objective of Supplementary Event 1 (EVA1) was to demonstrate the technical feasibility of the RAI Copilot and to establish a baseline architecture for the subsequent project phases. The resulting prototype indexed PDF documents in a local Chroma vector database and implemented a simple RAG pipeline built with LangChain. The pipeline consisted of a dense retriever and an LLM that generated answers based on the retrieved chunks. The application was implemented with a Vue.js web client and a FastAPI web server, a stack that the current version retains. Both the embedding model and the generator LLM were accessed through OpenAI's application programming interface (API).

Specialization Project 1

The goal of Specialization Project 1 (VT1) was to extend the prototype into a deployable multi-user system and to assess it in use. This phase introduced a MongoDB store for user accounts and conversations, and Qdrant as the vector database. The project further added user authentication, role-based access control, document access levels, and conversation history. The RAG pipeline was extended with query enhancement, maximal marginal relevance (MMR) retrieval, and reranking. A first benchmarking script was written as well. It evaluated 26 pipeline configurations using a manually authored question catalog. The dataset consisted of 43 single- and multi-turn questions derived from 12 PDF documents in the domain of RAI. An LLM judge rated each answer on a five-point ordinal scale according to its correctness and factual consistency with the retrieved documents. The usability of the system was assessed in a task-based study with two participants.

Specialization Project 2

Specialization Project 2 (VT2) pursued two objectives. First, to implement an improved version of the Copilot and assess its performance and usability. Second, to develop a benchmark framework tied to the Copilot in order to identify its best-performing RAG pipeline configuration. VT2 moved inference on-premise. The LLM, the embedding model, and the reranker were served with vLLM on the Zurich University of Applied Sciences (ZHAW) School of Engineering (SoE) compute cluster behind OpenAI-compatible interfaces. This removed the dependency on third-party providers and thereby made the system usable with confidential documents. The flexible response mode (Section 6.2.2) was introduced in this phase. The benchmark was extended as well. A new evaluation dataset was compiled from student-authored questions, with LLM-generated

reference answers and refusal flags. The latter marked questions that the SUT is expected to refuse. This dataset consisted of 120 questions derived from 50 PDF documents in the domain of RAI. The answer collection and evaluation were separated into two scripts. The evaluation script introduced seven judge-based metrics to assess the quality of the retrieved chunks and the generated answers. The evaluation script is the direct predecessor of the toolbox described in Chapter 4. The benchmark in this phase assessed 32 RAG pipeline configurations. VT2 further introduced a set of automated backend and end-to-end (E2E) tests, and a load test that Chapter 8 builds on. Finally, the updated version of the Copilot was assessed in a second task-based user study with six participants.

All projects described above formed the basis for the work presented in this thesis. Table 6.1 summarizes the key features of the Copilot after each project phase. In addition, Table 4.2 lists the changes made to the evaluation pipeline during this thesis. Table 8.2 compares the load test implementation in VT2 and in this thesis.

2.3 Project Activities

Following the description of the preceding projects, this section gives an overview of the activities carried out in this thesis. Both artifacts, the evaluation toolbox and the Copilot, underwent the same phases of design, implementation, and evaluation.

2.3.1 Design and Implementation

In the design and implementation phase, requirements were identified, translated into concrete design objectives, and finally implemented in the respective artifact. The resulting artifacts are presented in detail in Chapter 4 and Chapter 6.

Evaluation Toolbox

The objectives for the toolbox were derived from a gap analysis of existing RAG evaluation frameworks, reported in Section 4.1. A second source of objectives was a defect analysis of the evaluation pipeline developed in VT2. The shortcomings identified in that analysis, together with the changes made in response to them, are summarized in Table 4.2.

RAI Copilot

The objectives for the version developed in this thesis were derived from three sources. The first was a defect analysis of the VT2 Copilot. The second source was the benchmark results from the same project. The third was a review of advanced RAG techniques (Section 3.1.3), which identified retrieval and generation capabilities that the VT2 pipeline did not yet implement. Table 6.1 lists the resulting feature set alongside the state of the system after each preceding project phase.

2.3.2 Evaluation

Following the implementation, several empirical studies were conducted to evaluate both artifacts. All studies that involved the knowledge base operated on the same document corpus, consisting

of 50 English-language PDF documents on RAI. The corpus, totaling 2,539 pages, spans publicly and privately available regulations, guidelines, standards, and research papers. Document length ranged from 9–492 pages, with a median of 36 pages. The documents were the same as those used in VT2, with two documents replaced.

Furthermore, both artifacts were evaluated against synthetic Q&A datasets produced by the generation pipeline of the toolbox. Two such datasets were generated, differing only in random seed and size. First, a benchmark dataset of 536 items was used to compare different RAG pipeline configurations of the RAI Copilot (Section 7.1.1). Second, a separate validation set of 174 items was used exclusively for the human annotation study (Section 5.2.1). The reason for this separation is stated below.

Evaluation Toolbox

Two studies assessed the toolbox. Both analyzed whether the artifacts the toolbox produces can be trusted. The first concerns the generation pipeline. The author manually verified a sample of 50 items from the benchmark dataset in order to examine whether the synthetic Q&A pairs are fit to serve as ground truth for the Copilot benchmark (Section 5.1).

The second concerns the evaluation pipeline. A human annotation study compared the verdicts of five judge-based metrics and the refusal classifier against the labels of five annotators with domain expertise. The study estimated the reliability of the LLMs employed in the evaluation pipeline (Section 5.2). The five RAG configurations whose evaluated answers were annotated were selected to span the quality range observed in the Copilot benchmark. This ensured that agreement was estimated across the full spectrum of answer quality rather than on a narrow band of it.

RAI Copilot

The Copilot was assessed along two axes. The implemented toolbox was used to benchmark different RAG pipeline configurations. The toolbox had two roles in this benchmark. First, it was used to generate a synthetic dataset, consisting of 536 single-turn, multi-turn, multi-hop, and refusal questions. All questions were generated from the document corpus described at the beginning of this section. This dataset was then used to collect answers from the Copilot for each configuration. Second, the evaluation pipeline of the toolbox scored these generated answers for each configuration across two refusal, four retrieval, and six generation metrics. These scores were then analyzed to identify the best-performing configuration for the improved version of the RAI Copilot. The complete benchmark is described in Chapter 7.

Furthermore, the technical reliability and performance of the Copilot were assessed through an automated test suite and load tests. Both components were already introduced in VT2. The test suite was extended to cover the newly introduced features of the Copilot. The load tests were extended to assess the system's performance under increased load compared to the VT2 version. A detailed description of these system tests is provided in Chapter 8.

Order of Execution

The studies are reported in the order in which their results are needed, which is not the order in which they were conducted. Since Chapter 5 and Chapter 7 depend on each other, the actual sequence is clarified here.

The benchmark dataset was generated first and screened manually (Section 5.1), since every subsequent study rests on the assumption that it can serve as ground truth. It was then used to collect answers from 20 pipeline configurations of the RAI Copilot, which the evaluation pipeline scored in order to identify the best-performing configuration (Chapter 7). These scores also provided the quality ranking from which the five configurations for the annotation study were drawn.

The author screened the benchmark dataset, analyzed the resulting scores, and took part in the annotation study. Neither that dataset nor the answers scored against it could therefore be used to estimate judge reliability without risking contamination from prior exposure. A separate validation set was generated under identical settings, and the five selected configurations were re-run against it. The resulting answers were scored once more to obtain the judge verdicts against which the annotators' ratings were compared (Section 5.2.1).

A consequence of this order is that the judge reliability was assessed after the benchmark had been run. The resulting reliability estimates are nevertheless expected to transfer to the benchmark. The validation set was generated from the same corpus with an identical pipeline configuration, and differed only in the random seed and the number of items. In addition, the annotated answers stem from five configurations spanning the benchmark's quality ranking. Both the questions and the answers rated by the annotators can therefore be regarded as draws from the same distributions the judges score in Chapter 7. The outcome of the annotation study decides how much weight each metric carries when the benchmark is interpreted in Section 7.2. Metrics with strong agreement are used as acceptance criteria. Metrics with weak agreement are either excluded from the decision rule or retained with a conservative interpretation.

Table 2.2 summarizes all empirical studies conducted in this work and highlights where in this thesis each is reported in detail.

Method	**Purpose**	**Sample**	**Reported in**
Manual screening	Assess whether the synthetic dataset can serve as ground truth for the Copilot benchmark	Author ($n = 50$ samples)	Section 5.1
Human annotation study	Establish judge reliability of the evaluation pipeline	5 experts, 119 items	Section 5.2
RAI Copilot benchmarking	Optimize and compare RAG pipeline configurations	20 configurations, 536 items each	Chapter 7
System & load testing	Validate the technical reliability and performance of the RAI Copilot under load	–	Chapter 8

Table 2.2: Overview of the empirical studies conducted in this work.

Chapter 3

Background and Related Work

This chapter establishes the theoretical foundation for this thesis. Section 3.1 presents the fundamentals of RAG, its limitations, and the advanced techniques proposed in response to them. The same section also discusses responsible AI principles and their application to the RAG paradigm. Section 3.2 reviews methods for evaluating RAG systems, covering evaluation datasets, evaluation metrics, and existing open-source frameworks.

3.1 Retrieval-Augmented Generation

LLMs acquire linguistic and factual knowledge during pre-training by optimizing their parameters over large-scale training data [10]. This comes with several limitations, such as outdated knowledge, lack of domain-specific knowledge, and hallucinations [4], [11], [12]. While fine-tuning a pre-trained model can mitigate the first two issues to some extent, it does not outperform RAG on knowledge-intensive tasks [13]. RAG retrieves passages relevant to a user query from an external knowledge base and incorporates them into the response generation process [4], [10], [14]. The paradigm was originally introduced by Lewis et al. [3] as a jointly trained architecture that combines a parametric sequence-to-sequence generator with a non-parametric dense index over Wikipedia. Because the external knowledge acts as a non-parametric memory, it can be updated without retraining the generator and can accommodate long-tail or confidential information [14]. While Q&A remains the core task of RAG [4], it may also be applied to machine translation, text summarization or classification, and code generation or completion [14], [15].

This section first describes the stages of a basic pipeline. Then it discusses the limitations of this paradigm and the optimization techniques proposed in response to them. Finally, it closes with responsible AI principles for RAG.

3.1.1 Fundamentals of a RAG Pipeline

Gao et al. [4] define the earliest paradigm as Naive RAG, which consists of three stages: indexing, retrieval, and generation.

Indexing

Indexing converts a document corpus into a searchable representation. Source documents are first parsed into plain text, which is then partitioned into smaller chunks. Finally, the chunks are encoded into vectors that are stored in a suitable vector database.

Chunking is necessary because documents typically exceed both the context window of the generator and the input length of the embedding model [16]. The choice of the chunk granularity is a trade-off. Smaller chunks favor recall and efficiency over contextual sufficiency [16]. Larger chunks, on the other hand, provide more context at the cost of retrieval precision [17]. A common strategy is to split documents into fixed-size chunks, which is popular because of its simplicity and its effectiveness in many cases [18], [19]. Other strategies align chunk boundaries with the structure of the text, at the sentence, paragraph, or section level, or select chunk size and overlap adaptively [18].

Next, each chunk is mapped to a dense vector by an embedding model, which is trained so that semantically similar texts lie close together in the embedding space [20]. Chunks can also be indexed as high-dimensional sparse vectors (e.g., using BM25 [21]), which capture exact term frequencies. Sparse vectors are stored in an inverted index, which maps each term to the chunks in which it occurs [22].

Retrieval

During retrieval, the query is first encoded into a vector using the same embedding model as in the indexing stage. The retriever then selects the k chunks most likely to contain the information required to answer the query. Two common approaches are dense and sparse retrieval.

In dense retrieval, the query is mapped to a dense vector in the same embedding space as the indexed chunks. Similarity between dense vectors is computed using a decomposable similarity function, such as cosine similarity or the inner product [20]. This retrieval method can capture semantic similarity but may struggle with salient terms that occur rarely [20], [23].

In contrast, sparse retrieval models represent the query as a high-dimensional sparse vector. Ranking functions such as BM25 score a document by the frequency of the query terms it contains, weighted so that terms rare across the corpus contribute more [21]. These methods work well with very specific keywords but capture neither lexical variation nor semantic relatedness [20].

As with the chunk size, determining the optimal number of retrieved chunks k is challenging. Retrieving too few chunks may miss relevant information, while retrieving too many chunks may introduce irrelevant information [24].

Generation

In the final stage, the retrieved chunks are assembled into a prompt together with the user query and any system instructions. This prompt is then passed to a generator model, which produces the answer. Because the retrieved context enters through the prompt rather than through the model weights, the approach is applicable to black-box LLMs that do not expose their internal parameters [12].

Depending on the task, the generator is instructed either to ground its response strictly in the

retrieved chunks or to synthesize the retrieved context with its own parametric knowledge [4]. The model may be further instructed to attribute individual statements to the chunks they were derived from [25], [26]. Schreieder et al. [27] conclude that prompting remains the leading approach for attribution that does not require modifying the parameters of the generator.

3.1.2 Limitations of RAG

RAG mitigates but does not eliminate the failure modes of LLMs, and it introduces new ones. An analysis of three RAG systems identified seven recurring failure points: the answer is missing from the corpus; the chunk containing the answer is not among the top_k retrieved documents; it is retrieved but dropped during consolidation; it reaches the generator but is not used to generate the response; the output violates a required format; the response is at the wrong level of specificity; or it is incomplete [28].

Broadly, the underlying causes of these failures can be categorized into retrieval and generation challenges.

Retrieval Challenges

Limitations may arise from document indexing itself. The indexed documents may be outdated, lack domain coverage, or contain text that was incorrectly parsed [29]. For example, handling semi-structured data introduces two challenges. Text splitting processes can fragment tables and corrupt retrieval, while their presence in chunks complicates semantic similarity searches [4].

More generally, the performance of a RAG system is highly dependent on the quality of the retrieved chunks [15]. Retrieval often surfaces irrelevant, redundant, or semantically noisy chunks, which reduce the quality of the generated answer [30]. Furthermore, retrieval may miss relevant chunks or retrieve highly similar chunks that address the question only partially [31]. Such distracting documents degrade answer accuracy, especially if many of them are included in the context [32].

Naive RAG retrieves chunks based on the user's original query [4]. Therefore, ambiguous phrasing, domain-specific vocabulary, or abbreviations with several meanings may hinder effective retrieval [4].

Generation Challenges

A study by Jin et al. [33] concludes that generators prefer evidence that appears more frequently or that is consistent with their parametric knowledge. The authors further observe that models may favor incorrect parametric knowledge over correct evidence in retrieved chunks. Generators may also rely on their internal knowledge when the question cannot be answered based on the retrieved chunks [15]. In addition, an increased amount of misleading evidence in the retrieved context negatively affects the quality of the generated response [34].

The limited context window of the generator also constrains the amount of retrieved information that can be used to answer a query [14]. Providing too much context, on the other hand, may trigger the "lost in the middle" effect [35]. Model performance degrades when relevant information is positioned in the middle of a long input context rather than at its beginning or end.

3.1.3 Advanced RAG Techniques

In response to the limitations of the Naive RAG pipeline, several advanced techniques have been proposed. These advancements can be applied before, during, and after retrieval.

Pre-Retrieval Optimization

Contextual retrieval prepends an LLM-generated, chunk-specific summary of the surrounding document to each chunk before embedding [36]. Chunks may also be extended with document metadata, such as author names, the file name, or a timestamp [4]. To reduce manual labeling efforts, LLMs can be used to automatically extract keywords and micro-summaries from each chunk before indexing [18]. An approach that leverages this chunk metadata is metadata filtering. Keywords are extracted from the query and then used to filter the candidate chunks before similarity search, which reduces the search space [4].

Rewrite–Retrieve–Read inserts an explicit query rewriting step into retrieval pipelines [37]. A straightforward approach is to prompt an LLM to rewrite the query [37]. A complex query may further be partitioned into multiple simpler sub-queries [4]. Query rewriters may also be used to interpret preceding dialogue to produce context-aware queries [38]. RQ-RAG [39] instruction-tunes an LLM to dynamically choose between query rewriting, decomposition, and disambiguation. In contrast, HyDE [40] leverages an LLM to generate a synthetic document that answers the query. This hypothetical document is then embedded and used to retrieve relevant chunks.

Retrieval Optimization

While pre-retrieval techniques focus on enriching chunks and refining search queries, retrieval optimizations improve the search step itself. Hybrid retrieval performs both dense and sparse retrieval and merges the results into a single list. Reciprocal rank fusion (RRF) is a merging technique that yields one ranking without requiring comparable relevance scores across lists [41]. Several studies report that hybrid retrieval outperforms dense or sparse retrieval alone on retrieval benchmarks [16], [42]. MMR is another retrieval method that aims to increase the diversity of the retrieved chunks by selecting chunks that are both relevant to the query and dissimilar to the other retrieved chunks [43].

Post-Retrieval Optimization

An often-cited post-retrieval optimization technique is reranking [4], [14], [30]. Reranking rescores the retrieved chunks with a cross-encoder that attends jointly to the query and each passage [44]. Such models are often more accurate than the bi-encoder used for retrieval but too slow to apply to the full corpus [45]. An LLM may also be employed to remove low-quality or irrelevant chunks from the retrieved set to enhance the efficiency and effectiveness of downstream tasks [46].

For white-box generators that expose their internal parameters, retrieved information can be injected into the parameters themselves [15]. Beyond prompt-based injection, Su et al. [47] propose Parametric RAG, in which documents are encoded as low-rank adapters that are merged into the model weights at inference time. They report improvements over standard RAG, which injects knowledge through prompts, on several Q&A benchmarks.

3.1.4 Responsible AI in RAG Systems

A large number of organizations have published principles intended to guide responsible AI development. Jobin et al. [1] analyzed 84 such documents and identified convergence around the ethical principles of transparency, justice and fairness, non-maleficence, responsibility, and privacy. However, the authors also note that there is a significant divergence in how these principles are interpreted and implemented, why they are considered important, and which subjects they impact. One framework that translates such principles into concrete requirements is the Ethics Guidelines for Trustworthy AI of the European Commission's High-Level Expert Group [2]. The guidelines define seven requirements that a trustworthy system should satisfy. The requirements are: human agency and oversight; technical robustness and safety; privacy and data governance; transparency; diversity, non-discrimination and fairness; societal and environmental wellbeing; and accountability.

Trust-RAG Compass [48] defines six key dimensions of trustworthiness for RAG systems. Five of them correspond directly to a requirement of the Ethics Guidelines and are listed below. The sixth, factuality, concerns the accuracy and truthfulness of the generated response. It has no direct counterpart among the seven requirements and is addressed by the evaluation pipeline (Section 4.3) of the toolbox rather than by system features of the RAI Copilot.

- **Robustness**: requires the generator to extract relevant information from the retrieved chunks despite noise, varying levels of detail, different retrieval orders, and misinformation in these chunks.
- **Privacy**: requires RAG systems to protect personal and sensitive information during retrieval and response generation.
- **Transparency**: requires RAG systems to make both the retrieval of relevant information and its integration into the generated response clear and traceable to users.
- **Fairness**: requires RAG systems to mitigate bias throughout both the retrieval and generation stages.
- **Accountability**: requires RAG systems to verify that the retrieved chunks are accurate, relevant, and trustworthy, and to identify which chunks support which statements in the generated response.

These dimensions are listed since they informed the design of the RAI Copilot across all projects (Section 6.3.1).

3.2 Evaluation of RAG Systems

As stated above, a RAG system comprises several components, such as the embedding model, the retriever, and the generator. Determining the optimal configuration of these components is challenging due to the complex dependencies among them [49]. Es et al. [50] argue that automated evaluation is essential, as each of these components affects the performance of the system as a whole.

This section reviews the three ingredients any RAG evaluation requires: a dataset, a set of metrics, and an implementation. Both this literature review and the defect analysis of the pipeline developed in VT2 directly inform the implementation of the evaluation toolbox in Chapter 4.

3.2.1 Evaluation Datasets

The following paragraphs review open-domain, domain-specific, and synthetic datasets.

Existing Datasets

Van Elburg et al. [51] mention that most RAG benchmarks rely on open-domain or domain-specific datasets.

Open-domain datasets are built on general-purpose corpora. MTRAG [52] contains 110 human-generated multi-turn conversations (842 tasks) across four domains and was shown to be challenging even for state-of-the-art systems, in particular on questions that depend on earlier turns. Another notable dataset is HotpotQA [53], which was created to evaluate multi-hop reasoning, in which answering a question requires connecting information from multiple documents. It contains 113,000 Wikipedia-based question–answer pairs, sentence-level supporting facts, and comparison questions. This enables systems to be evaluated not only on answer accuracy but also on their reasoning process and explanations. CLAPNQ [54] is a human-annotated dataset of 4,946 long-form questions whose answers typically span two to three sentences and are grounded in a single gold passage. It includes both answerable and unanswerable questions, derived from Natural Questions [55] and grounded in Wikipedia passages. The dataset also provides a retrieval corpus, which allows retrieval, generation, and the full RAG pipeline to be evaluated separately.

The generality of these datasets poses limitations. LLMs may reproduce answers memorized from Wikipedia, since Wikipedia content is commonly used in pre-training [56]. However, assessing a RAG configuration requires a task on which the generator benefits from the retrieved information [57]. Open-domain datasets also often fail to reflect how users interact with a deployed system [58].

Domain-specific datasets address both concerns mentioned above. LegalBench-RAG [59] contains 6,858 expert-annotated query–answer pairs linked to precise source spans in legal documents, which enables fine-grained retrieval assessment. FinDER [60] is an expert-annotated financial dataset with 5,703 query–reference–evidence triples. It assesses generator models using short, ambiguous questions that strictly depend on external information retrieval.

However, datasets of both categories are expensive to produce. Creating questions, identifying supporting evidence, writing reference answers, and checking annotation quality require substantial time and domain expertise [52], [61]. Synthetic datasets avoid this cost by generating questions and reference answers from a given corpus. RGB [62] is an LLM-generated dataset based on news articles. It includes separate testbeds to assess the robustness to noisy retrieved documents, abstention when the retrieved documents are insufficient, integration of information from multiple documents, and the identification of factual errors in the retrieved documents. MultiHop-RAG [63] contains 2,556 LLM-generated multi-hop query–reference–evidence pairs constructed from news articles. This dataset also includes null queries, which require the generator

to abstain from answering because the answer is not present in the retrieved context. CRUD-RAG [64] broadens the scope beyond question answering and evaluates RAG systems on content generation, knowledge-intensive Q&A, error correction, and summarization. DragonBall [65] is another synthetic dataset, consisting of 6,711 query–reference–evidence triples across the finance, law, and medical domains.

The following paragraphs describe synthetic generation approaches in detail. Furthermore, they establish the foundation for the Q&A generation pipeline developed as part of the evaluation toolbox (Section 4.2).

Synthetic Generation Approaches

ARES [61] generates synthetic query–reference–evidence triples from passages in the target corpus using an LLM. The prompt for the generator is extended with at least five few-shot examples that map in-domain passages to queries and answers. Low-quality queries are discarded if the retriever cannot retrieve the passage from which the query was generated as its top result [61].

DataMorgana [66] generates synthetic Q&A datasets from a document corpus through two stages: configuration and generation. In the configuration stage, users define mutually exclusive question categories, such as question type, phrasing, and user expertise, and their desired distribution. During generation, a document and category combination is sampled, after which an LLM produces a question and answer constrained by the selected categories.

Teixeira de Lima et al. [58] invert the usual order to generate single-fact, summary, and reasoning questions. The input context is first summarized by an LLM into a theme. The LLM then extracts factual statements from the context. From these, summary statements are derived by combining several factual statements, and conclusion statements by inferring from the factual statements and the theme. A statement is then randomly sampled from one of these three lists, and a question is generated that is unambiguously answered by it. Finally, low-quality items are removed by LLM-based critiques.

DeepEval [67] follows a four-step pipeline. An LLM first generates query–reference–evidence goldens from the provided contexts. These are then filtered by a critic model, which scores each question regarding self-containment and clarity. Goldens that fall below a configurable threshold are regenerated a configurable number of times. The retained questions are then rewritten by an evolution step, in which one of seven transformations is sampled and applied to increase complexity. A final styling step adapts the questions and reference answers to a specified output format.

Van Elburg et al. [51] found that synthetic benchmarks generally agree with human benchmarks when ranking RAG systems with different retrieval settings. However, they are less reliable for comparing generator architectures. The authors state that synthetic questions tend to be simpler and more specific than human-written questions. Because they allow fewer valid answers, they may not reveal the stylistic differences between generators that become visible on more open-ended questions. In addition, because GPT-4o was used to generate the synthetic data, the benchmark may favor GPT-4o when it is included among the generators being compared.

3.2.2 Metrics

Yu et al. [68] organize RAG evaluation into retrieval, generation, and E2E evaluation. The main purpose of evaluating a retriever is to verify that it returns information relevant to the search query [69]. Generation evaluation assesses whether the generator produces coherent and relevant answers from the retrieved context [4]. Additionally, E2E evaluation assesses the system as a whole, since the interaction between retrieval and generation means that component-level results alone do not fully characterize overall performance [68].

Retrieval

Retrieval quality can be evaluated using set-based, ranking-based, and hit-based metrics [18]. Precision@k and Recall@k are set-based metrics computed over the top_k retrieved passages. Precision@k is the proportion of these k passages that are relevant, whereas Recall@k is the proportion of all relevant passages in the corpus that appear among them. Ranking-based metrics, such as mean average precision (MAP), mean reciprocal rank (MRR), and normalized discounted cumulative gain (NDCG), additionally account for the positions of relevant passages in the ranked list [70]. Hit@k records whether at least one relevant passage appears within the top_k results [70].

All of these metrics require relevance labels for query–passage pairs, which are costly to obtain through human annotation [71]. The authors mention a second approach that involves using the reference answer to obtain weak relevance labels. Claim recall measures the proportion of claims in the reference answer that are supported by the retrieved context [72]. Context precision, on the other hand, measures the proportion of retrieved passages that contain relevant reference claims [72]. Contextual relevance infers relevance from the query itself, by measuring how well the retrieved context focuses on the information needed to answer the query [73]. The effective information rate follows the same idea at the word level. It divides the number of words in matched reference passages by the total number of words in the whole retrieved context [65].

Generation

For short answers, exact-match metrics can be used to assess whether the generated answer is correct [69]. This matters where the answer must match word for word, as in code generation or fact retrieval [18]. Token-level Precision, Recall, and F1 measure the overlap between the tokens of the generated answer and those of the reference [56]. Embedding-based metrics, such as BERTScore [74] or SBERT [75], assess the semantic similarity between the response and a given reference answer. N-gram-based metrics, which measure overlap between contiguous word sequences, are another commonly used category [69]. Examples include BLEU [76] and ROUGE [77]. A drawback of these metrics is that they rely on lexical overlap and may fail to grasp the underlying meaning [6], [65], [78]. Furthermore, they require a reference answer, which is not always available [5].

Recent work has shifted toward using LLM-as-a-judge approaches to assess the correctness, relevance, and coherence of the generated answer [18]. Faithfulness evaluates whether the generated response is grounded in the retrieved context [73]. Answer relevancy measures whether the response addresses the user's query, without verifying its factual correctness [50]. Rau et al.

[56] further employ an LLM judge to assess the correctness of the answer given the query and a reference answer. Zhu et al. [65] introduce three additional LLM-based generation metrics, which are all computed on key points extracted from the reference answer. Completeness measures the proportion of key points covered by the generated answer, hallucination the proportion contradicted by it, and irrelevancy the proportion that are neither.

Generator evaluation need not be limited to answer quality. Gao et al. [25] propose two metrics to assess the citation behavior of a generator using a natural language inference (NLI) model. Citation recall verifies if a statement in the response is supported by at least one of its cited passages. Citation precision, on the other hand, identifies irrelevant citations.

Efficiency

In addition to retrieval and answer quality, the efficiency of a RAG system should also be evaluated. Relevant metrics include E2E latency, time to first token (TTFT), CPU and GPU utilization, and monetary cost [49], [70]. These metrics should be reported together with quality measures because optimization often involves a quality–efficiency trade-off. For example, adding a reranking stage can improve context precision and recall but increase the E2E latency [79].

3.2.3 LLM-as-a-Judge Evaluation

As the previous section showed, many metrics rely on LLMs to judge the quality of the retrieved context or the generated answer. Therefore, the following paragraphs provide a concise overview of the design and limitations as well as the evaluation of LLM-as-a-judge approaches.

Gu et al. [6] define LLM-as-a-judge evaluation as an in-context learning process in which an LLM is provided with the input, relevant context, and task-specific instructions that guide its reasoning and judgment. Evaluators can output a score, answer a yes/no question, conduct pairwise comparisons, or make a multiple-choice selection [6]. Zheng et al. [5] propose three evaluation paradigms: (1) pairwise comparison of two candidate responses, (2) single answer grading, and (3) reference-guided grading. Li et al. [80] further mention listwise evaluation, in which the judge ranks and evaluates multiple candidate items.

Limitations and Biases

Zheng et al. [5] identify several biases that can affect LLM-as-a-judge evaluation. Position bias occurs when the order in which candidate responses are presented influences the judgment. Verbosity bias arises when the evaluator prefers longer answers even when they are not more accurate or relevant. Self-enhancement bias occurs when the judge favors responses produced by itself.

Furthermore, the inherent generative randomness of LLMs can lead to judgments that do not adhere to the required output format and therefore complicate the automated extraction of judge verdicts [6]. In addition, the reliability of LLM judgments may be heavily undermined by hallucinations, which often manifest as fabricated evidence or misinterpreted facts [80].

Another limitation concerns reproducibility. Since results depend on the specific judge model, advances in LLM capabilities can invalidate earlier evaluations [69]. The authors further note that

the absence of a standardized prompt makes LLM-based evaluation results difficult to compare across studies. Furthermore, constructing suitable prompts requires substantial manual effort, and a given prompt may not be applicable to the varying requirements of different evaluation tasks [80].

Improvement Strategies

Several strategies have been proposed to improve the reliability of LLM-as-a-judge evaluation. An effective approach to help the LLM to better understand the evaluation task is few-shot prompting, which incorporates examples in the evaluation prompt [6], [81]. A related technique, mentioned by Gu et al. [6], is the decomposition of the evaluation task into smaller steps, each defined with its own criteria and constraints. To enhance interpretability, evaluators can be instructed to provide a reason for their judgments [69], [82].

A mitigation strategy for position bias is to query the judge twice with two answers in opposite orders and count a win only if the same answer is preferred both times [5]. Ye et al. [83] additionally recommend a prompt-based mechanism for analyzing evaluation templates and identifying potential sources of bias in their instructions.

The aggregation of multiple evaluation results for the same item, either across repeated runs of one judge or across several judge models, reduces the influence of random variation and model-specific bias [6].

Another strategy involves fine-tuning the judge model on meta-evaluation datasets constructed specifically for evaluation tasks to enhance the model's understanding of evaluation prompts and mitigate potential biases [6]. Prometheus 2 [84] is an open-source LLM evaluator that achieves high agreement with human judgments in both direct-assessment and pairwise-ranking tasks.

Meta-Evaluating LLM Judges

The validity, consistency, and reliability of LLM judges are established through meta-evaluation, which critically assesses the quality of such evaluators [80]. Bavaresco et al. [7] compared LLM and human judgments across 20 datasets and recommend that LLM evaluators should be calibrated and validated against task-specific human judgments before deploying them. A common approach for doing so treats the judge as an additional annotator and reports its agreement with human annotators [6]. A fundamental metric is accuracy, i.e., the proportion of judgments that match the human evaluator's annotations [80]. Agreement can further be measured through chance-corrected and correlation measures such as Cohen's κ and Spearman's ρ [6]. Furthermore, the consistency among the human annotators (inter-rater agreement) can be reported using Krippendorff's α [85].

3.2.4 Evaluation Frameworks

A number of evaluation frameworks implement the metrics discussed above.

Ragas [50] and DeepEval [86] provide RAG-specific judge metrics together with synthetic dataset generation capabilities. ARES [61] generates a synthetic, domain-specific training set from a target corpus and uses it to fine-tune lightweight judge models. The judges score query–passage–

answer triples on context relevance, answer faithfulness, and answer relevance. For more accurate results, the predictions of the judges are calibrated with a small set of human annotations. RAGChecker [72] uses claim-level entailment checking to assess both retrieval and generation. It decomposes text into claims and evaluates whether the claims are entailed in a reference text, which may be the retrieved context or a reference answer.

Other frameworks focus on hyperparameter optimization itself. AutoRAG [87] is an open-source framework that automatically constructs near-optimal RAG pipelines for a given dataset. It evaluates alternative modules across stages such as query expansion, retrieval, passage reranking, and text generation using stage-specific performance metrics.

FlashRAG [88] is an open-source toolkit for modular RAG research. It provides reusable datasets, retrievers, rerankers, refiners, generators, evaluation metrics, and predefined pipelines to benchmark and compare different RAG configurations. BERGEN [56] is an open-source library for standardizing and reproducing RAG experiments. Similar to FlashRAG, it provides retrievers, rerankers, language models, and evaluation metrics to compare different RAG configurations on configurable datasets. Like the frameworks discussed above, RAGPerf [49] evaluates the quality of the retrieved context and the generated response for a set of configurations. In addition, it captures performance metrics such as CPU and GPU utilization or throughput and provides workflows for stress testing RAG systems.

Part II

RAG Evaluation Toolbox

Chapter 4

Toolbox Design

This chapter introduces the two components of the evaluation toolbox: (1) a pipeline that generates synthetic Q&A datasets following a generate-then-validate approach, and (2) an evaluation pipeline that scores answers from RAG systems using LLM-as-a-judge and statistical metrics. An initial version of the evaluation pipeline was already developed in VT2 and considerably improved and extended within this thesis.

The chapter first summarizes existing evaluation frameworks and their limitations in Section 4.1. Section 4.2 then details the first component of the toolbox, the Q&A generation pipeline, while Section 4.3 describes the evaluation pipeline. Section 4.4 defines how the toolbox can be used and integrated into other projects, and Section 4.5 discusses the limitations of the toolbox.

4.1 Framework Analysis

Section 3.2.4 introduced the evaluation frameworks relevant to this thesis. This section compares them in order to justify why a dedicated toolbox was built instead of reusing an existing framework. Frameworks were included if they are open-source and target RAG systems specifically. Each framework was assessed from its documentation alone.

Table 4.1 compares the toolbox, described in this chapter, against nine evaluation frameworks along six dimensions. Two criteria separate the toolbox from the other frameworks. First, the toolbox is capable of identifying refusals and limits the evaluation of other metrics to non-refusal answers that actually carry a quality signal. Some frameworks, such as DeepEval or Ragas, allow users to define custom metrics, which may be used to determine whether a response is a refusal. Second, the toolbox is designed to generate various types of questions, including single-turn, multi-turn, multi-hop, and single- and multi-turn refusals. Most of the analyzed frameworks, however, only support part of this range. Another limitation of existing frameworks is that many require complex configuration or custom code to run an evaluation. The toolbox, on the other hand, is designed to be fully configurable through a YAML file and can be run entirely with a single command-line instruction.

However, the toolbox also has its own gaps. First, it does not support custom metrics out of the box. While the toolbox provides an interface for custom metrics, the implementation requires

code changes at several points. Second, the toolbox does not collect answers from the SUT itself. Users must collect answers from the SUT in the required format and specify them as input to the evaluation pipeline.

Framework	Cust. metrics	Data gen.	Question types	Refusal	Sweep	YAML + CLI
ARES [61]	✗	✓	∼	✗	✗	✓
AutoRAG* [87]	✗	✓	✗	✗	✓	✗
BERGEN* [56]	✗	✗	✗	∼	✓	✓
DeepEval [86]	✓	✓	∼	∼	✗	✗
FlashRAG* [88]	∼	✗	✗	✗	✓	✗
Ragas [89]	✓	✓	∼	∼	✗	✓
RAGChecker [72]	✗	✗	✗	✗	✗	✓
RAGPerf* [49]	✗	✗	✗	✗	✓	✓
TruLens [90]	✓	✗	✗	∼	✗	✗
Toolbox (this thesis)	∼	✓	✓	✓	✗	✓

Table 4.1: Analysis of existing RAG evaluation frameworks. ✓ = supported, ∼ = partially supported, ✗ = not supported. Frameworks marked with a * are not capable of evaluating answers from an external SUT. *Cust. metrics*: interfaces for custom metrics. *Data gen.*: generation of synthetic evaluation datasets. *Question types*: support for different types of questions. *Refusal*: metric calculation is conditioned on a refusal decision. *Sweep*: answer collection from the SUT is part of the evaluation run, so that several pipeline configurations can be benchmarked. *YAML + CLI*: evaluation is runnable entirely over YAML configuration and CLI commands, without code changes.

4.2 Q&A Generation Pipeline

Benchmarking a RAG system requires a test dataset whose questions and references are derived from the same documents from which the SUT retrieves its context. As stated in Section 3.2.1, several RAG-specific datasets exist. However, they are bound to their own corpora and may not be suitable to benchmark a system over a different knowledge base. The manual creation of a domain-specific dataset is time-consuming and expensive [58], [65].

Figure 4.1 shows the six stages of the Q&A pipeline, which is implemented in Python. The stages range from document processing to reference answer validation.

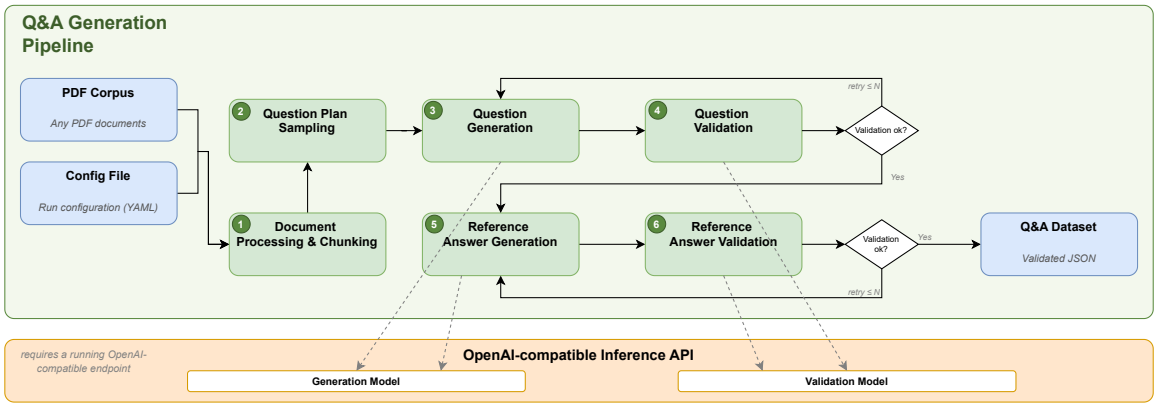

Figure 4.1: The Q&A generation pipeline of the evaluation toolbox. The pipeline produces LLM-validated question–reference pairs from a corpus of source PDFs.

As shown in Figure 4.1, the pipeline takes a path to a set of PDF documents as well as a YAML configuration file as input. The configuration file specifies a single generation run. It

includes dataset properties such as size, domain, and random seed; the question types and per-category rates (Section 4.2.1); the chunking granularity; and the generator and validator LLMs, including model, endpoints, temperature, sampling parameters, and prompt templates. Since every parameter that influences generation is captured in this file, a dataset can be closely reproduced or adapted to a new corpus without code changes. An example configuration file is provided in Section A.1.1.

The pipeline involves two LLMs: a generator and a validator, which are described in Section 4.2.2. The pipeline communicates with these models through the OpenAI-compatible chat-completions interface. Any inference service exposing this interface, such as the hosted OpenAI API or vLLM, can be used. To minimize the risk of self-enhancement bias, the generator and validator should be different models.

4.2.1 Question Taxonomy

The pipeline is designed to generate a wide range of question categories, which are described in the following list.

- **Single-turn**: Single-turn questions represent a standalone user query. Each question is grounded in a single document chunk and must be answerable using only the information contained in it. The question must be self-contained and ask directly about the subject matter, without referencing the source document.
- **Multi-turn**: Multi-turn questions embed the final question in a synthetic conversation history of T preceding user–assistant turns (configurable). The history simulates a user progressively exploring a topic, moving from broader questions toward the final one, and must neither state nor hint at the final question or its answer. The history must stay within the context of the source document chunk. If the chunk provides insufficient information to generate a coherent conversation, the conversation may range over the dataset's domain. The final question is grounded in a single chunk, like a single-turn question, but only needs to be understandable in the context of the conversation. This question category tests whether the SUT correctly resolves conversational context, such as follow-up questions.
- **Multi-hop**: Multi-hop questions require combining information from multiple document chunks. The pipeline first identifies bridges, i.e., keywords that appear in chunks of several distinct source documents (Section 4.2.2). A multi-hop question hinges on such a bridge keyword and requires a distinct fact from each of k chunks, such that no single chunk is sufficient to answer it. This question category tests whether the retriever gathers evidence spread across documents and whether the generator synthesizes it into a single answer.
- **Refusal**: Refusal questions are questions that a generator should refuse to answer. The pipeline supports three refusal categories: out-of-scope, harmful, and personal information. Out-of-scope questions are questions that are not related to the domain of the source documents. Harmful questions ask for information that could be used to harm others, such as hacking techniques. Personal information questions request sensitive information about a person, such as their home address or health information. For this category, the

pipeline targets fictional individuals to avoid generating questions about real persons. Each category must further be assigned a list of reasons (Section A.1.1). This ensures that the generated refusal questions are diverse and cover a wide range of different scenarios. The pipeline generates both single- and multi-turn refusal questions in a generation run. Multi-turn refusals verify whether the system still abstains when the preceding turns of the conversation are benign. Each refusal question is flagged (`should_refuse = True`) to distinguish it from the other question categories, which are flagged with `should_refuse = False`.

Except for refusals, each question is additionally assigned a type, i.e., factual, comparative, or definitional, that determines its interrogative focus. The generation and validation of each question category are described in detail in the following section.

4.2.2 Pipeline Architecture

In the following, each stage of the pipeline, visualized in Figure 4.1, is described in detail. As stated above, the pipeline takes a set of PDF documents and a configuration file as input.

4.2.2.1 Document Processing and Chunking

The first stage of the pipeline processes the provided set of PDF documents. Each document is first converted with Docling [91], an open-source document conversion tool. The structured conversion result is then chunked with Docling's `HybridChunker`[1]. This chunker tries to preserve logical document structures, such as headings and captions. The token limit for each chunk is configurable via the YAML file. The choice of this limit involves a trade-off. If chunks are too small, they may lack the context required to formulate meaningful questions and yield thin reference answers. If chunks are too large, they span several subtopics, which may result in overly broad questions. In practice, the limit should at least be large enough that content belonging to the same heading or caption is not split across chunks. Chunks that contain fewer than 48 tokens are discarded, as they may not contain enough information to generate a meaningful question. This stage finally returns a list of documents with their corresponding chunks.

4.2.2.2 Question Plan Sampling

After processing the input documents, the pipeline builds question plans according to the configuration file. A plan is an object that contains all information required to generate a specific question. This includes a unique question ID, the question category, the type, and category-specific properties. For each question category, the pipeline first determines the number of questions to generate based on the configured rates. The remaining budget is assigned to single-turn questions. The pipeline then constructs the corresponding number of question plans for each category.

For single- and multi-turn questions, the pipeline randomly samples a question type and a document chunk. To maintain the diversity of the dataset, the pipeline keeps track of already

[1] `https://docling-project.github.io/docling/concepts/chunking/`

sampled chunks to avoid sampling the same chunk twice where possible. For multi-turn questions, the pipeline additionally samples a number of preceding user–assistant turns T. This number is then used by the generator model to generate the history of T turns.

For multi-hop questions, the sampling behavior is different. If the pipeline sampled document chunks at random, they might be semantically too different to generate an adequate question. For that reason, the pipeline employs KeyBERT [92], which internally uses all-MiniLM-L6-v2[2] to extract the three highest-scoring keywords of at most two words each. Keywords are only kept if they exceed a relevancy threshold, which defaults to 0.4. The keyword-extraction parameters (number of keywords, n-gram range, relevancy threshold) are configurable. The defaults were chosen heuristically and not systematically tuned. After extraction, all document chunks are grouped by their keywords, and only groups that span at least $k_{\min}$ distinct source documents are retained as multi-hop bridges. To construct a plan, the pipeline randomly samples a bridge keyword and the number of hops k. The number of hops is drawn uniformly from the configurable range $[k_{\min}, k_{\max}]$ and clamped to the number of documents the bridge spans. It then samples k distinct source documents and a single chunk from each. This ensures that the sampled chunks are related through a shared keyword and that answering the question requires evidence from k different documents.

Finally, for refusals, the pipeline randomly samples whether the question should be single- or multi-turn, the number of preceding turns for multi-turn questions, and a refusal category and a corresponding reason. There is no need to sample a document chunk, as refusals do not require any information available in the knowledge base.

All sampling decisions described above are drawn from a single seeded random number generator (RNG). This makes the initial plan construction fully deterministic given the configured seed.

4.2.2.3 Question Generation and Validation

The sampled question plans are then passed to the generation stage. First, the pipeline partitions the plans by question category, as each category has a dedicated prompt template. This keeps each prompt focused on a single task and avoids long conditional instruction lists, which may be harder for the model to follow. Each template is filled with the properties of the corresponding plan, i.e., the sampled document chunks and question type as well as category-specific properties such as the number of preceding turns, the domain, the bridge keyword, or the refusal category and reason. The prompts instruct the generator model to produce questions that satisfy a fixed set of criteria: a question must be grounded in the provided chunk, self-contained, specific, and must neither contain multiple sub-questions nor meta-references to the chunk. For multi-turn plans, the generator additionally produces a conversation history, subject to the constraints described in Section 4.2.1. Plans for which generation fails, e.g., due to malformed model output, are retried with the same properties and dropped after a configurable number of attempts.

Every generated question is subsequently assessed by the validator model. The validator is configured independently of the generator, so a different model can be used to mitigate self-

[2] `https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2`

enhancement bias. For each category, a dedicated validation prompt restates the generation criteria and instructs the validator to reject the question if any criterion is violated. Since single- and multi-turn questions differ only in the conversational history, they are covered by the same validation prompt. For multi-turn questions, the validator additionally checks that the history is coherent and does not leak the final question or its answer. For multi-hop questions, the validator verifies that the question requires information from every chunk, and for refusals that the question matches its assigned category and reason. The validator returns a structured verdict consisting of a binary decision and a short justification. In addition to the LLM judgment, the pipeline performs deterministic structural checks on multi-turn questions, e.g., that the history is non-empty and ends with an assistant turn.

Rejected questions are retried for a configurable number of attempts. The pipeline samples a new plan with the same question category as the rejected one and repeats generation and validation. This decision is motivated by the observation that a rejection is often caused by the input material. A sampled chunk may consist of pure noise, such as reference lists or tables of contents, which may make it difficult to generate a question that matches the generation criteria. Resampling the plan while preserving the category therefore increases the chance of generating a valid question. A drawback of this approach is the loss of reproducibility, as the validation outcome of a question is non-deterministic. After the maximum number of retries is reached, persistently failed questions are excluded from the dataset. The pipeline further records the first-pass rate, the per-retry success rates, and the final drop rate as run statistics.

4.2.2.4 Reference Generation and Validation

The set of validated questions is then passed to the reference generation stage, which follows the same generate-then-validate approach. Generating references in a separate stage has three advantages. First, only rejected questions are discarded, which means that a valid question is not regenerated because the reference answer is rejected. Second, references are only generated for validated questions, which reduces generation costs. Third, per-stage pass rates are recorded, which allows quality issues to be attributed to a specific stage.

Reference answers are generated only for non-refusal questions, as the expected behavior for refusals is abstention rather than a factual answer. For each question, the generator model receives the question, its type, and the associated document chunks. For multi-turn questions, the model also receives the conversation history, which helps it understand the final question. The prompt instructs the model to answer the question using only the provided chunk(s).

The generated references are then assessed by the same validator model against five criteria: the reference must be grounded in the chunk(s), complete with respect to the question, concise, factually correct, and free of meta-references to the source documents. As with question validation, rejected references are regenerated and revalidated for at most the configured number of attempts. Persistently failing pairs are excluded from the final dataset. Finally, the same pass-rate statistics are recorded for this stage.

4.2.2.5 Dataset Output

As the final step, the validated question–reference pairs and refusal questions are assembled into the output dataset and serialized as a single JSON file. Besides the questions themselves, the file includes additional metadata about the generation run, such as the generation timestamp, the generator and validator models, the random seed, and the targeted and the actual number of questions. The run statistics collected during validation are stored in the file as well. Each item retains its ID, category, question type, conversation history, source chunks including their document of origin, reference answer, and both validation verdicts. Optionally, all rejected questions are written to a separate failures file together with the corresponding rejection reasons, which enables manual inspection and error analysis of the generation process.

This dataset can then be used to benchmark a RAG system, as described in Section 7.1.3. The schema of the dataset is provided in Section A.1.2.

4.3 Evaluation Pipeline

The second component of the toolbox scores the answers produced by a RAG system against a reference dataset. The pipeline computes both LLM-as-a-judge and statistical metrics. As stated at the beginning of this chapter, an evaluation pipeline was already developed in VT2. Table 4.2 highlights the contributions of this thesis to the evaluation pipeline.

Aspect	**VT2 version**	**This thesis**
Metrics	Judge-based metrics only	New: BERTScore, ROUGE, BLEU; rank agreement redesigned based on NDCG; refactored citation accuracy metric
Refusal handling	Integrated into generation metric prompts	Dedicated classification stage, metric gating, TRR/FRR metrics
Judge output	Free-text parsing	Structured output with per-metric response schemas
Judge variance	Single run	Configurable ensemble runs with mean aggregation
Error semantics	Failed metrics scored 0.0, biasing averages	Typed error records, excluded from aggregation
Reporting	Per-item scores appended to input files	Structured report with per-metric aggregates and refusal rates
Fault tolerance	Ad-hoc resume (skip computed, re-run errored)	Item-level checkpointing, repair runs with recomputed aggregates
Concurrency	Sequential metric loops	Concurrent metric computation, batched judge calls
Input validation	–	Pydantic schemas for dataset, answer files, and configuration
Model deployment	SLURM deployment coupled into the evaluator	Any OpenAI-compatible endpoint
Run configuration	CLI arguments	YAML configuration file
Packaging	Standalone Python script	Installable Python package with CLI

Table 4.2: Differences between the initial evaluation pipeline developed in VT2 and the redesigned version developed in this thesis.

Figure 4.2 shows the three stages of the enhanced evaluation pipeline, which is also implemented

in Python. All stages are explained in detail in Section 4.3.2.

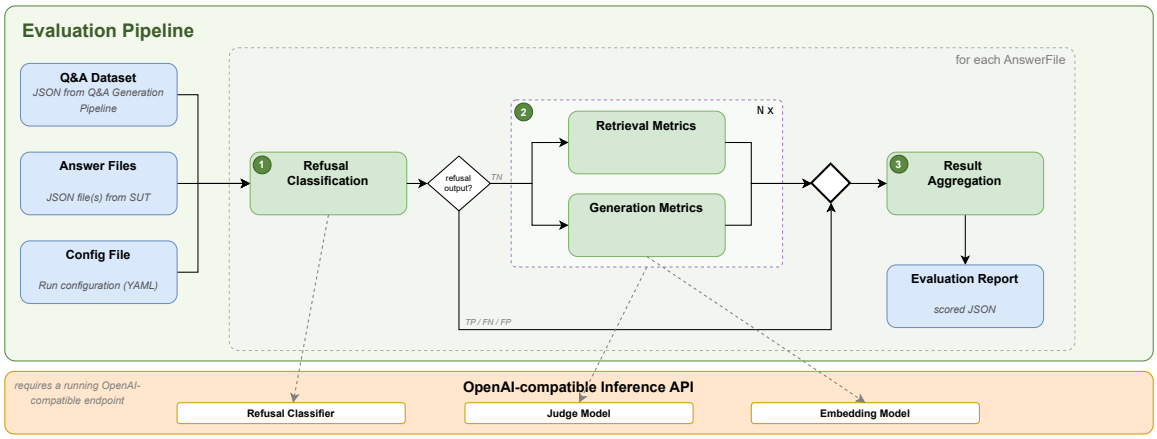

Figure 4.2: The evaluation pipeline of the toolbox. Given the reference question–answer pairs and the answers produced by the SUT, it computes LLM-as-a-judge and statistical metrics.

The pipeline requires three inputs: a reference dataset, a directory of answer files, and a YAML configuration file. The schemas of the dataset and answer files are defined in Section A.1.2 and Section A.2.2, respectively. As in the Q&A generation pipeline, the configuration file specifies a single evaluation run. It includes the metrics to compute; model configurations for the judge, the refusal classifier, and the embedding model; the number of ensemble runs; and additional parameters to configure specific metrics. An example configuration file is provided in Section A.2.1.

The pipeline interacts with the configured models through OpenAI-compatible chat-completions and embeddings interfaces.

4.3.1 Metric Taxonomy

The supported metrics assess three aspects of the SUT: refusal behavior, retrieval quality, and generation quality. All metrics are computed by an LLM judge or a refusal classifier, by embedding similarity, or by deterministic n-gram statistics. While some metrics require a reference answer (reference-based), others evaluate the system's output on its own (reference-free). Table 4.3 summarizes all supported metrics along these axes.

Several supported judge-based metrics follow the score definitions established by DeepEval [86] and Ragas [50]. In each case the aggregation formula is adopted unchanged, whereas the judge prompts were rewritten. For example, each prompt was supplemented with few-shot examples to guide the judge's reasoning.

As visualized in Table 4.3, some metrics require the retrieved chunks. These chunks must be exported with a corresponding identifier (e.g., doc_1, doc_2) and in rank order. The exported chunks are expected to be the ones the answer was actually generated from. For a system that applies a reranker, these are the reranked chunks, so the rank-sensitive metrics assess the ordering the reranker produced rather than the retriever's initial ranking.

Metric	Method	Reference-based	Chunks	Ensembled
Refusal				
True refusal rate	LLM classifier	✓		
False refusal rate	LLM classifier	✓		
Retrieval				
Contextual recall	LLM judge	✓	✓	✓
Contextual precision	LLM judge	✓	✓	✓
Contextual relevancy	LLM judge		✓	✓
Rank agreement	LLM judge		✓	✓
Generation				
Faithfulness	LLM judge		✓	✓
Answer relevancy	LLM judge + embeddings			
Citation accuracy	LLM judge		✓	✓
BERTScore	Embeddings	✓		
ROUGE-1/2/L	Statistical	✓		
BLEU	Statistical	✓		

Table 4.3: Supported metrics by evaluation aspect, computation method, and input requirements. Reference-based metrics require the ground truth label or reference answer from the dataset. Chunks marks metrics that require the retrieved chunks as input. Citation accuracy additionally requires citation markers in the answer. Ensembled metrics are computed N times per item and averaged.

Finally, all judge-based metrics require that the output of the judge LLM matches the expected structure. For that reason, judge responses are requested as structured output, where the expected format is defined by a per-metric Pydantic response schema.

4.3.1.1 Refusal Metrics

The reference dataset is expected to include a binary label for each question, which indicates whether the system should refuse to answer it. This is important because the evaluation pipeline assesses for each response of the SUT whether it is a refusal or not (Section 4.3.2.1). The label and the pipeline's refusal decision are then combined into one of four cases defined in Table 4.4.

	Predicted	
Ground truth	Refused	Answered
Should refuse	True positive (TP)	False negative (FN)
Should answer	False positive (FP)	True negative (TN)

Table 4.4: The system's refusal decision is combined with the dataset's ground truth label into one of four cases, which form the basis for the refusal metrics.

This classification enables the computation of two refusal metrics: the true refusal rate (TRR) and the false refusal rate (FRR). The TRR measures how reliably the system abstains from answering questions it should refuse. The FRR, on the other hand, measures how often the system incorrectly refuses questions it should answer.

$$\mathrm{TRR} = \frac{TP}{TP + FN}, \qquad \mathrm{FRR} = \frac{FP}{FP + TN} \tag{4.1}$$

4.3.1.2 Retrieval Metrics

Retrieval metrics assess the quality of the retrieved context independently of the generated answer. This isolates the retrieval step. A low score points to a retrieval failure regardless of the quality of the generated answer. Each supported retrieval metric is judge-based. In addition to the individual judgments, described in the following paragraphs, the judge also provides a short justification for each verdict. This facilitates the error analysis of individual evaluation results.

Contextual Recall

This reference-based metric, adapted from DeepEval [86], measures whether the retrieved context contains all information needed to reproduce the reference answer. The judge receives the question, the reference answer, and the retrieved chunks. It then splits the reference answer into individual sentences and issues a binary verdict per sentence on whether it is attributable to the retrieved chunks. Non-attributable sentences are labeled as partially grounded, not grounded, or contradicted. These verdict categories do not influence the score but are retained in the evaluation report to support error analysis. With S denoting the extracted sentences and $v_s \in \{0, 1\}$ the verdict for sentence s, the score is defined as:

$$\mathrm{CR} = \frac{1}{|S|} \sum_{s \in S} v_s \tag{4.2}$$

A score of 1.0 indicates that the retrieved context covers the complete reference answer, whereas a low score indicates missing evidence.

Contextual Precision

The second reference-based metric, adapted from Ragas [50], is contextual precision. It measures whether relevant chunks are ranked higher than irrelevant ones. The judge is queried for each retrieved chunk individually together with the question and the reference answer. It then issues a binary verdict on whether the chunk supports any part of the reference answer. Given all the verdicts $v_1, \ldots, v_K \in \{0, 1\}$ in retrieval rank order, the score is the rank-weighted average precision:

$$\mathrm{CP} = \frac{\sum_{k=1}^{K} \mathrm{Prec@}k \cdot v_k}{\sum_{k=1}^{K} v_k}, \qquad \mathrm{Prec@}k = \frac{1}{k} \sum_{j=1}^{k} v_j \tag{4.3}$$

The score rewards placing relevant chunks early in the ranking. A score of 1.0 represents a perfect retrieval ranking, where all relevant chunks are positioned at the very top of the retrieved list. Conversely, a low score indicates that the system either failed to retrieve any relevant context or placed the relevant chunks at the bottom of the retrieved list. If no chunk is judged relevant, the score is defined as zero.

Contextual Relevancy

This reference-free metric, adapted from DeepEval [86], measures how focused the retrieved context is. The judge receives a single retrieved chunk together with the question and decomposes the chunk into statements, where each sentence or list item counts as one statement. It then

issues a binary verdict per statement on whether it contributes information toward answering the question. Because the judge produces one verdict per statement rather than per chunk, this metric is the most output-intensive of the retrieval metrics. It therefore supports two per-metric overrides: the maximum number of chunks c to evaluate and the number of ensemble runs, which overrides the globally configured value. If c is set, only the first $c \leq k$ of the k retrieved chunks are evaluated. With S denoting the statements pooled across the evaluated chunks and $v_s \in \{0, 1\}$ the verdict for statement s, the score is defined as:

$$\text{CRel} = \frac{1}{|S|} \sum_{s \in S} v_s \tag{4.4}$$

A score of 1.0 means that every evaluated statement was judged relevant, whereas a low score indicates that the retrieved context contains a large amount of irrelevant information. Two properties of this definition should be noted. First, statements are pooled across chunks, so chunks contribute to the score in proportion to their number of statements. Second, since the cap retains the highest-ranked chunks, which are the most likely to be relevant, a value $c < k$ tends to overestimate the relevancy of the full retrieved context.

Rank Agreement

The second reference-free retrieval metric quantifies how well the retriever's or reranker's ordering agrees with an independent relevance assessment. The judge receives the question and all retrieved document chunks and assigns each chunk a graded label: directly answers, partially answers, or irrelevant. The chunks are presented to the judge in random order to avoid position bias. The resulting label is mapped to a gain $g \in \{0, 1, 2\}$. With $g_1, \ldots, g_K$ denoting the gains in the system's rank order and $g_{(1)} \geq \cdots \geq g_{(K)}$ the same gains sorted in descending order, the score is the NDCG:

$$\text{RA} = \frac{\text{DCG}}{\text{IDCG}}, \qquad \text{DCG} = \sum_{i=1}^{K} \frac{g_i}{\log_2(i+1)}, \qquad \text{IDCG} = \sum_{i=1}^{K} \frac{g_{(i)}}{\log_2(i+1)} \tag{4.5}$$

A score of 1.0 indicates that the system's ranking perfectly agrees with the judge's assessment. A low score, on the other hand, indicates that the system either failed to retrieve relevant chunks or placed them at the bottom of the ranking. Items whose chunks are all graded irrelevant yield no score, as they carry no ranking signal.

Rank agreement is related to contextual precision, since both assess the ordering of the retrieved chunks. Nevertheless, both metrics are supported, as they differ in what they compare the ordering against. Contextual precision judges each chunk in isolation against the reference answer, whereas rank agreement assesses all chunks in a single call and assigns graded labels relative to the question alone. This allows the judge to weigh the chunks against one another. Being reference-free, the metric is further suitable for datasets that do not contain reference answers.

4.3.1.3 Generation Metrics

In contrast to the retrieval metrics, the generation metrics assess the quality of the generated answer. The pipeline supports six generation metrics, of which three are judge-based. The other three metrics (BERTScore, ROUGE, and BLEU) are mostly deterministic. These metrics are included since they are inexpensive to compute and enable the comparison with other generation metrics. However, they must be seen as complementary, as they are only capable of capturing surface-level similarity.

Faithfulness

This reference-free metric, adapted from Ragas [50], measures whether the generated answer is fully grounded in the retrieved context. The metric is computed in two sequential steps. First, the judge receives only the question and the generated answer and decomposes the answer into standalone factual statements. In a second query, the judge receives these extracted statements together with the retrieved chunks. For each statement, the judge outputs a verdict $v_s \in \{0, 1\}$, a verdict category (fully grounded, partially grounded, contradicted, not grounded), and a short justification. The judge is instructed to set $v_s = 1$ only for statements it labels fully grounded. While the verdict itself is binary, the retained categories support a finer-grained error analysis. With S denoting the extracted statements and $v_s \in \{0, 1\}$ the verdict for statement s, the score is defined as:

$$\mathrm{F} = \frac{1}{|S|} \sum_{s \in S} v_s \tag{4.6}$$

A perfect score of 1.0 indicates that each factual statement in the answer is fully grounded in the retrieved context. A high score is therefore desirable in systems that are expected to only use information in the provided chunks without adding any hallucinated content. A low score, on the other hand, indicates that the answer contains a large amount of ungrounded information. This may indicate that the generation model tends to leverage its own parametric knowledge to generate the final answer.

Answer Relevancy

Answer relevancy, adapted from Ragas [50], is another reference-free metric, which measures whether the answer actually addresses the question. The judge generates N_q questions that the given answer could plausibly answer, where N_q is a configurable parameter. In the original implementation, the judge is queried N_q times to produce the questions. Manual inspection of the generated questions revealed that this often yielded near-identical questions, even at an elevated sampling temperature. Since the score averages over the N_q questions, such duplicates provide no additional information and the average degenerates to a single measurement. The pipeline therefore queries the judge a single time and instructs it to produce all N_q questions at once. The questions must differ in wording but ask for the same underlying information. Within a single response, the model conditions on the questions it has already generated and can vary the phrasing accordingly. With E denoting the configured embedding model, the score is the mean cosine similarity between the original question q and the generated questions $q'_1, \ldots, q'_{N_q}$:

$$\mathrm{AR} = \frac{1}{N_q} \sum_{i=1}^{N_q} \cos(E(q),\, E(q_i')) \tag{4.7}$$

A high score indicates that the answer is relevant to the original question. An answer that drifts from the question produces reverse-engineered questions dissimilar to the original one, which results in a lower score.

Citation Accuracy

This reference-free metric verifies the correctness of the citation markers in the response. It therefore requires the SUT to cite its sources using the marker format $[n]$, where e.g., [1] refers to the first chunk of all retrieved chunks. The judge receives the question, the answer, and the retrieved chunks. It identifies every citation in the answer and maps it to the corresponding chunk by its identifier. It then returns a binary verdict on whether that chunk supports the statement the citation is attached to. Failures are further labeled as contradicted, irrelevant, or missing document (chunk). Citations are evaluated per occurrence, so a statement carrying several markers contributes one verdict per marker. With C denoting the citations in the answer and $v_c \in \{0, 1\}$ the verdict for citation c, the score is defined as:

$$\mathrm{CA} = \frac{1}{|C|} \sum_{c \in C} v_c \tag{4.8}$$

A score of 1.0 indicates that all citations are correct. A low score indicates that a substantial fraction of the citations is not supported by the cited chunk. The metric measures citation precision rather than coverage, which means that uncited claims are not penalized. If an answer makes several claims but correctly cites only one of them, the score equals 1.0. Furthermore, the marker format is fixed in the judge prompt. Answers that use a different citation convention may not be recognized as citing at all. Answers without citations yield no score, so systems that do not cite are not penalized on this metric.

BERTScore

BERTScore [74] matches each token of the answer to its most similar token in the reference using contextual embeddings. It then averages the resulting cosine similarities, reported as precision, recall, and F1. Unlike the n-gram metrics below, it credits paraphrases that convey the reference content in different wording. The pipeline computes BERTScore with `bert_score` [74] using its default English model (RoBERTa-large) and baseline rescaling, which spreads the scores over a wider range and allows negative values.

ROUGE

ROUGE [77] measures overlap between the answer and the reference in terms of unigrams (ROUGE-1), bigrams (ROUGE-2), and the longest common subsequence (ROUGE-L). Each variant is reported as precision, recall, and F1, computed with `rouge_score` [93].

BLEU

BLEU [76] measures modified n-gram precision of the answer against the reference, combined with a brevity penalty that discourages overly short answers. The pipeline computes BLEU-4 with `sacrebleu` [94] and the default settings. The score lies in $[0, 1]$, where 1.0 denotes complete n-gram overlap with the reference.

4.3.2 Pipeline Architecture

Following the description of all supported metrics, this section explains how the evaluation pipeline computes them. The pipeline, shown in Figure 4.2, receives three inputs: a reference dataset produced by the Q&A generation pipeline or a dataset that matches the expected schema (Section A.1.2), a directory of answer files, and a configuration file. Each answer file contains the answers produced by the SUT for the entire dataset under one specific system configuration. Therefore, a single evaluation run can score several configurations of the same system. All inputs are validated against their respective Pydantic schemas to ensure that all information required for metric computation is available. These schema contracts decouple the toolbox from the SUT: any system that exports its answers in the specified JSON format can be evaluated without modifying the pipeline. The pipeline then processes each answer file independently. If no evaluation report exists for a file, its answers are evaluated one by one using the stages described below. If a report from a previous run already exists, the pipeline instead re-evaluates only the metric results that failed, as described in Section 4.3.2.4.

4.3.2.1 Refusal Classification

For each answer, the pipeline first determines whether the SUT refused to answer. For that task, a dedicated LLM receives the response and corresponding question and returns a binary refusal decision and a short justification. The classifier is further instructed to not classify an irrelevant or inconsistent answer as a refusal. For multi-turn questions, the model receives the system's rewritten, self-contained version of the question instead of the original. The classifier model is configurable independently of the judge, allowing the use of a smaller model for this comparatively simple classification task.

The predicted label is then combined with the dataset's ground truth refusal label into one of four cases, summarized in Table 4.4. The resulting label gates the metric calculation stage. Quality metrics are only computed for correctly answered questions (TN). This is a deliberate design decision, as the retrieval of relevant chunks or the quality of an answer is irrelevant if the system correctly refused (TP), should have refused (FN), or if it refused instead of answering (FP). For the other three cases, the label itself is the evaluation result, as it allows to compute the refusal metrics, described in Section 4.3.1.

4.3.2.2 Metric Calculation

For correctly answered questions (TN), the pipeline then concurrently computes all configured metrics (Section 4.3.1). For multi-turn items, which contain a rewritten question, the pipeline uses this rewritten version for all metrics that require the question as input. This ensures that

the judge receives a self-contained question, which encodes the conversational context. Each metric returns a result object containing the score and metric-specific details, such as the parsed judge responses. A failed computation does not abort the evaluation of the item. Instead, the exception is captured in the result object with an error flag and the error message. This enables the targeted re-evaluation of failed metrics, as described in Section 4.3.2.4. Furthermore, if an item lacks the input required by a metric, e.g., the retrieved chunks, the corresponding result object is flagged as unevaluable.

Unlike the Q&A generation pipeline, the evaluation pipeline does not include a validation stage to assess the quality of the judge-based metrics. Judge-based metrics are instead stabilized by ensembling. The judge is queried N times for each metric, where N is the configured number of ensemble runs. The final score represents the arithmetic mean of individual judgments. The goal of this approach is to reduce the variance of individual judgments. The ensemble approach is enabled for all judge-based metrics, except for the answer relevancy metric. This metric is not ensembled because it already averages over the N_q generated questions.

4.3.2.3 Result Aggregation

After all items of an answer file have been processed, the pipeline serializes the results into an evaluation report. This JSON report contains the per-item results, including the refusal verdict and all metric results. The report further contains aggregate statistics across all items. These statistics include the true and false refusal rates and, for each metric, the average score together with the number of evaluable items. The coverage information is required, as averages are only computed over items that produced a score. To make the report self-describing, it further records the configured judge, refusal classifier, and embedding models, as well as the evaluation timestamp. The full schema of the evaluation report is provided in Section A.2.3.

4.3.2.4 Error Handling

Evaluation runs are long-running and depend on external inference services. Therefore, the pipeline is designed to lose as little completed work as possible. First, all metric results for a single item are temporarily stored in a checkpoint file. If an evaluation run is interrupted and restarted, the pipeline resumes from the last checkpoint. The checkpoint file is deleted once the final evaluation report is written. At the metric level, the error results described above enable repair runs. If an evaluation report already exists for an answer file, the pipeline scans it for failed metric results and recomputes only those, leaving successful results untouched. If the refusal classification itself fails, the entire item is re-evaluated, as all metric results depend on the refusal label. After a repair run, the aggregate statistics are recomputed and the updated report is saved.

4.4 Packaging and CLI

The toolbox is packaged as the Python package `rag-eval-toolbox`, so that it can be installed with standard Python tooling into any project or environment. The package installs a single

console command, `rag-eval`, with one subcommand per pipeline:

```
rag-eval generate-qa --config qa_config.yaml \
  [--documents-dir <DIR>] [--output-dir <DIR>]

rag-eval evaluate --config eval_config.yaml \
  [--qa-dataset <FILE>] [--answers-dir <DIR>] [--output-dir <DIR>]
```

Both subcommands take the YAML configuration file described in the previous sections as their central argument. Integration with a SUT is file-based. The toolbox does not invoke the evaluated system itself. Instead, the two components communicate exclusively through the defined JSON schemas. The Q&A generation pipeline produces a dataset file, the SUT consumes it and produces answer files, and the evaluation pipeline scores these files against the dataset. This decoupling ensures that any system that can read and write the defined formats can be evaluated, regardless of programming language or framework.

4.5 Limitations

While the previous sections explained the design of the toolbox, this section examines the resulting boundaries of the toolbox. First, the scope conditions, which constrain the systems and datasets the toolbox can be applied to, are discussed. The section then concludes with an overview of limitations of the Q&A generation and evaluation pipelines.

4.5.1 Scope Conditions

A system-agnostic design is a key requirement of the evaluation toolbox. As stated in Section 4.4, the toolbox is not tied to any specific SUT. Nevertheless, the toolbox still requires inputs in a specific format and with certain properties.

The Q&A generation pipeline accepts a directory of PDF documents as input and imposes no further requirements on their structure or content. Its output, in contrast, follows a fixed schema (Section A.1.2), since the evaluation pipeline must be able to resolve questions, references, and metadata. This schema defines the boundary of the pipeline. SUT differ in how they expose their retrieval and generation interfaces. The toolbox therefore delegates answer collection to the SUT. An example of an answer collection pipeline can be found in Section 7.1.3, which describes how the RAI Copilot collects answers for a given dataset.

The evaluation pipeline imposes the corresponding requirement on the answer files. Beyond the generated answers themselves, the fields a system must export depend on which metrics are configured (Table 4.3). Metrics marked as reference-based require a reference answer from the dataset. Metrics marked as requiring retrieved chunks expect the system to export the chunks its answer was generated from, each with an identifier and in rank order. Two metrics impose conditions beyond these schemas. Citation accuracy assumes that the response contains markers following the $[n]$ convention. Furthermore, multi-turn items must include a self-contained question that encodes the conversational context. For external datasets in which the final question depends on the conversation history and no rewrite is provided, the judge may misinterpret the question, which invalidates the resulting scores.

4.5.2 Q&A Generation Pipeline

The following list summarizes the limitations of the Q&A generation pipeline.

- The pipeline does not assess the quality of individual document chunks. Therefore, a sampled chunk may consist largely of structural noise, such as tables of contents, reference lists, or conversion artifacts. The resampling retry loop described in Section 4.2.2.3 may mitigate the effect of such chunks. Nevertheless, a dedicated suitability check before sampling or a document cleaning stage would address this more directly.
- Every question and corresponding reference answer, except for multi-hop questions, is generated from exactly one sampled chunk. Semantically similar information may, however, also be present in other chunks of the corpus. If the SUT generates a correct answer grounded in a different chunk, reference-based metrics may falsely penalize the answer. Extending the pipeline by a post-generation stage that searches the corpus for additional chunks answering the question would yield a set of possible reference answers.
- Chunks for multi-hop questions are selected based on lexical keyword overlap. This does not guarantee that the connected chunks address the same concept. Since the generator is instructed to produce a question that requires information from all chunks, this may result in an inaccurate multi-hop question.
- The most fundamental limitation concerns the ground truth itself. Both the questions and the reference answers are produced by an LLM and assessed by a second LLM. No human verification is part of the pipeline. Errors in the reference answers propagate into every reference-based metric computed against the dataset.

4.5.3 Evaluation Pipeline

A second set of limitations concerns the evaluation pipeline and how the resulting scores should be interpreted.

- The refusal classification stage gates all quality metrics. A misclassification by the refusal LLM silently changes which items enter the metric calculation stage. Two configurations may therefore be compared on different averages. This limits the comparability of their scores where refusal rates differ substantially.
- The judge extracts and validates statements for faithfulness, contextual recall, and contextual relevancy. Therefore, the denominator is determined by the judge model rather than by the data. The judge may extract statements at a different granularity and thus assign different scores to an identical answer across different configurations. Furthermore, inaccurate extraction propagates into the verdicts of the validation stage.
- The evaluation pipeline does not support a judge-based metric to compare the generated answer against the reference answer. This axis is covered only by BERTScore, ROUGE, and BLEU, which capture surface-level similarity. An answer that is faithful, relevant, and correctly cited but incomplete with respect to the reference is therefore not penalized by any judge-based metric.

- Faithfulness, contextual recall, and citation accuracy retain a verdict category per statement. These categories do not influence the score. In the current design, a partially grounded statement is scored the same as a contradicted statement. Incompleteness and contradiction are therefore indistinguishable in the aggregate. Nevertheless, these categories support the error analysis of individual results. For example, for a system that is not expected to ground every statement in the retrieved context, a high number of partially grounded statements may be acceptable, whereas contradicted statements are not.
- If the judge belongs to the same model family as a generator under evaluation, its verdicts may favor that system's outputs. The judge should therefore be chosen independently.

A final limitation concerns both pipelines. The prompts for the Q&A generator and validator, as well as those of the evaluation judge, were developed iteratively and have not been varied systematically. All reported scores are therefore conditioned on these specific formulations. Quantifying the shift in metric scores using different prompt variants is left to future work.

Overall, the toolbox is only as trustworthy as the models it is configured with. Chapter 5 therefore validates the judgments of the evaluation pipeline against human annotations. This establishes the extent to which the results reported in the remainder of this thesis can be trusted.

Chapter 5

Toolbox Validation

The toolbox described in Chapter 4 produces two artifacts: a synthetic, domain-specific Q&A dataset and an evaluation report that scores answers produced by the SUT. Both are generated by LLMs and therefore cannot be assumed to be reliable. This chapter establishes how far each can be trusted.

Two studies are reported. Section 5.1 examines whether the generated Q&A dataset is fit to serve as ground truth for the benchmarking reported in Chapter 7. In order to establish the reliability of a subset of the judge-based metrics, Section 5.2 compares the verdicts of the judge and refusal classifier against the ratings of expert annotators. The order in which the studies were carried out is stated in Section 2.3.2.

Section 5.3 then compiles recommendations for conducting comparable annotation studies, and Section 5.4 discusses the limitations of both studies.

5.1 Screening of the Benchmark Dataset

The generate-then-validate design of the Q&A generation pipeline (Section 4.2.2) ensures that every item of the dataset is at least accepted by a validator LLM. The dataset used to benchmark the RAI Copilot, which is described in Section 7.1.1, consists of 547 judge-validated questions. To assess the dataset quality, the author manually screened a subset of this dataset.

Sampling

A sample of 50 questions, stratified by question category, was manually screened. The resulting subset contained 13 single-turn, 13 multi-turn, 12 multi-hop, and 12 refusal questions. Near-equal allocation was chosen because the categories are generated by different prompts and differ in difficulty.

Each item was assessed against six criteria. *Clarity* asks whether the question is unambiguous. *Answerable from source* asks whether the sampled reference chunks contain the information required to answer it. *Reference correct* and *reference grounded* ask whether the reference answer is factually correct and supported by those chunks. *No leakage* asks whether the question or, for multi-turn questions, the conversation history does not give away or hint at the answer.

Self-contained asks whether the question can be understood on its own without knowing what the reference chunks are about. Refusal questions carry no reference answer, no source chunks, and do not require an answer. Therefore, the three reference criteria and the leakage criterion do not apply to them. Each question received one of three verdicts: kept unchanged, edited, or dropped. A question was kept unchanged in most cases if it met every applicable criterion. Otherwise, it was edited or dropped.

Results

Table 5.1 reports the outcome of the screening. Of the 50 reviewed questions, 33 were kept unchanged, 6 were edited, and 11 were removed from the dataset. This resulted in a final dataset size of 536 questions.

Stratum	Reviewed	Kept	Edited	Dropped	Drop rate
Single-turn	13	7	1	5	38.5 %
Multi-turn	13	8	3	2	15.4 %
Multi-hop	12	6	2	4	33.3 %
Refusal	12	12	0	0	0.0 %
Total	50	33	6	11	22.0 %

Table 5.1: Outcome of the manual screening of 50 validated Q&A questions.

All refusal questions passed the screening, which is consistent with their acceptance rate of 94.8 % during the judge validation (Section 7.1.1). The 17 questions that were dropped or edited can be grouped into the following failure modes.

- **Unusable source chunk**: The reference chunk was either a reference list or a table of contents. Questions of this kind satisfy every criterion above, as the requested information does appear in the chunk. An example is "What is the title of the 2024 report by D. Atherton that surveys and analyzes 2023 incidents and responses related to deepfakes and child safety?". They were nevertheless dropped, as answering them requires only locating a string in a bibliographic listing rather than retrieving and interpreting content. For another question, the referenced chunks contained too little information to support a valid multi-hop question.
- **Invalid multi-hop bridge**: The observed cases were a bridge entity that denotes different things in the two chunks or a bridge forced onto unrelated passages. This highlights the limitations of the keyword-extraction mechanism employed to identify related document chunks. Even if the reference chunks share a keyword, that keyword may not refer to the same entity.
- **Leakage**: Some multi-turn questions had the answer stated in an assistant turn of the conversational history. Since the history is part of the prompt passed to the generator of the SUT, such questions can be answered without leveraging the retrieved chunks at all. Affected assistant turns were removed together with their corresponding user turns.
- **Remaining causes**: Three further items had individual defects. A question asking for

two items where the chunk provides more. One question that was neither clear nor self-contained. A multi-hop question where the answer was fully contained in a single chunk. The latter was re-labeled as a single-turn question and kept.

Two further observations illustrate the difficulty of specifying rejection criteria in a validator prompt. A number of multi-turn questions are self-contained, meaning that their conversation history is not required to answer them. Furthermore, some questions contain meta-references to the source document, such as "According to ..., as described in ...". Together with the defects mentioned above, these examples highlight a fundamental challenge in prompt design. It is difficult to cover all potential failure cases in the instructions, and harder still to anticipate them in the first place.

Two thirds (66 %) of the questions passed the manual screening without requiring any modifications. This figure increases to 78 % after minimal editing. Overall, the screening shows that the generate-then-validate pipeline produces largely valid questions, but does not reliably filter out edge cases. Manual screening therefore remains necessary to catch them and to guarantee a strict standard of quality.

5.2 Human Annotation Study

The previous section assessed the quality of the generated benchmark dataset. This section addresses the reliability of the LLM judge and the refusal classifier employed in the evaluation pipeline of the toolbox.

5.2.1 Study Design

The annotation study did not assess the judge verdicts obtained during the RAI Copilot benchmark (Chapter 7). This decision is justified by two main considerations. First, the author screened and corrected part of the benchmark dataset, as reported above. Second, the author analyzed the scores of all benchmarked configurations on this dataset. This previous exposure would have precluded an independent and unbiased rating. Therefore, the study operated on an independent validation dataset and set of scores, as described in the following paragraphs.

Validation Set

For this study, a second validation dataset comprising 174 questions was generated. The generation pipeline utilized the exact same configuration (Section A.1.1) as the benchmark dataset. The only altered parameters were the random sampling seed and the total number of generated questions. As this validation set was not screened manually, the conditions under which the judge was examined are no more favorable than those of the benchmark.

Configuration Selection

To generate scorable answers for the validation set, five configurations from the Copilot benchmark were selected. These were identified using a weighted composite of faithfulness, answer relevancy, and citation accuracy to deliberately span the observed quality range. The rationale for this diverse selection is that evaluating only champion-quality outputs would not allow for assessing

judge behavior on low-performance configurations. Furthermore, the selection was constrained by feasibility and component diversity. The chosen configurations span several generator and embedding models, and k_{final} was restricted to 4–8. k_{final} is the number of chunks passed to the generator. The configurations and their benchmark scores are listed in Table B.1.

These configurations were re-run against the validation set using the answer collection pipeline described in Section 7.1.3. The resulting answers were then scored by the evaluation pipeline to obtain the judge verdicts. As with the Q&A dataset, the judge configuration was identical to the one used in the Copilot benchmark (Section 7.1.4).

Participants and Metrics

Annotations were carried out by five expert annotators. The annotator pool consisted of four researchers from the RAI group at the ZHAW SoE, and the author.

Table 5.2 lists the six annotation tasks that were carried out. Each question was answered with either *yes* or *no*. The questions were designed to mirror the corresponding judge instructions as closely as possible.

Task	Shown	Units	Question shown to the annotator
Statement extraction	response & statements	97	Do these statements capture all claims of the response, without adding or changing any?
Faithfulness	chunks & statements	329	For each statement below: is it fully supported by the chunks alone?
Contextual recall	chunks & reference sentences	185	For each sentence below: can all of its factual claims be attributed to the chunks?
Contextual precision	question, reference, chunks	544	For each chunk below: does it contain at least one fact that matches or contributes to the reference answer?
Citation accuracy	question, response, chunks	274	For each sentence below: does the document cited by the highlighted marker support the statement it is attached to? [1] corresponds to doc_1, [2] to doc_2, etc.
Refusal	question & response	119	Did the system refuse to answer?
Total		**1548**	

Table 5.2: The six annotation tasks. *Shown* lists the context provided to the annotator. The same context is also provided to the LLM judge. *Units* counts the distinct binary annotation tasks the items yield.

Not every supported metric (Table 4.3) can be validated in this way. BERTScore, ROUGE, and BLEU are mostly deterministic and produce no verdict for a human to agree or disagree with. The score for answer relevancy is computed as the cosine similarity between the question and N_q questions generated from the response. Rank agreement scores an entire ranking with graded labels. Contextual relevancy decomposes every retrieved chunk into statements and assesses their relevancy, which would result in high annotation cost.

Statement extraction is included because of the two-step nature of the faithfulness metric. To calculate the faithfulness metric, the judge first decomposes the response into factual statements. In a second call, the judge assesses whether each statement is supported by the retrieved chunks. For this metric and for contextual recall, annotators labeled the units the judge extracted, statements for faithfulness and reference sentences for contextual recall, since unit-level

agreement requires identical units. The refusal task assesses the refusal classifier, which is a different model from the judge that evaluates the other metrics.

Rating Procedure

A tool was developed to collect the human annotations and deployed on the ZHAW SoE compute cluster. The web application consists of a FastAPI backend that serves HTML pages to the annotators. A screenshot of the annotation tool is provided in Figure B.1.

In total, 119 question–answer pairs including judge verdicts (items) were sampled evenly across the five configurations. As in the benchmark (Section 7.1.4), each judge-based metric was computed five times and averaged. Judge verdicts were selected from the ensemble run whose score was the median of the five. If several runs shared the median score, the first of them was selected. Nineteen items were assigned to all five annotators, so that the agreement between them could be measured. The shared set spanned all five configurations and all four question categories. In addition to the shared set, each annotator received four unique items from every configuration, to ensure that no annotator rated the output of a single system. In total, this resulted in 39 items per annotator. For each item, all tasks listed in Table 5.2 had to be completed. In combination with the shared items, this yielded 2,424 binary annotation tasks across all annotators.

For each annotator, the tasks were presented in consecutive blocks grouped by metric, following the sequence outlined in Table 5.2. The 19 shared items were presented first. This was a deliberate design choice to minimize context switching and reduce the cognitive load associated with continuously adapting to different tasks.

5.2.2 Judge Reliability

The author completed 100 % of their assignments (539 annotations) and the external annotators between 2.3 % and 15.3 % of theirs. In total, the external annotators contributed 107 annotations on the shared items, covering statement extraction, faithfulness, and contextual recall. Contextual precision, citation accuracy, and the refusal classification, on the other hand, were annotated by the author alone.

Table 5.3 reports the agreement between the judge and the author on five of the six tasks that the author rated. The statement extraction task is excluded, as this metric does not have a corresponding judge verdict. It is included in Table 5.4, which reports the inter-rater agreement. Both tables are discussed in detail in the following paragraphs.

Task	n	Agreement	κ	AC1	Human p	Judge p
Faithfulness	114	0.904	0.221	0.890	0.912	0.956
Contextual recall	68	0.897	0.721	0.837	0.735	0.779
Contextual precision	184	0.668	0.394	0.347	0.397	0.728
Citation accuracy	101	0.861	−0.019	0.841	0.871	0.990
Refusal	39	1.000	1.000	1.000	0.154	0.154

Table 5.3: Agreement between the LLM judge and the author on the units the author rated. *Agreement* is the share of units on which both assign the same verdict, κ is Cohen's kappa, and AC1 is Gwet's first-order agreement coefficient. *Human p* and *Judge p* give the share of positive verdicts of each rater. The table does not list the 33 statement extraction units, since the judge did not produce a verdict for this task.

Table 5.4 reports the agreement among the human annotators themselves, on the units that were rated by at least two of them.

Task	Units	Raters	Ratings	Agreement	α	AC1
Statement extraction	14	5	64	0.857	0.185	0.828
Faithfulness	46	2	92	0.870	0.188	0.845
Contextual recall	11	2	22	0.909	0.000	0.900

Table 5.4: Agreement among the human annotators on the units rated by at least two of them. *Agreement* is the observed pairwise agreement, α is Krippendorff's alpha, and AC1 is Gwet's first-order agreement coefficient. The three remaining tasks were rated by the author alone and admit no inter-rater estimate.

Statement Extraction

Statement extraction is the only task that was annotated by all five annotators, with 64 ratings on 14 decompositions. Furthermore, it is the only task that does not have a corresponding judge verdict. Therefore, the following numbers are drawn from Table 5.4. The annotators agree with each other in 85.7 % of the ratings, with AC1 = 0.828 and $\alpha = 0.185$. The reason for the low value of α is that most statements were accepted by all annotators. The consequence of this is further described in the next paragraph.

Overall, human raters accept the judge's statement decomposition in the majority of cases.

Faithfulness

Faithfulness reaches 90.4 % raw agreement over 114 statements. However, $\kappa = 0.221$ indicates only fair agreement [95]. This divergence is a known phenomenon, often referred to as the kappa paradox, which occurs when a dataset is highly imbalanced [96]. Of the 114 statements, 101 are accepted as grounded by both raters and only two are rejected by both (Table B.2). AC1 [97], which is designed for exactly this situation, yields 0.890.

The same pattern is observed among the human annotators themselves. They agree on 87 % of the 46 shared statements with $\alpha = 0.188$ and AC1 = 0.845. 91 % of the ratings fall on *grounded*. This indicates that judge–author agreement is comparable to the agreement between the human annotators themselves.

Because the verdicts are heavily concentrated on *grounded*, this metric may not be suitable for ranking configurations with similar faithfulness.

Contextual Recall

Contextual recall is the task on which judge and author agree most cleanly. Over the 68 rated sentences the agreement is 89.7 %, with $\kappa = 0.721$ and AC1 = 0.837. According to Landis and Koch [95], this corresponds to substantial agreement. Five sentences were rated as attributable by the judge but not by the author, and two the other way around.

The inter-rater agreement among the human annotators is 90.9 % with $\alpha = 0.000$ and AC1 = 0.900. The low value of α is caused by one rater who marked all sentences as attributable to the retrieved chunks.

Nevertheless, the scores indicate that the judge is a reliable proxy for human annotators on this metric.

Contextual Precision

Contextual precision is the task with the lowest raw agreement (66.8 %) over 184 chunks, with $\kappa = 0.394$ and AC1 $= 0.347$. The disagreement is entirely one-directional (Table B.2). The judge marks 61 chunks as relevant to the reference answer, while the author does not. There is no chunk that the author accepts and the judge rejects. The judge accordingly labels 72.8 % of all chunks relevant against the author's 39.7 %.

Contextual precision as computed by this judge is therefore best read as an upper bound on retrieval precision rather than as an estimate of it.

Citation Accuracy

Citation accuracy reaches 86.1 % raw agreement, but $\kappa = -0.019$ and AC1 $= 0.841$. The gap between the two coefficients arises from the same mechanism as for faithfulness, in a more extreme form. The judge returned *supported* for 100 of the 101 rated citations, a positive rate of 99 %. The author rejected 13 of them.

A manual inspection of the responses after the annotation study revealed the cause of the judge's near-perfect positive rate. Several responses did not contain any citation markers at all. Nevertheless, the judge invented citations and marked them as *supported*. This behavior is not consistent with the judge's instructions, which require an empty verdict list when the response contains no citations. Since the same pattern appeared in the RAI Copilot benchmark, it is examined further in Section 7.2.4.

For the purposes of this chapter, the citation accuracy score cannot be read as a measure of citation quality in this deployment.

Refusal Classification

The refusal classifier achieves perfect agreement with the author on the 39 rated items ($\kappa = 1.000$). Six responses were classified as refusals and 33 as answered. The result is plausible given the simpler nature of the task. The classifier judges a single property of the response as a whole and does not have to decide on individual statements or chunks, nor does it require the retrieved context to reach a verdict.

This is the strongest reliability evidence in the study. The refusal classifier is an important component in the evaluation pipeline, as it gates the metric computation. Metrics are only computed for items whose classifier label, combined with the ground-truth refusal flag, marks them as correctly answered (Section 4.3.2). The reliability of the classifier is thus a precondition for the reliability of the remaining metrics.

5.3 Recommendations for Human Annotation Studies

The design decisions of this study, alongside their identified limitations, generalize to comparable validations of an LLM judge. Based on these insights, the following recommendations are proposed.

- **Group the tasks strategically:** The ordering of tasks presents a trade-off between

cognitive load and data diversity. While grouping tasks by metric reduces context switching for the annotator, it inherently limits the diversity of the evaluated items. In time-constrained studies, it is recommended to present the tasks of each item together to ensure ratings are collected across a broader, more diverse set of metrics.

- **Match annotation instructions to the judge's criteria, not the raw prompt**: The prompts utilized in the evaluation pipeline comprise extensive instructions spanning several thousand words. Presenting these verbatim to human annotators would be infeasible. Therefore, questions (Table 5.2) should be designed to mirror the core instructions of the judge. Some questions may be further supplemented with concise hints to clarify the task. For instance, in the refusal task, annotators were explicitly told that "an attempted answer, even partial or incorrect, is not a refusal."
- **Provide a note field for edge cases**: Low-quality cases, such as responses missing a citation marker or containing citations in the wrong format, are not covered by a strict binary annotation task. It is recommended to include a note field to let annotators flag such cases. This not only facilitates the identification of edge cases, but also provides valuable insights into how annotators perceive these anomalies.
- **Decouple decomposition from evaluation**: Several metrics, such as contextual recall and citation accuracy, rely on a decompose-then-validate approach in a single judge call. Because errors in the decomposition step directly propagate to the final metric score, the reliability of the decomposition should be assessed separately. It is recommended to split decomposition and evaluation into two distinct judge calls. While this requires adjustments to the evaluation pipeline itself, it decouples the evaluation of the judge's extraction capabilities from its final scoring logic.

5.4 Summary and Limitations

Taken together, the two studies show that the artifacts of the toolbox are usable but not uniformly trustworthy. Roughly two thirds of the generated Q&A items required no correction. The reliability of the judges proved to be metric-dependent. Reliability ranges from perfect agreement (refusal classifier) and substantial agreement (contextual recall) to systematic, one-directional over-acceptance (contextual precision). In addition, the annotation study revealed that the judge invents verdicts for non-existent citations, which is inconsistent with its instructions.

In addition to the limitations of the toolbox design itself (Section 4.5), the studies have their own limitations.

Dataset Quality

The screening of Section 5.1 was carried out only by the author on 50 of 547 items. While the screening identified a number of defects, including invalid source chunks and answer leakage, it is not guaranteed that the remaining 497 items are free of such defects. Furthermore, the validation of judges used in the Q&A generation pipeline was not part of the human annotation study. This implies that the quality of both the benchmark and validation datasets depends on

the reliability of the validator LLM that assesses each generated question and reference answer.

Metric Coverage

The annotation study covered four of the seven judge-based metrics and the refusal classifier. The remaining metrics, contextual relevancy, rank agreement, and answer relevancy were not assessed. As they are part of the evaluation pipeline, future work should examine their reliability as well.

Single Rater for Half the Tasks

The annotation of 39 items was a time-consuming task. Since the available time of the external annotators was limited, contextual precision, citation accuracy, and the refusal classification were rated only by the author. For that reason, the study only reports the agreement between the judge and one person. Further annotations would have been valuable, especially for contextual precision, where the judge and the author disagreed at a rate of 33.2 %.

Moreover, the author also wrote the judge prompts. Agreement on these tasks may therefore be influenced by a shared interpretation of the criteria.

Single Ensemble Run

The judge verdicts came from a single ensemble run per item. The aggregate score the evaluation pipeline reports is computed as the mean over all five runs. Therefore, the agreement measured here characterizes a representative run rather than the aggregate. Majority voting across runs was not applicable, because the judge re-extracts the units in every run, so the verdicts of different runs do not align one-to-one.

Transferability

The results characterize one judge model and one refusal classifier, instructed with a specific set of prompts. The results are therefore not transferable to different judge models, revised prompts, or a corpus from another domain.

Part III

RAI Copilot

Chapter 6

RAI Copilot

The preceding part introduced a toolbox for evaluating RAG systems. This chapter describes the RAI Copilot, a RAG-based assistant, which was developed iteratively across three preceding project phases (EVA1, VT1, and VT2) and substantially extended within this thesis.

The chapter documents the system as it now exists. Section 6.1 presents the overall architecture, while Section 6.2 describes the E2E RAG flow, covering both the document-ingestion and query-time pipelines. Furthermore, Section 6.3 details the key features of the Copilot. This section also highlights in which project phase the key features were implemented. The chapter concludes with the limitations of the current implementation. The environment variables referenced throughout the chapter are collected in Table C.1.

6.1 System Architecture

This section provides a high-level overview of the RAI Copilot, describing its architecture, as well as the deployment topology.

6.1.1 Interaction of System Components

The RAI Copilot consists of a web client and server, a Qdrant vector database, a MongoDB database, and a set of external services.

Figure 6.1 provides a container-level view of the architecture using the C4 model notation [98], showing these containers and external services and how they interact. The diagram also highlights the chosen technologies for each container and external system.

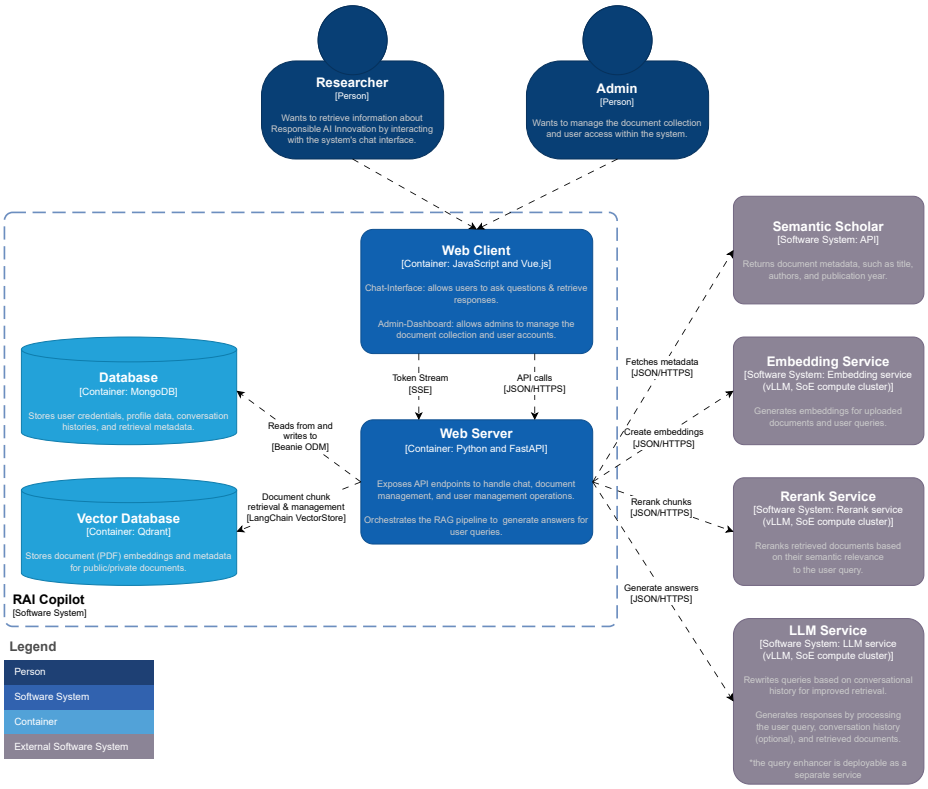

Figure 6.1: Container-level view of the RAI Copilot architecture in C4 notation. The web client interacts with the web server, which further interacts with external software systems (grey boxes), the document store, and the vector database.

Each of these containers and external services is briefly described below:

- **Web Client**: The web client, implemented with Vue.js, is the interface through which users interact with the Copilot. It also provides an admin interface for privileged users (admins) to manage the knowledge and user base. The admin interface further includes a retrieval testing view, in which an admin can issue a query and inspect the raw chunks returned by the retriever, along with their metadata and access levels. This offers transparency into the retrieval step and helps to diagnose retrieval quality independently of the generation LLM. A visual overview of the user interface and its key functionalities is provided in Section C.1.
- **Web Server**: The web server, implemented with FastAPI, exposes the application's API and orchestrates the core RAG workflow. The workflow consists of document ingestion, retrieval, and answer generation. Additionally, the server coordinates the data stores and the external model services.
- **Vector database (Qdrant)**: Stores the vector embeddings and metadata of document chunks and conducts the similarity search to retrieve relevant chunks at query time.
- **MongoDB Database**: Persists the application's structured data, which includes user accounts and conversation histories.
- **External services**: The embedding, query-enhancement, generation, and reranking models, served via vLLM on the compute cluster's GPU nodes. Additionally, the system

queries the Semantic Scholar API[1] to enrich document metadata.

6.1.2 Deployment Topology

Since the system is designed to operate over a user-provided knowledge base, which may contain private or otherwise sensitive material, data confidentiality is a key requirement for the application. Using third-party providers would transmit potentially confidential passages to an external service. For that reason, the system is deployed entirely on the ZHAW SoE compute cluster. The following list highlights how each individual component is deployed:

- **Web Client**: Served by nginx, running as a `systemd` service inside a virtual machine (VM).
- **Web Server**: Runs as a `systemd` service inside the same VM with access to an NVIDIA Tesla T4 GPU.
- **Vector database (Qdrant)**: Runs as a Docker container inside the VM.
- **Database (MongoDB)**: Runs as a native service on the VM, installed from the operating system's package repositories.
- **Language models**: Deployed separately on dedicated GPU nodes of the compute cluster, distinct from the application VM. All models (embedding model, query enhancer, generator, and reranker) are served with vLLM, which exposes OpenAI- and Cohere-compatible APIs.

This self-hosted topology keeps every document, embedding, and generated response inside institutional boundaries. Another advantage of this deployment strategy is that the application is only accessible from the ZHAW internal network.

6.2 Document and Query Pipelines

The RAG pipeline, which spans the entire flow from a raw uploaded document to a generated answer, forms the core of the web server. It is organized into two complementary sub-pipelines: the document pipeline and the query pipeline. The document pipeline chunks, embeds, and indexes documents into the knowledge base, while the query pipeline retrieves relevant content and generates a response at query time. Both pipelines are visualized in Figure 6.2 and described in detail in the following two sections.

[1] `https://www.semanticscholar.org/product/api`

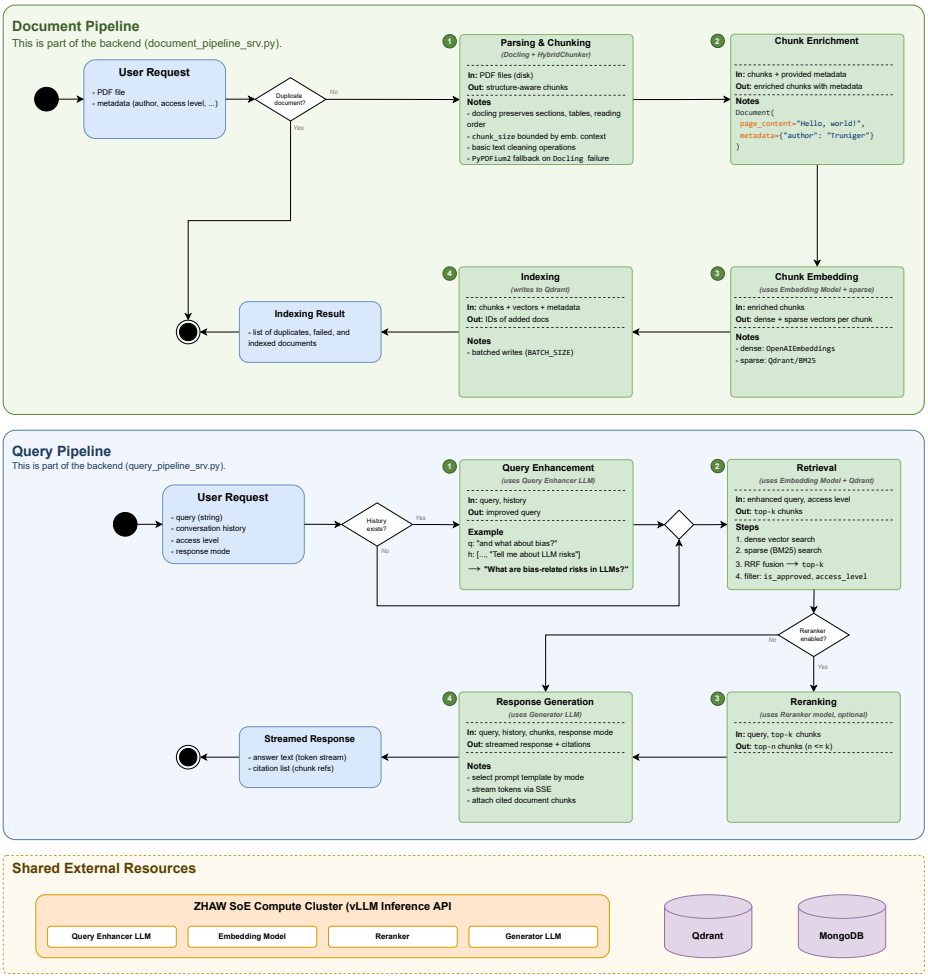

Figure 6.2: The document pipeline (top) handles document uploads. The query pipeline (middle) handles chat requests. External resources (bottom) are shared across both pipelines.

The bottom lane in Figure 6.2 shows all models and data stores that are shared across both pipelines. Both pipelines are instantiated once and reused across requests to ensure that shared resources and connections are only initialized once at application startup.

6.2.1 Document Pipeline

The document pipeline, shown in the top lane of Figure 6.2, transforms an uploaded PDF document into an indexed, searchable representation. It is triggered whenever an admin or collaborator adds PDF documents to the vector database, either as individual files or as a ZIP archive for bulk upload.

To avoid redundant processing and duplicate chunks in the knowledge base, the pipeline first computes a SHA-256 hash of each uploaded document. This hash is then compared against the hashes of already indexed documents to detect and skip the processing of duplicates. The hash is stored as part of the chunk metadata under the `file_hash` field. Non-duplicate documents then proceed through the following four steps.

Step 1: Parsing & Chunking

First, each document is parsed with Docling [91]. Docling was selected because it produces a structured document representation (`DoclingDocument`) that preserves layout elements such as headings, paragraphs, and tables.

Its models run locally, which keeps document processing on-premise in line with the privacy requirements of the Copilot. The structured output is then passed to the chunking stage, where the `HybridChunker` splits the document into chunks that respect its structure while staying within the token limit of the embedding model. First, the `HierarchicalChunker` creates one chunk for each detected document element. If these chunks exceed the maximum token limit, they are further split. Conversely, if chunks are undersized, they are merged with successive chunks if they share the same heading and caption. The maximum token limit for each chunk is configurable via environment variables.

Step 2: Chunk Enrichment

Then, each chunk is contextualized using Docling's `contextualize` method, which prepends the chunk's structural context, such as a section heading, to its raw text. Next, a LangChain `Document`[2] is created for each chunk, which contains the contextualized text and additional metadata, such as the document title, author, publication year, and access type. The full set of metadata is described in Section C.2.

All metadata fields are specified when a document is uploaded in the user interface (UI). For bulk uploads, the metadata is defined in a `metadata.json` file that specifies metadata for each document. This metadata is later used to filter the search results and to provide bibliographic information in the user interface.

Step 3: Chunk Embedding

Each chunk is then embedded into both a dense and a sparse vector representation, which together enable the hybrid retrieval described in Section 6.2.2. The sparse vectors are produced by the `Qdrant/bm25`[3] model, which captures exact keyword matches, while the dense embedding model is configurable.

Because the dense model is served through vLLM, any embedding model that vLLM supports, instruction-tuned or not, can be swapped in without code changes. The instructions for query and document embeddings are configurable via environment variables.

Step 4: Indexing

Finally, the embedded document chunks and corresponding metadata are stored in the Qdrant vector database. In the current implementation, the embedding (Step 3) and this step (Step 4) are executed together via a single vector database call. They are visualized as separate stages to reflect their distinct conceptual roles.

Beyond ingestion, the pipeline is also responsible for document management operations: users can retrieve and delete indexed document chunks and manage their metadata. Deleting a document

[2] `https://reference.langchain.com/python/langchain-core/documents/base/Document`
[3] `https://huggingface.co/Qdrant/bm25`

removes its chunks from the vector database. Metadata management is described in detail in Section 6.3.5.

6.2.2 Query Pipeline

The query pipeline, visualized in the middle lane of Figure 6.2, generates a response to a user query grounded in the indexed document chunks. It is triggered for every chat request. The query pipeline is implemented using LangGraph[4]. This open-source framework models the RAG workflow as a directed graph of nodes. Nodes are functions that encode the logic of each step in the pipeline, while edges determine the execution order of the nodes. The `State` is a shared data structure from which nodes read and to which they write. This object contains all the necessary information for the pipeline, such as the user query, the retrieved document chunks, and the generated response.

Step 1: Query Enhancement

A user query may be asked as part of an ongoing conversation. An example is a follow-up question that refers to a previous answer, such as "What are limitations of this approach?". Since this query is not self-contained, it must be reformulated into a question that includes the necessary context to understand it.

This is done by the query enhancer LLM, which takes the user query and the conversation history as input and outputs a self-contained reformulation where necessary. The number of previous conversation turns to include in the prompt is configurable via environment variables. The reformulated or original query (if no history is available) is then stored in the shared `State` and passed to the retrieval step.

As with the embedding model, the query enhancer is served via vLLM and can be swapped with any other model that vLLM supports without code changes. The prompt used to instruct the query enhancer model can be found in Section C.4.1.

Step 2: Retrieval

In this step, the query is embedded into both dense and sparse vector representations, using the same embedding models as in the document pipeline. These query vectors are then used to search the vector store for chunks most relevant to the query.

Dense retrieval captures semantic similarity, whereas sparse retrieval contributes exact keyword matches. The two result lists are fused using RRF (Section 3.1), weighting the dense and sparse lists equally. The search is restricted to the chunks the requesting user is authorized to access. A filter on the chunk access level, which is defined by the `access` metadata field, is applied during retrieval. This ensures that a user with public access retrieves only public chunks, whereas a user with private access retrieves from the entire knowledge base (Section 6.3.3). Furthermore, only approved chunks (Section 6.3.2) are considered for retrieval. Finally, the configurable top_k most relevant chunks are forwarded to the reranking step.

In addition to hybrid retrieval, the pipeline also supports dense and MMR search, selected via

[4] `https://www.langchain.com/langgraph`

environment variables.

Step 3: Reranking

After retrieval, the top_k chunks are reranked using a cross-encoder. This model takes the original or enhanced query and each retrieved chunk as input and outputs a relevance score for each chunk. The chunks are then sorted by their scores, and the top_n chunks are selected for the response generation step.

The reranking model can be swapped with any other model that vLLM supports without code changes. The reranking step can be disabled via environment variables. In that case, the top_k chunks from the retrieval step are passed directly to the response generation step.

Step 4: Response Generation

In the final step, the retrieved or reranked chunks, the (possibly enhanced) query, and the conversation history are assembled into a prompt and passed to the generation LLM, which produces the answer.

The system offers two response modes that the user can switch between in the UI. In the default mode (Section C.4.3), the model is instructed to strictly ground its answer in the retrieved chunks. If the information in the chunks is insufficient to answer the question, the model must explicitly state that it cannot answer the question, rather than drawing on its own parametric knowledge. Since this mode may result in many answer refusals, a flexible mode (Section C.4.2) was introduced in VT2. This mode allows the model to fall back on its parametric knowledge if the context is insufficient. However, the model is only allowed to do so if the question remains within the RAI domain.

Regardless of the mode, every claim drawn from a retrieved chunk must carry the corresponding citation marker (e.g., `[1]`). In the flexible mode, claims drawn from general knowledge are marked as such instead. These citation markers form the basis for the citation highlighting described in Section 6.3.4.

Finally, the response is streamed to the client token by token as it is produced. This reduces the perceived latency, which is particularly relevant for long answers the Copilot may produce.

6.3 Key Features

This section presents the key features of the RAI Copilot and highlights how these features operationalize responsible AI principles. The section is introduced with a feature map (Section 6.3.1) that maps the core features to the project phases in which they were introduced and to the principles they serve. The remaining sections then describe some of these features in detail.

6.3.1 Feature Map

Each project phase introduced new features and improvements to the RAI Copilot. For that reason, Table 6.1 maps key features of the Copilot to the specific project phases in which they were implemented. The list of features is not exhaustive, but rather highlights the most significant

additions and improvements made during each phase.

Table 6.1 further assigns each feature to the principle it primarily serves, using the requirements for trustworthy AI introduced in Section 3.1.4.

Feature	EVA1	VT1	VT2	MT	Principle
Baseline RAG Prototype	✓				–
User Authentication & Role-Based Access Control		✓			Privacy & data governance
Conversation History		✓			–
Query Enhancement		✓			Technical robustness & safety
Document Access Control		✓			Privacy & data governance
Response Rating		✓			Accountability
Dense & MMR Retrieval		✓			Technical robustness & safety
Reranking		✓			Technical robustness & safety
vLLM Model Support			✓		Privacy & data governance
Model Serving on the ZHAW SoE Cluster			✓		Privacy & data governance
Flexible Response Mode			✓		Transparency
ZIP Document Upload			✓		–
Hybrid Text Chunking (Docling)				✓	Technical robustness & safety
Hybrid Retrieval				✓	Technical robustness & safety
Collaborator Role & Document Approval Workflow				✓	Human oversight
Metadata Management				✓	Transparency
Citation Highlighting				✓	Transparency
Application Deployment on ZHAW SoE Compute Cluster				✓	Technical robustness & safety

Table 6.1: Key features of the RAI Copilot, mapped to the project phase in which each was introduced and to the RAI principle it primarily serves (Section 3.1.4).

Each feature implemented in this thesis addresses a gap that the VT2 version left open. Hybrid text chunking and retrieval respond to the retrieval quality limitations observed in the VT2 benchmark. Citation highlighting and metadata management address the need for traceability. The collaborator role enables collaboration, while the document approval workflow ensures that admin users retain final control over the vector database.

6.3.2 Collaboration and Governance

The RAI Copilot operates over a manually curated and potentially confidential knowledge base. Therefore, control over who may use the system and who may extend the knowledge base is a central concern. The Copilot addresses this through a role-based access control model combined with a document approval workflow. This keeps a human in control of both the user base and the contents of the knowledge base.

User Roles

The system currently distinguishes three user roles, which are enforced on the web server for specific operations.

- **User**: This is the default role. Users can interact with the Copilot and manage their own conversations. Their retrieval scope is additionally restricted by their document access level (Section 6.3.3).
- **Collaborator**: In addition to the user permissions, collaborators may extend the knowledge base by uploading documents. However, documents added by a collaborator become

retrievable only after an admin has approved them (see below).

- **Admin**: Privileged users with full control over the application. Admins manage the user base, as well as the knowledge base. This functionality is exposed through a dedicated admin interface in the web client.

Document Approval Workflow

The document approval workflow works as follows. Each document chunk carries an `is_approved` flag in its metadata, to ensure that the retrieval step (Section 6.2.2) only considers approved chunks. Documents uploaded by an admin are approved automatically, whereas documents uploaded by a collaborator are stored in the vector database with `is_approved` set to `false`. This ensures that they remain invisible to retrieval until an admin reviews and approves them. Newly proposed documents can be manually approved or deleted directly in the document overview in the admin dashboard. Proposed documents are indexed directly, because the default ingestion pipeline (Section 6.2.1) can be reused by changing only a single metadata field. Furthermore, this approach keeps a single source of truth without the need for a separate temporary staging store.

In summary, this workflow allows the knowledge base to be extended collaboratively. It further ensures that an admin retains final control over which sources the Copilot may use for generating its responses.

6.3.3 Privacy

As stated in Section 6.1.2, the knowledge base may contain confidential documents. Beyond the deployment-level isolation described there, the Copilot enforces document-level access control, so that sensitive document chunks are only retrievable by authorized users. Although this feature was already introduced in VT1, it is restated here because it directly operationalizes the responsible AI principle of privacy.

Access control is expressed along two dimensions:

- **Document access label**: Each document is labeled as either `public` or `private` through the `access` metadata field, which is set at upload time. Private documents represent sensitive sources that must not be exposed to unauthorized users.
- **User access level**: Each user is assigned a document access level (public or private) that determines which documents the user may retrieve. This access level is stored as part of the user entry in the MongoDB database and can only be changed by admin users.

The access levels are enforced during retrieval (Section 6.2.2) to ensure that users can only retrieve document chunks they are allowed to view. Since the filter is applied at the database level, private chunks are excluded from the candidate set entirely, and therefore cannot be referenced in an answer.

6.3.4 Citation Highlighting

When LLMs omit citations, their responses are interpreted as independently produced rather than drawn from external sources [99]. This has motivated evidence-based text generation,

which aims to increase verifiability by linking model responses to supporting evidence [27]. Such attribution allows users to verify whether each generated statement is supported by the cited source [100].

VT2 Version

In the VT2 version of the RAI Copilot, the generation LLM was instructed to cite every factual claim in its response by referencing the author, publication year, and source URL of the supporting evidence. This citation style proved unreliable, because the retrieved chunks themselves contain in-text citations of the underlying papers. As a result, the model occasionally copied such an inline reference instead of citing the source document of the chunk it actually based its answer on. Moreover, even correct markers provided limited benefit on their own, as the user must manually locate the referenced passage in the cited source document. Furthermore, the previous version did not allow the user to directly view the document chunk of the cited passage in the UI. Citation highlighting closes these gaps by resolving each citation to the specific chunk it is based on and visually highlighting that passage in the source document. The remainder of this section describes how this feature is implemented in the current version of the RAI Copilot.

Current Version

In the current version, as stated in Section 6.2.2, the generation model is instructed to emit a citation marker (e.g., `[1]`) for every factual claim in its response. The marker is the index of the chunk in the retrieved context passed to the model (e.g., `doc_1 = [1]`). The server resolves each index to the corresponding chunk in the vector database. Restricting the citation targets to these chunk IDs may mitigate the mis-citation issue of the previous version, as the model is now clearly instructed to reference a retrieved chunk as a whole. Furthermore, the model is prompted to place a citation block at the end of its response, which lists all cited chunk IDs in the following format: `<citations>[1, 6, 9]</citations>`. This list of chunk IDs is then used to determine the specific chunks on which the model may have based its answer. The identified chunks are stored in the metadata of the corresponding assistant response in the MongoDB database. The chunks must be persisted to display them in the UI when necessary.

The highlighting pipeline is executed when the user clicks on a citation marker in the UI. First, the referenced chunk is displayed in a pop-up window alongside document metadata, such as author, citation count, and source URL. Beyond attribution, this metadata supports the user in assessing the credibility of the cited source without leaving the application. Concurrently, the highlighting pipeline is executed, which consists of the steps described below. The highlights are computed on demand rather than eagerly for every response, which avoids unnecessary embedding computations for citations the user never inspects.

The main challenge of this feature is that the generation model typically paraphrases the retrieved content instead of quoting it verbatim. Consequently, exact string matching between the response and the chunk is insufficient, and the cited statements must be matched semantically instead. The highlighting pipeline consists of the following steps.

Step 1: Cited statement extraction: First, the response is segmented into sentences using the `en_core_web_sm` model of the spaCy [101] library. This model was chosen because it is

lightweight and optimized for CPU inference. Because the model treats citation markers as sentence boundaries, the markers are temporarily masked during segmentation. Each citation marker is then identified using regular expressions to attribute it to a sentence. Sentences without a marker are buffered and assigned to the next marker that occurs, since the generation model frequently places a single marker at the end of a multi-sentence claim. Conversely, sentences containing several markers are split at the marker positions, and each resulting clause is attributed to its respective marker. After removing Markdown formatting and very short fragments, this step yields the response statements associated with the clicked citation.

Step 2: Segmentation & embedding: Next, the cited chunk is segmented into sentences with the same spaCy model. The chunk sentences and the extracted response statements are then embedded using the configured embedding model. Currently, the same embedding model as in the document and query pipelines is used to avoid redundant model deployments.

Step 3: Semantic matching: For each extracted response statement, the cosine similarity to every sentence in the chunk is computed. The three most similar chunk sentences that exceed a similarity threshold are selected for highlighting. Both parameters are configurable directly in the UI and default to $k_{\text{hl}} = 3$ and a threshold of 0.75. These defaults were determined through inspection of highlighting results during development and not through a systematic parameter search.

Step 4: Rendering: Finally, the selected chunk sentences are wrapped in HTML tags that render as highlighted passages in the UI. Each highlighted passage carries a tooltip listing the response statements it may support. This allows the user to directly compare a claim with its supporting evidence. If no statement can be attributed to the clicked citation or an error occurs during the process, the original chunk is displayed without highlights. Figure C.3 shows the citation highlighting feature in the UI.

Overall, this feature represents an initial approach to enhancing transparency and verifiability in the RAI Copilot. However, there remain some limitations, such as the reliance on LLMs or inaccurate statement extraction, which are discussed in Section 6.4.

6.3.5 Metadata Management

As stated in Section 6.3.4, the metadata associated with the document chunks supports a lightweight credibility assessment. For that reason, it must be ensured that fields such as the citation count are kept up to date. In the previous version, metadata could only be set during the document upload, without an option to update it afterward. This version of the Copilot allows admin users to change all metadata fields (Section C.2) after document uploads. However, manual updates are error-prone and time-consuming. Therefore, the system integrates the Semantic Scholar API. Given the digital object identifier (DOI) assigned to a document, the API returns its source URL, title, publication year, citation count, journal, and author names. Semantic Scholar was chosen because it is freely accessible without a subscription, its corpus spans a wide range of publications, and it provides the citation counts required for the credibility assessment.

The metadata is retrieved from the Semantic Scholar API at three points:

- **Document upload**: If a DOI is provided during the upload, the remaining metadata fields can be fetched automatically, so that newly indexed documents are enriched without manual effort.
- **Metadata edits**: When editing the metadata of an individual document in the document overview, admin users can trigger an update that overwrites the stored fields with the current values from Semantic Scholar.
- **Bulk update**: Admin users can refresh the metadata of all indexed documents that have an assigned DOI in a single operation. This keeps volatile fields, most notably the citation count, up to date as the literature evolves.

Finally, the document overview allows admin users to download the metadata of all indexed documents. Combined with the ZIP bulk upload (Section 6.2.1), this export enables rebuilding the knowledge base without losing the curated metadata. This is helpful when all documents must be re-ingested after changing the chunking configuration or the embedding model.

6.4 Limitations

While the preceding sections described the RAI Copilot as it now exists, this section examines the boundaries of that implementation.

Document Cleaning

The current version indexes documents without dedicated content-level cleaning. Academic PDFs typically contain structural noise. Docling already omits some of it, such as page headers, footers, and watermarks. However, other elements, such as reference lists and tables of contents, are chunked and indexed like regular text. A table-of-contents chunk may be retrieved when a user asks for the definition of a specific term, since the term appears there as a heading. Such semantically related but answerless chunks occupy top-k retrieval slots and can degrade answer accuracy [102].

One possible mitigation strategy is to use an LLM for semantically aware document cleaning. Chen et al. [103] propose MARCUS, a pipeline that uses LLM agents to clean and re-chunk documents from the BRIGHT[5] dataset. SafeClean is an agent in this pipeline that is instructed to remove irrelevant navigation elements, such as buttons or footers, from HTML pages. On BRIGHT-Plus, the resulting cleaned dataset, NDCG@10 improved substantially for dense retrievers (e.g., from 15.2 % to 31.5 % averaged across domains for the BGE embedding model). While MARCUS was applied to web-crawled corpora, a similar approach could be adapted to clean PDF documents. However, LLM-based cleaning would further increase the ingestion latency described below.

Document Ingestion Latency

A further limitation concerns document ingestion latency. Because Docling's conversion pipeline relies on AI models for layout analysis and table structure recognition, ingesting documents is computationally expensive. Livathinos et al. [104] report a median conversion time of 0.79 s per page on an x86 CPU under the default pipeline settings, with up to 16.3 s per page at the

[5] `https://huggingface.co/datasets/xlangai/BRIGHT`

95th percentile. With an NVIDIA L4 GPU, the median drops to 0.11 s per page, with 2.08 s at the 95th percentile [104]. Since the RAI Copilot's web server runs on a VM with Tesla T4 GPU access (Section 6.1.2), document conversion can be expected to lie between the reported CPU and L4 figures. Indexing times measured during the benchmark on the machine described in Section 7.1.2 are reported in Figure D.2. Despite GPU acceleration, bulk uploads of long documents still take minutes to complete.

The CPU- or GPU-bound parsing and chunking are offloaded to worker threads. For that reason, document ingestion does not block the server's event loop. The application remains responsive to concurrent requests while documents are processed. The upload request itself, however, remains synchronous. This means that the uploading user waits until processing completes, as ingestion is not yet dispatched to a background job queue.

Model Availability

Since infrastructure resources within the ZHAW SoE cluster are shared, continuous system uptime cannot be guaranteed. Furthermore, the model deployments are constrained by maximum job times. These challenges pose an operational constraint, because the RAI Copilot requires uninterrupted model availability. To minimize manual intervention, SLURM jobs for model deployments are resubmitted automatically after the maximum job time is exceeded. Nevertheless, since data privacy is a key requirement of the Copilot, this limitation can be seen as a deliberate trade-off: self-hosting the models on ZHAW infrastructure keeps all data on-premise, at the cost of the guaranteed availability that a managed external inference provider would offer.

Citation Highlighting

Another limitation concerns the newly introduced citation highlighting feature, which depends on the generation model correctly emitting citation markers. If the model outputs the markers or the citation block (Section 6.3.4) in the wrong format, or omits them entirely, a claim cannot be attributed to a document chunk. Moreover, even correctly included citations do not guarantee that the cited chunk supports the claim.

Beyond the generation model, the highlighting pipeline itself introduces uncertainty. The identification of citation markers using regular expressions fails if the generator outputs markers in an invalid format, such as `[document_1]`. The similarity-based matching can fail in both directions: strongly paraphrased claims may fall below the similarity threshold, while thematically related but non-supporting sentences may exceed it. Since cosine similarity captures semantic relatedness, a highlighted passage is an indication of support, not proof. Consequently, the feature improves transparency, but the user remains responsible for verifying that the evidence supports the claim.

Several mitigations are conceivable, such as a feedback mechanism that would allow users to flag incorrect attributions. Instead of cosine similarity, the matching could rely on an NLI model. NLI is the task of classifying whether a hypothesis is entailed by, contradicts, or is neutral with respect to a given premise [105]. Such a model could be used to identify which passages entail each factual statement in the response.

Chapter 7

RAG Pipeline Benchmark

This chapter applies the evaluation toolbox of Chapter 4 to the RAI Copilot (Chapter 6). It serves two purposes. It identifies the pipeline configuration under which the Copilot performs best, and it demonstrates the toolbox on a system. The dataset, the judge, and the metrics used here are those examined in Chapter 5. The order in which the two chapters were carried out is stated in Section 2.3.2.

7.1 Experimental Setup

This section describes the design of the benchmark. It first introduces the dataset, the configuration space, and the execution environment. It then details the answer collection pipeline, which instantiates the RAI Copilot under a given configuration to generate responses. Next, it describes the configuration of the evaluation pipeline that scores them. Finally, it states how configurations were selected for the evaluation and how the champion was determined.

7.1.1 Benchmark Dataset

The benchmark dataset was generated with the Q&A generation pipeline of the toolbox (Section 4.2) from the 50 RAI PDF documents introduced in Section 2.3. The pipeline was configured to produce 550 items and returned 547 validated questions. The configuration file that was used to produce this dataset is provided in Section A.1.1. During the manual screening of the dataset (Section 5.1), a further 11 questions were removed.

Questions were generated with Qwen3-Next-80B-A3B-Instruct-FP8 (temperature 0.7, top-p 0.8) and validated with gpt-oss-20b (temperature 0). The sampling parameters of the generator were chosen based on the recommendations of the model provider. The temperature for the validator was set to zero to make its decision as reproducible as possible. Both models were served with vLLM on the ZHAW SoE compute cluster. The two models were selected under the following constraints. Both had to be openly licensed and servable with vLLM on the available hardware. Furthermore, the generator and validator had to be different to reduce self-enhancement bias. Finally, neither model could be one of the four candidate generators of the benchmark or the judge of the evaluation pipeline.

Table 7.1 provides an overview of the judge-validated benchmark dataset.

Dimension	Value	Count	Share
Question category			
Single-turn	One question, one source chunk	270	50.4 %
Multi-turn	One question preceded by a conversation history, one source chunk	106	19.8 %
Multi-hop	One question spanning two or three source chunks	105	19.6 %
Refusal	One question the system is expected to refuse, single- or multi-turn	55	10.3 %
Question type			
Factual	Asks for a specific fact in the chunk	182	34.0 %
Definitional	Asks for a definition of a concept	170	31.7 %
Comparative	Asks how two or more concepts relate or differ	129	24.1 %
Refusal	–	55	10.3 %
Refusal category			
Personal information	–	28	5.2 %
Harmful	–	15	2.8 %
Out of scope	–	12	2.2 %

Table 7.1: Composition of the 536-item benchmark dataset. The single-turn, multi-turn, and multi-hop categories contain no refusal questions, while refusal questions are either single- or multi-turn. Question types and refusal categories are reported over all finally retained 536 questions. Shares may not sum to 100 due to rounding.

Multi-turn questions include a conversation history of two ($n = 36$), four ($n = 31$), or six ($n = 39$) preceding messages (one to three user–assistant turns). The 105 multi-hop questions reference two ($n = 85$) or three ($n = 20$) chunks. The 55 refusal questions comprise 28 single- and 27 multi-turn questions.

The following statistics describe the dataset before the manual screening, that is, the 547 judge-validated items.

Of the 550 questions generated in the first pass, 72.4 % were accepted by the validator. After resampling the rejected questions, 549 questions were accepted within the configured 10 attempts. This raises the final pass rate to 99.8 %. Reference answers were only generated for the 494 validated non-refusal questions. 95.5 % of them were accepted on the first pass. After retrying the rejected references, 492 were accepted, which corresponds to a final pass rate of 99.6 %. Together with the 55 refusal questions, this results in 547 validated items.

In total, 776 questions were generated to obtain the 547 validated items. The share that yielded a validated item ranges from 94.8 % for refusal questions down to 55.8 % for multi-hop questions. For the latter, 197 generated questions were needed to obtain 110 validated items. Single-turn and multi-turn questions lie in between, at 73.5 % and 73.0 %, respectively. This indicates that multi-hop questions are the most difficult category for the generator to produce.

7.1.2 Configuration Space and Execution Environment

The following paragraphs describe the hyperparameter search space of the benchmark, the selection of candidate models, and the execution environment.

Hyperparameter Search Space

A configuration of the RAI Copilot comprises the tunable parameters of the implemented RAG pipeline: document chunking and indexing, retrieval and reranking, and answer generation. The benchmark varies the parameters that are expected to influence retrieval or answer quality across all these stages. Table 7.2 lists the resulting search space.

Component	Values	#
Sampled		
Generator LLM	gemma-4-31B-it, NVIDIA-Nemotron-3-Super-120B-A12B-NVFP4, Mistral-Small-3.2-24B-Instruct-2506, Apertus-8B-Instruct-2509	4
Embedding model	bge-m3, llama-nemotron-embed-1b-v2, snowflake-arctic-embed-l-v2.0, Octen-Embedding-8B, harrier-oss-v1-27b	5
Search method	hybrid, dense, MMR	3
Retriever k	4, 8, 12, 16, 24, 32, 64, 128	8
Reranker	none, mxbai-rerank-large-v2, bge-reranker-v2-m3, llama-nemotron-rerank-1b-v2	4
Reranker top$_n$	4, 8, 12, 16	4
Chunk size	128, 256, 512, 1024, 2048	5
Fixed		
Query enhancer LLM	Phi-4	1
Response mode	default	1
Context token budget	55,000	1
Access level	private	1
Sampling seed	42	1

Table 7.2: Hyperparameter search space of the RAG pipeline benchmark. Each generator was run with its recommended sampling parameters. Instruction-tuned embedding models were run with the recommended instruction prefix for query and document embeddings. The fixed components were held constant across all configurations. Full model identifiers and licensing information are given in Section D.1.2. All models were sourced from the HuggingFace Hub.

The sampled dimensions span the full pipeline. First, the chunk size determines the granularity of the index. The embedding model, search method, and retriever k determine which chunks are retrieved. Next, the reranker and its top$_n$ filter which chunks reach the generator. Finally, the generator determines how this context is turned into the final answer.

The remaining parameters were held constant. The query enhancer was fixed to a single model, as query rewriting is a comparatively narrow task. The response mode and the access level were fixed so that all configurations answer under identical conditions. To reserve space for prompt instructions, the context token budget was limited to 55,000, which is below the maximum context window of some generators. Finally, the fixed seed makes the configuration sampling reproducible.

Altogether, the sampled dimensions span 38,400 distinct combinations. Two constraints reduce this number to 21,840 valid configurations. First, the reranker top$_n$ must be smaller than the number of retrieved chunks, as a reranker that returns at least as many chunks as it receives filters nothing. Second, the retrieved context, estimated as the product of the chunk size and the effective number of chunks (k_{final}), must fit within the token budget of 55,000. k_{final} corresponds to the reranker top$_n$ if a reranker is used and to the retriever k otherwise. Configurations without a reranker are counted once, since the reranker top$_n$ has no effect on them.

Model Selection

The selection of the candidate models in Table 7.2 was driven by the following criteria. Models had to be available under a license that permits self-hosted deployment, servable with vLLM, and

fit on a single NVIDIA H200 GPU. Furthermore, models had to cover a range of parameter sizes and model families. Finally, some models were selected based on their performance in popular benchmarks, such as Arena AI [106], MTEB [107], and RTEB [108].

Before the benchmark was started, every candidate was deployed and tested individually. The tests verified that each model was servable over vLLM and succeeded in its intended role.

Execution Environment

All models presented in Table 7.2 were deployed on the ZHAW SoE compute cluster. Generator models were served with a single NVIDIA H200 SXM GPU with 140 GB of VRAM. The enhancer model was served with a single NVIDIA L40S GPU with 48 GB of VRAM. All embedding models and rerankers were served with a single NVIDIA A100 SXM GPU with 40 GB of VRAM.

The answer collection pipeline itself, including the Qdrant vector database in server mode, was executed on a local machine. The machine is equipped with an Intel Core i9-14900HX processor, 64 GB of RAM, and an NVIDIA RTX 4080 Laptop GPU with 12 GB of VRAM. The pipeline communicated with the deployed models over the network. The reported response times therefore comprise local retrieval, network round trips to the cluster, and model inference.

7.1.3 Answer Collection

As stated in Section 4.4, the toolbox does not invoke the SUT to generate answers. Therefore, collecting model responses is the responsibility of the benchmark itself. Figure 7.1 shows the five stages of the pipeline built for this purpose. In contrast to the toolbox, this pipeline is specific to the RAI Copilot, as it instantiates the components of the system directly. An answer collection pipeline was already introduced in VT2. Since this version differs substantially from the one implemented in VT2, it is described here as a new component.

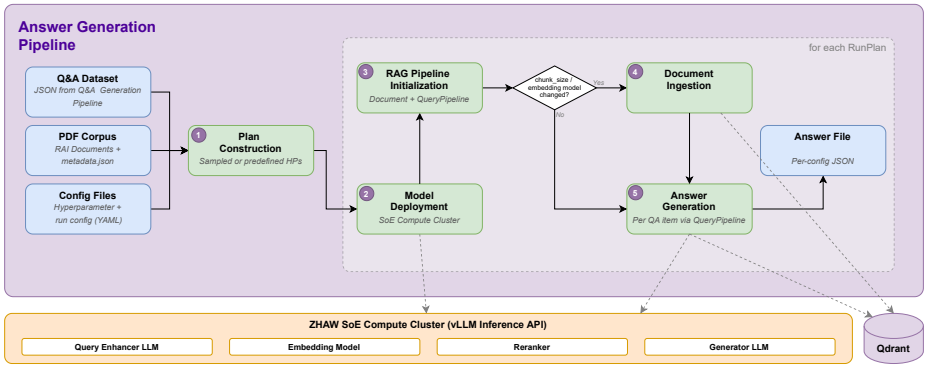

Figure 7.1: The answer collection pipeline used during the benchmark. For each sampled configuration, the pipeline instantiates the RAI Copilot with the corresponding components and generates answers for the benchmark dataset.

The pipeline requires three inputs: a dataset (Section 4.2.2.5), a directory of PDF documents and a JSON document metadata file, and a set of YAML configuration files. The set of PDF documents must be identical to the one used to generate the dataset. Otherwise, the source chunks referenced by the dataset would not be part of the system's index. One YAML file configures the benchmarking pipeline itself. It defines the input and output paths, the mode of

configuration selection, and the model and cluster settings. An example of this file is provided in Section D.1.1. For each configuration, the pipeline produces one answer file in the format required by the toolbox (Section A.2.2), which serves as the input of the evaluation pipeline (Section 4.3.2).

The pipeline supports two modes of configuration selection: predefined and random sampling. Depending on the selected mode, a second YAML file lists either every configuration for which answers should be generated, or the full hyperparameter search space from which configurations are sampled.

Plan Construction

The first stage builds a list of run plans. Each plan contains a unique ID, one hyperparameter configuration, and the path of the answer file it produces. The ID is a truncated SHA-256 hash of the configuration's parameter values. In the predefined mode, one plan is created per listed configuration. In the random sampling mode, a configurable number of configurations are drawn from the search space by a seeded RNG. Sampled configurations are deduplicated by their ID and checked against the two validity constraints stated in Section 7.1.2. Configurations that violate a constraint are discarded and redrawn.

Finally, all plans are sorted by generator, embedding model, reranker, and enhancer. This ordering groups configurations that share models and therefore reduces the number of model deployments required for a run.

Model Deployment

For each configuration, the pipeline determines the set of models that must be served. This set includes a generator, an embedding model, optionally a reranker, and optionally a query enhancer if the configuration specifies a model different from the generator. Models are served as vLLM instances on the ZHAW SoE compute cluster. The pipeline submits one SLURM job per model over SSH, waits until the job has been scheduled, and then continually polls the health endpoint of each server until it accepts requests. Deployments are reused across configurations whenever the required model is already being served. Together with the plan ordering described above, this keeps the number of deployments close to the number of distinct model combinations rather than the number of configurations. The pipeline also keeps track of the SLURM job IDs to terminate a deployment when a new model must be served, or all deployments at the end of the answer collection run.

RAG Pipeline Initialization

Once all models for a single configuration are deployed, the pipeline instantiates the clients for the embedding model, the generator, and optionally the reranker and query enhancer. It then initializes the document and query pipelines of the RAI Copilot with the required parameters. Instantiation is performed from scratch for every configuration, so that no state, cached client, or model instance is carried over between configurations.

Document Ingestion

The provided documents are then ingested into the Qdrant vector database using the document pipeline of the RAI Copilot itself, described in Section 6.2.1. Ingestion is performed only if no collection exists for the current combination of embedding model, chunk size, and search method. Otherwise, the existing collection is reused. The metadata for each document must be provided in a separate metadata JSON file. If any document fails to be ingested, the configuration is aborted. This ensures that answers are generated against a complete indexed corpus. The duration of the ingestion process is recorded in the answer file.

Answer Generation

In the final stage, the pipeline iterates over each item in the dataset and submits it to the query pipeline one at a time. For multi-turn items, the conversation history is passed alongside the question. For all items, the response mode of the generator LLM is set according to the configured value. In addition, the access level is set to private, which ensures that all indexed documents are considered during retrieval. The responses are generated in a blocking, non-streaming manner. For every item, the pipeline stores the identifier of the question from the dataset, the rewritten question produced by the query enhancer, and the generated answer. In addition, it records the retrieved chunks with their content, similarity score, and rank. Finally, the pipeline also captures the token usage and the processing time per pipeline step as well as the E2E response time.

The answer file, which contains the information described above, is written after every generated answer to the configured location. A run that is interrupted can therefore be resumed without repeating already completed work. Already answered items are skipped, and configurations with missing items are completed on the next run. Failed items are not included in the answer file and are re-attempted when the configuration is run again.

These answer files are then used as input to the evaluation pipeline (Section 4.3), which produces an evaluation report for each configuration.

7.1.4 Metrics and Judge Configuration

This section states which metrics of the toolbox (Section 4.3.1) were computed and how the judges were configured. The section also describes over which items each metric is aggregated. All metrics were computed with the pipeline described in Section 4.3.2.

Selected Metrics

All available metrics were computed, as each addresses a different stage of the pipeline and the search space varies components at every stage. Table 7.3 lists all metrics together with the property they measure.

Group	Metric	Measures
Refusal	TRR, FRR	Whether the system refuses when it should
Retrieval	Contextual Recall	Coverage of the reference answer by the retrieved chunks
	Contextual Precision	Rank-weighted share of retrieved chunks relevant to the reference
	Contextual Relevancy	Share of statements in the retrieved chunks relevant to the question
	Rank Agreement	Agreement of the retrieved ranking with a judge-assigned ranking
Generation	Faithfulness	Grounding of the answer in the retrieved chunks
	Answer Relevancy	Alignment of the answer with the question asked
	Citation Accuracy	Correctness of the citation markers in the answer
	BERTScore	Semantic similarity to the reference answer
	ROUGE, BLEU	Lexical overlap with the reference answer

Table 7.3: Metrics computed for every configuration of the benchmark.

For answer relevancy, the llama-nemotron-embed-1b-v2 embedding model was used.

Metrics were not aggregated over the same set of items. The refusal metrics were computed over all 536 items, since each item should either be answered or refused. All remaining metrics were only computed for items that should be answered and were answered, that is, the true negatives of the refusal classification. Every aggregate is therefore reported together with the number of items it was computed over. Items whose metric could not be computed are excluded from the average rather than counted as zero.

Judge Configuration

Llama-3.3-70B-Instruct was used as the judge for all LLM-as-a-judge metrics, with a temperature of 0.7 and a top-p of 0.8. A non-zero temperature was used so that the five ensemble runs are not identical. The judge model and its model family differ from the two models used to generate the dataset and all four candidate generators of the search space. This ensured that no configuration was scored by a model that produced either the question, answer, or the reference. Furthermore, it is classified as a human-like judge for response accuracy evaluation in the Judge's Verdict benchmark [109]. Every judge-based metric, except answer relevancy and contextual relevancy, was computed five times and averaged. Contextual relevancy is the most expensive metric to compute, as it requires a verdict for every statement in the first five retrieved chunks. The ensemble size N was therefore reduced from five to one mid-run to keep the evaluation within the available compute budget. Because that reduction took effect part-way through, 7 of the 20 configurations hold a mixture of ensembled and single-sample scores. All configurations therefore only use the first ensemble run for contextual relevancy.

Refusal classification was performed by Phi-4 with a temperature of 0.0, so that the classification that gates all remaining metrics was deterministic.

7.1.5 Configuration Selection

The evaluation of all 21,840 valid configurations was infeasible. Each configuration requires deploying multiple models, indexing the documents, and generating and evaluating 536 answers. The benchmark therefore proceeded in three stages. First, answers to all 536 questions were collected for a random subset of configurations with the pipeline described above. In a second

step, all answers from every configuration were evaluated with the implemented evaluation pipeline. Finally, a champion was selected from the results by a fixed decision rule, which is described below.

Sampling

Twenty configurations were drawn from the 21,840 valid configurations using the random sampling mode of the answer collection pipeline. This corresponds to roughly 0.09 % of the valid configuration space. The density does not permit an estimate of the individual contribution of each component, which is discussed in Section 7.3. However, it was sufficient to identify a configuration that performed well across all metrics and to observe how widely performance varied.

Decision Rule

Eligible champion configurations were selected by a conjunctive decision rule, rather than by a single metric. A conjunctive decision rule rejects objects that fail to meet the minimum cutoff on at least one criterion [110]. This decision rule was chosen for the following reasons. First, the criteria are non-compensatory, meaning that a high answer relevancy score cannot balance out a low faithfulness score. This is important because a system that must strictly ground its answers in the retrieved context but often leverages its own parametric knowledge is unusable regardless of how relevant its answers are. Second, the result identifies not only which configurations are acceptable but also on which criteria the remaining ones fail. Third, thresholds can be adjusted to reflect the requirements of a specific system. A system may not be required to be completely faithful, while still being required to strictly refuse non-answerable questions.

Table 7.4 lists the seven criteria applied in this benchmark to select the champion configuration. Among the configurations that met all criteria, the one with the lowest p95 response time was selected as the champion.

Criterion	**Requirement**
Refusal	
TRR	$\geq$0.95
FRR	$\leq$0.035
Retrieval	
Contextual recall	$\geq$0.85
Contextual precision	$\geq$0.80
Generation	
Faithfulness tail	$\leq$2 % below 0.5
Citation coverage	$\geq$95 %
Operational	
Response time (p95)	$\leq$6 s

Table 7.4: Acceptance criteria applied to select a configuration. A configuration is eligible only if it meets every criterion.

The following list describes the rationale for each criterion applied. The thresholds were set to

reflect the requirements of the RAI Copilot.

- TRR $\geq$0.95: the threshold is set below 1.0 because the Copilot implements no dedicated safety guardrail. The refusal behavior is solely defined by the prompt instructions.
- FRR $\leq$0.035: the threshold is set to allow the Copilot to refuse an answerable question only once in roughly thirty cases.
- Contextual recall $\geq$0.85: the threshold is set below 1.0 because a reference answer may also be supported by chunks other than the ones it was generated from.
- Contextual precision $\geq$0.80: the generator may still synthesize an answer from lower-ranked, but relevant chunks. Therefore, the threshold is set below 1.0.
- Faithfulness tail $\leq$2 %: 2 % of the answers may be scored with a faithfulness score below 0.5. A threshold on the mean would hide a small cluster of highly unfaithful answers if the remaining answers score high.
- Citation coverage $\geq$95 %: the share of answered (TN) responses that contain at least one marker in the instructed format. The citation highlighting feature relies on the presence of markers in the given format. A wrong or missing marker prevents users from verifying a claim. The reason for not selecting citation accuracy itself is stated in Sections 5.2.2 and 7.2.4.
- Response time (p95) $\leq$6 s: responsiveness is governed by the slow tail rather than by the average response time. Since the Copilot streams tokens to the user, a maximum E2E response time of 6 s is considered acceptable for 95 out of 100 requests.

7.2 Results and Analysis

Before the results of the benchmark are presented, the outcomes of the human annotation study (Section 5.2.2) are concisely summarized. The refusal classifier agreed with the human annotator on every rated item, and contextual recall reached substantial agreement. Both are therefore used to gate and to compare configurations. Faithfulness agreed at a high raw rate, but its verdicts concentrate almost entirely on *grounded*. This limits its power to separate configurations of similar quality. Contextual precision was found to be permissive and is read as an upper bound on retrieval precision rather than as an estimate of it. Citation accuracy was found to be unsuitable for ranking configurations, as the judge invented citations that the responses did not contain. Contextual relevancy, rank agreement, and answer relevancy were not covered by the annotation study.

The benchmark evaluated 20 configurations, which are listed in Table D.2, on 536 questions. Answer collection for the configurations required approximately 34 h of compute time. Scoring these answers was roughly seven times more expensive, requiring 238 h of compute time. The total compute time of 272 h includes repair runs for failed metrics. Detailed results for every configuration and metric are provided in Section D.2.

7.2.1 Acceptance Criteria and Champion Selection

Figure 7.2 applies the thresholds defined in Table 7.4 to the metric outcome of all 20 configurations. Each row indicates which of the seven criteria a configuration meets. The count above each column states how many configurations that criterion rejects. Configurations range from meeting two criteria (C03) to meeting all seven. Only C09 and C11 meet all seven, and both are mistral-24b configurations with a bge-m3 reranker and $k_{\text{final}} = 4$.

Figure 7.2: Outcome of the conjunctive decision rule. For every configuration the figure shows which of the seven criteria are met. Only 2 of 20 configurations meet all seven.

Every criterion rejects at least four configurations. Citation coverage is the most constraining, rejecting nine, followed by TRR and the p95 response time, which reject seven each. Of the seven configurations rejected by the p95 response time, four are deemed non-admissible based on this criterion alone. Contextual precision is the least constraining, rejecting four.

The rejections cluster by generator rather than by retrieval. All gemma-31b configurations, except C06 (which also fails FRR), fail on response time alone, which ranges from 8.195 s to 9.762 s. All apertus-8b configurations fail on TRR, faithfulness tail, and citation coverage.

Figure 7.3 examines the sensitivity of the chosen thresholds. For every criterion the dashed line traces how many configurations would meet that criterion alone as its threshold moves. The solid line, on the other hand, indicates how many would be accepted if that threshold moves while the other six are held constant. The gap between the two lines is the extent to which a criterion is not the binding constraint.

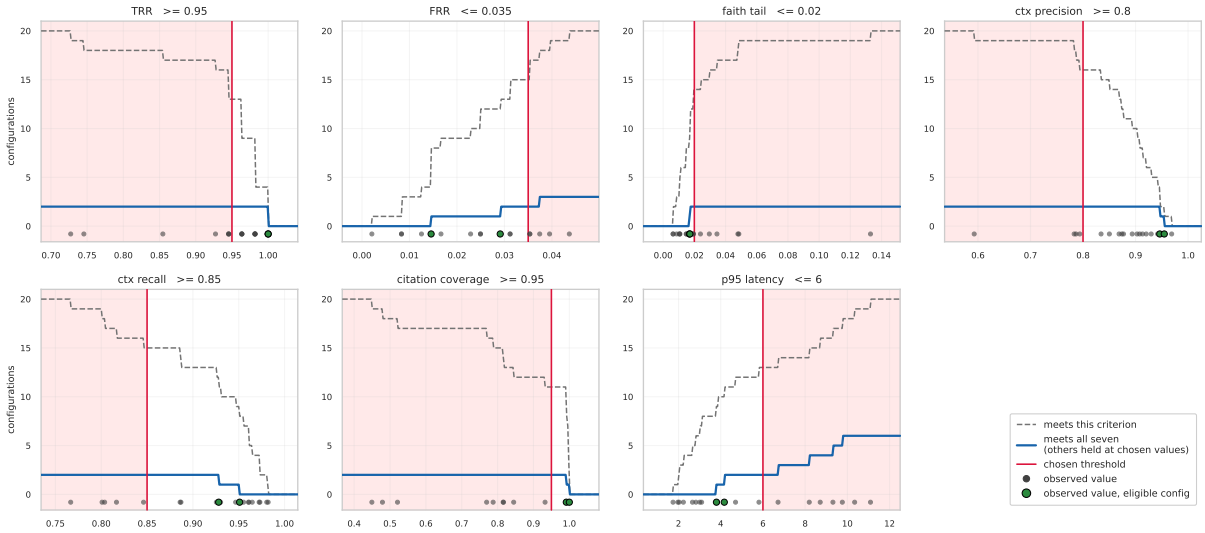

Figure 7.3: Sensitivity of the eligible set to each acceptance threshold taken on its own. The dashed line represents the number of configurations meeting that criterion alone as its threshold moves. The solid line the number meeting all seven with the other six held at their chosen values. The vertical line marks the chosen threshold. The shaded side is the side a configuration must not fall on. Observed values are plotted along the bottom axis. Both eligible configurations are highlighted green.

The eligible set is stable under all but two of the seven thresholds. Relaxing TRR, faithfulness tail, contextual precision or recall, or citation coverage alone would not admit further configurations. This is the case because every configuration that fails one of those criteria also fails another one. However, loosening FRR from 0.035 to 0.0375 admits C10, and loosening the p95 response time from 6 s to 6.74 s admits C13. As stated above, four out of seven configurations fail only due to the response time.

As stated in Section 7.1.5, the champion was selected by the lowest p95 response time. C11 achieves 3.797 s, while C09 achieves 4.169 s.

7.2.2 Refusal Behavior

Figure 7.4 illustrates the refusal behavior of all 20 configurations. The ideal configuration is marked with a star and located in the upper left corner of the figure. It answers all questions it should answer (FRR = 0.00) and refuses all questions it should refuse (TRR = 1.00). Both red lines mark the threshold of the refusal criteria. Configurations that lie in the shaded area satisfy both refusal criteria. Detailed refusal scores are provided in Table D.3.

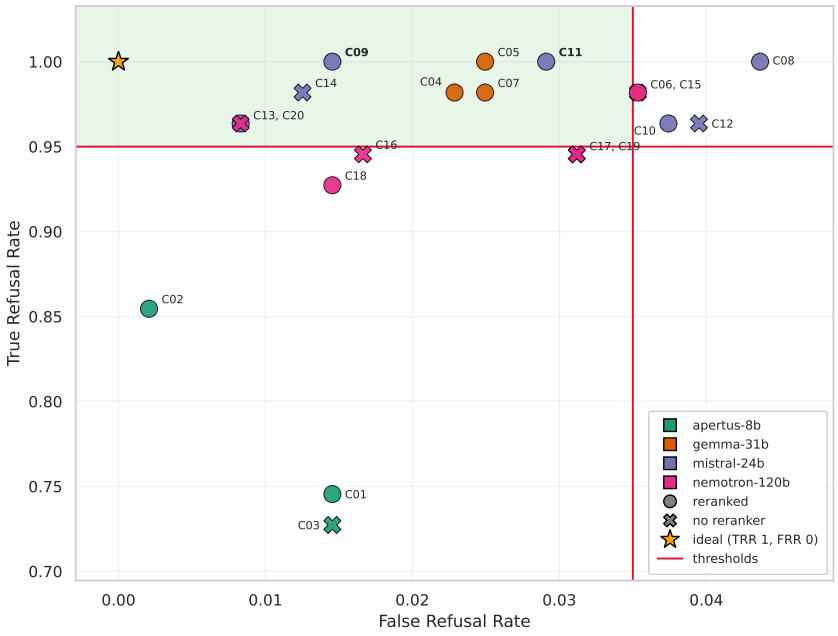

Figure 7.4: TRR against FRR for all 20 configurations. The shaded region marks the area satisfying both refusal criteria of Table 7.4. The ideal configuration is located in the upper left corner of the figure.

Figure 7.4 clearly indicates that failures are strictly one-sided. Configurations either answer questions that should be refused (e.g., C02), or refuse answerable questions (e.g., C08). The three apertus-8b configurations (C01–C03) reach the lowest TRR of the entire benchmark, between 0.727 and 0.855. No other configuration falls below 0.927. Four of the six nemotron-120b configurations (C16–C19) also miss the TRR threshold. Neither the gemma-31b nor the mistral-24b configurations fail the TRR criterion at all. Three out of seven mistral-24b configurations exceed the FRR threshold.

For the Copilot, a high TRR is more important than a low FRR. This follows from the design principle that the generator must refuse ungrounded and out-of-domain questions.

7.2.3 Effect of Retrieval Depth

The left panel in Figure 7.5 illustrates the effect of k_{final} on contextual recall and precision. Each configuration is plotted twice, once per metric. The orange and blue lines represent the mean of each metric across all configurations that share the same k_{final}. The right panel plots the two metrics against each other. The star marker in the upper right corner indicates an ideal configuration. Configurations in the shaded area satisfy both retrieval criteria.

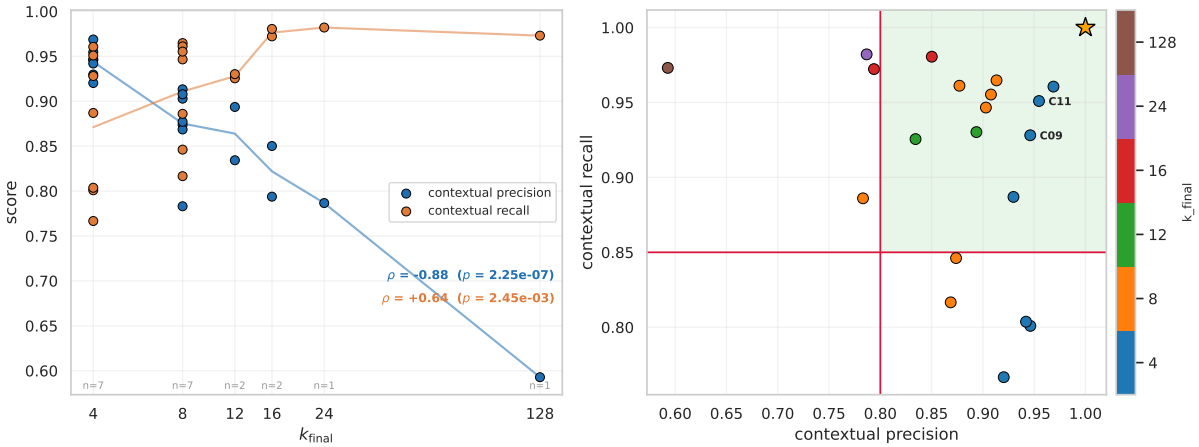

Figure 7.5: Contextual precision and contextual recall against k_{final}, the number of chunks passed to the generator. Left: both metrics against k_{final}. Right: the two metrics against each other, colored by k_{final}.

The panel on the left shows that both metrics move in opposite directions to some extent as k_{final} increases. There is a strong and statistically significant negative correlation between k_{final} and contextual precision (Spearman's $\rho = -0.88$, $p = 2.25 \times 10^{-7}$). Contextual recall shows a moderate positive correlation (Spearman's $\rho = +0.64$, $p = 2.45 \times 10^{-3}$). These properties are expected rather than surprising. Contextual precision averages Prec@k over the positions of the relevant chunks. The chunks admitted by a larger k_{final} are relevant less often, and those that are relevant may sit at low ranks with low Prec@k, which lowers the average. Contextual recall, on the other hand, benefits from the additional chunks up to a certain level, as the judge has more evidence to ground the reference answer in.

The two criteria consequently bound k_{final} from opposite sides (right panel). Eleven configurations satisfy both criteria, with a k_{final} ranging from 4–16.

7.2.4 Citation Defect

Figure 7.6 visualizes the citation defect, already introduced in Section 5.2.2. The orange bars show the share of answers carrying at least one marker in the instructed `[n]` format. The blue bars show the mean citation accuracy the judge assigned to the same configuration. The dashed red line marks the 95 % coverage threshold.

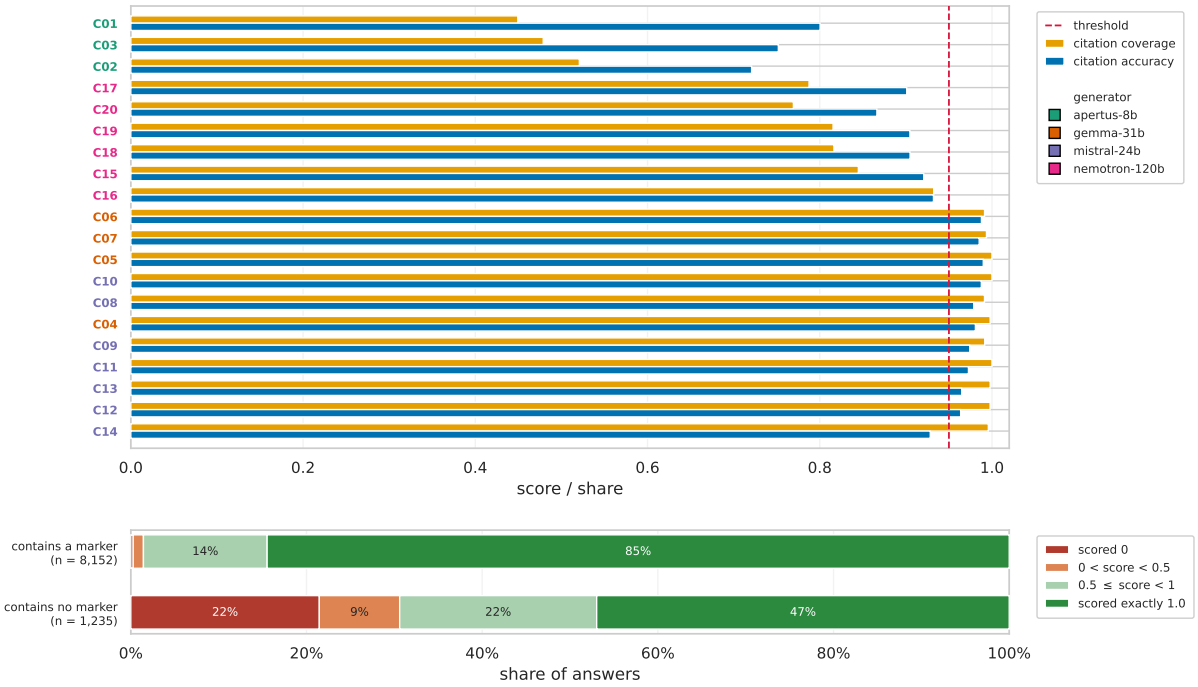

Figure 7.6: Citation coverage and citation accuracy per configuration. The panel on the bottom illustrates the citation defect.

Coverage is determined almost entirely by the generator. The three apertus-8b configurations include a citation marker in no more than 52 % of their responses. The six nemotron-120b configurations include a marker in 77–93 % of their responses. All nine configurations are rejected by the coverage criterion. The other two model families, gemma-31b and mistral-24b, include a marker in more than 99 % of their responses and therefore meet the coverage criterion.

The citation defect is visualized in the lower panel of Figure 7.6. Of the 9,387 answers scored for citation accuracy across all configurations, 1,235 contain no citation marker at all. Instead of returning an empty verdict list, as the evaluation instructions require, the judge scored 47 % of these 1,235 responses with a score of 1.0. In total, only 9 answers without a marker were scored according to the instructions.

These findings indicate a serious defect in the judge's scoring behavior. Therefore, citation accuracy is replaced by the citation coverage criterion. Coverage is read deterministically from the response text and requires no judge verdict.

7.2.5 Traditional Metrics

The results section concludes with a discussion of the non-judge metrics, which are BLEU, ROUGE, and BERTScore. The objective of this analysis is to examine whether these metrics are suitable for ranking configurations. Figure 7.7 examines the effect of the answer length on these three metrics. Answerable responses across all configurations were grouped into fixed logarithmic bins based on their length relative to the reference answer. The bins span ratios from 0.2 to 10.0. Each point in the figure represents the mean of a metric across all responses in a bin. Of the 9,396 responses for which these metrics could be computed, 222 fall outside this range and were

excluded.

All four curves peak in the same bin between 1.00 and 1.25 times the reference length. The scores fall off on both sides of this bin. An answer around a third of the reference length scores no better than one several times longer. The metrics are therefore dominated by length agreement with the reference.

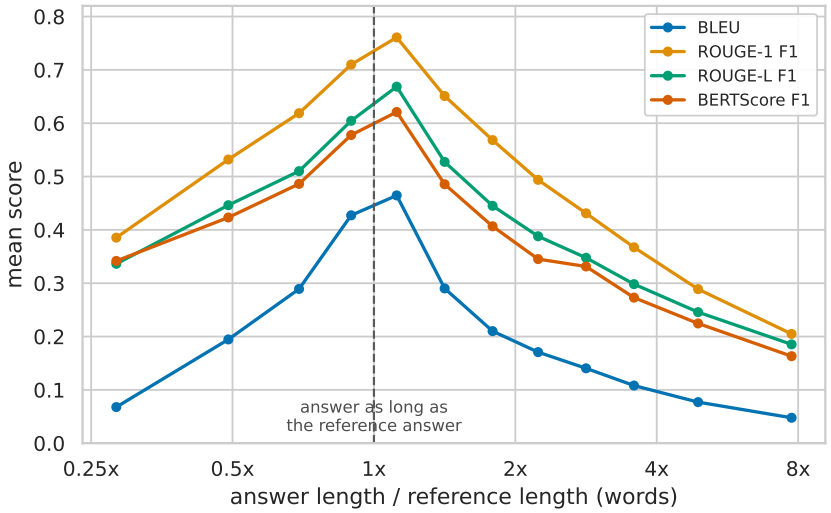

Figure 7.7: Mean BLEU, ROUGE-1 F1, ROUGE-L F1, and BERTScore F1 of answerable responses, binned by the ratio of answer length to reference length. All four curves peak when the answer is 1.0–1.25 times the reference length.

To conclude the analysis, Figure 7.8 compares the overall length of the generated answers across the four generator models over answerable items. The dotted line represents the median reference answer length over the 481 items with a reference answer. The whiskers of the boxplots extend to the 5th and 95th percentiles, since some generator LLMs produced no or unconventionally long answers.

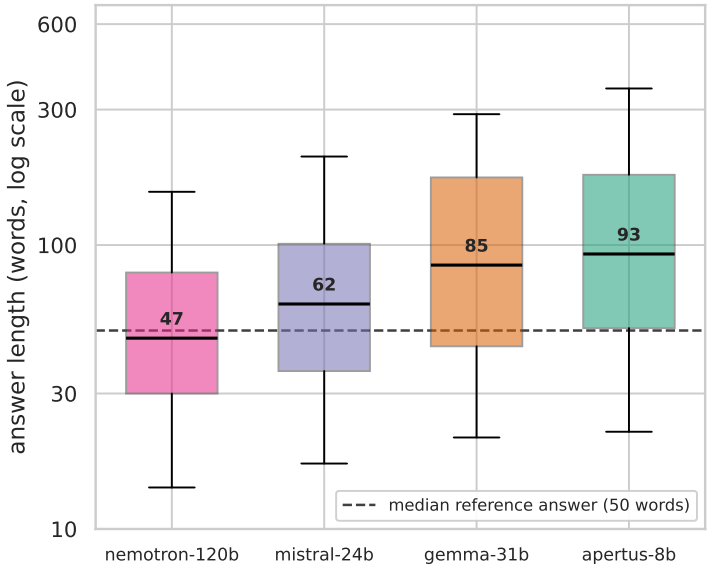

Figure 7.8: Distribution of answer length (words) per generator over answerable items, pooled across configurations. Whiskers span the 5th–95th percentile. The dotted line marks the median reference length.

The boxplots separate the four generator models. Nemotron-120b produces the shortest answers, with a median of 47 words, followed by mistral-24b with 62. Gemma-31b reaches 85 words and apertus-8b 93. Against a median reference answer of 50 words, this spans 0.94–1.86 times the expected length.

This indicates that non-judge metrics may not be suitable to rank configurations, as they are sensitive to the answer length. Such metrics would penalize the generally longer answers from gemma-31b and apertus-8b, even if they are factually correct. The study further highlights the need for a judge-based metric that assesses the correctness of generated responses, which is currently not supported by the toolbox.

7.3 Summary and Limitations

The benchmark evaluated 20 of the 21,840 valid pipeline configurations against 536 synthetic questions. Two configurations, C09 and C11, satisfy all seven acceptance criteria, and C11 was selected as champion for its lower p95 response time. Both are mistral-24b configurations with a bge-m3 reranker and $k_{\text{final}} = 4$. The following conclusions can be drawn from the benchmarking.

- The p95 response time and the FRR are the binding constraints. They are the only criteria whose relaxation admits further configurations. For instance, three out of four gemma-31b configurations are rejected solely due to the p95 response time.
- Configurations either answer questions that should be refused or refuse answerable questions. No configuration fails on both criteria. Apertus-8b configurations tend to answer questions that should be refused, while three out of five configurations that refuse answerable questions are mistral-24b configurations.
- Contextual recall and precision are correlated with k_{final} in opposite directions. Both metrics should therefore be included in the acceptance criteria, as they bound k_{final} from opposite sides.
- Citation accuracy is unusable in the current version of the toolbox, because the judge assigns scores to answers that contain no citation markers. Citation coverage is used as a deterministic replacement criterion. It represents the share of answers that contain at least one citation marker in the instructed format.
- Reference-based non-judge metrics rank configurations by length agreement with the reference rather than by answer quality. This makes them unsuitable for configuration selection on this dataset.

The following limitations arise from the design of this benchmark specifically. They extend the limitations of the toolbox and the annotation study, which are discussed in Sections 4.5 and 5.4.

Sampling Density

Only 0.09 % of the valid configuration space was evaluated. All 20 configurations were randomly sampled instead of varying one hyperparameter at a time. Therefore, the effect of individual hyperparameters cannot be examined. Furthermore, the benchmark misses a baseline configuration

to compare against. For instance, a sparse-retrieval baseline with a small generator would have shown whether the more complex pipeline adds value.

Acceptance Criteria

The thresholds of Table 7.4 reflect the requirements of the Copilot and were determined by the author only. A limitation of these thresholds is that they accept or reject configurations based on a single score. A configuration that misses a threshold by a small margin is rejected, while one that exceeds it is accepted. For instance, C10 misses the FRR threshold by 0.0024 and is therefore rejected.

Computational Constraints

The evaluation of 20 configurations required 272 h of compute time. The majority of this time was spent on answer evaluation. This clearly highlights the computational cost of optimizing RAG pipelines. The efficiency of the benchmark was furthermore negatively affected by the shared nature of the ZHAW SoE compute cluster.

Chapter 8

System Evaluation

While the previous chapter assessed what the RAI Copilot answers, this chapter assesses the technical reliability and performance of the system. An automated test suite (Section 8.1) verifies the functional correctness of important backend components and of the E2E user journeys. Additionally, a load test (Section 8.2) analyzes the system's latency, throughput, and error behavior under concurrent use. Both were introduced in VT2 and were extended in this thesis.

8.1 Automated Test Suite

The automated test suite serves as a regression safeguard for the RAI Copilot. In the current implementation, the core functionality of the Copilot is verified along two dimensions. Backend tests verify the pipeline logic in isolation. E2E tests cover the complete application, including web client and server, and external services.

8.1.1 Backend Tests

The backend tests were implemented with pytest [111]. Pytest is an open-source testing framework for Python. It was selected for its support of fixtures and parameterized tests, which allow the same behavior to be verified across several inputs without duplicating test code. Another widely used framework is unittest, which belongs to the Python standard library. Barbosa and Hora [112] find that developers prefer pytest to unittest for its simpler syntax, easier maintenance, and interoperability with unittest.

Each test is structured according to the Arrange–Act–Assert pattern [113] to ensure a uniform structure. In each test, the SUT and its dependencies are first brought into the required state. Then, the behavior under test is invoked. Finally, the resulting state or return value is verified against the expectation.

The complete suite consists of 50 test functions, which expand to 57 executed cases. They verify the behavior of the document and query pipelines, as well as the citation highlighting service. The tests are organized into three modules, which are described in Table 8.1. The existing suite was extended to cover the new functionality (Table 6.1) added in this thesis.

Module	Covered behavior	Tests
`test_document_pipeline`	Singleton initialization, vector database setup across all three modes, document ingestion, collection management, retrieval with access-level filtering, test-environment reset	23
`test_query_pipeline`	Query enhancement, retrieval, reranking, answer generation, LangGraph wiring, conversation history preparation, citation extraction utilities	18
`test_citation_srv`	Cited-statement extraction, statement segmentation, sentence-level chunk highlighting	9

Table 8.1: Coverage of the backend test suite. Test counts refer to test functions. Tests invoked with different parameters expand to 57 executed cases in total.

Every external dependency is replaced by a mock. The vector database, the MongoDB database, and all model clients are stubbed. Furthermore, a shared fixture supplies a deterministic configuration object in place of the application settings. This ensures that each test runs in a consistent, predictable state. The suite verifies orchestration logic and error handling. A test can confirm that the query pipeline calls the reranker with the expected arguments and that the output matches the expected format. However, it does not verify whether the reranker deployment itself responds as expected. This is covered by the E2E tests and, ultimately, by the load test.

8.1.2 End-to-End Tests

E2E testing is a black-box evaluation method that ensures a software system functions correctly from the end-user's point of view [114]. Black-box tests solely assess whether the system behaves as expected, without considering its internal implementation [115]. E2E tests cover the complete application flow, including the web client and server, and integrated third-party services.

The E2E tests were implemented with Cypress [116]. Cypress is a testing framework that simulates user interactions within a web client. While Selenium remains the most widely used framework in the surveyed literature [115], Cypress was selected for three reasons. First, it executes in the same run loop as the application under test, which enables automatic waiting when locating web elements [117]. Second, unlike Selenium, Cypress provides assertions and test retries out of the box [117]. Third, prior experience with the framework reduced the implementation effort in VT2 and in this thesis.

The E2E tests are organized into three specifications, which are described in the following list. The specifications focus on the core user journeys of the RAI Copilot, which include document management and chatbot interaction. The collaborator workflow, metadata editing, citation dialog, and MongoDB resetting were added in this thesis.

- **Document pipeline**: These tests cover the upload of a single PDF with manually entered metadata and the upload of a ZIP archive containing multiple PDFs. They further verify the collaborator proposal and admin approval workflow and cover document deletion and metadata editing with Semantic Scholar.
- **Query pipeline**: These tests cover the functionality exposed by the chat interface. This

includes querying the Copilot, switching between response modes, and opening the citation dialog.

- **Environment teardown**: This specification resets the vector database and the MongoDB database after the suite has completed.

The suite runs against the web server started in a dedicated test environment. This ensures that the vector database operates in in-memory mode and that a reset endpoint is exposed. Each specification resets the vector database before and after execution, which keeps the specifications independent of one another and of their execution order.

8.2 Load Testing

The RAI Copilot is intended to be used by researchers at the ZHAW Centre for Artificial Intelligence (CAI). For that reason, the objective of the load test is to verify the Copilot's reliability under concurrent use. Jiang and Hassan [118] group load-related problems into two categories. First, functional defects that manifest only under load, such as deadlocks or memory leaks. Second, violations of non-functional requirements such as reliability, stability, and robustness. The test reported here targets the latter.

The load test is implemented with Locust [119], an open-source load testing tool implemented with Python. Flexibility was a key requirement for the selection of the load testing framework. Locust enables the tracking of custom metrics and captures them alongside the standard metrics, such as E2E latency. In addition, Locust provides a number of event-hooks that can be used for seeding as well as environment cleanup, which makes test runs reproducible.

As stated before, a load test was already conducted in VT2. It established a closed-loop model with two tasks: document ingestion and Copilot querying. The extensions made in this thesis focused on increasing concurrency, implementing a dedicated test environment, and recording additional timing metrics. Table 8.2 highlights the differences between the VT2 version and this thesis.

8.2.1 Load Test Design

The implemented load test follows a use-case-based design. In this design, requests represent realistic use cases [118]. Furthermore, the test follows a closed-loop model, in which each simulated user issues a request, waits for the response, and then pauses before issuing the next one [120].

The test simulates both regular users and administrators. The number of simulated users can be set via a CLI argument (Section 8.2.3). Before the test starts, the user accounts are added to the MongoDB database. The number of administrators is fixed at two. Then, the vector database is reset and seeded with the configured document collection. After the setup is complete, both user classes authenticate once, open a new conversation, and then repeatedly query the following two endpoints. Both tasks below are executed with a random pause of five to ten seconds, approximating the time a real user spends reading the response before asking the next question.

Aspect	VT2 version	This thesis
Concurrency ceiling	15 users (assumed to exist beforehand)	No ceiling. The configured number of users is generated and seeded per run
User population	A single user class. Each user both uploaded and queried	Separate administrator and regular-user classes
Environment state	No reset or seeding	MongoDB and vector database reset before every run
Document corpus	Whatever the target instance contained before a run	The 50 documents of the benchmark corpus (Section 7.1.1), which are ingested before every run
Query set	10 questions inlined in the test file	536 questions of the benchmark dataset, sampled uniformly at random
Latency measurement	Header latency and stream duration	Header latency, time to first token, and stream duration
Error accounting	Rejected requests and in-stream errors reported; exceptions logged only	Every attempt recorded exactly once and failures classified by cause
Result granularity	Aggregates over the whole run	Per-request log and aggregates over the whole run
Experimental design	Single 40 min test	Three load levels × three repetitions

Table 8.2: Differences between the load test developed in VT2 and the extended load test developed in this thesis.

- **Query Endpoint**: The first task queries the Copilot in the default response mode, using the same endpoint that serves queries submitted through the UI. This task is performed by both user classes. Queries are sampled uniformly from the benchmark dataset (Section 7.1.1). Each request carries a conversation ID, so that the exchange is persisted to the conversation history. This also ensures that the query enhancement step is triggered once the conversation contains more than one turn. A new conversation is started after 15 queries. Locust records the header latency, the TTFT, and the full stream duration, which are described in Section 8.2.2.
- **Document Upload Endpoint**: Administrators additionally upload PDF documents. As with the query task, the same endpoint is used that serves non-ZIP uploads through the admin interface. During this task, a single PDF is uniformly sampled from the 50 documents from which the benchmark dataset was generated. The document pipeline performs a duplicate check (Section 6.2.1). Since the admins may upload each PDF multiple times, each request carries a flag to skip this check. For this task, Locust records the E2E processing time. This task is weighted at one upload per ten queries to reflect that querying dominates ingestion in normal usage.

The tasks are executed until the configured run duration is reached. After the test ends, the vector database is reset. The conversation database is not, so that the stored histories remain available for inspection after the run. The MongoDB database is cleared only when the next run starts. The timing metrics recorded during a run are described in the following section.

8.2.2 Measured Timing Metrics

Locust does not record meaningful response times for streaming endpoints: it stops timing when the response headers arrive, before the pipeline has produced any content. Because the chat

endpoint streams its response as a sequence of server-sent events, the response time recorded by Locust by default is not meaningful. The load test therefore records three separate quantities per streaming request, summarized in Table 8.3.

Quantity	Definition
Header latency	Time until the response headers arrive, i.e., how quickly the server accepts the request.
TTFT	Time from issuing the request until the first content token is streamed. Comprises query enhancement, embedding, retrieval, reranking, and generation prefill. This is the latency a user perceives until the first token is displayed in the UI.
Stream duration	Time from issuing the request until the stream terminates. This metric is dominated by the number of generated tokens.

Table 8.3: Timing metrics recorded for each request to the streaming chat endpoint.

For non-streaming endpoints, such as the document upload or login, the E2E response time is recorded. Locust aggregates the recorded metrics over the entire run and reports statistics such as the mean, median, and percentiles. These aggregates include the warm-up phase, which may not be representative of the steady-state behavior of the application. Therefore, all requests are also written to a per-request log, enabling the analysis over any time window.

8.2.3 Load Test Execution

The load test was run against the RAI Copilot deployed on the ZHAW SoE compute cluster (Section 6.1.2). The RAG pipeline was configured with configuration C11 (Table D.2). This configuration was selected because it was identified as the champion in the Copilot benchmark. This allowed both the behavior of the application under load and the model reliability under concurrent use to be observed. Load generation tools and the SUT should be kept on separate machines to prevent overlapping resource usage and ensure accurate measurements [118]. Therefore, the Locust script was executed on a local machine that communicated with the Copilot over virtual private network (VPN). The reported response times therefore include network latency and correspond to the E2E latency experienced by a user rather than to server-side processing time alone.

The test examined three load levels that are summarized in Table 8.4. The first level establishes a baseline latency with five concurrent users. The expected load mirrors the expected number of concurrent users at the ZHAW CAI. The load test is considered passed if the Copilot successfully handles the expected load without non-functional issues. Finally, the last level identifies the saturation point of the system, where error rates might increase and response times degrade.

Level	Users	Spawn rate	Duration	Purpose
Baseline	5	1/s	15 min	Baseline run
Expected load	30	1/s	15 min	Expected load at ZHAW CAI
Saturation	100	1/s	15 min	Probe behavior beyond the expected load

Table 8.4: The three load levels examined. Two of the simulated users are admin users at every level. All users are spawned at a rate of one per second, and every run lasts 15 min.

Each level was executed three times. A single run yields no estimate of between-run variation,

and therefore no basis for deciding whether a difference between levels exceeds it.

8.3 Results and Analysis

This section presents the results of the system tests. First, it details the results of the automated test suite to verify system correctness. Subsequently, it analyzes the system's steady-state performance and scalability across multiple simulated load levels.

8.3.1 Automated Test Suite

Both test suites were executed against the current implementation. All 57 executed backend test cases passed in under 15 s. The E2E suite was run against the web server started in the test environment. All specifications passed as well, which yields a pass rate of 100 % for both suites.

Overall, the result highlights that the functionality added in this thesis (Table 6.1) integrates with the components carried over from the previous projects without breaking them.

8.3.2 Load Testing

Table 8.5 reports steady-state performance across the three load levels. The steady-state window comprises all requests submitted between 2 min and 15 min after the start of the run. The delay of 2 min was defined to account for the time needed to spawn all users and for the system to warm up. Requests still streaming when the run time elapsed were allowed to complete and are included. The reported values are pooled over the three runs of each level. The per-run breakdown is provided in Table E.1.

Metric	**Baseline** 5 users	**Expected** 30 users	**Saturation** 100 users
Query endpoint			
Completed requests n	1,043	5,049	7,149
Failed requests (%)	0.00	0.04	0.03
Throughput (req/min)	26.74	129.46	183.31
Header latency p95 (s)	0.30	1.06	2.10
TTFT p50 (s)	0.71	1.27	9.47
TTFT p95 (s)	1.88	4.96	21.76
Per-run range	1.60–2.25	4.12–5.42	18.34–26.52
Stream duration p50 (s)	2.30	5.79	23.68
Stream duration p95 (s)	5.34	11.65	38.62
Document upload endpoint			
Completed uploads n	32	21	17
Failed uploads (%)	0.00	0.00	0.00
E2E time p50 (s)	19.82	31.28	24.88
E2E time p95 (s)	83.76	96.03	97.96

Table 8.5: Results per load level, pooled over the three runs of each level. All values are computed over the requests submitted between 2–15 min after the start of a run, which excludes the ramp-up and the subsequent warm-up phase. The per-run range gives the smallest and largest TTFT p95 observed across the three runs of a level.

A notable insight is that throughput did not scale proportionally with the number of users. An increase in concurrent users from 5 to 30 increased throughput by a factor of 4.84 (from 26.74 to 129.46 req/min). The next level, which more than tripled the concurrency (from 30 to 100 users), raised throughput by a factor of 1.42 (from 129.46 to 183.31 req/min). Furthermore, the TTFT p50 grew roughly 7.5 times (from 1.27 to 9.47 s) and the TTFT p95 grew roughly 4.4 times (from 4.96 to 21.76 s). This is a clear indication that the system approached its concurrency limit, where throughput flattens and latency increases.

The per-run TTFT p95 ranges do not overlap between levels (Baseline: 1.60–2.25 s; Expected: 4.12–5.42 s; Saturation: 18.34–26.52 s). This indicates that the three load levels are distinguishable despite run-to-run variability and do not reflect overlapping noise.

Failure rates remained negligible across all levels ($\leq$0.04 %). Four stream requests failed in total. Three of them exceeded the configured stream deadline of 180 s, which means that the system did not deliver a complete response within that budget. A plausible cause is that the generator entered a degeneration loop and kept emitting tokens without reaching a stop condition [121]. The fourth request terminated at the connection level before a status code was received.

No errors occurred during the upload tasks. However, only 32, 21, and 17 uploads fell into the steady-state windows of the three levels. Since the number of administrators was fixed at two at every level, the ingestion load did not scale with the load level (19.82 s, 31.28 s and 24.88 s). For that reason, the differences between levels mainly reflect the lengths of the sampled documents rather than load effects.

The load test demonstrates that the RAI Copilot can handle the expected load of 30 concurrent users. A median TTFT of 1.27 s and a p95 of 4.96 s are considered acceptable for a research assistant application. In conclusion, the system manifests degradation under load as increased latency rather than reduced reliability.

8.4 Limitations

Despite notable findings, the following limitations arise from the design of the two evaluations.

Scope of the Test Suite

The pass rate reported in Section 8.3.1 only quantifies the proportion of successful tests, as no code coverage metrics were collected. While the backend tests cover error handling and edge cases to some extent, the E2E tests verify only the expected path of each user journey. Consequently, these test suites serve primarily as a regression safeguard for their specifically targeted functionality.

Under-sampled Upload Path

The number of administrators was fixed at two at every load level, and the upload task was weighted at one upload per ten queries. This choice reflects the expected usage pattern of the Copilot, where querying dominates ingestion. However, it also means that this endpoint is not meaningfully load-tested by this design.

Absence of Server-Side Monitoring

All measurements were taken at the client. This means that system resources, such as GPU utilization or memory consumption on the VM or the compute cluster, were not monitored. Therefore, the saturation observed at 100 concurrent users cannot be attributed to a specific component. This also limits the test to assessing non-functional load-related problems. Functional defects, such as memory leaks that occur only under load, would require resource monitoring to become detectable [118].

Part IV

Synthesis

Chapter 9

Conclusion and Future Work

This chapter reflects on the objectives defined in Chapter 1 and summarizes the key contributions of this thesis. It further discusses threats to validity and concludes with recommendations for future work.

9.1 Conclusion

Regarding objective O1, the evaluation toolbox was designed and implemented as an installable Python package with two pipelines. The Q&A generation pipeline produces synthetic domain-specific datasets spanning single-turn, multi-turn, multi-hop, and refusal questions. The generate-then-validate approach ensures that every item is assessed by a dedicated validator LLM. The evaluation pipeline scores external RAG systems across two refusal, four retrieval, and six generation metrics, gated by a dedicated refusal classifier. Both pipelines are system-agnostic and fully configurable with YAML files.

Regarding objective O2, the dataset screening and the human annotation study showed that the toolbox's artifacts are usable but not uniformly trustworthy. Two thirds of the screened dataset items required no correction. Judge reliability proved metric-dependent. The refusal classifier agreed perfectly with the author, and contextual recall reached substantial agreement, while contextual precision was systematically permissive. The annotation study further highlighted that the citation accuracy judge invented verdicts for responses without citations. These findings directly shaped how the Copilot benchmark was interpreted, and were condensed into recommendations for future annotation studies.

Regarding objective O3, the RAI Copilot was extended with hybrid chunking and retrieval. Furthermore, a collaborator role for proposing new documents, combined with an admin approval workflow, was introduced. To keep document metadata up to date, such metadata can be updated directly in the UI. Citation highlighting improves the transparency of the system by resolving each claim in the response to the passage that may support it. Moreover, the Copilot was deployed on a VM on the ZHAW SoE compute cluster. Each service the application interacts with is self-hosted to adhere to the data confidentiality requirement.

Regarding objective O4, the Copilot was benchmarked with the evaluation toolbox on a dataset

of 536 questions generated from 50 RAI documents across all supported metrics. The benchmark applied seven acceptance criteria to 20 sampled configurations. Two configurations met every criterion. C11 (mistral-24b, hybrid retrieval with a bge-m3 reranker, $k_{\text{final}} = 4$) was selected as champion for its lowest p95 response time and deployed. The refusal behavior of the Copilot is one-sided. Configurations either refuse answerable questions or answer questions that should be refused. No configuration failed both the TRR and the FRR criterion. The benchmark also showed that the citation accuracy metric is unusable, as it scored 47 % of the responses without citations with a score of 1.0. The benchmark further revealed that surface-level metrics, such as ROUGE and BLEU, cannot be used to rank configurations as they prefer answers whose length is similar to the reference answer. The automated test suite passed all 57 backend test cases and every E2E specification. Finally, the load test confirmed that the deployed system handles the expected 30 concurrent users with a median TTFT of 1.27 s and degrades under overload through latency rather than failures.

9.2 Key Contributions

The results of this thesis can be summarized in three key contributions.

- **Evaluation Toolbox**: A system-agnostic, refusal-aware toolbox for evaluating domain-specific RAG systems. It generates synthetic domain-specific datasets through an LLM-based generate-then-validate approach and evaluates external RAG applications across twelve judge-based and non-judge-based metrics.
- **Validated Judges and Annotation Study Design**: A human annotation study that quantifies the reliability of four judge-based metrics and the refusal classifier against expert labels. It further provides an annotation tool together with transferable recommendations for designing such studies.
- **Enhanced and Benchmarked RAI Copilot**: An improved, responsible-by-design version of the RAI Copilot. It was benchmarked with the toolbox to identify a deployable RAG pipeline configuration, and load-tested to 30 concurrent users without non-functional issues.

9.3 Threats to Validity

This section outlines important potential threats to the validity of the conducted studies. The detailed limitations are stated where they arise (Sections 4.5, 5.4, 7.3 and 8.4).

Construct Validity

Both the dataset and the scores for each configuration are mostly LLM-generated. Errors in synthetic reference answers propagate into every reference-based metric. The Q&A generation and the evaluation prompts clearly define the tasks and the expected outcome. However, both the manual verification of the dataset and the Copilot benchmark revealed that it is challenging to capture all possible edge cases in a single prompt.

Internal Validity

The author wrote the judge prompts, generated the dataset, defined the acceptance thresholds, and was the sole annotator for half of the annotation tasks. Agreement on those tasks may therefore be inflated by a shared interpretation of the criteria. Furthermore, the selection of the champion configuration was based solely on the defined acceptance criteria. The benchmark does not permit conclusions about the effect of individual hyperparameters. The champion configuration is therefore a recommendation, rather than a proven optimum.

External Validity

The results of every conducted study (Table 2.2) are conditioned on specific components. For instance, the Copilot benchmark was conducted on a specific dataset, generated from 50 RAI documents and evaluated with one judge model and one refusal classifier. The same applies to the human annotation study, which establishes the reliability of the chosen judge models on a specific set of prompts and a specific dataset.

9.4 Future Work

This thesis concludes with several recommendations to address the identified limitations, improve the evaluation toolbox, and further enhance the RAI Copilot application.

Evaluation Toolbox

The Q&A generation pipeline should be validated with human annotators, as this thesis validated only the evaluation pipeline. The annotation study should assess both the quality of the generated questions and references, and the reliability of the validation judge. Furthermore, a document preprocessing stage should be added to the pipeline, which should remove reference lists and tables of contents. The pipeline currently only generates questions from PDF documents. However, the knowledge base of existing RAG systems may also include other document formats, such as DOCX or Markdown. The generation pipeline should therefore be extended to support these formats as well.

The most valuable extension to the evaluation pipeline would be a judge-based answer correctness metric. Correctness is currently only covered by surface-level metrics. The analysis of the benchmark revealed that these are dominated by length agreement with the reference answer. However, a correctness metric would further require adjustments to the Q&A generation pipeline, since a question may be answerable from multiple chunks.

The instructions for the citation accuracy metric must be redesigned. A possible solution would be to extract citation markers deterministically before judging. This would follow the decompose-then-validate approach already established for the faithfulness metric. The same applies to the contextual recall and contextual relevancy metrics, where the judge also decomposes and evaluates in a single pass.

Furthermore, the annotation study should be repeated to capture more expert labels and to further assess the reliability of the metrics not assessed in this thesis. Finally, applying the toolbox to a third-party system and comparing its verdicts with existing frameworks would

establish its claimed generality.

RAI Copilot

On the system side, a similar document preprocessing stage, as described above, should be added to improve the quality of the indexed chunks. The prompt-based refusal instructions should be replaced with dedicated guardrails, such as input filters or a guardrail model. Furthermore, replacing the similarity-based citation highlighting with an NLI model would turn an indication of support into a verifiable entailment check.

A future benchmark should focus on ablation studies to quantify the effect of individual hyperparameters on the quality of the retrieved context and the generated answer. This also includes evaluating the effect of different prompt designs, such as chain-of-thought or few-shot prompting, on the quality of the generated answers.

Finally, the load test could be extended to stress-test the system with a higher number of document uploads. The same applies to the Copilot benchmark, which was conducted on a corpus of 50 documents only and may not reflect production use cases. A larger corpus would enable an evaluation of the performance and stability of the implemented retrieval methods.

Bibliography

[1] A. Jobin, M. Ienca, and E. Vayena, "The global landscape of AI ethics guidelines," *Nature Machine Intelligence*, vol. 1, no. 9, pp. 389–399, Sep. 2, 2019, ISSN: 2522-5839. DOI: 10.1038/s42256-019-0088-2 [Online]. Available: https://www.nature.com/articles/s42256-019-0088-2

[2] High-Level Expert Group on Artificial Intelligence, "Ethics guidelines for Trustworthy AI," European Commission, Brussels, Apr. 8, 2019. [Online]. Available: https://digital-strategy.ec.europa.eu/en/library/ethics-guidelines-trustworthy-ai

[3] P. Lewis et al., "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, ser. NIPS '20, Red Hook, NY, USA: Curran Associates Inc., Dec. 6, 2020, pp. 9459–9474, ISBN: 978-1-7138-2954-6. [Online]. Available: https://dl.acm.org/doi/10.5555/3495724.3496517

[4] Y. Gao et al. "Retrieval-Augmented Generation for Large Language Models: A Survey." arXiv: 2312.10997 [cs]. [Online]. Available: http://arxiv.org/abs/2312.10997

[5] L. Zheng et al., "Judging LLM-as-a-judge with MT-bench and Chatbot Arena," in *Proceedings of the 37th International Conference on Neural Information Processing Systems*, ser. NIPS '23, Red Hook, NY, USA: Curran Associates Inc., Dec. 10, 2023, pp. 46 595–46 623.

[6] J. Gu et al., "A survey on LLM-as-a-judge," *The Innovation*, vol. 7, no. 6, p. 101 253, Jun. 2026, ISSN: 26666758. DOI: 10.1016/j.xinn.2025.101253 [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S2666675825004564

[7] A. Bavaresco et al., "LLMs instead of Human Judges? A Large Scale Empirical Study across 20 NLP Evaluation Tasks," in *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, W. Che, J. Nabende, E. Shutova, and M. T. Pilehvar, Eds., Vienna, Austria: Association for Computational Linguistics, Jul. 2025, pp. 238–255, ISBN: 979-8-89176-252-7. DOI: 10.18653/v1/2025.acl-short.20 [Online]. Available: https://aclanthology.org/2025.acl-short.20/

[8] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design Science in Information Systems Research," 2004.

[9] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, "A Design Science Research Methodology for Information Systems Research," *Journal of Management Information Systems*, vol. 24, no. 3, pp. 45–77, Dec. 2007, ISSN: 0742-1222, 1557-928X. DOI: 10.2753/MIS0742-1222240302 [Online]. Available: https://www.tandfonline.com/doi/full/10.2753/MIS0742-1222240302

[10] W. X. Zhao et al., "A Survey of Large Language Models," *Frontiers of Computer Science*, vol. 20, no. 12, p. 2 012 627, May 9, 2026, ISSN: 2095-2236. DOI: 10.1007/s11704-026-60308-3 [Online]. Available: https://doi.org/10.1007/s11704-026-60308-3

[11] L. M. Amugongo, P. Mascheroni, S. G. Brooks, S. Doering, and J. Seidel, "Retrieval Augmented Generation for Large Language Models in Healthcare: A Systematic Review," Jun. 11, 2025. DOI: 10.20944/preprints202407.0876.v1 [Online]. Available: https://www.preprints.org/manuscript/202407.0876/v1

[12] W. Fan et al., "A Survey on RAG Meeting LLMs: Towards Retrieval-Augmented Large Language Models," in *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, Barcelona Spain: ACM, Aug. 25, 2024, pp. 6491–6501, ISBN: 979-8-4007-0490-1. DOI: 10.1145/3637528.3671470 [Online]. Available: https://dl.acm.org/doi/10.1145/3637528.3671470

[13] O. Ovadia, M. Brief, M. Mishaeli, and O. Elisha, "Fine-Tuning or Retrieval? Comparing Knowledge Injection in LLMs," in *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, Y. Al-Onaizan, M. Bansal, and Y.-N. Chen, Eds., Miami, Florida, USA: Association for Computational Linguistics, Nov. 2024, pp. 237–250. DOI: 10.18653/v1/2024.emnlp-main.15 [Online]. Available: https://aclanthology.org/2024.emnlp-main.15/

[14] P. Zhao et al., "Retrieval-Augmented Generation for AI-Generated Content: A Survey," *Data Science and Engineering*, Jan. 2, 2026, ISSN: 2364-1541. DOI: 10.1007/s41019-025-00335-5 [Online]. Available: https://doi.org/10.1007/s41019-025-00335-5

[15] S. Wu et al., "Retrieval-augmented generation for natural language processing: A survey," *Artificial Intelligence Review*, vol. 59, no. 9, p. 192, Jun. 1, 2026, ISSN: 1573-7462. DOI: 10.1007/s10462-026-11605-7 [Online]. Available: https://doi.org/10.1007/s10462-026-11605-7

[16] X. Wang et al., "Searching for Best Practices in Retrieval-Augmented Generation," in *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, Y. Al-Onaizan, M. Bansal, and Y.-N. Chen, Eds., Miami, Florida, USA: Association for Computational Linguistics, Nov. 2024, pp. 17 716–17 736. DOI: 10.18653/v1/2024.emnlp-main.981 [Online]. Available: https://aclanthology.org/2024.emnlp-main.981/

[17] S. Li, L. Stenzel, C. Eickhoff, and S. A. Bahrainian, "Enhancing Retrieval-Augmented Generation: A Study of Best Practices," in *Proceedings of the 31st International Conference on Computational Linguistics*, O. Rambow, L. Wanner, M. Apidianaki, H. Al-Khalifa, B. D. Eugenio, and S. Schockaert, Eds., Abu Dhabi, UAE: Association for Computational Linguistics, Jan. 2025, pp. 6705–6717. [Online]. Available: https://aclanthology.org/2025.coling-main.449/

[18] A. Brown, M. Roman, and B. Devereux, "A Systematic Literature Review of Retrieval-Augmented Generation: Techniques, Metrics, and Challenges," *Big Data and Cognitive Computing*, vol. 9, no. 12, Dec. 12, 2025, ISSN: 2504-2289. DOI: 10.3390/bdcc9120320 [Online]. Available: https://www.mdpi.com/2504-2289/9/12/320

[19] M. Dinzinger, L. Caspari, K. G. Dastidar, J. Mitrovic, and M. Granitzer, "Investigating the Robustness of Embedding Models on Noisy Input Texts," in *Proceedings of the 2nd International Workshop on Open Web Search Co-Located with the 47th European Conference on Information Retrieval (ECIR 2025), Lucca, Italy, April 10, 2025*, S. M. Farzana et al., Eds., ser. CEUR Workshop Proceedings, vol. 4137, CEUR-WS.org, 2025, pp. 2–12. [Online]. Available: https://ceur-ws.org/Vol-4137/WOWS_2025_paper_1.pdf

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

[21] S. Robertson and H. Zaragoza, "The Probabilistic Relevance Framework: BM25 and Beyond," *Foundations and Trends® in Information Retrieval*, vol. 3, no. 4, pp. 333–389, 2009, ISSN: 1554-0669, 1554-0677. DOI: 10.1561/1500000019 [Online]. Available: http://www.nowpublishers.com/article/Details/INR-019

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

[23] C. Sciavolino, Z. Zhong, J. Lee, and D. Chen, "Simple Entity-Centric Questions Challenge Dense Retrievers," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, M.-F. Moens, X. Huang, L. Specia, and S. W.-t. Yih, Eds., Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 6138–6148. DOI: 10.18653/v1/2021.emnlp-main.496 [Online]. Available: https://aclanthology.org/2021.emnlp-main.496/

[24] A. J. Oche, A. G. Folashade, T. Ghosal, and A. Biswas. "A Systematic Review of Key Retrieval-Augmented Generation (RAG) Systems: Progress, Gaps, and Future Directions." version 1. [Online]. Available: https://arxiv.org/abs/2507.18910

[25] T. Gao, H. Yen, J. Yu, and D. Chen, "Enabling Large Language Models to Generate Text with Citations," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, H. Bouamor, J. Pino, and K. Bali, Eds., Singapore: Association for Computational Linguistics, Dec. 2023, pp. 6465–6488. DOI: 10.18653/v1/2023.emnlp-main.398 [Online]. Available: https://aclanthology.org/2023.emnlp-main.398/

[26] H. Qian et al., "VeriCite: Towards Reliable Citations in Retrieval-Augmented Generation via Rigorous Verification," in *Proceedings of the 2025 Annual International ACM SIGIR Conference on Research and Development in Information Retrieval in the Asia Pacific Region*, Xi'an China: ACM, Dec. 7, 2025, pp. 47–54, ISBN: 979-8-4007-2218-9. DOI:

10.1145/3767695.3769505 [Online]. Available: https://dl.acm.org/doi/10.1145/3767695.3769505

[27] T. Schreieder, T. Schopf, and M. Färber, "Attribution, Citation, and Quotation: A Survey of Evidence-based Text Generation with Large Language Models," in *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, M. Liakata, V. P. Moreira, J. Zhang, and D. Jurgens, Eds., San Diego, California, United States: Association for Computational Linguistics, Jul. 2026, pp. 30 956–31 000, ISBN: 979-8-89176-390-6. [Online]. Available: https://aclanthology.org/2026.acl-long.1430/

[28] S. Barnett, S. Kurniawan, S. Thudumu, Z. Brannelly, and M. Abdelrazek, "Seven Failure Points When Engineering a Retrieval Augmented Generation System," in *Proceedings of the IEEE/ACM 3rd International Conference on AI Engineering - Software Engineering for AI*, Lisbon Portugal: ACM, Apr. 14, 2024, pp. 194–199, ISBN: 979-8-4007-0591-5. DOI: 10.1145/3644815.3644945 [Online]. Available: https://dl.acm.org/doi/10.1145/3644815.3644945

[29] A. Garani, "A Systematic Taxonomy of Failure Modes in Retrieval-Augmented Generation Systems," in *Proceedings of the 6th Workshop on Trustworthy NLP (TrustNLP 2026)*, K.-W. Chang et al., Eds., San Diego, California: Association for Computational Linguistics, Jul. 2026, pp. 413–424, ISBN: 979-8-89176-418-7. DOI: 10.18653/v1/2026.trustnlp-main.27 [Online]. Available: https://aclanthology.org/2026.trustnlp-main.27/

[30] C. Sharma. "Retrieval-Augmented Generation: A Comprehensive Survey of Architectures, Enhancements, and Robustness Frontiers." arXiv: 2506.00054 [cs]. [Online]. Available: http://arxiv.org/abs/2506.00054

[31] K. K. Leung et al., "Classifying and Addressing the Diversity of Errors in Retrieval-Augmented Generation Systems," in *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, V. Demberg, K. Inui, and L. Marquez, Eds., Rabat, Morocco: Association for Computational Linguistics, Mar. 2026, pp. 3185–3207, ISBN: 979-8-89176-380-7. DOI: 10.18653/v1/2026.eacl-long.147 [Online]. Available: https://aclanthology.org/2026.eacl-long.147/

[32] P. Mandikal and R. Mooney, "Sparse meets dense: A hybrid approach to enhance scientific document retrieval," *The 4th workshop on scientific document understanding, AAAI*, Feb. 2024. [Online]. Available: http://www.cs.utexas.edu/users/ai-labpub-view.php?PubID=128046

[33] Z. Jin et al., "Tug-of-War between Knowledge: Exploring and Resolving Knowledge Conflicts in Retrieval-Augmented Language Models," in *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, N. Calzolari, M.-Y. Kan, V. Hoste, A. Lenci, S. Sakti, and N. Xue, Eds., Torino, Italia: ELRA and ICCL, May 2024, pp. 16 867–16 878. [Online]. Available: https://aclanthology.org/2024.lrec-main.1466/

[34] H. Wang, A. Prasad, E. Stengel-Eskin, and M. Bansal. "Retrieval-Augmented Generation with Conflicting Evidence." arXiv: 2504.13079 [cs]. [Online]. Available: http://arxiv.org/abs/2504.13079

[35] N. F. Liu et al., "Lost in the Middle: How Language Models Use Long Contexts," *Transactions of the Association for Computational Linguistics*, vol. 12, pp. 157–173, Feb. 23, 2024, ISSN: 2307-387X. DOI: 10.1162/tacl_a_00638 [Online]. Available: https://doi.org/10.1162/tacl_a_00638

[36] "Contextual Retrieval in AI Systems. "[Online]. Available: https://www.anthropic.com/engineering/contextual-retrieval

[37] X. Ma, Y. Gong, P. He, H. Zhao, and N. Duan, "Query Rewriting in Retrieval-Augmented Large Language Models," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, H. Bouamor, J. Pino, and K. Bali, Eds., Singapore: Association for Computational Linguistics, Dec. 2023, pp. 5303–5315. DOI: 10.18653/v1/2023.emnlp-main.322 [Online]. Available: https://aclanthology.org/2023.emnlp-main.322/

[38] Y. Zhu et al., "Large Language Models for Information Retrieval: A Survey," *ACM Transactions on Information Systems*, vol. 44, no. 1, 12:1–12:54, Nov. 14, 2025, ISSN: 1046-8188. DOI: 10.1145/3748304 [Online]. Available: https://dl.acm.org/doi/10.1145/3748304

[39] C.-M. Chan et al. "RQ-RAG: Learning to Refine Queries for Retrieval Augmented Generation." arXiv: 2404.00610 [cs]. [Online]. Available: http://arxiv.org/abs/2404.00610

[40] L. Gao, X. Ma, J. Lin, and J. Callan, "Precise Zero-Shot Dense Retrieval without Relevance Labels," in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, A. Rogers, J. Boyd-Graber, and N. Okazaki, Eds., Toronto, Canada: Association for Computational Linguistics, Jul. 2023, pp. 1762–1777. DOI: 10.18653/v1/2023.acl-long.99 [Online]. Available: https://aclanthology.org/2023.acl-long.99/

[41] G. V. Cormack, C. L. A. Clarke, and S. Buettcher, "Reciprocal rank fusion outperforms condorcet and individual rank learning methods," in *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, Boston MA USA: ACM, Jul. 19, 2009, pp. 758–759, ISBN: 978-1-60558-483-6. DOI: 10.1145/1571941.1572114 [Online]. Available: https://dl.acm.org/doi/10.1145/1571941.1572114

[42] M. G. Arivazhagan, L. Liu, P. Qi, X. Chen, W. Y. Wang, and Z. Huang, "Hybrid Hierarchical Retrieval for Open-Domain Question Answering," in *Findings of the Association for Computational Linguistics: ACL 2023*, A. Rogers, J. Boyd-Graber, and N. Okazaki, Eds., Toronto, Canada: Association for Computational Linguistics, Jul. 2023, pp. 10 680–10 689. DOI: 10.18653/v1/2023.findings-acl.679 [Online]. Available: https://aclanthology.org/2023.findings-acl.679/

[43] J. Carbonell and J. Goldstein, "The use of MMR, diversity-based reranking for reordering documents and producing summaries," in *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '98, New York, NY, USA: Association for Computing Machinery, Aug. 1, 1998, pp. 335–336, ISBN: 978-1-58113-015-7. DOI: 10.1145/290941.291025 [Online]. Available: https://dl.acm.org/doi/10.1145/290941.291025

[44] R. Nogueira and K. Cho. "Passage Re-ranking with BERT." arXiv: 1901.04085 [cs.IR]. [Online]. Available: http://arxiv.org/abs/1901.04085

[45] S. Humeau, K. Shuster, M.-A. Lachaux, and J. Weston. "Poly-encoders: Transformer Architectures and Pre-training Strategies for Fast and Accurate Multi-sentence Scoring." arXiv: 1905.01969 [cs.CL]. [Online]. Available: http://arxiv.org/abs/1905.01969

[46] Y. Huang and J. X. Huang, "A Survey on Retrieval-Augmented Text Generation for Large Language Models," *ACM Comput. Surv.*, vol. 58, no. 12, 300:1–300:38, May 15, 2026, ISSN: 0360-0300. DOI: 10.1145/3805774 [Online]. Available: https://dl.acm.org/doi/10.1145/3805774

[47] W. Su et al., "Parametric Retrieval Augmented Generation," in *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*, Padua Italy: ACM, Jul. 13, 2025, pp. 1240–1250, ISBN: 979-8-4007-1592-1. DOI: 10.1145/3726302.3729957 [Online]. Available: https://dl.acm.org/doi/10.1145/3726302.3729957

[48] Y. Zhou et al. "Trustworthiness in Retrieval-Augmented Generation Systems: A Survey." arXiv: 2409.10102 [cs.IR]. [Online]. Available: http://arxiv.org/abs/2409.10102

[49] S. Li et al. "RAGPerf: An End-to-End Benchmarking Framework for Retrieval-Augmented Generation Systems." arXiv: 2603.10765 [cs.PF]. [Online]. Available: http://arxiv.org/abs/2603.10765

[50] S. Es, J. James, L. Espinosa Anke, and S. Schockaert, "RAGAs: Automated Evaluation of Retrieval Augmented Generation," in *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, N. Aletras and O. De Clercq, Eds., St. Julians, Malta: Association for Computational Linguistics, Mar. 2024, pp. 150–158. DOI: 10.18653/v1/2024.eacl-demo.16 [Online]. Available: https://aclanthology.org/2024.eacl-demo.16/

[51] J. Van Elburg, P. Van Der Putten, and M. Marx, "Can We Evaluate RAGs with Synthetic Data?" In *Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, I. Koprinska, J. Mendes-Moreira, and P. Branco, Eds., vol. 2841, Cham: Springer Nature Switzerland, 2026, pp. 177–192, ISBN: 978-3-032-19101-4 978-3-032-19102-1. DOI: 10.1007/978-3-032-19102-1_11 [Online]. Available: https://link.springer.com/10.1007/978-3-032-19102-1_11

[52] Y. Katsis et al., "MTRAG: A Multi-Turn Conversational Benchmark for Evaluating Retrieval-Augmented Generation Systems," *Transactions of the Association for Computational Linguistics*, vol. 13, pp. 784–808, 2025. DOI: 10.1162/tacl.a.19 [Online]. Available: https://aclanthology.org/2025.tacl-1.36/

[53] Z. Yang et al., "HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, E. Riloff, D. Chiang, J. Hockenmaier, and J. Tsujii, Eds., Brussels, Belgium: Association for Computational Linguistics, Oct. 2018, pp. 2369–2380. DOI: 10.18653/v1/D18-1259 [Online]. Available: https://aclanthology.org/D18-1259/

[54] S. Rosenthal, A. Sil, R. Florian, and S. Roukos, "CLAPnq: Cohesive Long-form Answers from Passages in Natural Questions for RAG systems," *Transactions of the Association for Computational Linguistics*, vol. 13, pp. 53–72, 2025. DOI: 10.1162/tacl_a_00729 [Online]. Available: https://aclanthology.org/2025.tacl-1.3/

[55] T. Kwiatkowski et al., "Natural questions: A benchmark for question answering research," *Transactions of the Association for Computational Linguistics*, vol. 7, pp. 453–466, Aug. 2019, ISSN: 2307-387X. DOI: 10.1162/tacl_a_00276 eprint: https://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00276/1923288/tacl_a_00276.pdf. [Online]. Available: https://doi.org/10.1162/tacl_a_00276

[56] D. Rau et al., "BERGEN: A Benchmarking Library for Retrieval-Augmented Generation," in *Findings of the Association for Computational Linguistics: EMNLP 2024*, Y. Al-Onaizan, M. Bansal, and Y.-N. Chen, Eds., Miami, Florida, USA: Association for Computational Linguistics, Nov. 2024, pp. 7640–7663. DOI: 10.18653/v1/2024.findings-emnlp.449 [Online]. Available: https://aclanthology.org/2024.findings-emnlp.449/

[57] T. Kenneweg, P. Kenneweg, and B. Hammer. "Retrieval Augmented Generation Systems: Automatic Dataset Creation, Evaluation and Boolean Agent Setup." arXiv: 2403.00820 [cs.IR]. [Online]. Available: http://arxiv.org/abs/2403.00820

[58] R. Teixeira de Lima et al., "Know Your RAG: Dataset Taxonomy and Generation Strategies for Evaluating RAG Systems," in *Proceedings of the 31st International Conference on Computational Linguistics: Industry Track*, O. Rambow et al., Eds., Abu Dhabi, UAE: Association for Computational Linguistics, Jan. 2025, pp. 39–57. [Online]. Available: https://aclanthology.org/2025.coling-industry.4/

[59] N. Pipitone and G. H. Alami. "LegalBench-RAG: A Benchmark for Retrieval-Augmented Generation in the Legal Domain." arXiv: 2408.10343 [cs]. [Online]. Available: http://arxiv.org/abs/2408.10343

[60] C. Choi et al., "FinDER: Financial Dataset for Question Answering and Evaluating Retrieval-Augmented Generation," in *Proceedings of the 6th ACM International Conference on AI in Finance*, ser. ICAIF '25, New York, NY, USA: Association for Computing Machinery, Nov. 14, 2025, pp. 638–646, ISBN: 979-8-4007-2220-2. DOI: 10.1145/3768292.3770361 [Online]. Available: https://dl.acm.org/doi/10.1145/3768292.3770361

[61] J. Saad-Falcon, O. Khattab, C. Potts, and M. Zaharia, "ARES: An Automated Evaluation Framework for Retrieval-Augmented Generation Systems," in *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, K. Duh, H. Gomez, and S. Bethard, Eds., Mexico City, Mexico: Association for Computational Linguistics,

Jun. 2024, pp. 338–354. DOI: 10.18653/v1/2024.naacl-long.20 [Online]. Available: https://aclanthology.org/2024.naacl-long.20/

[62] J. Chen, H. Lin, X. Han, and L. Sun, "Benchmarking Large Language Models in Retrieval-Augmented Generation," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 16, pp. 17 754–17 762, Mar. 24, 2024, ISSN: 2374-3468. DOI: 10.1609/aaai.v38i16.29728 [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/29728

[63] Y. Tang and Y. Yang, "MultiHop-RAG: Benchmarking Retrieval-Augmented Generation for Multi-Hop Queries," presented at the First Conference on Language Modeling, Aug. 26, 2024. [Online]. Available: https://openreview.net/forum?id=t4eB3zYWBK

[64] Y. Lyu et al., "CRUD-RAG: A Comprehensive Chinese Benchmark for Retrieval-Augmented Generation of Large Language Models," *ACM Transactions on Information Systems*, vol. 43, no. 2, pp. 1–32, Mar. 31, 2025, ISSN: 1046-8188, 1558-2868. DOI: 10.1145/3701228 [Online]. Available: https://dl.acm.org/doi/10.1145/3701228

[65] K. Zhu et al., "RAGEval: Scenario Specific RAG Evaluation Dataset Generation Framework," in *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, W. Che, J. Nabende, E. Shutova, and M. T. Pilehvar, Eds., Vienna, Austria: Association for Computational Linguistics, Jul. 2025, pp. 8520–8544, ISBN: 979-8-89176-251-0. DOI: 10.18653/v1/2025.acl-long.418 [Online]. Available: https://aclanthology.org/2025.acl-long.418/

[66] S. Filice, G. Horowitz, D. Carmel, Z. Karnin, L. Lewin-Eytan, and Y. Maarek, "Generating Q&A Benchmarks for RAG Evaluation in Enterprise Settings," in *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 6: Industry Track)*, G. Rehm and Y. Li, Eds., Vienna, Austria: Association for Computational Linguistics, Jul. 2025, pp. 469–484, ISBN: 979-8-89176-288-6. DOI: 10.18653/v1/2025.acl-industry.33 [Online]. Available: https://aclanthology.org/2025.acl-industry.33/

[67] "Golden Synthesizer | DeepEval - The LLM Evaluation Framework. "[Online]. Available: https://deepeval.com/docs/golden-synthesizer

[68] H. Yu, A. Gan, K. Zhang, S. Tong, Q. Liu, and Z. Liu, "Evaluation of Retrieval-Augmented Generation: A Survey," in *Big Data*, W. Zhu et al., Eds., Singapore: Springer Nature, 2025, pp. 102–120, ISBN: 978-981-96-1024-2. DOI: 10.1007/978-981-96-1024-2_8

[69] L. Brehme, T. Ströhle, and R. Breu, "Can LLMs be Trusted for Evaluating RAG Systems? A Survey of Methods and Datasets," in *2025 IEEE Swiss Conference on Data Science (SDS)*, Jun. 2025, pp. 16–23. DOI: 10.1109/SDS66131.2025.00010 [Online]. Available: https://ieeexplore.ieee.org/document/11081490/

[70] A. Gan et al. "Retrieval Augmented Generation Evaluation in the Era of Large Language Models: A Comprehensive Survey." arXiv: 2504.14891 [cs]. [Online]. Available: http://arxiv.org/abs/2504.14891

[71] A. Salemi and H. Zamani, "Evaluating Retrieval Quality in Retrieval-Augmented Generation," in *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '24, New York, NY, USA: Association for Computing Machinery, Jul. 11, 2024, pp. 2395–2400, ISBN: 979-8-4007-0431-4. DOI: 10.1145/3626772.3657957 [Online]. Available: https://dl.acm.org/doi/10.1145/3626772.3657957

[72] D. Ru et al., "RAGCHECKER: A Fine-grained Framework for Diagnosing Retrieval-Augmented Generation," in *Advances in Neural Information Processing Systems 37 (NeurIPS 2024)*, Curran Associates, Inc., Dec. 10, 2024.

[73] S. Knollmeyer et al., "Evaluating Retrieval Augmented Generation: A Comprehensive Review of Evaluation Dimensions, Question Types, and Application," *SN Computer Science*, vol. 7, no. 5, p. 576, Jun. 16, 2026, ISSN: 2661-8907. DOI: 10.1007/s42979-026-05134-x [Online]. Available: https://doi.org/10.1007/s42979-026-05134-x

[74] T. Zhang, V. Kishore, F. Wu, K. Q. Weinberger, and Y. Artzi, "BERTScore: Evaluating Text Generation with BERT," 2020.

[75] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, K. Inui, J. Jiang, V. Ng, and X. Wan, Eds., Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 3982–3992. DOI: 10.18653/v1/D19-1410 [Online]. Available: https://aclanthology.org/D19-1410/

[76] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: A Method for Automatic Evaluation of Machine Translation," in *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, P. Isabelle, E. Charniak, and D. Lin, Eds., Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, Jul. 2002, pp. 311–318. DOI: 10.3115/1073083.1073135 [Online]. Available: https://aclanthology.org/P02-1040/

[77] C.-Y. Lin, "ROUGE: A Package for Automatic Evaluation of Summaries," in *Text Summarization Branches Out*, Barcelona, Spain: Association for Computational Linguistics, Jul. 2004, pp. 74–81. [Online]. Available: https://aclanthology.org/W04-1013/

[78] C.-W. Liu, R. Lowe, I. Serban, M. Noseworthy, L. Charlin, and J. Pineau, "How NOT To Evaluate Your Dialogue System: An Empirical Study of Unsupervised Evaluation Metrics for Dialogue Response Generation," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, J. Su, K. Duh, and X. Carreras, Eds., Austin, Texas: Association for Computational Linguistics, Nov. 2016, pp. 2122–2132. DOI: 10.18653/v1/D16-1230 [Online]. Available: https://aclanthology.org/D16-1230/

[79] A. Ammar, A. Koubaa, O. Nacar, and W. Boulila. "Optimizing Retrieval-Augmented Generation: Analysis of Hyperparameter Impact on Performance and Efficiency." arXiv: 2505.08445 [cs]. [Online]. Available: http://arxiv.org/abs/2505.08445

[80] H. Li et al. “LLMs-as-Judges: A Comprehensive Survey on LLM-based Evaluation Methods.” arXiv: 2412.05579 [cs.CL]. [Online]. Available: http://arxiv.org/abs/2412.05579

[81] T. Zhang, L. Patterson, B. Webb, Z. Jin, and M. Beiting-Parrish, “Evaluating generative AI chatbots for large-scale assessment data: Comparing LLM-as-a-judge and human ratings,” *Large-scale Assessments in Education*, vol. 14, no. 1, p. 19, Mar. 12, 2026, ISSN: 2196-0739. DOI: 10.1186/s40536-026-00287-w [Online]. Available: https://link.springer.com/10.1186/s40536-026-00287-w

[82] D. Chan, S. Petryk, J. Gonzalez, T. Darrell, and J. Canny, “CLAIR: Evaluating Image Captions with Large Language Models,” in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, Singapore: Association for Computational Linguistics, 2023, pp. 13 638–13 646. DOI: 10.18653/v1/2023.emnlp-main.841 [Online]. Available: https://aclanthology.org/2023.emnlp-main.841

[83] J. Ye et al. “Justice or Prejudice? Quantifying Biases in LLM-as-a-Judge.” arXiv: 2410.02736 [cs]. [Online]. Available: http://arxiv.org/abs/2410.02736

[84] S. Kim et al., “Prometheus 2: An Open Source Language Model Specialized in Evaluating Other Language Models,” in *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, Y. Al-Onaizan, M. Bansal, and Y.-N. Chen, Eds., Miami, Florida, USA: Association for Computational Linguistics, Nov. 2024, pp. 4334–4353. DOI: 10.18653/v1/2024.emnlp-main.248 [Online]. Available: https://aclanthology.org/2024.emnlp-main.248/

[85] C.-H. Chiang and H.-y. Lee, “Can Large Language Models Be an Alternative to Human Evaluations?” In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, A. Rogers, J. Boyd-Graber, and N. Okazaki, Eds., Toronto, Canada: Association for Computational Linguistics, Jul. 2023, pp. 15 607–15 631. DOI: 10.18653/v1/2023.acl-long.870 [Online]. Available: https://aclanthology.org/2023.acl-long.870/

[86] J. Ip and K. Vongthongsri, *Deepeval*, version 4.1.3, Jul. 2026. [Online]. Available: https://github.com/confident-ai/deepeval

[87] D. Kim, B. Kim, D. Han, and M. Eibich. “AutoRAG: Automated Framework for optimization of Retrieval Augmented Generation Pipeline.” arXiv: 2410.20878 [cs]. [Online]. Available: http://arxiv.org/abs/2410.20878

[88] J. Jin et al., “FlashRAG: A Modular Toolkit for Efficient Retrieval-Augmented Generation Research,” in *Companion Proceedings of the ACM on Web Conference 2025*, Sydney NSW Australia: ACM, May 8, 2025, pp. 737–740, ISBN: 979-8-4007-1331-6. DOI: 10.1145/3701716.3715313 [Online]. Available: https://dl.acm.org/doi/10.1145/3701716.3715313

[89] *Vibrantlabsai/ragas*, Vibrant Labs, Aug. 23, 2026. [Online]. Available: https://github.com/vibrantlabsai/ragas

[90] *Truera/trulens*, TruEra, Aug. 11, 2026. [Online]. Available: https://github.com/truera/trulens

[91] C. Auer et al. “Docling Technical Report.” arXiv: 2408.09869 [cs]. [Online]. Available: http://arxiv.org/abs/2408.09869

[92] M. Grootendorst, *MaartenGr/KeyBERT: BibTeX*, version v0.1.3, Zenodo, Jan. 25, 2021. DOI: 10.5281/ZENODO.4461265 [Online]. Available: https://zenodo.org/record/4461265

[93] “Google-research/rouge at master · google-research/google-research,” GitHub. [Online]. Available: https://github.com/google-research/google-research/tree/master/rouge

[94] M. Post, “A Call for Clarity in Reporting BLEU Scores,” in *Proceedings of the Third Conference on Machine Translation: Research Papers*, O. Bojar et al., Eds., Brussels, Belgium: Association for Computational Linguistics, Oct. 2018, pp. 186–191. DOI: 10.18653/v1/W18-6319 [Online]. Available: https://aclanthology.org/W18-6319/

[95] J. R. Landis and G. G. Koch, “The measurement of observer agreement for categorical data,” *Biometrics. Journal of the International Biometric Society*, vol. 33, no. 1, pp. 159–174, 1977, ISSN: 0006341X, 15410420. JSTOR: 2529310. [Online]. Available: http://www.jstor.org/stable/2529310

[96] A. R. Feinstein and D. V. Cicchetti, “High agreement but low Kappa: I. the problems of two paradoxes,” *Journal of Clinical Epidemiology*, vol. 43, no. 6, pp. 543–549, Jan. 1990, ISSN: 08954356. DOI: 10.1016/0895-4356(90)90158-L [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/089543569090158L

[97] K. L. Gwet, “Computing inter-rater reliability and its variance in the presence of high agreement,” *British Journal of Mathematical and Statistical Psychology*, vol. 61, no. 1, pp. 29–48, 2008. DOI: 10.1348/000711006X126600 eprint: https://bpspsychub.onlinelibrary.wiley.com/doi/pdf/10.1348/000711006X126600. [Online]. Available: https://bpspsychub.onlinelibrary.wiley.com/doi/abs/10.1348/000711006X126600

[98] Brown Simon, *The C4 Model*. 2026, ISBN: 979-8-3416-6011-3. [Online]. Available: https://learning.oreilly.com/library/view/the-c4-model/9798341660113/

[99] J. Huang and K. Chang, “Citation: A Key to Building Responsible and Accountable Large Language Models,” in *Findings of the Association for Computational Linguistics: NAACL 2024*, K. Duh, H. Gomez, and S. Bethard, Eds., Mexico City, Mexico: Association for Computational Linguistics, Jun. 2024, pp. 464–473. DOI: 10.18653/v1/2024.findings-naacl.31 [Online]. Available: https://aclanthology.org/2024.findings-naacl.31/

[100] N. Liu, T. Zhang, and P. Liang, “Evaluating Verifiability in Generative Search Engines,” in *Findings of the Association for Computational Linguistics: EMNLP 2023*, H. Bouamor, J. Pino, and K. Bali, Eds., Singapore: Association for Computational Linguistics, Dec. 2023, pp. 7001–7025. DOI: 10.18653/v1/2023.findings-emnlp.467 [Online]. Available: https://aclanthology.org/2023.findings-emnlp.467/

[101] “spaCy · Industrial-strength Natural Language Processing in Python. ”[Online]. Available: https://spacy.io/

[102] F. Cuconasu et al., "The Power of Noise: Redefining Retrieval for RAG Systems," in *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, Washington DC USA: ACM, Jul. 10, 2024, pp. 719–729, ISBN: 979-8-4007-0431-4. DOI: 10.1145/3626772.3657834 [Online]. Available: https://dl.acm.org/doi/10.1145/3626772.3657834

[103] L. Chen, Y. Cai, J. Dong, and Y. Wang. "BRIGHT+: Upgrading the BRIGHT Benchmark with MARCUS, a Multi-Agent RAG Clean-Up Suite." arXiv: 2506.07116 [cs.AI]. [Online]. Available: http://arxiv.org/abs/2506.07116

[104] N. Livathinos et al. "Docling: An Efficient Open-Source Toolkit for AI-driven Document Conversion." arXiv: 2501.17887 [cs]. [Online]. Available: http://arxiv.org/abs/2501.17887

[105] S. R. Bowman, G. Angeli, C. Potts, and C. D. Manning, "A large annotated corpus for learning natural language inference," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, L. Màrquez, C. Callison-Burch, and J. Su, Eds., Lisbon, Portugal: Association for Computational Linguistics, Sep. 2015, pp. 632–642. DOI: 10.18653/v1/D15-1075 [Online]. Available: https://aclanthology.org/D15-1075/

[106] W.-L. Chiang et al., "Chatbot arena: An open platform for evaluating LLMs by human preference," in *Proceedings of the 41st International Conference on Machine Learning*, ser. ICML'24, vol. 235, Vienna, Austria: JMLR.org, Jul. 21, 2024, pp. 8359–8388.

[107] K. Enevoldsen et al., "MMTEB: Massive Multilingual Text Embedding Benchmark," 2025.

[108] "Introducing RTEB: A New Standard for Retrieval Evaluation. "[Online]. Available: https://huggingface.co/blog/rteb

[109] S. Han, G. T. Junior, T. Balough, and W. Zhou. "Judge's Verdict: A Comprehensive Analysis of LLM Judge Capability Through Human Agreement." arXiv: 2510.09738 [cs.CL]. [Online]. Available: http://arxiv.org/abs/2510.09738

[110] T. Elrod, R. D. Johnson, and J. White, "A new integrated model of noncompensatory and compensatory decision strategies," *Organizational Behavior and Human Decision Processes*, vol. 95, no. 1, pp. 1–19, Sep. 2004, ISSN: 07495978. DOI: 10.1016/j.obhdp.2004.06.002 [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0749597804000573

[111] *Pytest-dev/pytest*, pytest-dev, Aug. 10, 2026. [Online]. Available: https://github.com/pytest-dev/pytest

[112] L. Barbosa and A. Hora, "How and Why Developers Migrate Python Tests," in *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, Mar. 2022, pp. 538–548. DOI: 10.1109/SANER53432.2022.00071 [Online]. Available: https://ieeexplore.ieee.org/document/9825863/

[113] V. Khorikov, *Unit Testing Principles, Practices, and Patterns*. New York: Manning Publications Co. LLC, 2020, 1 p., ISBN: 978-1-61729-627-7 978-1-63835-029-3.

[114] B. García, F. Ricca, M. Leotta, and M. Munoz-Organero, "Test automation with selenium: A survey," *Information and Software Technology*, vol. 194, p. 108 077, Jun. 1, 2026, ISSN: 0950-5849. DOI: 10.1016/j.infsof.2026.108077 [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584926000662

[115] I. Kertusha, G. Assres, O. Duman, and A. Arcuri, "A survey on web testing: On the rise of AI and applications in industry," *Science of Computer Programming*, vol. 253, p. 103 473, Aug. 2026, ISSN: 01676423. DOI: 10.1016/j.scico.2026.103473 [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0167642326000390

[116] *Cypress-io/cypress*, Cypress.io, Aug. 10, 2026. [Online]. Available: https://github.com/cypress-io/cypress

[117] B. García, J. M. del Alamo, M. Leotta, and F. Ricca, "Exploring Browser Automation: A Comparative Study of Selenium, Cypress, Puppeteer, and Playwright," in *Quality of Information and Communications Technology*, A. Bertolino, J. Pascoal Faria, P. Lago, and L. Semini, Eds., Cham: Springer Nature Switzerland, 2024, pp. 142–149, ISBN: 978-3-031-70245-7. DOI: 10.1007/978-3-031-70245-7_10

[118] Z. M. Jiang and A. E. Hassan, "A Survey on Load Testing of Large-Scale Software Systems," *IEEE Transactions on Software Engineering*, vol. 41, no. 11, pp. 1091–1118, Nov. 2015, ISSN: 1939-3520. DOI: 10.1109/TSE.2015.2445340 [Online]. Available: https://ieeexplore.ieee.org/document/7123673/

[119] J. Heyman, *Locustio/locust*, Locust, Aug. 9, 2026. [Online]. Available: https://github.com/locustio/locust

[120] B. Schroeder, A. Wierman, and M. Harchol-Balter, "Open Versus Closed: A Cautionary Tale," in *3rd Symposium on Networked Systems Design & Implementation (NSDI 06)*, San Jose, CA: USENIX Association, May 2006. [Online]. Available: https://www.usenix.org/conference/nsdi-06/open-versus-closed-cautionary-tale

[121] A. Holtzman, J. Buys, L. Du, M. Forbes, and Y. Choi, "The curious case of neural text degeneration," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=rygGQyrFvH

[122] OECD, *Recommendation of the council on artificial intelligence*, OECD/LEGAL/0449, 2019. [Online]. Available: https://legalinstruments.oecd.org/en/instruments/oecd-legal-0449

Appendix A

Evaluation Toolbox: Technical Details

This appendix describes the technical artifacts of the evaluation toolbox (Chapter 4). It shows the configuration files of both pipelines and the schemas that define their inputs and outputs.

A.1 Q&A Generation Pipeline

This section documents the configuration of a generation run and the schema of the resulting dataset.

A.1.1 Configuration File

The complete YAML configuration used to generate the datasets for the RAI Copilot benchmark and the human annotation study is presented below. For the validation dataset, only the seed and the number of questions were changed.

```yaml
dataset:
  name: rai_qa
  description: 'RAI Q&A dataset'
  language: en
  domain: responsible_ai
  num_questions: 550
  seed: 42

documents_dir: ../../examples/documents/docs_rai
output_dir: ../../examples/datasets

store_failures: True

max_validation_attempts: 10

question_types: [factual, comparative, definitional]

multi_turn:
  rate: 0.2
  min_turns: 2
  max_turns: 6

refusal:
  rate: 0.1
  categories:
```

```
    out_of_scope:
      - cooking recipes
      - sports results
      - travel planning
      - music recommendations
      - gardening
      - movie reviews
    harmful:
      - hacking techniques
      - bypassing safety filters
      - system misuse
      - offensive language
      - false information
    personal_info:
      - home address
      - contact details
      - salary details
      - health information

multi_hop:
  rate: 0.2
  min_chunks: 2
  max_chunks: 3

chunking:
  max_tokens: 4096

generator:
  model: Qwen/Qwen3-Next-80B-A3B-Instruct-FP8
  api_key: EMPTY
  base_url: http://<host>.zhaw.ch:8688/v1
  temperature: 0.7
  top_p: 0.8
  single_turn_prompt: ./qa_generation/llms/prompts/question_single_turn.txt
  multi_turn_prompt: ./qa_generation/llms/prompts/question_multi_turn.txt
  refusal_prompt: ./qa_generation/llms/prompts/question_refusal.txt
  multi_hop_prompt: ./qa_generation/llms/prompts/question_multi_hop.txt
  reference_prompt: ./qa_generation/llms/prompts/reference.txt

validator:
  model: openai/gpt-oss-20b
  api_key: EMPTY
  base_url: http://<host>.zhaw.ch:8689/v1
  temperature: 0.0
  standard_prompt: ./qa_generation/llms/prompts/validation_standard.txt
  refusal_prompt: ./qa_generation/llms/prompts/validation_refusal.txt
  multi_hop_prompt: ./qa_generation/llms/prompts/validation_multi_hop.txt
  reference_prompt: ./qa_generation/llms/prompts/validation_reference.txt
```

A.1.2 Dataset Schema

The following snippet presents the Pydantic schema of the Q&A dataset generated by the Q&A generation pipeline. This data structure is also enforced by the evaluation pipeline (Section 4.3) on the input dataset.

```
class ChatMessage(BaseModel):
  role: Literal['assistant', 'user']
  content: str

```

```
class ReferenceChunk(BaseModel):
  text: str
  source: Path

class ValidationVerdict(BaseModel):
  is_valid: bool
  reason: str | None = None

class QAItem(BaseModel):
  id: str
  question: str
  reference_answer: str | None = None
  reference_chunks: list[ReferenceChunk] = []
  history: list[ChatMessage] = []
  category: Literal['single_turn', 'multi_turn', 'multi_hop', 'refusal']
  is_multi_turn: bool = False
  is_multi_hop: bool = False
  should_refuse: bool = False
  refusal_domain: str | None = None
  refusal_category: str | None = None
  refusal_reason: str | None = None
  question_type: str | None = None
  bridge_entity: str | None = None
  question_validation: ValidationVerdict | None = None
  reference_validation: ValidationVerdict | None = None

class QADataset(BaseModel):
  name: str
  version: str = '1.0'
  language: str = 'en'
  description: str
  items: list[QAItem]

  num_question_target: int | None = None
  num_questions: int | None = None
  generated_at: str | None = None
  generator_config: dict = {}
  run_stats: dict = {}
```

A.2 Evaluation Pipeline

This section documents the configuration of an evaluation run. It further shows the schema of the answer files the pipeline consumes, and the schema of the reports it produces.

A.2.1 Configuration File

The following YAML file configures the evaluation pipeline to evaluate the SUT. The contextual-relevancy override shows the final state after the ensemble size was reduced mid-run (Section 7.1.4).

```
answers_dir: ../../examples/answer-files
qa_dataset: ../../examples/datasets/rai_qa.json
output_dir: ../../examples/eval-reports

n_ensemble_runs: 5

metrics:
```

```
  retrieval: [contextual_recall, contextual_precision, rank_agreement,
      contextual_relevancy]
  generation: [faithfulness, answer_relevancy, citation_accuracy, bert_score]
  statistical: [rouge_score, bleu_score]

judge:
  model: meta-llama/Llama-3.3-70B-Instruct
  base_url: http://<host>.zhaw.ch:8820/v1
  api_key: EMPTY
  temperature: 0.7
  top_p: 0.8
  max_completion_tokens: 32768

refusal_clf:
  model: microsoft/phi-4
  base_url: http://<host>.zhaw.ch:8821/v1
  temperature: 0.0
  api_key: EMPTY
  prompt: ./evaluation/refusal/prompts/refusal_clf.txt

embedding:
  model: nvidia/llama-nemotron-embed-1b-v2
  base_url: http://<host>.zhaw.ch:8822/v1
  api_key: EMPTY

metric_overrides:
  answer_relevancy:
    n_generations: 5
  contextual_relevancy:
    max_chunks: 5
    n_ensemble: 1
```

A.2.2 Answer File Schema

The following Pydantic schema defines the structure of an answer file evaluated by the evaluation pipeline. The SUT is expected to produce answer files adhering to this schema.

```
class ChunkMetadata(BaseModel):
  filename: str | None = None
  doi: str | None = None
  file_hash: str | None = None
  title: str
  author: str
  year: int
  url: str
  citationCount: int | str | None = None
  journal: dict | str | None = None
  category: str
  access: str
  is_approved: bool
  created_at: str | None = None

class RetrievedChunk(BaseModel):
  id: str
  content: str
  score: float | None = None
  rank: int
  metadata: ChunkMetadata | None = None

```

```
class AnswerRecord(BaseModel):
  qa_item_id: str
  response: str
  enhanced_question: str | None = None
  retrieved_chunks: list[RetrievedChunk] = []
  num_retrieved_chunks: int = 0
  timing: dict = {}
  token_usage: dict = {}

class AnswerFile(BaseModel):
  version: str = '1.0'
  config_id: str
  config_mode: str
  config: dict
  seed: int | None = None
  qa_dataset_name: str | None = None
  records: list[AnswerRecord]
  num_records: int = 0
  vdb_update_time_ms: float | None = None
  generated_at: str | None = None
```

A.2.3 Evaluation Report Schema

The following Pydantic schema defines the structure of the evaluation report the pipeline produces for each answer file (Section 4.3.2). The `aggregates` field holds the per-metric averages, their evaluable counts, and the refusal rates.

```
class MetricResult(BaseModel):
  metric: str
  ensemble_scores: list[float] | None = None
  score: float | None
  raw: dict | list | None = None
  error: bool = False
  error_txt: str | None = None
  generated_by: str | None = None

class ItemEvaluation(BaseModel):
  qa_item_id: str
  refusal: bool
  refusal_reason: str
  refusal_handling: Literal['tp', 'fn', 'fp', 'tn', 'error'] | None
  metric_results: list[MetricResult]

class EvalReport(BaseModel):
  config_id: str
  n_total: int
  n_evaluated: int
  results: list[ItemEvaluation]
  aggregates: dict = {}
  evaluated_at: str | None = None
  judge_config: dict = {}
```

Appendix B

Toolbox Validation: Study Materials

This appendix provides additional material for the human annotation study described in Section 5.2.

B.1 Annotation Tool

Figure B.1 shows the interface through which the annotations were collected. The context shown in the left panel is the same context that was provided to the LLM judge, so that both raters decide on identical evidence.

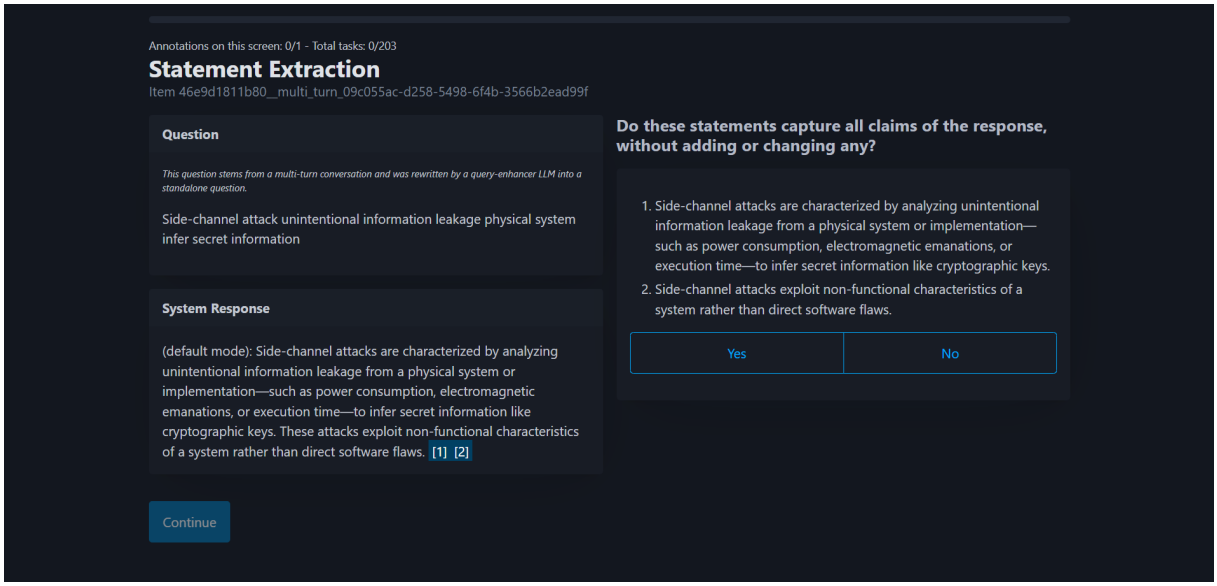

Figure B.1: The interface of the annotation study tool, which was used to collect human expert labels. The left panel shows the context that was also provided to the LLM judge. The right panel shows the binary annotation task for the given context.

B.2 Selected Configurations

Table B.1 lists the five RAI Copilot configurations that were re-run against the validation set for the human annotation study (Section 5.2.1). The query enhancer was Phi-4 in all five

configurations. The composite is defined as $0.65 \cdot \text{faithfulness} + 0.25 \cdot \text{answer relevancy} + 0.10 \cdot \text{citation accuracy}$ and was used solely to spread the sample across the observed quality range. The full hyperparameters and benchmark scores of these configurations are provided in Section D.2.

ID	Generator	Benchmark Composite	Selected as	Items	Units	Annotated
C05	gemma-31b	0.910	top	24	417	152
C11	mistral-24b	0.900	top	24	289	117
C08	mistral-24b	0.881	middle	24	261	87
C16	nemotron-120b	0.867	middle	24	330	113
C01	apertus-8b	0.767	low	23	251	70
Total				119	1,548	539

Table B.1: Configurations used in the human annotation study, ordered by the composite score. *Items* counts the distinct question–answer pairs drawn from each configuration. Of the 119 items in total, 19 were assigned to all five annotators and the remaining 100 to one annotator each, so that every annotator received 39 items. *Units* counts the resulting binary annotation tasks, which vary per configuration because the number of retrieved chunks, extracted statements, and citation markers differs between them. *Annotated* counts the units that received at least one label.

B.3 Verdict Distributions

Table B.2 reports how the verdicts of the LLM judge and of the author are jointly distributed for each annotation task. The agreement coefficients of Table 5.3 are computed from these counts.

		Positive verdicts (%)		Joint outcomes			
Task	n	Human	Judge	Both yes	Both no	Judge only	Human only
Faithfulness	114	91.2	95.6	101	2	8	3
Contextual recall	68	73.5	77.9	48	13	5	2
Contextual precision	184	39.7	72.8	73	50	61	0
Citation accuracy	101	87.1	99.0	87	0	13	1
Refusal	39	15.4	15.4	6	33	0	0

Table B.2: Joint distribution of the LLM judge's and the author's verdicts, over the units the author rated. *Positive verdicts* gives the share of units each rater marked *yes*. *Judge only* and *Human only* count the units on which exactly one of the two did so.

Appendix C

RAI Copilot: Technical Details

This appendix supplements Chapter 6 with a visual overview of the user interface, the document metadata schema, the deployed configuration, and the prompt templates of the RAI Copilot.

C.1 User Interface

This section provides a visual overview of the RAI Copilot's user interface.

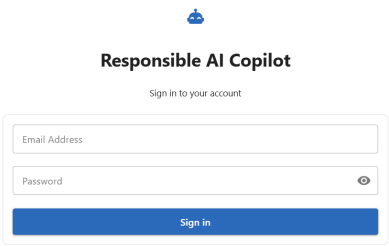

Figure C.1: The login view of the RAI Copilot.

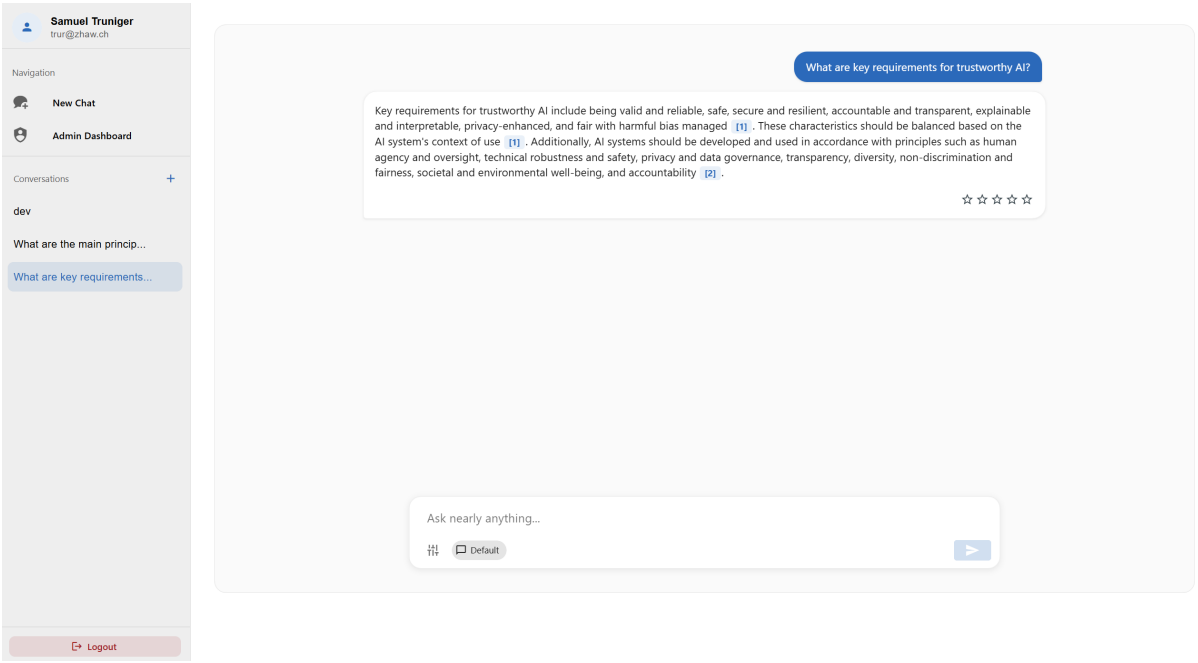

Figure C.2: The chat interface after authentication.

Figure C.3: Citation highlighting in the RAI Copilot. Clicking a citation marker opens the referenced chunk [122], in which the passages supporting the cited claims are highlighted.

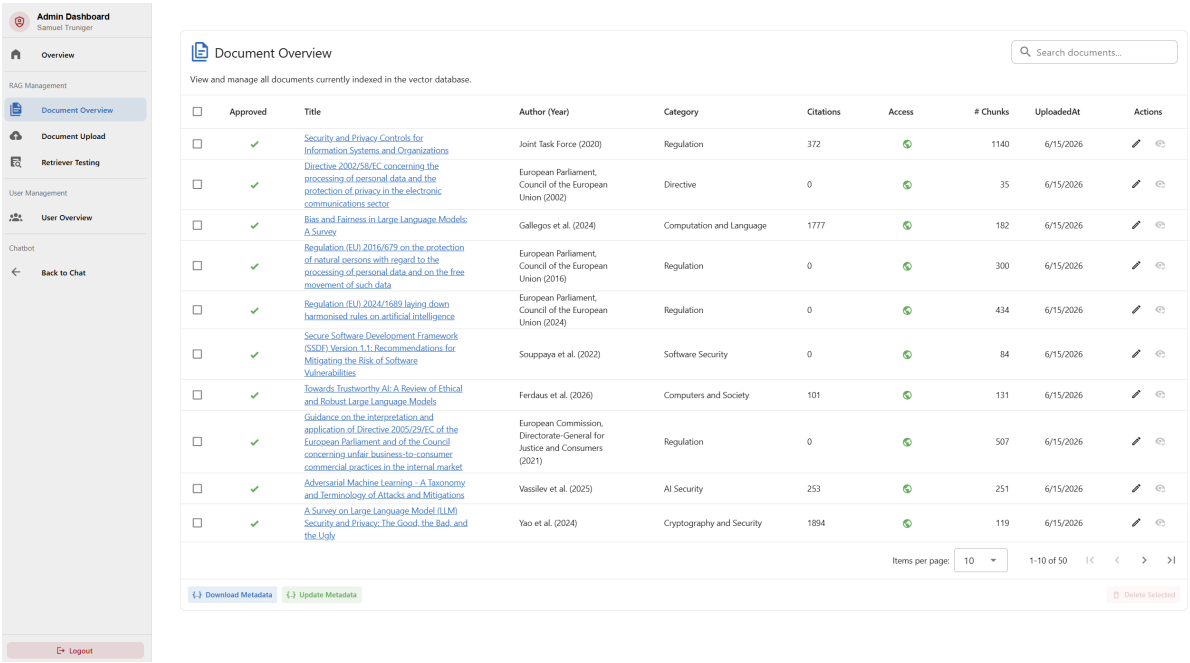

Figure C.4: The document overview, which lists all indexed documents and provides access to the metadata management operations described in Section 6.3.5.

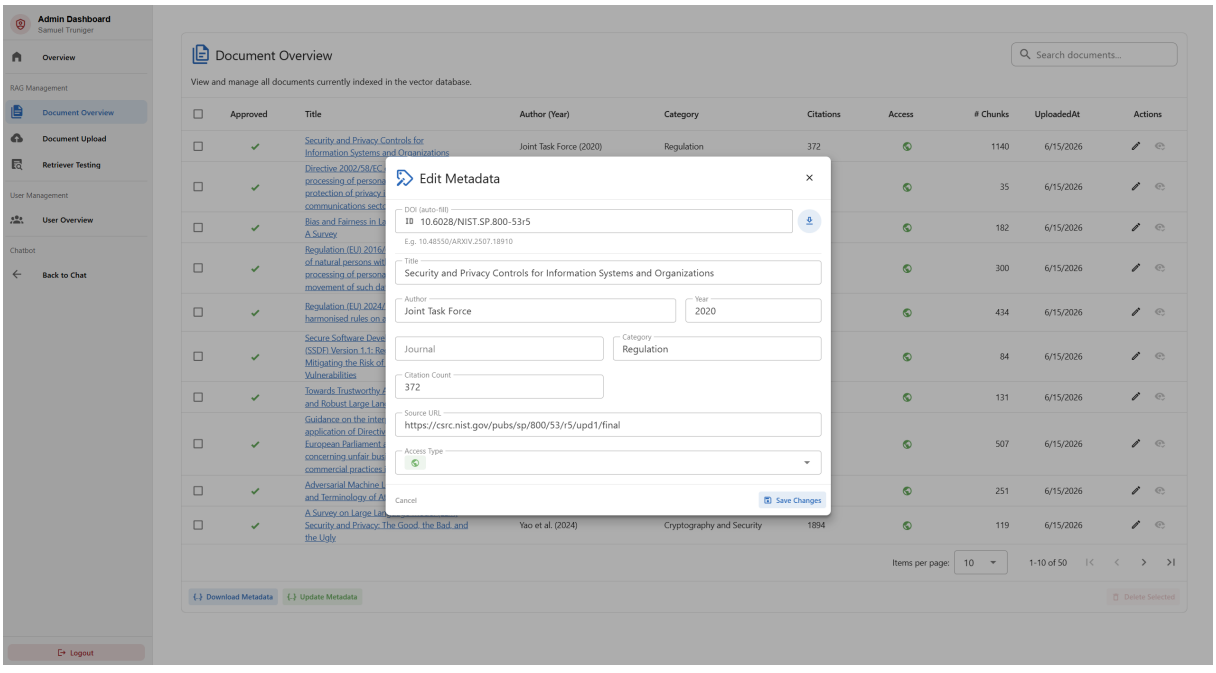

Figure C.5: The metadata management window that allows admin users to edit the metadata of indexed documents.

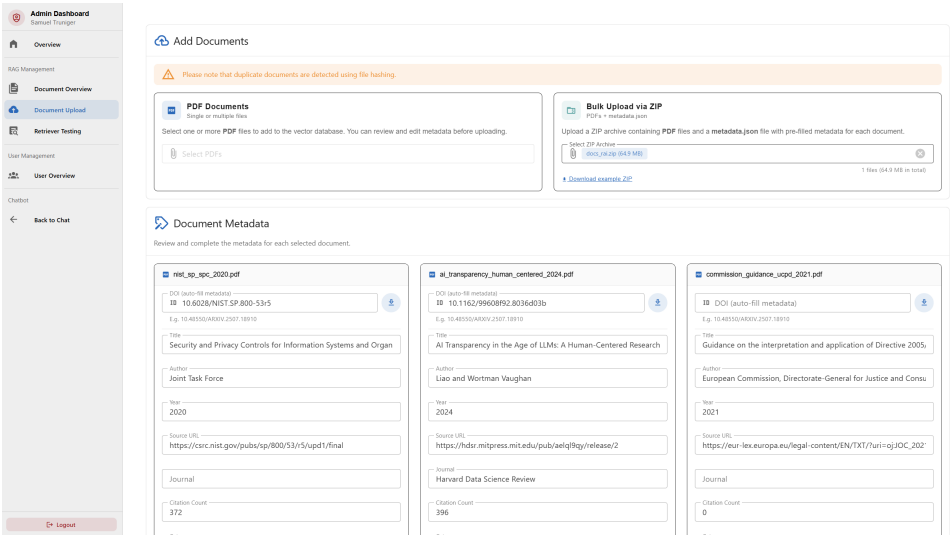

Figure C.6: The document upload interface, which triggers the document pipeline (Section 6.2.1) for individual PDF files or ZIP archives.

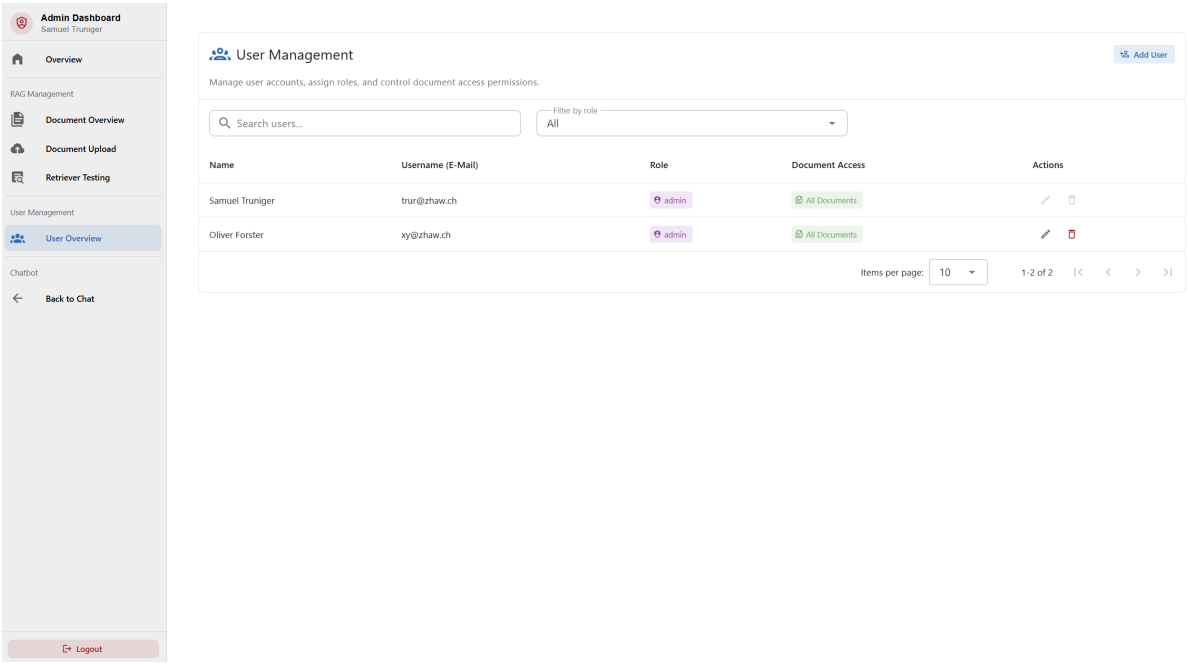

Figure C.7: The user management interface, through which admin users manage accounts and roles.

C.2 Document Metadata Schema

Each document chunk in the knowledge base is associated with a metadata record. The fields below are specified when a document is uploaded. The `access` and `is_approved` fields are governance flags that control visibility and approval status, while the remaining fields capture bibliographic information.

```
{
  "id": 1,
  "filename": "nist_sp_spc_2020.pdf",
  "doi": "10.6028/NIST.SP.800-53r5",
  "file_hash": "aec0f6d8ba5f435c7924d24ff83ae92889414cb27329d0ed5dd399ac465e8714",
```

```
  "title": "Security and Privacy Controls for Information Systems and Organizations",
  "author": "Joint Task Force",
  "year": 2020,
  "url": "https://csrc.nist.gov/pubs/sp/800/53/r5/upd1/final",
  "citationCount": 372,
  "journal": "",
  "category": "Regulation",
  "access": "public",
  "is_approved": true,
  "created_at": "2026-06-23T19:25:03.737892+00:00"
}
```


C.3 Configuration

Table C.1 lists the environment variables relevant to the system behavior described in this thesis. The complete set of deployment and runtime settings is documented in the project repository. The values shown are those of the deployed system, i.e., the champion configuration C11 (Table D.2).

Variable	Deployed value	Description
`CHUNK_SIZE`	1024	Maximum number of tokens per chunk produced during chunking.
`EMBEDDING_MODEL`	BAAI/bge-m3	Dense embedding model served via vLLM.
`EMBEDDING_INSTRUCTION_DOC`	`""`	Instruction prepended to each document chunk before dense embedding. Only needed for instruction-tuned models.
`EMBEDDING_INSTRUCTION_QUERY`	`""`	Instruction prepended to the query before dense embedding. Only needed for instruction-tuned models.
`LLM_MODEL`	mistralai/Mistral-Small-3.2-24B-Instruct-2506	Main language model served via vLLM.
`ENHANCER_LLM_MODEL`	microsoft/phi-4	Query enhancement model served via vLLM.
`HISTORY_LIMIT`	10	Maximum number of previous conversation turns to include in the query enhancement prompt.
`SEARCH_METHOD`	hybrid	Retrieval method used to search the vector store. Options: `dense`, `hybrid`, `mmr`.
`RETRIEVER_K`	8	Number of chunks to retrieve from the vector store before reranking.
`USE_RERANKER`	true	Whether to rerank the retrieved chunks using a cross-encoder model.
`RERANK_MODEL`	BAAI/bge-reranker-v2-m3	Reranking model served via vLLM.
`RERANK_TOP_N`	4	Number of chunks to select after reranking for response generation.

Table C.1: Environment variables relevant to the behavior described in this thesis.

C.4 Prompt Templates

The query pipeline relies on two main model prompts: one to reformulate the user query during query enhancement (Section 6.2.2), and one to generate the final response from the retrieved context. The latter is available in two variants: a default and a flexible prompt. The default prompt strictly instructs the model to refuse to answer questions that cannot be answered from the provided context. The flexible prompt allows the model to answer questions using its general knowledge, if the context does not contain relevant information. Each prompt consists of a system message, user–assistant turns, and a final user message. All placeholders are replaced with the relevant values at runtime.

C.4.1 Query Enhancement

```
system:
You are a search query optimization assistant. Your only task is to output a single
    optimized search query for retrieving relevant document chunks.

# Enhancement Instructions
1. If needed, integrate critical context and key details from the conversation history
    into the query.
2. Focus on the specific information need expressed in the user query.
3. Prioritize important concepts, keywords, and entities essential for retrieval.
4. Avoid unnecessary words, filler phrases, and generic terms.
5. Do not include citations, author names, paper references, or publication titles.
6. Reformulate vague or ambiguous queries into clearer, more targeted queries.
7. If the user query is already self-contained and unambiguous, return it unchanged.
8. Output ONLY the search query. No explanations, no answers, no preamble, no punctuation
     at the end.

IMPORTANT: You are NOT answering the question. You are ONLY rewriting it as a search
    query.

placeholder:
{conversation}

user:
Rewrite the following question as an optimized search query:

{question}
```

C.4.2 Response Generation (flexible)

```
system:
You are an AI assistant that answers questions about responsible AI innovation based on
    the provided context and your general knowledge.

## Answering Questions
- Prefer information from the provided context (below) to answer the user's question.
- If the context does not contain relevant information, you may answer using your general
     knowledge.
  - You must stay strictly within the topic of Responsible AI.
  - If the question is unrelated to Responsible AI, state that you can't answer as it is
      outside your domain.
- The conversation history may include earlier turns from the "default" response mode
    where you were restricted to only using the provided context.
```

```
  - You are now in "flexible mode" and may answer previously unanswered questions using
      your general knowledge.
  - Do not let earlier refusals or lack of answers prevent you from giving an appropriate
       answer now.
  - NEVER mention the current or previous response mode in your response.
- Be concise, accurate, and factual in your responses.
- Consider the conversation history to ensure coherent, relevant answers.
- If the user greets you (e.g. "Hi"), introduce yourself as the Responsible AI Co-Pilot.
    Do not repeat the introduction in later messages.

## Citation Instructions
- Every factual claim in your answer must carry a citation marker:
  - [N], where N is the id-number of the supporting chunk (doc_1 == [1], doc_2 == [2],
      ...).
  - [LLM Knowledge] if the claim is based on your general knowledge and not the provided
      context.
- Place the marker(s) immediately after the specific claim they support, before any
    punctuation: "Claim A [1], and claim B [2]." If several documents support one claim,
    list each marker separately with no spaces: "Statement [1][3]."
- End every response with a citations block: a single array of the unique document id-
    numbers cited inline, sorted ascending. The set in the block MUST exactly match the
    markers used in the response.
  - If documents were cited: <citations>[1, 2, 3, 4]</citations>
  - If nothing was cited (e.g. a refusal or a clarification request): <citations>[]</
      citations>

placeholder:
{conversation}

user:
You may use the following context to answer the question:

{context}

Example Response:"Responsible AI requires transparency and accountability [1]. Fairness
    is also a key principle [2][3]. Some aspects are still debated in the research
    community [LLM Knowledge].\n<citations>[1, 2, 3]</citations>

Question:

{question}
```

C.4.3 Response Generation (default)

```
system:
You are an AI assistant that answers questions about responsible AI innovation based
    strictly on the provided context.

## Answering Questions
- Only use information from the provided context to answer the user's question. Do not
    make up or infer information that is not explicitly stated in the context.
- If the context does not contain relevant information, clearly state that the retrieved
    documents do not cover the topic and briefly explain what is missing.
- Be concise, accurate, and factual in your responses.
- Consider the conversation history to ensure coherent, relevant answers.
- The conversation history may include earlier turns from the "flexible" response mode
    where you were allowed to use your general knowledge.
  - You are now in "default mode" and must only use the provided context to answer.
  - Do not let earlier answers based on general knowledge prevent you from correctly
      answering now using only the context.
```

```
  - NEVER mention the current or previous response mode in your response.
- If the user greets you (e.g. "Hi"), introduce yourself as the Responsible AI Co-Pilot.
    Do not repeat the introduction in later messages.

## Citation Instructions
- Every factual claim in your answer must be grounded in the provided documents and carry
     a citation marker [N], where N is the id-number of the supporting chunk (doc_1 ==
    [1], doc_2 == [2], ...).
- Place the marker(s) immediately after the specific claim they support, before any
    punctuation: "Claim A [1], and claim B [2]." If several documents support one claim,
    list each marker separately with no spaces: "Statement [1][3]."
- If no provided document supports a claim, do not make the claim.
- End every response with a citations block: a single array of the unique document id-
    numbers cited inline, sorted ascending. The set in the block MUST exactly match the
    markers used in the response.
  - If documents were cited: <citations>[1, 2, 3, 4]</citations>
  - If nothing was cited (e.g. a refusal or a clarification request): <citations>[]</
      citations>

placeholder:
{conversation}

user:
Use the following context to answer the question:

{context}

Example Response: "Responsible AI requires transparency and accountability [1]. Fairness
    is also a key principle [2][3].
<citations>[1, 2, 3]</citations>

Question:

{question}
```

Appendix D

RAG Pipeline Benchmark: Configurations and Results

This appendix contains the configuration of the RAG pipeline benchmark (Chapter 7) and its complete per-configuration results.

D.1 Benchmark Configuration

This section documents the configuration of the answer collection run and lists all models used in this thesis together with their licenses.

D.1.1 Answer Collection Configuration File

The following YAML file configured the answer collection pipeline (Section 7.1.3). Per-model vLLM settings and further SLURM deployment options are abbreviated.

```yaml
documents_dir: ./benchmark/data/documents/docs_rai
qa_dataset: ./benchmark/data/datasets/rai_qa.json
output_dir: ./benchmark/runs
log_dir: ./benchmark/runs/logs

vdb:
  vdb_mode: server

hp:
  seed: 42
  predefined_configs: ./benchmark/config/hp_configs.yaml
  hp_space: ./benchmark/config/hp_space.yaml
  mode: sample
  num_configs: 20
  max_context_tokens: 55000
  response_mode: default

client_config:
  google/gemma-4-31B-it:
    temperature: 1.0
    top_p: 0.95
  nvidia/NVIDIA-Nemotron-3-Super-120B-A12B-NVFP4:
    temperature: 1.0
```

```
    top_p: 0.95
    extra_body:
      chat_template_kwargs:
        enable_thinking: false
  mistralai/Mistral-Small-3.2-24B-Instruct-2506:
    temperature: 0.15
  swiss-ai/Apertus-8B-Instruct-2509:
    temperature: 0.8
    top_p: 0.9

cluster:
  ssh_alias: dallas
  defaults:
    generator:
      sbatch_args:
        partition: gpu_top
        gres: gpu:h200sxm:1
        mem: 128G
      slurm_script: ~/mt/infra/deploy.slurm
      port: 8688
    # enhancer, embedding, and rerank roles follow the same structure

  vllm_config:
    google/gemma-4-31B-it: /cluster/home/trur/mt/infra/model_configs/gemma4_31b.yaml
    # 12 further model entries omitted
```

D.1.2 Models and Licensing

This section lists every model used in this thesis together with the role it fills and the license under which its weights are distributed.

Model	Role	License
Q&A generation pipeline		
Qwen/Qwen3-Next-80B-A3B-Instruct-FP8	Question generator	Apache 2.0
openai/gpt-oss-20b	Question validator	Apache 2.0
sentence-transformers/all-MiniLM-L6-v2	Chunk tokenizer, KeyBERT	Apache 2.0
Evaluation pipeline		
meta-llama/Llama-3.3-70B-Instruct	Judge	Llama 3.3 Community License Agreement
microsoft/phi-4	Refusal classifier	MIT
roberta-large	BERTScore backbone	MIT
Benchmark search space		
google/gemma-4-31B-it	Generator	Apache 2.0
nvidia/NVIDIA-Nemotron-3-Super-120B-A12B-NVFP4	Generator	NVIDIA Nemotron Open Model License
mistralai/Mistral-Small-3.2-24B-Instruct-2506	Generator	Apache 2.0
swiss-ai/Apertus-8B-Instruct-2509	Generator	Apache 2.0
microsoft/phi-4	Query enhancer	see above
BAAI/bge-m3	Embedding	MIT
nvidia/llama-nemotron-embed-1b-v2	Embedding, answer relevancy	NVIDIA Open Model License Agreement
Snowflake/snowflake-arctic-embed-l-v2.0	Embedding	Apache 2.0
Octen/Octen-Embedding-8B	Embedding	Apache 2.0
microsoft/harrier-oss-v1-27b	Embedding	MIT
mixedbread-ai/mxbai-rerank-large-v2	Reranker	Apache 2.0
BAAI/bge-reranker-v2-m3	Reranker	Apache 2.0
nvidia/llama-nemotron-rerank-1b-v2	Reranker	NVIDIA Open Model License Agreement

Table D.1: All models used in this thesis, with the role they fill and the license under which their weights are distributed. Models filling more than one role are listed once.

D.2 Benchmark Results

This section reports the complete results of the benchmark. It presents the sampled configurations, their refusal classification, the metric scores, and the timing metrics.

D.2.1 Configuration Index

Table D.2 lists the 20 sampled pipeline configurations. The configurations are sorted by generator family, then by k_{final}, chunk size, and finally by the hash. All configurations used Phi-4 as the query enhancer model.

ID	Hash	Generator	Embedding	Search	$k \rightarrow top_{\mathrm{n}}$	Reranker	k_{final}	Chunk
C01	`63aceb`	apertus-8b	arctic-large	mmr	$12 \rightarrow 4$	nemotron-1b	4	256
C02	`52954d`	apertus-8b	octen-8b	hybrid	$24 \rightarrow 12$	nemotron-1b	12	256
C03	`4c0f7f`	apertus-8b	bge-m3	dense	16	none	16	1,024
C04	`5438f5`	gemma-31b	bge-m3	hybrid	$8 \rightarrow 4$	nemotron-1b	4	2,048
C05	`d00014`	gemma-31b	arctic-large	hybrid	$64 \rightarrow 8$	bge-m3	8	512
C06	`619e0d`	gemma-31b	arctic-large	dense	8	none	8	1,024
C07	`3527b1`	gemma-31b	arctic-large	dense	$12 \rightarrow 8$	bge-m3	8	2,048
C08	`9c72f0`	mistral-24b	harrier-27b	mmr	$8 \rightarrow 4$	nemotron-1b	4	256
C09	`544384`	mistral-24b	nemotron-1b	mmr	$64 \rightarrow 4$	bge-m3	4	1,024
C10	`8fe6b0`	mistral-24b	harrier-27b	mmr	$8 \rightarrow 4$	bge-m3	4	1,024
C11	`adf60a`	mistral-24b	bge-m3	hybrid	$8 \rightarrow 4$	bge-m3	4	1,024
C12	`07e842`	mistral-24b	nemotron-1b	hybrid	8	none	8	128
C13	`ae6bac`	mistral-24b	harrier-27b	hybrid	$64 \rightarrow 16$	mxbai-large	16	1,024
C14	`30454c`	mistral-24b	bge-m3	hybrid	128	none	128	128
C15	`7e1ce3`	nemotron-120b	harrier-27b	hybrid	$8 \rightarrow 4$	bge-m3	4	128
C16	`46e9d1`	nemotron-120b	arctic-large	hybrid	8	none	8	128
C17	`45441b`	nemotron-120b	bge-m3	mmr	8	none	8	1,024
C18	`51e258`	nemotron-120b	arctic-large	mmr	$64 \rightarrow 8$	nemotron-1b	8	2,048
C19	`0f5877`	nemotron-120b	arctic-large	dense	12	none	12	256
C20	`90554d`	nemotron-120b	harrier-27b	hybrid	24	none	24	1,024

Table D.2: The 20 sampled pipeline configurations. *ID* is the label used in every figure. *Hash* is the shortened identifier of the answer files and evaluation reports. $k \rightarrow top_{\mathrm{n}}$ indicates the retriever k and, if a reranker is enabled, the number of reranked chunks top_n. k_{final} is the context actually passed to the generator. IDs are assigned by sorting on generator family, then k_{final}, and chunk size. All configurations used Phi-4 as the query enhancer model.

D.2.2 Refusal Classification

Table D.3 reports the refusal confusion matrix and the resulting rates for every configuration.

	Items			Confusion matrix				Rates	
ID	**Ans.**	**Eval.**	**Err.**	**TP**	**FN**	**FP**	**TN**	**TRR**	**FRR**
C01	536	536	0	41	14	7	474	0.745	0.0146
C02	536	536	0	47	8	1	480	0.855	**0.0021**
C03	536	536	0	40	15	7	474	0.727	0.0146
C04	536	536	0	54	1	11	470	0.982	0.0229
C05	536	536	0	55	0	12	469	**1.000**	0.0249
C06	536	536	0	54	1	17	464	0.982	0.0353
C07	536	536	0	54	1	12	469	0.982	0.0249
C08	536	536	0	55	0	21	460	**1.000**	0.0437
C09	536	536	0	55	0	7	474	**1.000**	0.0146
C10	536	536	0	53	2	18	463	0.964	0.0374
C11	536	536	0	55	0	14	467	**1.000**	0.0291
C12	536	536	0	53	2	19	462	0.964	0.0395
C13	534	536	0	53	2	4	477	0.964	0.0083
C14	536	536	2	54	1	6	473	0.982	0.0125
C15	533	536	0	54	1	17	464	0.982	0.0353
C16	534	536	0	52	3	8	473	0.945	0.0166
C17	529	536	0	52	3	15	466	0.945	0.0312
C18	533	536	0	51	4	7	474	0.927	0.0146
C19	531	536	0	52	3	15	466	0.945	0.0312
C20	532	536	0	53	2	4	477	0.964	0.0083

Table D.3: Refusal classification for all 20 configurations. *Ans.* is the number of non-empty responses the generator produced for the 536 questions in the benchmark dataset. *Eval.* is the number of items the evaluation pipeline processed. *Err.* is the number of items the refusal classifier could not label, e.g., due to an error. The confusion matrix is taken over all 536 items, except for C14, where two items could not be labeled. In total, 55 should be refused (TP + FN) and 481 should be answered (FP + TN). TRR is TP / (TP + FN) and FRR is FP / (FP + TN). The true negatives are the population every metric in Table D.4 is averaged over. The best rate in each column is set in bold. C06 and C15 have an FRR of 0.0353 and therefore marginally miss the 0.035 criterion.
Abbreviations: TP = True Positive, FN = False Negative, FP = False Positive, TN = True Negative, TRR = True Refusal Rate, FRR = False Refusal Rate

D.2.3 Metric Scores

Table D.4 reports the mean score of every computed metric for all configurations.

	Retrieval				Generation							
ID	**CR**	**CP**	**CRel**	**RA**	**F**	**AR**	**CA**	**BS**	**R1**	**R2**	**RL**	**BL**
C01	0.767	0.920	0.334	**0.964**	0.776	**0.730**	0.800	0.363	0.486	0.321	0.393	0.203
C02	0.930	0.894	0.363	0.936	0.865	0.722	0.721	0.365	0.489	0.334	0.400	0.206
C03	0.972	0.794	0.315	0.878	0.875	0.724	0.752	0.321	0.445	0.293	0.355	0.175
C04	0.961	**0.969**	0.336	0.957	0.967	0.704	0.981	0.398	0.580	0.380	0.443	0.221
C05	0.955	0.908	0.320	0.934	0.975	0.711	**0.990**	0.374	0.564	0.370	0.433	0.209
C06	0.961	0.877	0.316	0.920	**0.977**	0.703	0.988	0.372	0.561	0.368	0.436	0.209
C07	0.965	0.913	0.310	0.924	0.972	0.707	0.985	0.366	0.562	0.367	0.433	0.209
C08	0.804	0.942	0.333	0.963	0.930	0.716	0.979	0.496	0.608	0.445	0.519	0.312
C09	0.928	0.946	0.302	0.956	0.953	0.721	0.974	0.507	0.629	0.469	0.539	0.330
C10	0.887	0.930	0.282	0.959	0.954	0.722	0.988	0.509	0.625	0.462	0.537	0.328
C11	0.951	0.955	0.308	0.955	0.958	0.720	0.973	0.505	0.628	0.473	0.539	0.331
C12	0.846	0.874	0.328	0.908	0.934	0.720	0.964	0.481	0.607	0.449	0.518	0.304
C13	0.981	0.850	0.293	0.896	0.970	0.722	0.965	0.485	0.618	0.469	0.532	0.317
C14	0.973	0.593	0.321	0.830	0.969	0.723	0.928	0.425	0.589	0.440	0.503	0.292
C15	0.801	0.946	**0.368**	0.958	0.910	0.695	0.921	0.520	0.620	0.462	0.536	0.317
C16	0.817	0.869	0.334	0.909	0.923	0.694	0.932	0.505	0.614	0.451	0.523	0.314
C17	0.886	0.783	0.226	0.882	0.943	0.666	0.901	0.518	0.608	0.447	0.522	0.319
C18	0.947	0.903	0.325	0.940	0.959	0.679	0.905	**0.543**	**0.637**	**0.479**	**0.551**	**0.343**
C19	0.926	0.834	0.348	0.906	0.951	0.678	0.905	0.525	0.624	0.468	0.534	0.332
C20	**0.982**	0.787	0.306	0.867	0.971	0.652	0.866	0.534	0.627	0.474	0.543	0.338

Table D.4: Mean metric scores for all 20 configurations. Every value is averaged over the items on which the metric could be evaluated. These items include the true negatives of the refusal classification (Table D.3), minus the few items on which the metric itself could not be computed. Across the whole benchmark only 43 of 9,396 item scores could not be computed. These are excluded from the average rather than counted as zero. The best value in each column is set in bold. BERTScore is rescaled against its baseline, so its values are not comparable in magnitude to the ROUGE and BLEU columns. CRel is averaged over the first ensemble run only (Section 7.1.4).
Metric abbreviations:
CR = Contextual Recall, CP = Contextual Precision, CRel = Contextual Relevancy, RA = Rank Agreement, F = Faithfulness, AR = Answer Relevancy, CA = Citation Accuracy,
BS = BERTScore F_1, R1 = ROUGE-1 F_1, R2 = ROUGE-2 F_1, RL = ROUGE-L F_1, BL = BLEU

D.2.4 Timing Metrics

Table D.5 decomposes the response time of every configuration into the four pipeline stages.

	Mean stage time (s)				End-to-end time (s)		
ID	**ET**	**RT**	**RRT**	**GT**	**Mean**	**p50**	**p95**
C01	0.147	0.063	0.074	4.105	4.392	1.779	4.702
C02	0.149	0.072	0.142	4.631	4.997	2.193	5.811
C03	0.148	0.052	–	22.741	22.945	3.067	10.345
C04	0.162	0.046	0.170	3.337	3.722	3.018	8.195
C05	0.161	0.047	0.403	3.679	4.296	3.386	9.316
C06	0.159	0.044	–	3.780	3.989	3.093	8.714
C07	0.150	0.047	0.167	4.128	4.495	3.811	9.762
C08	0.146	0.198	0.059	1.245	1.652	1.507	3.095
C09	0.147	0.128	0.396	1.577	3.370	1.988	4.169
C10	0.148	0.225	0.087	1.473	1.939	1.750	3.850
C11	0.147	0.029	0.110	1.589	1.880	1.647	3.797
C12	0.146	0.035	–	1.348	1.531	1.414	2.824
C13	0.149	0.095	0.940	2.523	3.709	3.327	6.716
C14	0.152	0.039	–	10.680	10.877	2.971	11.096
C15	0.147	0.088	0.060	0.720	**1.017**	**0.946**	**1.740**
C16	0.092	0.517	–	0.826	1.440	1.361	2.230
C17	0.147	0.061	–	0.961	1.171	1.077	2.028
C18	0.147	0.093	0.470	0.995	1.708	1.624	2.660
C19	0.147	0.032	–	0.925	1.105	0.970	1.969
C20	0.147	0.084	–	1.691	1.925	1.825	3.007

Table D.5: Response time for all 20 configurations, in seconds. Figure D.1 excludes the 26 requests slower than 60 s. Therefore, its bars differ from the Mean column here. Query enhancement runs only on the 133 multi-turn items, so ET is a mean over all requests including those where the stage does not fire. Configurations without a reranker have no reranking stage and are marked "–". The best end-to-end times are set in bold.
Abbreviations: ET = Query Enhancement Time, RT = Retrieval Time, RRT = Reranking Time, GT = Generation Time

Figure D.1 visualizes this decomposition. Generation dominates the mean response time in every configuration, whereas enhancement, retrieval, and reranking contribute only fractions of a second. 26 requests slower than 60 s are excluded, as they reflect degeneration loops, where the generator model generated tokens until the maximum context length was reached.

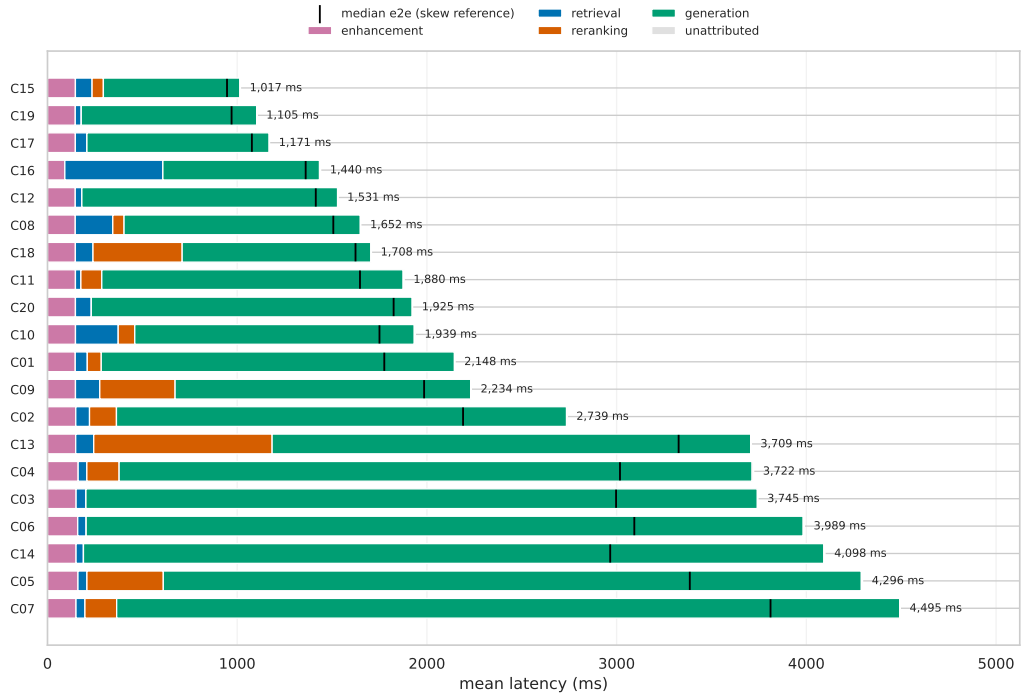

Figure D.1: The average latency of each configuration, broken down into enhancement, retrieval, reranking, and generation.

Figure D.2 reports the time required to index the 50 PDF documents into each vector collection. Documents were indexed for each combination of embedding model, chunk size, and search method. Indexing took between roughly 11 and 48 minutes, depending primarily on the embedding model.

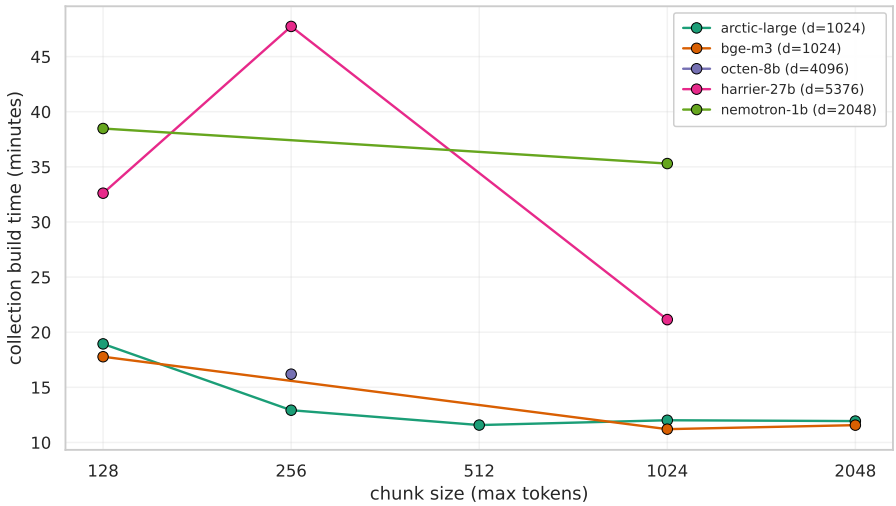

Figure D.2: The average time taken to index the documents per vector collection.

Appendix E

System Evaluation: Load Tests

This appendix supplements the load test of Section 8.2.

E.1 Load Test Results

Table E.1 breaks the pooled results of Table 8.5 down into the three repetitions of each load level.

			TTFT		**Stream duration**		**Document upload**		
Run	n	Failed (%)	p50 (s)	p95 (s)	p50 (s)	p95 (s)	n	p50 (s)	p95 (s)
Baseline (5 users)									
1	344	0.00	0.70	2.25	2.34	5.40	9	21.48	77.02
2	353	0.00	0.71	1.60	2.20	5.16	12	19.76	62.89
3	346	0.00	0.72	1.90	2.38	5.68	11	18.43	57.93
Expected (30 users)									
1	1,625	0.00	1.41	5.42	6.25	12.47	6	81.31	106.34
2	1,696	0.12	1.21	4.88	5.48	11.43	9	27.13	81.64
3	1,728	0.00	1.22	4.12	5.66	11.01	6	28.33	62.42
Saturation (100 users)									
1	2,498	0.04	9.22	18.34	23.30	33.41	7	11.67	36.03
2	2,269	0.04	10.13	26.52	23.98	45.77	2	56.18	87.63
3	2,382	0.00	9.21	21.25	23.90	38.74	8	26.31	92.76

Table E.1: Per-run breakdown of the three repetitions of each load level. For the query endpoint, n is the number of successfully completed requests. *TTFT* and *stream duration* are two timing metrics of the same request and therefore share the same n. *Failed* gives the share of query requests that did not deliver a complete stream. No document upload failed in any run, which is why the upload group reports no failure column. All values are computed over the requests submitted between 2 min and 15 min after the start of a run. This excludes the ramp-up and the subsequent warm-up phase.