

XRAG: eXamining the Core - Benchmarking Foundational Components in Advanced Retrieval-Augmented Generation

Qianren Mao¹, Yangyifei Luo^{2*}, Jinlong Zhang^{2*}, Hanwen Hao²,
Zhilong Cao², Xiaolong Wang², Xiao Guan², Zhenting Huang², Weifeng Jiang³, Shuyu Guo²,
Zhentao Han², Qili Zhang², Siyuan Tao², Yujie Liu², Junnan Liu², Zhixing Tan¹,
Jie Sun¹, Bo Li^{1,2†}, Xudong Liu^{1,2}, Richong Zhang^{1,2}, Jianxin Li^{1,2†}
¹Zhongguancun Laboratory, ²Beihang University, ³Nanyang Technological University
maoqr@zgclab.edu.cn

🔗 <https://github.com/DocAILab/XRAG>

Abstract

Retrieval-augmented generation (RAG) synergizes the retrieval of pertinent data with the generative capabilities of Large Language Models (LLMs), ensuring that the generated output is not only contextually relevant but also accurate and current. We introduce XRAG, an open-source, modular codebase that facilitates exhaustive evaluation of the performance of foundational components of advanced RAG modules. These components are systematically categorized into four core phases: pre-retrieval, retrieval, post-retrieval, and generation. We systematically analyse them across reconfigured datasets, providing a comprehensive benchmark for their effectiveness. As the complexity of RAG systems continues to escalate, we underscore the critical need to identify potential failure points in RAG systems. We formulate a suite of experimental methodologies and diagnostic testing protocols to dissect the failure points inherent in RAG engineering. Subsequently, we proffer bespoke solutions aimed at bolstering the overall performance of these modules. Our work thoroughly evaluates the performance of advanced core components in RAG systems, providing insights into optimizations for prevalent failure points.

1 Introduction

Retrieval-Augmented Generation (RAG) [1, 4, 13, 19] represents a pivotal strategy in Q&A tasks, demonstrating enhanced performance by delivering more informative and accurate answers compared to relying solely on large language models (LLMs). The efficacy of basic RAG systems is contingent upon the seamless operation of four core components: pre-retrieval, retrieval, post-retrieval, and generation. The pre-retrieval stage indexes the corpus and reforms queries for efficient retrieval. The retrieval stage focuses on identifying and extracting documents relevant to a given query. The post-retrieval stage refines, summarizes, or compacts information to ensure contextual clarity. Finally, the generation stage employs the LLM to produce responses. These sequential stages critically influence output quality, highlighting the RAG framework’s interdependence. Advanced RAG modules (e.g. reranker, refiner) offer sophisticated algorithms for tailored search solutions, surpassing standardized methodologies.

*Equal Contribution. Jinlong Zhang did this work during his internship at Zhongguancun Laboratory.

†Corresponding Author.

Toolkits like LangChain [6] and LlamaIndex [25], modularize the RAG process, increasing adaptability and broadening its applications. However, they are typically cumbersome, making adaptation to new data challenging and validating or optimising innovative methods inconvenient. Although ongoing efforts like FastRAG [17], RALLE [15], LocalRQA [39], AutoRAG [23], FlashRAG [20], and RAGLAB [40], address these challenges through modular RAG processes (e.g., retrieval engines and generative agents), an implicit gap persists in the comparative performance evaluation of these advanced RAG modules within the overall RAG workflow. Comprehensive assessments of these modules are notably absent, making it challenging for researchers to evaluate their approaches in consistent experimental conditions. [12].

To address the abovementioned issues, we focus on the core components of advanced RAG modules and conduct comprehensive experiments across four aspects: pre-retrieval, retrieval, post-retrieval, and generation. We introduce XRAG, an open-source, modular codebase designed to comprehensively evaluate foundational components of advanced RAG modules. The key capabilities of XRAG are summarized as follows:

Modular RAG Process: Fine-Grained Comparative Analysis. Extensive experiments are conducted on the advanced RAG modules across four stages: pre-retrieval, retrieval, post-retrieval, and generation. The core components analysis covers 3 query rewriting strategies, six retrieval units, three post-processing techniques, and LLM generators from 3 different vendors — OpenAI, Meta, and Google. This categorization provides an in-depth understanding of the capabilities of RAG components.

Unified Benchmark Datasets: Dual Assessment of Retrieval and Generation. To enhance the uniformity and reusability of datasets in RAG research, XRAG compiles and formats three prevalent benchmark datasets, preprocessing them into a unified format. This standardization enables concurrent assessment of both retrieval and generation capabilities, streamlining comparative evaluations across diverse RAG systems.

Comprehensive Testing Methodologies: Multidimensional Evaluation Framework. To overcome the absence of a holistic evaluation system for RAG components, XRAG introduces an evaluation benchmark encompassing three perspectives. It comprises Conventional Retrieval Evaluation for retrieval-unit matching, Conventional Generation Evaluation for generation tests based on generative-token matching, and Cognitive LLM Evaluation for generation tests based on semantic understanding. XRAG ensures a standardized and thorough evaluation of retrieval and generation.

Identification and Mitigation of RAG Failure Points: Systematic Analysis and Improvement. Recognizing the lack of systematic experiments and improvement methods addressing RAG failure points, XRAG develops a set of evaluation methods to pinpoint and rectify specific issues. Targeted enhancement strategies are proposed and employed to verify the resolution of identified problems. Analyzing failure points and implementing feasible optimization and validation solutions can bolster the optimization of RAG components.

2 Related Works

Existing Retrieval-Augmented Generation (RAG) toolkits, such as LangChain [6] and LlamaIndex [25], enable rapid RAG system construction using pre-built models. These toolkits enhance flexibility and expand potential applications by modularising the RAG process. However, excessive encapsulation limits their transparency and usability.

Although ongoing efforts, such as FastRAG [17], RALLE [15], LocalRQA [39], AutoRAG [23], FlashRAG [20], and RAGLAB [40], address these issues by featuring and implementing modular RAG process, such as retrieval engines and generative agents. There remains an implicit gap in the comparative performance evaluation of these advanced RAG modules within the overall RAG workflow. FastRAG [17] and RALLE [15] allow users to assemble RAG systems with core components, fostering a more adaptable RAG implementation. AutoRAG [23] further supports users by identifying optimal RAG pipelines for custom data, facilitating bespoke RAG systems. LocalRQA [39] and RAGLAB [40] focus on RAG training, offering scripts for various component training. Nevertheless, FastRAG, RALLE, AutoRAG, and LocalRQA require users to reproduce published algorithms independently and offer limited component options, restricting the flexibility of RAG systems despite modular designs. FlashRAG [20] and RAGLAB [40] advance algorithmic

Table 1: Comparation of RAG Libraries. Modular Design (Mod.Dsgn) indicates toolkit modularity. Fair Comparison [40] (Fair.Comp) indicates evaluation by aligning key components like seeds, generators, retrievers, and instructions. Unified Datasets (Unif.Data) ensures unified dataset formats for retrieval and generation. Modular Evaluation (Mod.Eva) assesses RAG modular differences. Failure Management (Fail.Mgmt) systematically implements strategies for identifying and mitigating RAG failure points. ConR uses token-matching for evaluating retrieval, ConG uses token-matching for evaluating generation, and CogL is based on LLM-based instructions for retrieval and generation evaluation. ‘u’ refers to No. Of unified metrics.

Library	Mod.Dsgn	Fair.Comp	Unif.Data	Mod.Eva	Fail.Mgmt	ConR	ConG	CogL
LangChain [6]	✓	✗	✗	✗	✗	0	0	0
LlamaIndex [25]	✓	✗	✗	✓	✗	6	0	7
FastRAG [17]	✓	✗	✗	✗	✗	0	0	0
RALLE [15]	✓	✗	✗	✗	✗	0	0	0
LocalRQA [39]	✗	✗	✓	✓	✗	3	2	1
AutoRAG [23]	✓	✗	✓	✓	✗	6	5	4
FlashRAG [20]	✓	✗	✓	✓	✗	4	5	0
RAGLAB [40]	✓	✓	✓	✗	✗	0	4	0
XRAG (ours)	✓	✓	✓	✓	✓	8^u	9^u	33^u

reproducibility in RAG systems by integrating numerous algorithms into a unified framework. This supports efficient replication of existing methods and promotes innovation in algorithm development. However, FlashRAG [20] lacks uniformity in fundamental evaluation components, such as random seeds, generators, retrievers, and instructions, which hinders result comparability. RAGLAB [40], despite its offering a fair experimental setup, lacks comprehensive evaluation strategies for assessing individual RAG component performance. Similarly, FlashRAG [20] exhibits the same lack of modularity analysis, restricting to derive meaningful conclusions.

Neither toolkit addresses RAG system failure points, such as missing contextual knowledge or top-ranking confusion. The absence of systematic frameworks for identifying and mitigating failures limits algorithm resilience and undermines user trust, particularly in critical applications.

3 XRAG

Figure 1 delineates the integrated modules and schematic structure of the XRAG framework. The framework is stratified into datasets and corpus, advanced components, and evaluators, integrated through XRAG’s board and config hook (illustrated in the Appendix A.6.1). The overall structure progresses from foundational to application-oriented components.

Designed with a modular architecture, XRAG enables users to accomplish: preparation of normalized RAG datasets (Section 3.2), assembly of the RAG components (Section 3.1), evaluation of RAG system’s core components (Section 3.3), and diagnosing and optimizing of RAG system failures (Section 2 & Appendix A.2).

3.1 Advanced Component Modules

♣ **Pre-retrieval** Before retrieval, the pre-retrieval components leverage LLMs to refine user queries, enhancing the quality and relevance of the information retrieval process. Key methodologies include Step-back Prompting (SBPT [41]): Broaden’s queries to enrich contextual grounding for answers, enhancing the contextual foundation for answer generation. Hypothetical Document Embedding (HyDE [11]): Transmutes the original query into a form that better aligns with the indexed documents, improving retrieval alignment and efficacy. Chain-of-Verification (CoVe [9]): Executes a verification plan for further refining system responses into an enhanced one. Both HyDE and CoVe mandate pre-retrieval instructions, which involve generating hypothetical documents and responses consultation using query first, affirming their role as core components of the pre-retrieval process.

♣ **Retriever** For advanced retrieval strategies, we integrate the LlamaIndex to facilitate standard advanced methods. LexicalBM25 retriever ranks documents based on query term occurrence and rarity across the corpus. Simple Fusion Retriever (SQFusion) augments the query by generating related sub-queries and returns top-k nodes across all queries and indexes. Reciprocal Rerank Fusion

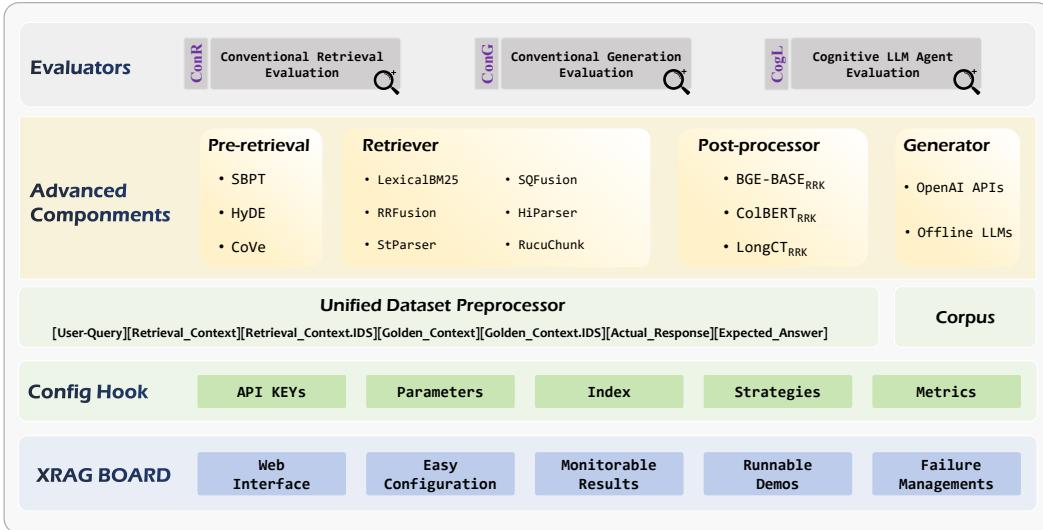


Figure 1: Schematic overview of the Xrag framework.

Retriever (RRFusion [8]) fuse indexes with a BM25-based retriever, capturing both semantic relations and keyword relevance. Both retrievers assign scores, enabling reciprocal reranking for node sorting without additional models or excessive computation.

For dense retrieval, Auto Merging Hierarchy Retriever (HiParser) recursively merges subsets of leaf textual chunk nodes linked to a parent node, constructing a ‘coarse-to-fine’ hierarchy of nodes. SentenceWindow Retriever (StParser) parses documents into single sentences per node, incorporating surrounding sentences for added context. RecursiveChunk Retriever (RecuChunk) traverses node relationships to fetch nodes based on references. Lastly, TreeSelectLeaf retriever (TreeLeaf) uses the embedding similarity between the query and chunked text to traverse the index graph, optimizing node selection.

▼ **Post-processor** Postprocessor strategies aim to transform and filter nodes before returning them, enhancing retrieval accuracy and efficiency. Xrag incorporates rerankers to enhance relevance evaluation by leveraging contextual understanding models instead of embedding matching models. We leverage Huggingface transformers to integrate the (BGE-BASE_{RRK}) reranker which directly outputs similarity scores by processing questions and documents through a Cross-Encoder model. ColBERT reranker [22, 35] (ColBERT_{RRK}) employs multi-vector representations for granular query-document matching. Additionally, LongContextReorder [26] (LongCT_{RRK}) postprocessor repositions high-scoring nodes at both the top and bottom of the list, expediting the identification of relevant information.

▲ **Generator** Xrag framework integrates various LLM generators, including those from HuggingFace Transformers APIs, ensuring compatibility with open-source LLMs. Anticipating the need for users to localize and adapt open-source models within private RAG algorithms, we have developed a system that supports deploying localized models on GPU and CPU. In addition to open-source models, the generator module of Xrag includes support for closed-source LLM APIs, including those from Meta and Gemini, enabling users to access diverse capabilities while retaining the option to use proprietary models.

3.2 Unified Benchmark Datasets & Corpus

We collect and preprocess three benchmark datasets for the Xrag framework, emphasising rigorous experimental validation of RAG systems. We develop a unified dataset structured to facilitate performance testing for both retrieval and generation modules, incorporating standardized formats:

User-Query Retrieval-Context Retrieval-Context.IDS Golden-Context Golden-Context.IDS Actual-Response Expected-Answer.
--

XRAG provides a unified architecture for retrieval and question-answering datasets, specifically HoppotQA [38], DropQA [10], and NaturalQA [24]. The corpus used for indexing derives from the metadata of these datasets’ training, validation, and test sets. This methodology aligns with previous works [36] and supports the efficient deployment of vector databases. Table 2 shows that the HotpotQA corpus contains the highest document count, followed by NaturalQA, indicating that retrieval difficulty escalates with the number of documents required per query. Moreover, these datasets address complex RAG question-answering scenarios, including Multi-hop Questions, Constrained Questions, Numerical Reasoning, and Logical Reasoning tasks. Multi-hop questions depend on iteratively retrieved documents and their interrelations to deduce the answer. Constrained Q&A generates each answer alongside a corresponding constraint or condition rather than providing a standalone response. Numerical reasoning involves performing arithmetic operations such as addition, subtraction, sorting, and counting. Set-logical reasoning addresses more complex logical problems involving relationships among retrieved textual samples.

We also retain metadata to enable users to customize their dataset construction strategies, addressing training or fine-tuning requirements for retrieval and generative models in future RAG research (not considered in our work yet). Additionally, we provide filtering tools that allow users to refine the dataset, enabling random selection or fixed data testing of samples for evaluation. This reduces resource utilization and token costs, especially when using LLM APIs.

Table 2: Summary of Benchmark Datasets & Corpus.

Dataset	Size			Corpus		Multi-hop	Constrained	Numerical	Set-logical
	Train	Validation	Test	Documents	Source				
HotpotQA	86,830	8,680	968	508,826	wikipedia	✓	✗	✓	✗
DropQA	78,241	7,824	870	6,147	wikipedia	✓	✗	✓	✓
NaturalQA	100,093	10,010	1,112	49,815	wikipedia	✓	✓	✓	✗

3.3 Evaluation Methods

XRAG supports a variety of evaluation metrics to assess the quality of the RAG systems. We integrate the Jury [5], a comprehensive package for the evaluation of NLG systems, with RAG community evaluation tools such as UpTrain³ and DeepEval⁴. Our metrics are pivotal in determining the effectiveness of both the retrieval and generation components. They are categorized into three groups: Conventional Retrieval Evaluation and Conventional Generation Evaluation, along with an additional category for Cognitive LLM Evaluation.

Conventional Retrieval Evaluation (ConR). It supports six primary metrics: F1, Exact Match (EM), Mean Reciprocal Rank (MRR), and Mean Average Precision (MAP), along with Hit@1 and Hit@5. Additionally, it includes the DCG family of metrics, which assesses the effectiveness of ranking models by evaluating the quality of ordered results (`retrieval-context.ids`). This family comprises Discounted Cumulative Gain (DCG), Normalized Discounted Cumulative Gain (NDCG), and Ideal Discounted Cumulative Gain (IDCG). IDCG is the maximum DCG that can be obtained if the results are ideally ranked – arranged in descending order of their relevance (`golden-context.ids`).

Conventional Generation Evaluation (ConG). These generative-token matching metrics can be classified into three broad categories. N-gram similarity metrics: ChrF [31], ChrF++ [32], METEOR [2], ROUGE F1 [34] (ROUGE-1, ROUGE-2, ROUGE-L) focus on overlap in n-grams between generation (`actual-response`) and reference (`expected-answer`). Divergence-based metrics: MAUVE [30], Perplexity [18] measure content quality, diversity, and model learning by comparing the distribution between the generation and reference. Error-based accuracy metrics: Word Error Rate (WER) [29], Character Error Rate (CER) [29] and Exact Match (EM), assess the accuracy of `actual-response` by calculating the differences or errors when compared with the `expected-answer`.

Cognitive LLM Evaluation (CogL). Based on their evaluation focus, they can be classified into three main categories: Retrieval, Generation, and Combined Retrieval & Generation. Cognitive LLM Evaluation metrics, derived from UpTrain and DeepEval, are classified based on the parameters used by our framework. Metrics involving response-related parameters, such as `actual-response`

³<https://github.com/uptrain-ai/uptrain>

⁴<https://github.com/confident-ai/deepeval>

or `expected-answer`, are Generation Metrics. Conversely, metrics that lack response parameters include retrieval-related parameters, such as `retrieval-context` or `retrieval-context.ids`, are designated as Retrieval Metrics. The rest are Combined Metrics. Retrieval Metrics assess context quality and consist of Context Relevance (Up-CRel) and Context Conciseness (Up-CCns), which comes from UpTrain. Response Metrics include Response Relevance (Dp-ARel), Response Completeness (Up-RCmp) from DeepEval, Response Conciseness (Up-RCnc), Response Relevance (Up-RRel), and Response Validity (Up-RVal) and Response Matching (Up-RMch) from Uptrain. Combined Metrics evaluate the impact of retrieval on final responses and include Context Precision (Dp-CPre), Context Recall (Dp-CRec), Context Relevance (Dp-CRel), Response Consistency (Up-RCns), Context Utilization (Up-CUti), and Factual Accuracy (Up-FAcc), Faithfulness (Dp-Faith), Hallucination (Dp-Hall). Metrics prefixed with ‘Up’ originate from UpTrain, while those prefixed with ‘Dp’ are from DeepEval. Detailed usage patterns of CogL evaluation are illustrated in Appendix A.4. Moreover, we utilize **GPT-4 Turbo** as the LLM agent for cognitive evaluation.

The XRAM evaluator presents several significant advantages:

- **Evaluating with Multiple RAG Metrics in One Go:** The XRAM evaluator allows users to assess various RAG-specific metrics simultaneously. This capability streamlines the evaluation process, enabling comprehensive performance analysis without sequential evaluations.
- **Standardizes the Structure of Evaluation Metrics:** A unified data format simplifies comparing across different RAG components on both retrieval and generation.
- **Character and Semantic × Retrieval and Generation:** It encompasses 4-fold cross-dimensional analysis, including character-level matching tests and semantic-level understanding tests for both retrieval and generation.

3.4 Systematic Diagnostics of RAG Failures

RAG systems hold great potential for delivering accurate, context-aware responses but face reliability challenges [3, 7]. These challenges include the tendency of models to generate deceptive responses under uncertainty, improper ranking of retrieval results, incomplete answers, sensitivity to noise, and limitations in handling complex reasoning tasks, as shown in Figure 2. Understanding these issues is crucial for recognizing the current boundaries of RAG technology and identifying areas where further research and development are needed.

Negative Refusal: The challenge of negative refusal in RAG systems, where models tend to produce deceptive responses instead of acknowledging uncertainty, will severely erode user trust. The issue of negative refusal often stems from the model’s lack of awareness of its knowledge boundaries. When confronted with a query that lacks sufficient information, the model may generate a factually incorrect or misleading response rather than transparently admitting the absence of relevant knowledge.

Ranking Confusion: Existing studies have shown that LLMs are more attentive to the earlier parts of input sequences. This characteristic poses a significant issue for RAG systems: even if the retrieval module successfully locates the correct document segments, the system may still be affected if the most relevant segments do not appear early in the input sequence. In such cases, the generation module will initially encounter suboptimal or erroneous information, ultimately impacting the accuracy and quality of the RAG system’s output.

Answer Absence: The RAG system is designed to integrate information retrieved by the retrieval module with the inference capabilities of the generative model. However, a common challenge is that LLMs may overlook relevant details during the answer generation phase, even when all related contexts have been correctly retrieved. This issue often arises from the limitations in the reasoning capabilities of these models and problems associated with the format or order in which the context is presented. For instance, the existence of multiple methods for inputting context to LLMs in LlamaIndex can result in disparate response generation mechanisms, potentially impeding the RAG system’s capacity to access and utilize critical contextual information effectively.

Noise Impact: The robustness of RAG systems against noise is another critical aspect of their performance, especially when dealing with sources containing irrelevant or misleading information. Due to variations in the retrieval model’s accuracy, the way users formulate their queries, and stylistic differences, the document chunks retrieved often contain some degree of irrelevant content, which, if fed into the LLMs, can significantly affect its reasoning performance and the final response.

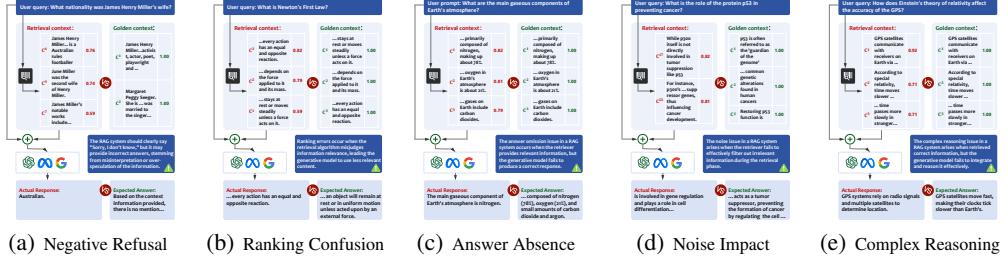


Figure 2: Typical RAG Failures.

Complex Reasoning: In practical tasks, handling complex reasoning scenarios that require integrating information from multiple documents is often necessary. In these scenarios, answering user queries depends on relevant clues scattered across different documents, necessitating cross-document information retrieval and integrated reasoning. However, RAG systems may struggle to fully grasp the complexity and diverse implicit information needs of such tasks, affecting the efficiency of the retrieval module and leading to the failure to identify all relevant cross-document segments. Coupled with the limitations in the reasoning capabilities of LLMs when faced with complex document inputs, this can ultimately impact the accuracy of the reasoning outcomes.

In-depth details of this investigation, including systematic analysis and improvement strategies, are presented in Appendix A.2.

4 Experimental Results and Discussion

4.1 Experimental Setup

Both the retriever and Q&A LLMs are essential modules of the RAG system. To focus on evaluating the Q&A capabilities of different LLMs, we fixed the retriever to the BGE-LARGE model version, as the retriever serves as the primary entry point influencing RAG performance. For document preprocessing, we utilized SentenceSplitter to divide documents into chunks and construct a vector index. SentenceSplitter was configured with a chunk size of 1024 tokens, representing the maximum sequence length and a chunk overlap of 20 tokens, denoting the overlap between consecutive chunks. We adhered to LlamaIndex configurations for other RAG components, including the refine module for response synthesis. To ensure compatibility and efficiency, XRAG integrates Huggingface Transformers. All Q&A LLMs were set with Temperature= 0 to ensure experimental consistency. For each query, five chunks were retrieved as contextual data. Consequently, the evaluation metrics measure retrieval accuracy based on five retrieval nodes, fully encompassing the assumptions of most datasets that typically consider only one or two golden contextual nodes. The metrics include HIT, DCG, NDCG, and IDCG, with a search depth $K = 5$. The generator model’s context window, encompassing the query, prompt, retrieved-context, and response content, is set to 4096 tokens.

We perform RAG testing on the entire test set in the two rule-based evaluations (Conventional Retrieval Evaluation and Conventional Generation Evaluation). For the Cognitive LLM Evaluation, token costs are substantial, as they stem from both the input and output tokens in the LLM’s reasoning and LLM’s testing processes. Therefore, we randomly sample the complete test set E_{no} times. The test result calculation is $\sum_{i=0}^{E_{no}} ((\sum_{j=0}^{P_{sc}} score)/P_{sc})/n$ and E_{no} is 3 in our experiments, and P_{sc} denotes the number of successful LLM API requests out of the E_{sp} samples. In the benchmark experiments evaluated by LLM retrieval and response metrics (Section 4.2), we set E_{sp} to 100. For the evaluation of RAG failures, we set E_{sp} to 20, considering the test quantity to be adequate, given that we have specifically curated datasets to investigate failures (Appendix A.2).

4.2 Benchmark Retrieval and Generation Evaluation

4.2.1 Retrieval Quality

The retrieval performance varies sensibly across the three datasets, with the poorest quality observed on DropQA in Table 3. Since DropQA requires advanced paragraph understanding and discrete

Table 3: Upstream RAG retrieval performance using basic retriever of BGE-Large. The conventional retrieval evaluation, denoted as ConR, is performed. The symbol \circ signifies metric values ranging from 0 to positive infinity, while all other values are within the $[0, 1]$ interval (%).

Dataset & Methods		ConR								
		F1	MRR	Hit@1	Hit@5	MAP	NDCG	DCG \circ	IDCG \circ	EM
HQA	BGE-Large	45.59	57.60	66.27	92.86	48.65	65.30	0.8996	1.2415	15.73
DQA	BGE-Large	17.57	21.15	32.33	32.33	21.15	24.27	0.2765	0.3648	12.45
NQA	BGE-Large	68.51	74.59	85.08	85.08	74.59	78.10	1.3861	1.7587	64.48

Table 4: Downstream RAG generation performance comparison of various LLMs, utilizing BGE-Large as the retrieval model. The conventional generation evaluation, denoted as ConG, is conducted. The top scores in each dataset are emphasized in bold. The symbol \circ signifies metric values ranging from 0 to positive infinity, while all other values are within the $[0, 1]$ interval (%). The symbol \downarrow denotes that lower metric values are preferable. The P_{sc} shows the average success rate of LLM API for generation across three 100-entry samples.

Methods		ConG									P_{sc}
		ChrF	ChrF++	METEOR	R1	R2	RL	EM	PPL \downarrow \circ	CER \downarrow \circ	
HQA	GPT-3.5 Turbo	15.49	18.54	16.35	27.71	14.44	21.57	14.66	133.73	0.8714	0.9570
	Gemini	15.36	18.47	15.91	25.57	14.29	21.00	14.14	129.25	0.8753	0.9692
	Llama3.1-8B	14.60	17.52	15.30	25.02	13.42	20.56	15.71	136.37	0.8689	0.9554
	GPT-4o Mini	15.52	18.65	16.37	25.68	14.81	21.21	17.80	131.28	0.8670	0.9540
DQA	GPT-3.5 Turbo	25.78	23.85	15.34	21.74	05.28	21.74	21.04	322.77	1.5747	1.1400
	Gemini	26.35	24.40	15.39	22.50	05.11	22.50	22.24	322.86	1.4352	1.0743
	Llama3.1-8B	26.03	24.06	15.24	21.62	05.02	21.62	21.44	314.41	1.4849	1.0905
	GPT-4o Mini	26.36	24.36	15.41	22.57	05.39	22.57	24.25	316.52	1.5361	1.1397
NQA	GPT-3.5 Turbo	42.29	41.12	30.55	42.57	16.75	42.55	53.93	310.54	1.9939	1.8732
	Gemini	39.70	38.52	28.03	39.79	15.32	39.79	52.36	306.21	2.2171	2.0444
	Llama3.1-8B	39.38	38.21	27.37	39.69	14.22	39.67	52.88	304.05	1.8704	1.7292
	GPT-4o Mini	45.52	43.34	30.42	45.78	19.67	45.78	56.02	315.95	1.4162	1.4043

reasoning [33], it presents a greater retrieval challenge. The NDCG metric reflects that the basic retrieval model performs reasonably well in relevance and ranking accuracy on HotpotQA and NaturalQA datasets. Ignoring ranking accuracy, a high Hit@5 score (>0.8) indicates a substantial likelihood of retrieving semantically relevant document blocks. The Hit@1 metric measures the ability to return the correct answer as the top-ranked result, making it suitable for tasks with a single retrieval target (e.g., the golden single passage in the NaturalQA dataset). However, it is less effective for scenarios requiring multiple equally prioritized context targets, as seen in the HotpotQA dataset.

Overall, the retrieval system performs well on HotpotQA and NaturalQA but faces challenges on DropQA, possibly due to the complexity of the answer. This suggests the need for further optimization of the prompt engineering algorithm to handle complex logical queries effectively.

4.2.2 Generation Quality

The RAG framework, as evaluated by the Cognitive LLM Evaluation in Table 4, Table 5 and Table 6, exhibits robust performance on NaturalQA dataset. From the perspective of the HotpotQA and DropQA datasets, there is clear potential for optimizing both the query understanding and reasoning capabilities of LLMs in answering, as indicated by the generative-token matching metrics in Table 4. However, an examination of three datasets in Table 5 and Table 6 reveals that even the basic RAG system, when evaluated with LLMs, shows high success rates for retrieval and response, reaching scores greater than 0.9 on many metrics.

The probability score P_{sc} in Table 4 indicates that LLM testing depends on API requests, which may incur packet loss due to unstable requests, long processing times, timeout limits, or server-side errors. This reflects a particular weakness of LLM testing, even though the average request success rate

Table 5: Upstream & Downstream performance of the RAG framework using Cognitive LLM Evaluation focusing on Separate Retrieval and Generation Metrics. The second row shows the average success rate of LLM API for evaluation across three 100-entry samples. All green-highlighted results representing the average across 3-times sampling tests.

Dataset & Methods		CogL [Retrieval]		CogL [Generation]					
		Up-CRel	Up-CCns	Dp-ARel	Dp-RCmp	Dp-RCnc	Dp-RRel	Dp-RVal	Dp-RMch
HQA	GPT-3.5 Turbo	0.7352	0.9201	0.9367	0.7667	0.9358	0.7856	0.9733	0.1854
		84.33	89.67	100	100	98.67	99.00	100	98.67
DQA	GPT-3.5 Turbo	0.6856	0.3604	0.9900	0.8417	0.9609	0.8441	0.9867	0.2060
		97.00	94.33	100	100	98.00	98.33	100	89.33
NQA	GPT-3.5 Turbo	0.8878	0.4133	0.9100	0.8061	0.8827	0.8214	0.9763	0.4574
		98.67	94.33	100	100	98.00	98.33	98.33	94.00

Table 6: Upstream & Downstream performance of the RAG framework using Cognitive LLM Evaluation focusing on Combined Retrieval and Generation Metrics.

Datasets & Methods		CogL [Combined Retrieval & Generation]							
		Dp-CPre	Dp-CRec	Dp-CRel	Up-RCns	Up-CUti	Up-FAcc	Dp-Fath	Dp-Hall
HQA	GPT-3.5 Turbo	0.9457	0.6455	0.6812	0.8318	0.6870	0.7516	0.9197	0.5537
		92.00	99.67	99.33	99.67	92.67	92.00	99.67	99.33
DQA	GPT-3.5 Turbo	0.8691	0.6254	0.8333	0.7450	0.6702	0.5326	0.7637	0.7333
		99.33	97.00	100	100	95.67	98.33	97.00	100
NQA	GPT-3.5 Turbo	0.9742	0.8769	0.9384	0.8948	0.6888	0.8485	0.9070	0.3417
		100	100	100	98.67	98.67	97.33	86.33	100

reaches over 85%. In Tables 4, rule-based metrics assess performance through ‘strict’ token-overlap criteria, whereas model-based evaluations in Table 5 and Table 6 offer a ‘soft’ token-match approach. Conventional rule-based metrics like EM and ROUGE focus on n-gram matching and fail to capture the overall accuracy of expression. Rule-based metrics are adequate for tasks requiring precise and unique answers. Cognitive LLM-based metrics assess generative expression quality, offering an advantage as evaluative understanding is complex to quantify with rule-based metrics.

5 Limitation

Our Xrag toolkit currently has several limitations we intend to address in future updates. i. Although we aim to incorporate core modules of the RAG system, time cost and compatibility constraints prevent comprehensive coverage of all RAG advancements, which may require more open-source contributions. ii. The toolkit lacks support for training RAG components. While training was considered in the design phase, the diversity of methods and the availability of dedicated repositories led to its exclusion. In the future, we may provide auxiliary scripts to assist with training needs. iii. Currently, our toolkit includes datasets focused on multi-hop question answering, numerical reasoning, and logical reasoning. Future updates could expand to encompass OpenQA [21, 27], long-form Q&A [28, 37], and multiple-choice Q&A [14, 16]. We encourage the open-source community to overcome these limitations. Our goal is to persistently refine the Xrag framework by delivering a more efficient and dependable platform with comprehensive evaluation and development tools.

6 Conclusion

This work introduces Xrag, an open-source framework for benchmarking advanced RAG systems that facilitates exhaustive evaluation of foundational components across pre-retrieval, retrieval, post-retrieval, and generation phases. It introduces a modular RAG process, unified benchmark datasets, comprehensive testing methodologies, and strategies for identifying and mitigating RAG failure points. The framework supports a range of evaluation metrics, including conventional retrieval and

generation assessments and cognitive LLM evaluations. We believe XRAM will empower researchers to construct and evaluate RAG modules, streamlining workflows efficiently.

References

- [1] Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. Self-rag: Learning to retrieve, generate, and critique through self-reflection. In *ICLR*. OpenReview.net, 2024.
- [2] Satanjeev Banerjee and Alon Lavie. METEOR: an automatic metric for MT evaluation with improved correlation with human judgments. In *IEEvaluation@ACL*, pages 65–72. Association for Computational Linguistics, 2005.
- [3] Scott Barnett, Stefanus Kurniawan, Srikanth Thudumu, Zach Brannelly, and Mohamed Abdelrazek. Seven failure points when engineering a retrieval augmented generation system. In *CAIN*, pages 194–199. ACM, 2024.
- [4] Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George van den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego de Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, Tom Hennigan, Saffron Huang, Loren Maggiore, Chris Jones, Albin Cassirer, Andy Brock, Michela Paganini, Geoffrey Irving, Oriol Vinyals, Simon Osindero, Karen Simonyan, Jack W. Rae, Erich Elsen, and Laurent Sifre. Improving language models by retrieving from trillions of tokens. In *ICML*, volume 162 of *Proceedings of Machine Learning Research*, pages 2206–2240. PMLR, 2022.
- [5] Devrim Cavusoglu, Ulas Sert, Secil Sen, and Sinan Altinuc. Jury: A comprehensive evaluation toolkit, 2023.
- [6] Harrison Chase. LangChain, October 2022. URL <https://github.com/langchain-ai/langchain>.
- [7] Jiawei Chen, Hongyu Lin, Xianpei Han, and Le Sun. Benchmarking large language models in retrieval-augmented generation. In *AAAI*, pages 17754–17762. AAAI Press, 2024.
- [8] Gordon V. Cormack, Charles L. A. Clarke, and Stefan Büttcher. Reciprocal rank fusion outperforms condorcet and individual rank learning methods. In *SIGIR*, pages 758–759. ACM, 2009.
- [9] Shehzaad Dhuliawala, Mojtaba Komeili, Jing Xu, Roberta Raileanu, Xian Li, Asli Celikyilmaz, and Jason Weston. Chain-of-verification reduces hallucination in large language models. In *ACL (Findings)*, pages 3563–3578. Association for Computational Linguistics, 2024.
- [10] Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In *Proc. of NAACL*, 2019.
- [11] Luyu Gao, Xueguang Ma, Jimmy Lin, and Jamie Callan. Precise zero-shot dense retrieval without relevance labels. In *ACL (1)*, pages 1762–1777. Association for Computational Linguistics, 2023.
- [12] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Qianyu Guo, Meng Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. *CoRR*, abs/2312.10997, 2023.
- [13] Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. REALM: retrieval-augmented language model pre-training. *CoRR*, abs/2002.08909, 2020.
- [14] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *ICLR*. OpenReview.net, 2021.
- [15] Yasuto Hoshi, Daisuke Miyashita, Youyang Ng, Kento Tatsuno, Yasuhiro Morioka, Osamu Torii, and Jun Deguchi. Ralle: A framework for developing and evaluating retrieval-augmented large language models. In *EMNLP (Demos)*, pages 52–69. Association for Computational Linguistics, 2023.

- [16] Zixian Huang, Ao Wu, Jiaying Zhou, Yu Gu, Yue Zhao, and Gong Cheng. Clues before answers: Generation-enhanced multiple-choice QA. In *NAACL-HLT*, pages 3272–3287. Association for Computational Linguistics, 2022.
- [17] Peter Izsak, Moshe Berchansky, Daniel Fleischer, and Ronen Laperdon. fastRAG: Efficient Retrieval Augmentation and Generation Framework, February 2023. URL <https://github.com/IntelLabs/fastrag>.
- [18] Fred Jelinek, Robert L Mercer, Lalit R Bahl, and James K Baker. Perplexity—a measure of the difficulty of speech recognition tasks. *The Journal of the Acoustical Society of America*, 62(S1): S63–S63, 1977.
- [19] Zhengbao Jiang, Frank F. Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. Active retrieval augmented generation. In *EMNLP*, pages 7969–7992. Association for Computational Linguistics, 2023.
- [20] Jiajie Jin, Yutao Zhu, Xinyu Yang, Chenghao Zhang, and Zhicheng Dou. Flashrag: A modular toolkit for efficient retrieval-augmented generation research. *CoRR*, abs/2405.13576, 2024.
- [21] Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. In *ACL (1)*, pages 1601–1611. Association for Computational Linguistics, 2017.
- [22] Omar Khattab and Matei Zaharia. Colbert: Efficient and effective passage search via contextualized late interaction over BERT. In *SIGIR*, pages 39–48. ACM, 2020.
- [23] Bwook Kim and Jeffrey Kim. AutoRAG, October 2024. URL <https://github.com/Marker-Inc-Korea/AutoRAG>.
- [24] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur P. Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural questions: a benchmark for question answering research. *Trans. Assoc. Comput. Linguistics*, 7:452–466, 2019.
- [25] Jerry Liu. LlamaIndex, 11 2022. URL https://github.com/jerryjliu/llama_index.
- [26] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Trans. Assoc. Comput. Linguistics*, 12:157–173, 2024.
- [27] Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Daniel Khashabi, and Hannaneh Hajishirzi. When not to trust language models: Investigating effectiveness of parametric and non-parametric memories. In *ACL (1)*, pages 9802–9822. Association for Computational Linguistics, 2023.
- [28] Sewon Min, Kalpesh Krishna, Xinxi Lyu, Mike Lewis, Wen-tau Yih, Pang Wei Koh, Mohit Iyyer, Luke Zettlemoyer, and Hannaneh Hajishirzi. Factscore: Fine-grained atomic evaluation of factual precision in long form text generation. In *EMNLP*, pages 12076–12100. Association for Computational Linguistics, 2023.
- [29] Andrew Cameron Morris, Viktoria Maier, and Phil D. Green. From WER and RIL to MER and WIL: improved evaluation measures for connected speech recognition. In *INTERSPEECH*, pages 2765–2768. ISCA, 2004.
- [30] Krishna Pillutla, Swabha Swayamdipta, Rowan Zellers, John Thickstun, Sean Welleck, Yejin Choi, and Zaïd Harchaoui. MAUVE: measuring the gap between neural text and human text using divergence frontiers. In *NeurIPS*, pages 4816–4828, 2021.
- [31] Maja Popovic. chrf: character n-gram f-score for automatic MT evaluation. In *WMT@EMNLP*, pages 392–395. The Association for Computer Linguistics, 2015.
- [32] Maja Popovic. chrf++: words helping character n-grams. In *WMT*, pages 612–618. Association for Computational Linguistics, 2017.

- [33] Qiu Ran, Yankai Lin, Peng Li, Jie Zhou, and Zhiyuan Liu. Numnet: Machine reading comprehension with numerical reasoning. In *EMNLP/IJCNLP (1)*, pages 2474–2484. Association for Computational Linguistics, 2019.
- [34] Lin CY Rouge. A package for automatic valuation of summaries. 5:74–81, 2004.
- [35] Keshav Santhanam, Omar Khattab, Jon Saad-Falcon, Christopher Potts, and Matei Zaharia. Colbertv2: Effective and efficient retrieval via lightweight late interaction. In *NAACL-HLT*, pages 3715–3734. Association for Computational Linguistics, 2022.
- [36] Weijia Shi, Sewon Min, Michihiro Yasunaga, Minjoon Seo, Richard James, Mike Lewis, Luke Zettlemoyer, and Wen-tau Yih. REPLUG: retrieval-augmented black-box language models. In *NAACL-HLT*, pages 8371–8384. Association for Computational Linguistics, 2024.
- [37] Ivan Stelmakh, Yi Luan, Bhuwan Dhingra, and Ming-Wei Chang. ASQA: factoid questions meet long-form answers. In *EMNLP*, pages 8273–8288. Association for Computational Linguistics, 2022.
- [38] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018.
- [39] Xiao Yu, Yunan Lu, and Zhou Yu. Localrqa: From generating data to locally training, testing, and deploying retrieval-augmented QA systems. *CoRR*, abs/2403.00982, 2024.
- [40] Xuanwang Zhang, Yunze Song, Yidong Wang, Shuyun Tang, Xinfeng Li, Zhengran Zeng, Zhen Wu, Wei Ye, Wenyuan Xu, Yue Zhang, et al. Raglab: A modular and research-oriented unified framework for retrieval-augmented generation. *arXiv preprint arXiv:2408.11381*, 2024.
- [41] Huaixiu Steven Zheng, Swaroop Mishra, Xinyun Chen, Heng-Tze Cheng, Ed H. Chi, Quoc V. Le, and Denny Zhou. Take a step back: Evoking reasoning via abstraction in large language models. In *ICLR*. OpenReview.net, 2024.

Table 7: The CogR performances of advanced pre-retrieval prompt engineering modules in HotpotQA, including Hypothetical Document Embeddings (HyDE), Step Back Prompting (SBPT), and Chain of Verification (CoVe). **GPT-3.5 Turbo** is used as the LLM within the RAG framework. Few-shot performance is provided to clarify the effect of refined prompts.

Dataset & Methods	CogR								
	F1	MRR	Hit@1	Hit@5	MAP	NDCG	DCG°	IDCG°	EM
NaN	69.85	77.22	88.33	98.33	21.11	79.14	1.0705	0.7699	15.03
HyDE	71.57	75.14	95.00	100	19.08	69.57	1.0674	0.7928	15.07
HyDE (5-SHOT)	71.33	73.11	93.33	100	19.83	65.16	1.1128	0.8348	16.11
SBPT	69.77	72.97	80.00	100	18.50	68.94	1.0205	0.7699	15.20
SBPT (5-SHOT)	70.51	82.64	90.00	100	22.76	74.72	1.1208	0.8170	16.11
CoVe	40.87	82.50	55.00	85.00	65.83	86.47	1.6746	1.6996	15.03
CoVe (5-SHOT)	39.00	84.17	61.67	86.67	66.83	84.66	1.9453	0.9803	15.03

Table 8: The CogL performances of advanced pre-retrieval prompt engineering modules in HotpotQA, including Hypothetical Document Embeddings (HyDE), Step Back Prompting (SBPT), and Chain of Verification (CoVe). **GPT-3.5 Turbo** is used as the LLM within the RAG framework. Few-shot performance is provided to clarify the effect of refined prompts.

Dataset & Methods	CogL [Merge Retrieval & Generation]								P_{sc}
	Dp-CPre	Dp-CRec	Dp-CRel	Up-RCns	Up-CUti	Up-FAcc	Dp-Fath	Dp-Hall	
NaN	0.8153	0.6966	0.9333	0.3000	0.2667	0.5306	0.9138	0.9833	97.79
HyDE	0.8814	0.8167	0.9567	0.2650	0.2167	0.5291	0.9310	0.9667	98.67
HyDE (5-SHOT)	0.8667	0.7833	0.9333	0.2667	0.2750	0.5469	0.9667	0.9833	98.33
SBPT	0.8833	0.7836	0.9667	0.2983	0.2750	0.5513	0.9000	0.9833	99.33
SBPT (5-SHOT)	0.8750	0.7833	0.8333	0.2650	0.3417	0.5099	0.9474	0.9667	98.67
CoVe	0.7667	0.6552	1.0000	0.9983	0.7833	0.7250	0.6833	0.9500	94.33
CoVe (5-SHOT)	0.8467	0.7414	1.0000	1.0000	0.8250	0.7569	0.7333	0.9500	98.67

A Appendix / supplemental material

A.1 More RAG Core Components Analysis

A.1.1 Advanced Pre-Retrieval Modules

In Table 8, SBPT and its 5-shot variant excel in Dp-CRel and Up-RCns, highlighting their ability to enhance the relevance and consistency of the retrieval context, which is crucial for the subsequent generation task. CoVe, despite achieving high relevance, shows a trade-off with factual accuracy, as evidenced by lower Dp-Fath and Dp-Hall scores.

A.1.2 Advanced Retriever Modules

The experimental results from Table 9 demonstrate a clear performance hierarchy among the retrieval modules. RecuChunk and LexicalBM25 lead in terms of MRR and Hit@1, suggesting their robustness in identifying relevant documents early in retrieval. HiParser and StParser have the highest F1 scores of 91.50 and 92.50, respectively, and both achieve an MRR of 93.75 and 97.50, with Hit@1 scores of 90.00. Their NDCG scores are 95.31 and 98.16, respectively, which are among the highest, suggesting excellent retrieval performance. Conversely, TreeLeaf and LongCTrrk lag in multiple metrics, including F1 and MRR, which may point to limitations in their retrieval strategies.

A.2 Optimization Strategies & Evaluation of RAG Failures

In the previous section, we summarized five types of failures in RAG systems. These issues affect the performance and reliability of RAG systems and limit their effectiveness in practical applications. To address these challenges, we have proposed various optimization strategies, constructed evaluation datasets, and utilized detailed evaluation metrics to assess the effectiveness of the improvements, as shown in Table 10.

Table 9: Performance of advanced retrieval modules in HotpotQA. Some are common, like keyword search (BM25) and reranking (SQFusion, RRFusion), alongside specialized techniques tailored for RAG workflows, including hierarchical retrieval (HiParser), sentence window parsing (StParser), recursive nodes (RecuChunk), and tree leaf nodes (TreeLeaf). The last three are reranked models.

Methods	ConR								CogL [Retrieval]	
	F1	MRR	Hit@1	Hit@5	MAP	NDCG	DCG°	IDCG°	Up-CRel	Up-CCns
TreeLeaf	37.78	56.67	28.33	56.67	42.50	56.67	0.5667	0.5667	0.2877	0.8947
RecuChunk	70.00	100	40.00	100	85.00	100	1.0946	1.0946	0.5250	0.9215
LexicalBM25	85.00	86.67	95.00	100	84.17	90.16	1.6297	1.7744	0.7303	0.9583
SQFusion	86.94	95.00	80.00	100	90.00	96.31	1.4994	1.5363	0.7493	0.9491
RRFusion	87.22	95.00	81.67	100	90.42	96.31	1.4889	1.5258	0.7412	0.9288
HiParser	91.50	93.75	90.00	100	90.00	95.31	1.6894	1.7363	0.8048	0.9648
StParser	92.50	97.50	90.00	100	95.00	98.16	1.5178	1.5363	0.8083	0.9833
CoLBERT_{rrk}	70.11	74.58	86.67	98.33	58.54	72.99	1.0608	1.4816	0.4083	0.3917
BGE-BASE_{rrk}	82.50	87.50	75.00	98.33	50.42	77.00	1.1242	1.0166	0.6417	0.5250
LongCT_{rrk}	71.56	75.97	86.67	100	61.83	79.66	1.3223	0.8453	0.5000	0.3684

Table 10: Experimental setup for RAG failure analysis. ♣ denotes the pre-retrieval method, ▼ represents the post-processor method, and ♠ represents the advanced retriever.

RAG Failures	Optimization Strategies	Dataset Settings	Evaluation Metrics
Negative Refusal	▼ Prompt Engineering ▼ Two-step Reasoning	Randomly sampled queries with unrelated context	Rejection Rate
Ranking Confusion	▼ Re-ranking ♣ Hybrid Retrieval ♣ ▼ Hybrid Retrieval & Re-ranking	Samples with lower F1 scores	F1, Hit@1, EM, MRR MAP, DCG, IDCG, NDCG
Answer Absence	▼ Simple Summarize ▼ Refine ▼ Compact ▼ Compact Accumulate	Samples with missing answers	Factual Accuracy (Up-FAcc) Response Consistency (Up-RCns) Context Utilization (Up-CUti)
Noise Impact	▼ Re-ranking	Random samples with varying numbers of noisy context	Response Completeness (Up-RCmp) Response Matching (Up-RMch)
Complex Reasoning	♣ Query Rewriting ♣ Query Decomposition ▼ Few-shot Prompting	Random samples from HotpotQA Hard	

A.2.1 Negative Refusal

Optimization Strategies.

- **Prompt Engineering:** Explicit prompts can encourage LLMs to engage in more thoughtful and judgmental reasoning, aiming to elicit negative responses at appropriate times. As an effective means commonly employed in the domain of LLMs to address a wide range of problems, prompt engineering represents the simplest solution to the challenge of negative refusal.
- **Two-step Reasoning:** The two-step reasoning process leverages the capabilities of LLMs first to assess their ability to answer a given query. Only when the model determines it has adequate information to respond does it proceed to the second step to provide a specific answer, thus arriving at the final inference. Conversely, if the initial assessment concludes that there is insufficient information to answer, the second step of reasoning is declined, thereby preventing the generation of hallucinatory responses.

Experimental Settings. To quantify the system’s capacity for negative refusal, we first constructed a dataset comprising 20 randomly sampled questions, each paired with unrelated context to simulate scenarios where the model cannot access necessary information. We adopted the rejection rate as an evaluation metric to measure the system’s ability to recognize and acknowledge situations that cannot provide a valid response. A higher rejection rate indicates a higher level of self-awareness and reliability in the system’s responses. Without any optimizations, we established a baseline by measuring the system’s rejection rate, then applied each optimization method, recording the rejection

rates under new conditions to reveal the feasibility of each common strategy in enhancing the model’s knowledge awareness and proper negation.

Table 11: Evaluation results of the strategies of negative refusal.

Strategies		Rejection Rate
Wrong Context	Prompt Engineering	1.000
	Two-step Reasoning	1.000
Correct Context	Prompt Engineering	0.600
	Two-step Reasoning	0.100

Evaluation & Results. Regarding the problem of negative refusal, employing prompt engineering to explicitly require the LLMs to possess the ability to refuse, along with the two-step reasoning approach—where the LLMs first evaluate whether the available information is sufficient to respond to the user’s query—both effectively increase the rejection rate of the RAG system, as shown in Table 11. However, the experimental results further indicate that, in scenarios with the correct context, prompt engineering leads to a much higher probability (up to 60%) of still refusing to provide an answer compared to the two-step reasoning method, severely undermining the usability of the RAG system. This highlights the importance of adequately designing system prompts and suggests that fixed prompt content might lack flexibility when dealing with diverse real-world situations.

A.2.2 Ranking Confusion

Optimization Strategies.

- **Re-ranking:** Employing a re-ranking model for more refined similarity calculations, although computationally more expensive than initial retrieval and typically applied only to smaller sample sizes, can significantly enhance the relevance and precision of retrieval results. In this section, we opt for the commonly used ColBERTv2 (Cohere rerank model) as the re-ranking model to perform finer-grained matching on the coarse retrieval outcomes.
- **Hybrid Retrieval:** Integrating multiple retrieval methods can effectively broaden the diversity of retrieval results, allowing for the identification of relevant document segments from various perspectives, which could improve ranking accuracy. Specifically, we implement a hybrid retrieval approach combining BM25 with vector-based retrieval to construct the RAG system’s hybrid retriever.
- **Hybrid Retrieval and Re-ranking:** After implementing the hybrid retrieval strategy, further application of the re-ranking method combines the dual advantages of diversified retrieval and precise similarity calculation. This approach aims to more effectively identify and rank the most relevant document segments, thereby enhancing the overall performance of the RAG system.

Experimental Settings. For the verification, we selected 20 samples with lower F1 scores under standard RAG retrieval conditions to facilitate the validation of the effectiveness of various methods. In terms of evaluation metrics, we use traditional measures such as F1 and EM to assess retrieval accuracy, along with additional metrics like Hit@1, MRR, MAP, DCG, IDCG, and NDCG to precisely evaluate the extent to which the aforementioned optimization strategies improve the correct ordering of retrieval results.

Table 12: Evaluation results of the strategies of ranking confusion (RR: Re-ranking, HR: Hybrid Retrieval).

Strategies	Evaluation Metrics							
	F1 (↑)	EM (↑)	MRR (↑)	HIT@1 (↑)	MAP (↑)	DCG° (↑)	IDCG° (↑)	NDCG (↑)
Basic-RAG	0.7400	0.0000	0.6670	0.9000	0.6800	1.2100	1.5300	0.7800
w/ RR	0.8000	0.0000	0.7500	1.0000	0.7917	1.6309	1.3809	0.8467
w/ HR	0.9250	0.8500	0.9250	0.9000	0.9125	1.4800	1.5360	0.9450
w/ HR + RR	0.9750	0.9500	0.9750	1.0000	1.0000	1.6000	1.6309	0.9816

Evaluation & Results. Regarding ranking confusion, re-ranking and hybrid retrieval strategies significantly improve the retrieval performance of RAG systems. As shown in Table 12, the gain from hybrid retrieval is noticeably superior to re-ranking. This advantage may be attributed to the

hybrid retrieval’s capability to perceive a broader range of relevant information snippets and its comprehensive integration of multi-source information, leading to enhanced accuracy and better ranking quality. Notably, the simultaneous application of both techniques achieves the most optimal performance improvement.

A.2.3 Answer Absence

Optimization Strategies.

- **Simple Summarize:** All retrieved document chunks are concatenated into a single text block and fed into the LLMs in one go.
- **Refine:** Each retrieved document chunk undergoes a separate Q&A session with the LLMs, with inputs including the original query, the answer from the previous round, and the current document chunk. A predefined prompt is used to refine each answer to elicit more detailed responses.
- **Compact:** Document chunks are first merged into longer blocks as much as possible before applying the Refine method, reducing the number of calls to LLMs.
- **Compact Accumulate:** Document chunks are similarly merged into longer blocks, but each chunk undergoes independent Q&A sessions with inputs being the original query and the current document chunk, without the answer from the previous round. All results are then combined to form the final response.

Experimental Settings. To conduct our experiments, we randomly sampled a batch of questions, providing the questions and their golden contexts to the LLMs to obtain responses for each question. Subsequently, through manual screening, we selected 20 questions and their corresponding golden contexts that exhibited missing answers to serve as the evaluation dataset for this section. To comprehensively validate the effectiveness of various methods, we adopt Factual Accuracy, Response Consistency, Context Utilization, Response Completeness, and Response Matching as evaluation metrics.

Table 13: Evaluation results of the strategies of answer absence. All metrics are discrete values within the interval [0, 1].

Strategies	Evaluation Metrics				
	Up-FAcc (↑)	Up-RCns (↑)	Up-CUti (↑)	Up-RCmp (↑)	Up-RMch (↑)
Simple Summarize	0.0250	1.0000	0.8250	0.8500	0.2790
Refine	0.9400	1.0000	0.6750	0.5750	0.2050
Compact	0.8750	1.0000	0.7000	0.4250	0.2930
Compact Accumulate	0.8191	0.9650	0.9750	0.8500	0.3840

Evaluation & Results. Concerning the phenomenon of answer absence, we tested various methods for inputting document chunks into the LLMs. The results show that different input methods affect the correctness and comprehensiveness of the model’s responses, as illustrated in Table 13. Among these, independently querying each document chunk produced the best results, whereas the more complex iterative response generation methods, such as Refine and Compact modes, performed poorly. This indicates that simply increasing the complexity of the RAG system does not always yield benefits and can sometimes have counterproductive effects.

A.2.4 Noise Impact

Optimization Strategies.

- **Re-ranking:** Employing a re-ranking model for more refined similarity calculations not only serves as a potential solution to the ranking confusion problem but can also help filter out irrelevant documents through threshold filtering or quantity filtering, thereby reducing the amount of noise input to LLMs and improving the performance of the RAG system.

Experimental Settings. We randomly sampled 20 query instances and combined them with their corresponding golden contexts and varying numbers of irrelevant document chunks to form the evaluation dataset. This allows us to explore the changes in RAG system performance under different proportions of noisy documents, as well as the enhancement of system capability after incorporating a

post-processing re-ranking module. We adopt the same set of metrics used in A.2.3 for the evaluation metrics.

Table 14: Evaluation results of the strategies of noise impact. All metrics are discrete values within the interval [0, 1].

Noise number	Strategies	Evaluation metrics				
		Up-FAcc (\uparrow)	Up-RCns (\uparrow)	Up-CUti (\uparrow)	Up-RCmp (\uparrow)	Up-RMch (\uparrow)
0	w/o Re-ranking	0.8950	1.0000	0.6750	0.5000	0.3920
	w/ Re-ranking	0.9000	0.9950	0.8000	0.4500	0.4700
1	w/o Re-ranking	0.6790	0.7976	0.8571	0.5714	0.2357
	w/ Re-ranking	0.5883	1.0000	0.6500	0.5500	0.4000
2	w/o Re-ranking	0.8650	0.9500	0.7500	0.4000	0.3760
	w/ Re-ranking	0.5883	1.0000	0.6500	0.5500	0.4000
3	w/o Re-ranking	0.7925	0.9500	0.7000	0.4000	0.2975
	w/ Re-ranking	0.8017	1.0000	0.7000	0.7000	0.4800

Evaluation & Results. When the document chunks within retrieval results contain noise, the output accuracy of the RAG system deteriorates as the amount of noise increases, as seen in Table 14. Post-processing the retrieval results using re-ranking methods can somewhat mitigate this issue, and the improvement becomes more pronounced as the number of noisy document chunks increases.

A.2.5 Complex Reasoning

Optimization Strategies.

- **Query Rewriting:** By rewriting user queries using an LLM to add and introduce more information, this approach aims to guide the retrieval module to access more relevant documents explicitly.
- **Query Decomposition:** Breaking down complex questions into simpler sub-questions that focus on individual aspects simplifies the reasoning process, enabling a comprehensive search for all required information.
- **Few-shot Prompting:** By prepending a few complex reasoning examples to the input sequence of the LLM, this method aims to guide and stimulate the model’s latent reasoning abilities, thereby enhancing the effectiveness of complex reasoning.

Experimental Settings. For verification of complex reasoning, we randomly sampled 20 items from the Hard subset of HotpotQA to serve as the task set for complex logic. Similarly, we adopted the same metrics used in A.2.3 to evaluate the improvement effects of each solution comprehensively.

Table 15: Evaluation results of the strategies of complex reasoning. All metrics are discrete values within the interval [0, 1].

Strategies	Evaluation Metrics				
	Up-FAcc (\uparrow)	Up-RCns (\uparrow)	Up-CUti (\uparrow)	Up-RCmp (\uparrow)	Up-RMch (\uparrow)
Basic-RAG	0.5700	1.0000	0.7500	0.4000	0.3830
Query Rewriting	0.6750	0.9900	0.7750	0.5500	0.4430
Query Decomposition	0.6250	0.9500	0.5500	0.3250	0.2790
Few-shot Prompting	0.6250	0.8900	0.5250	0.2500	0.2410

Evaluation & Results. When the RAG system faces complex reasoning tasks, our experimental results demonstrate that query rewriting significantly enhances the system’s performance in addressing such challenges, as evidenced in Table 15. This suggests that unclear or insufficiently detailed user queries may be a critical factor limiting system performance in complex scenarios. Additionally, neither query decomposition nor few-shot prompting positively impacted this challenge; instead, they led to a further decline in system performance. This reaffirms that increasing the complexity of the RAG system may introduce potential performance risks and that system prompts need to be carefully and thoroughly tested rather than made more complex and detailed.

A.3 Detail of Datasets & Corpus

The corpora for the three datasets originate from Wikipedia. We construct retrievable documents using metadata from the original datasets, standardizing retrieval objects as document IDs for

testing. Retrieval is successful if the retrieved chunk node corresponds to the annotated document ID. This method ensures consistency in retrieval labels and eliminates discrepancies caused by varying document chunking strategies.

We also generate metadata for all three datasets, including training and validation sets, to facilitate fine-tuning RAG systems or other customized tasks. For the test set, we limit the number of samples to mitigate the high token cost associated with RAG and LLM evaluations. Instead of evaluating the entire test set, we apply a sampling-based averaging method, reducing token usage while preserving reliability.

Moreover, Figure 3 illustrates the distribution of context lengths across three Q&A datasets: HotpotQA, DropQA, and NaturalQA. HotpotQA shows a tight clustering of context lengths between 0.1 to 0.3k tokens, focusing on shorter contexts. DropQA presents a broader range, with most contexts falling within the 0.1 to 0.5k tokens range but with a more extended tail. NaturalQA exhibits the most diverse distribution, with context lengths spanning from very short to as long as 140k tokens, reflecting a design that accommodates various text lengths for Q&A tasks.

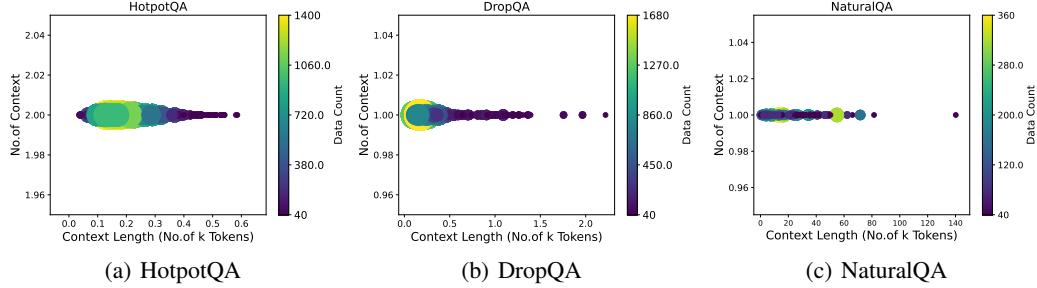


Figure 3: The golden contextual distribution of the corpora across three datasets providing the quantity and length of annotated contexts. This distribution aids in analyzing the contextual structure, enabling a clearer understanding of how contexts vary in detail for each dataset.

A.3.1 Detail of HotpotQA

HotpotQA is a dataset with Wikipedia-based question-answer pairs with four key features:

- Questions require reasoning across multiple supporting documents.
- Questions are diverse and independent of pre-existing knowledge schemas.
- It has the largest set of retrievable documents but lacks logical operations in its queries.
- Answers are typically long, often in a paragraph.

A.3.2 Detail of DropQA

DropQA is a reading comprehension benchmark requiring discrete reasoning over paragraphs:

- Answers span multiple text positions and involve discrete operations such as addition, counting, and sorting, requiring a deeper understanding of paragraph content than prior datasets.
- Answers are generally short, comprising one or more entities.
- The dataset contains a significant number of logic-based reasoning questions.

A.3.3 Detail of NaturalQA

NaturalQA consists of questions derived from an actual Google search engine, emphasizing real-world, open-ended inquiries.

- Questions are diverse, covering a broad range of topics.
- Answers are typically short, consisting of one or more entities.

A.4 Detail of Cognitive LLM Evaluation

‘Ex.’ provides descriptions or explanations of the metrics, while ‘Pa.^u’ specifies their standardized input parameters (all parameters are shown in Section 3.2). We utilize **GPT-4 Turbo** as the LLM agent in Cognitive LLM Evaluation.

- **Up-CRel [Retrieval]: Uptrain-Context-Relevance**
 - Ex. : Context relevance measures if the retrieved context has enough information to answer the question being asked.
 - Pa.^u : < Question, Retrieval-Context >
- **Up-CCns [Retrieval]: Uptrain-Context-Conciseness**
 - Ex. : Context conciseness refers to the quality of a reference context generated from the retrieved context in terms of being clear, brief, and to the point.
 - Pa.^u : < Question, Golden-Context, Retrieval-Context >
- **Dp-ARel [Generation]: DeepEval-Response-Relevancy**
 - Ex. : Response relevancy measures how relevant the actual response is compared to the provided input.
 - Pa.^u : < Question, Actual-Response >
- **Up-RCmp [Generation]: Uptrain-Response-Completeness**
 - Ex. : Response completeness measures if the generated response adequately answers all aspects of the question being asked.
 - Pa.^u : < Question, Actual-Response >
- **Up-RCnc [Generation]: Uptrain-Response-Conciseness**
 - Ex. : Response conciseness measures whether the generated response contains any additional information irrelevant to the question asked.
 - Pa.^u : < Question, Actual-Response >
- **Up-RRel [Generation]: Uptrain-Response-Relevance**
 - Ex. : Response conciseness measures whether the LLM-generated text contains any additional information irrelevant to the question asked.
 - Pa.^u : < Question, Actual-Response >
- **Up-RVal [Generation]: Uptrain-Response-Valid**
 - Ex. : In some cases, an LLM might fail to generate a response due to limited knowledge or unclear questions. Response Validity score can be used to identify these cases where a model is not generating an informative response.
 - Pa.^u : < Question, Actual-Response >
- **Up-RMch [Generation]: Uptrain-Response-Matching**
 - Ex. : Response matching compares the LLM-generated text with the gold (ideal) response using the defined score metric.
 - Pa.^u : < Question, Actual-Response, Expected-Answer >
- **Dp-CPre [Merge Retrieval & Response]: DeepEval-Context-Precision**
 - Ex. : Contextual precision measures your RAG pipeline’s retriever by evaluating whether nodes in your retrieval context relevant to the given input are ranked higher than irrelevant ones.
 - Pa.^u : < Question, Actual-Response, Expected-Answer, Retrieval-Context >
- **Dp-CRec [Merge Retrieval & Response]: DeepEval-Context-Recall**
 - Ex. : Contextual recall metric measures the quality of your RAG pipeline’s retriever by evaluating how much the retrieval context aligns with the expected output.
 - Pa.^u : < Question, Actual-Response, Expected-Answer, Retrieval-Context >
- **Dp-CRel [Merge Retrieval & Response]: DeepEval-Context-Relevance**
 - Ex. : Contextual relevancy metric measures the quality of your RAG pipeline’s retriever by evaluating the overall relevance of the information presented in your retrieval context for a given input.
 - Pa.^u : < Question, Actual-Response, Retrieval-Context >
- **Up-RCns [Merge Retrieval & Response]: Uptrain-Context-Consistency**

- Ex. : Response Consistency measures how well the generated response aligns with both the question asked and the context provided.
 - Pa.^u : < Question, Actual-Response, Retrieval-Context >
 - **Up-CUti [Merge Retrieval & Response]: Uptrain-Context-Utilization**
 - Ex. : Context Utilization score measures if the generated response has sufficiently used the retrieved context to answer the question being asked.
 - Pa.^u : < Question, Actual-Response, Retrieval-Context >
 - **Up-FAcc [Merge Retrieval & Response]: Uptrain-Factual-Accuracy**
 - Ex. : Factual accuracy measures the degree to which a claim made in the response is true according to the context provided.
 - Pa.^u : < Question, Actual-Response, Retrieval-Context >
 - **Dp-Fath [Merge Retrieval & Response]: DeepEval-Context-Faithfulness**
 - Ex. : The response faithfulness measures whether the actual output factually aligns with the contents of your retrieval context.
 - Pa.^u : < Question, Actual-Response, Retrieval-Context >
 - **Dp-Hall [Merge Retrieval & Response]: DeepEval-Context-Hallucination**
 - Ex. : The response hallucination measures whether your LLM generates factually correct information by comparing the output to the provided context.
 - Pa.^u : < Question, Actual-Response, Golden-Context >
-

A.5 Prompt for XRAG Instructions

Prompt for XRAG suitable for HotpotQA datasets

Context information is below.

 {context_str}

 Given the context information and no prior knowledge, answer the question:
 {query_str}
 We have the opportunity to refine the original answer.
 (only if needed) with some more context below.

 {context_msg}

 Given the new context, refine the original answer to better.
 Answer the question: query_str
 If the context isn't proper, output the original answer again.
 Original Answer: existing_answer

Prompt for XRAG suitable for DropQA and NaturalQA datasets

Context information is below.

 {context_str}

 Given the context information and no prior knowledge. Please provide a brief, shortest possible answer,
 ideally just one word for the following question:
 Question: who has sold more records Oasis or Coldplay?
 Expected Answer: Oasis
 We have the opportunity to refine the original answer.
 (only if needed) with some more context below.

 {context_msg}

 Given the new context, refine the original answer to better.

Answer the question: query_str
If the context isn't proper, output the original answer again.
Original Answer: existing_answer

A.6 Prompt for Pre-Retrieval Instructions

Prompt for Few-shot HyDE

You are a sophisticated AI model. Please write a passage to answer the question. Try to include as many key details as possible:

1.

Question: At It Again contains lyrics co-written by the singer and actor from what city?

Passage: The song "At It Again" features lyrics co-written by the singer and actor who hail from the vibrant city of Los Angeles. This city, known for its thriving entertainment industry, has been a hub for countless talented individuals who have made their mark in music, film, and television. The singer and actor, both born and raised in Los Angeles, have been deeply influenced by the diverse and creative atmosphere of their hometown. Their collaboration on "At It Again" showcases their unique perspectives and storytelling abilities as they draw inspiration from their personal experiences and the rich cultural tapestry of Los Angeles. Through their lyrics, they paint a vivid picture of the city's energy, its dreams, and its challenges, capturing the essence of their beloved hometown in every verse.

2.

Question: {question}

Passage: {generated passage}

/* Few-shot prompt-engineering */ /* ToDo: bundle original prompt */

Prompt for Few-shot SBPT

You are an expert at world knowledge. Your task is to step back and paraphrase a question to a more generic step-back question, which is easier to answer.

Here are a few examples:

1.

Original Question: Musician and satirist Allie Goertz wrote a song about the "The Simpsons" character Milhouse, who Matt Groening named after who?

Stepback Question: Who are some notable figures that have influenced the names of "The Simpsons" characters?

2.

Original Question: {question}

Stepback Question: {generated stepback question}

/* Few-shot prompt-engineering */ /* ToDo: bundle original prompt */

Prompt for Few-shot CoVe

You are a world-class, state-of-the-art agent.

You have access to multiple tools, each representing a different data source or API. Each tool has a name and a description, formatted as a JSON dictionary. The keys of the dictionary are the names of the tools, and the values are the descriptions. Your purpose is to help answer a complex user question by generating a list of sub-questions that can be answered by the tools.

These are the guidelines you consider when completing your task:

- * Be as specific as possible.
- * The sub-questions should be relevant to the user question.
- * The sub-questions should be answerable by the tools provided.
- * You can generate multiple sub-questions for each tool.
- * Tools must be specified by their name, not their description.
- * You don't need to use a tool if you don't think it's relevant.

Here are some examples:

1.

Original Question: How was Paul Graham's life different before, during, and after YC?

Sub-questions:

- What did Paul Graham work on before YC?

```

- What did Paul Graham work on during YC?
- What did Paul Graham work on after YC?

2.
Original Question: {question}
Sub-questions: {generated sub-questions}
Output the list of sub-questions by calling the SubQuestionList function.
## Tools
... json
{tools_str}
...
## User Question
{query_str}
/* Few-shot prompt-engineering */

```

A.6.1 Development Web UI of Xrag

The Web UI of Xrag enables developers to construct tailored action chains involving multiple inference steps. Figures 5, 4, 6, 7 and 8 illustrate examples of Xrag serving methods. Xrag offers an intuitive front-end interface for uploading datasets in Figure 4, whether from this study or custom sources. It includes an interactive configuration tool in Figure 5 that allows users to configure API keys and parameter settings and build vector database indices directly.

Additionally, Xrag features a static evaluation page in Figure 6 for evaluating pre-generated responses. Figure S shows developers can create query-specific prompt templates using the integrated query engine. Xrag also provides comprehensive evaluation (Figure 7) and facilitates the execution of individual actions with customized prompts, enabling immediate review of retrieval and generation outcomes.

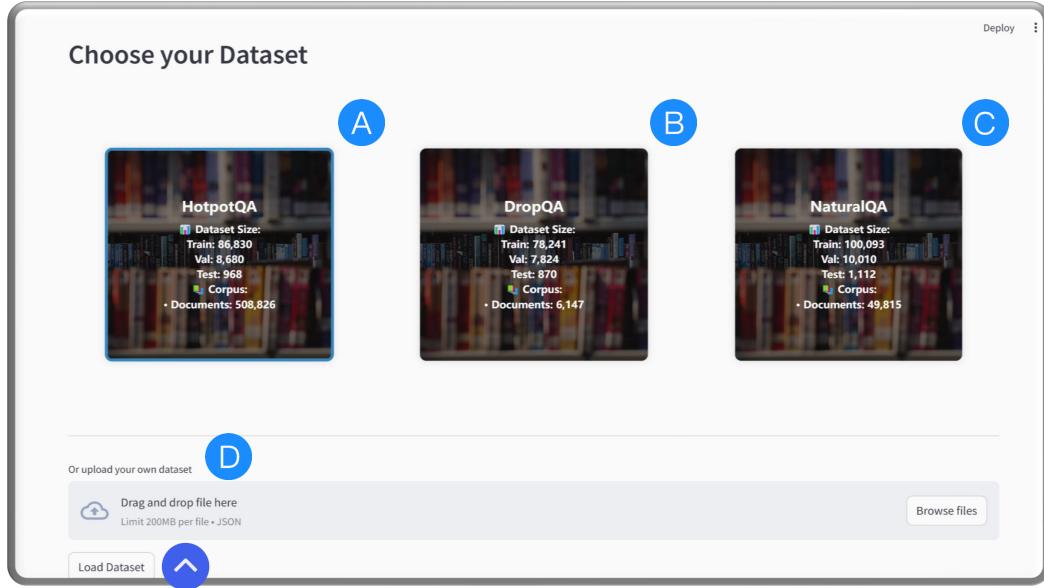


Figure 4: A screenshot of the Development Web UI of Xrag. Xrag's serving front-end interface for dataset uploads, supporting datasets from this study (HotpotQA, DropQA, NaturalQA) and custom datasets. The back-end automatically converts uploaded datasets into a unified format (see Section 3.2).

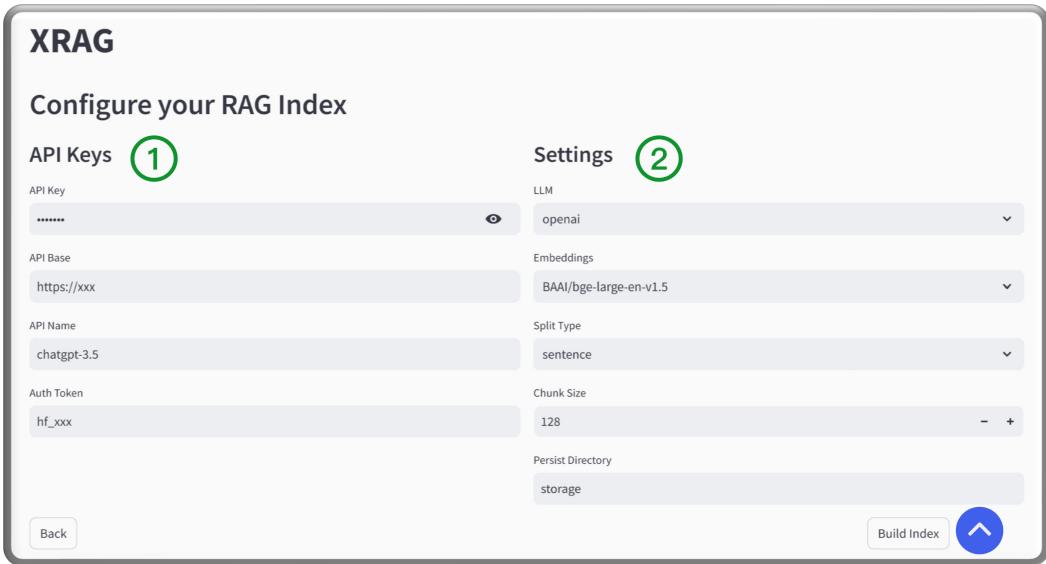


Figure 5: A screenshot of the Development Web UI of Xrag. Xrag’s interactive configuration tool for setting API keys and parameters and directly building vector database indices.

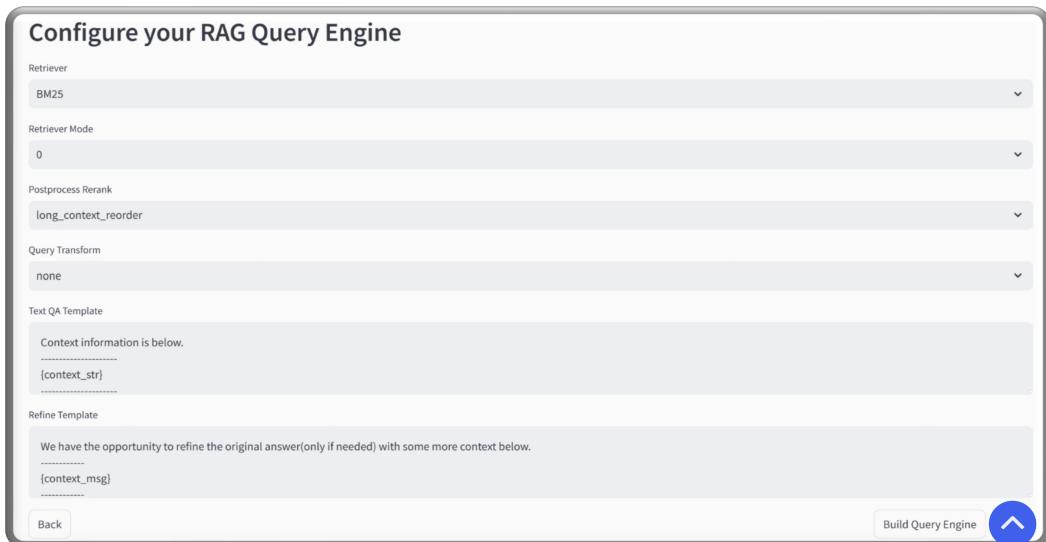


Figure 6: A screenshot of the Development Web UI of Xrag. Xrag’s interactive configuration tool for defining strategies for RAG advanced modules, including pre-retrieval methods, retrievers, and post-processors. Developers can also create prompt templates using the integrated query engine.

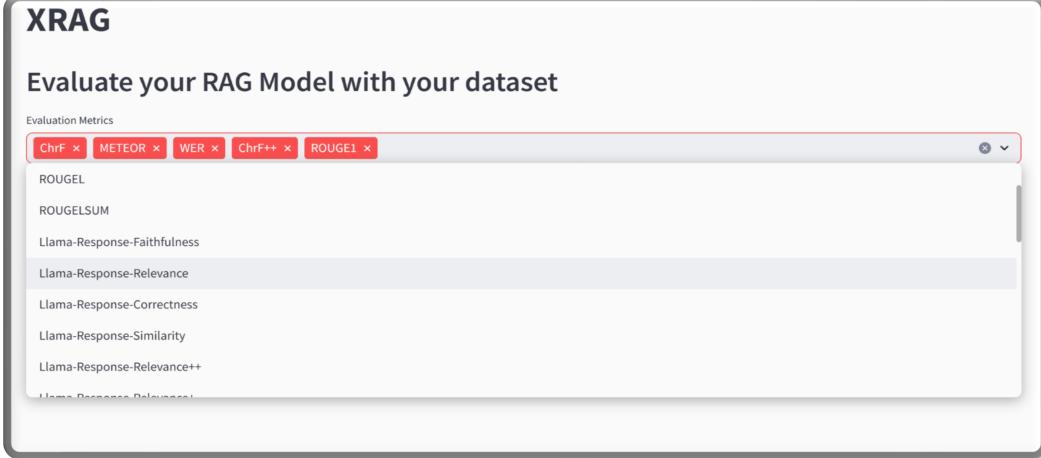


Figure 7: A screenshot of the Development Web UI of Xrag. Xrag's static evaluation page for directly assessing the quality of pre-generated responses.

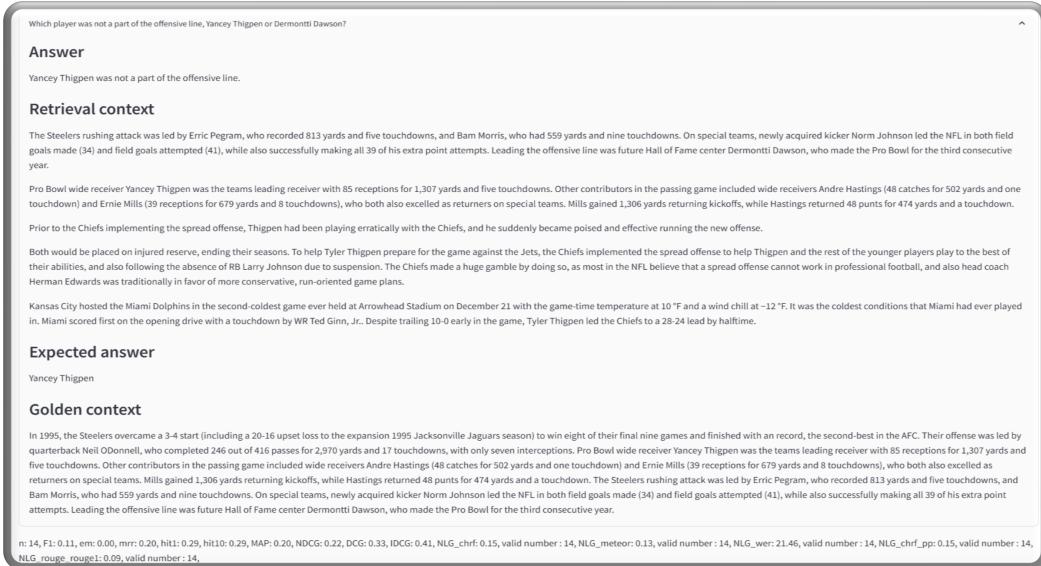


Figure 8: A screenshot of the Development Web UI of Xrag. Action execution with newly defined prompts in Xrag allows users to review retrieval and generate results efficiently.