RAGLAB: A Modular and Research-Oriented Unified Framework for Retrieval-Augmented Generation

Xuanwang Zhang[1,2,*], **Yunze Song**[2,*], **Yidong Wang**[3,2], **Shuyun Tang**[5], **Xinfeng Li**[4], **Zhengran Zeng**[3], **Zhen Wu**[1,†], **Wei Ye**[3], **Wenyuan Xu**[4], **Yue Zhang**[6], **Xinyu Dai**[1], **Shikun Zhang**[3], **Qingsong Wen**[2]

[1]Nanjing University [2]Squirrel AI [3]Peking University,
[4]Zhejiang University [5]Google [6]Westlake University

{zxw.ubw, YunzeSong77}@gmail.com

Abstract

Large Language Models (LLMs) demonstrate human-level capabilities in dialogue, reasoning, and knowledge retention. However, even the most advanced LLMs face challenges such as hallucinations and real-time updating of their knowledge. Current research addresses this bottleneck by equipping LLMs with external knowledge, a technique known as Retrieval Augmented Generation (RAG). However, two key issues constrained the development of RAG. First, there is a growing lack of comprehensive and fair comparisons between novel RAG algorithms. Second, open-source tools such as LlamaIndex and LangChain employ high-level abstractions, which results in a lack of transparency and limits the ability to develop novel algorithms and evaluation metrics. To close this gap, we introduce RAGLAB, a modular and research-oriented open-source library. RAGLAB reproduces 6 existing algorithms and provides a comprehensive ecosystem for investigating RAG algorithms. Leveraging RAGLAB, we conduct a fair comparison of 6 RAG algorithms across 10 benchmarks. With RAGLAB, researchers can efficiently compare the performance of various algorithms and develop novel algorithms.

https://github.com/fate-ubw/RAGLab

1 Introduction

Retrieval augmentation generation(RAG) leverages external knowledge to mitigate hallucination issues, ensure real-time knowledge updates, and protect private data with no parametric knowledge(Chen et al., 2017; Lewis et al., 2020; Guu et al., 2020). However, researchers face two main barriers to investigating new RAG algorithms. On the one hand, many published works are either not open-source or have difficulty setting up the environment. While open-source works lack modular design, it is hard to develop new algorithms or extend new datasets for evaluation. Researchers have to waste a lot of time developing new algorithms from scratch. On the other hand, a multitude of novel RAG algorithms have merged, including ITER-RETGEN(Shao et al., 2023), RRR(Ma et al., 2023), Self-Ask(Press et al., 2023), Active RAG(Jiang et al., 2023), Self-RAG(Asai et al., 2024), etc. However, these RAG algorithms are not well aligned in their fundamental components and evaluation methodologies, making it difficult for researchers to accurately assess their improvements. As a result, the absence of a unified framework makes it difficult for researchers and engineers to select appropriate algorithms for varied contexts, potentially hindering the advancement of the field.

Various current works are investigating these questions, such as LlamaIndex (Liu, 2022), LangChain(Chase, 2022), Haystack(Pietsch et al., 2019), FastRAG(Izsak et al., 2023), RALLE (Hoshi et al., 2023), LocalRQA(Yu et al., 2024), AutoRAG(Jeffrey Kim, 2024), and FlashRAG(Jin et al., 2024). LlamaIndex, LangChain, and Haystack are excessively encapsulated and lack transparency in internal operational mechanisms. Consequently, even experienced experts abandon tools like LangChain due to the lack of transparency(Woolf, 2023). FastRAG and RALLE offer light and transparent frameworks that enable users to assemble their own RAG systems using core components. AutoRAG provides comprehensive metrics to assist users in selecting an optimal RAG system for customized data. LocalRAG provides a wide selection of model training algorithms and evaluation methods. However, LocalRAG, FastRAG, AutoRAG, and RALLE do not reproduce published algorithms. Researchers still need to invest time in replicating algorithms using the provided components. FlashRAG addressed this issue by reproducing a substantial number of existing algorithms. However, FlashRAG lacks training functionalities and fails to properly align generators

[1]*Equal contribution
[2]†Corresponding author

Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing
System Demonstrations, pages 408–418

Table 1: Comparison of Different RAG Libraries and Frameworks. Fair Comparison refers to aligning all fundamental components during evaluation, including random seeds, generator, retriever, and instructions. Data Collector refers to the ability to gather or generate training and test data, either by sampling from existing raw datasets or by constructing labeled data using LLMs.

Library	Fair Comparison*	Data Collector*	Trainer	Auto Evaluation	Modular Design
Langchain(Chase, 2022)	✗	✗	✗	✗	✓
LlamaIndex(Liu, 2022)	✗	✗	✗	✓	✓
Haystack(Pietsch et al., 2019)	✗	✗	✗	✓	✓
FastRAG(Izsak et al., 2023)	✗	✗	✗	✗	✓
RALLE(Hoshi et al., 2023)	✗	✗	✗	✗	✓
LocalRQA(Yu et al., 2024)	✗	✓	✓	✓	✗
AutoRAG(Jeffrey Kim, 2024)	✗	✗	✗	✓	✓
FlashRAG(Jin et al., 2024)	✗	✗	✗	✓	✓
RAGLAB(ours)	✓	✓	✓	✓	✓

during inference, leading to unfair comparisons among various algorithms. For a more detailed comparison, refer to Table 1.

To close this gap, we present RAGLAB, a researcher-oriented RAG toolkit for a fair comparison of existing RAG algorithms and simplify the process of developing new algorithms. RAGLAB provides a modular architecture for each component of the RAG system, providing an ideal platform for fair comparison of algorithms. Additionally, RAGLAB designs an interactive mode and user-friendly interface, facilitating both educational purposes and demonstrations.

In this paper, we introduce the RAGLAB framework, giving an overview of core components and system workflows(section 2). We standardized key experimental variables: generator fine-tuning, instructions, retrieval configurations, knowledge bases, and benchmark. As a result, we present a comprehensive and fair comparison of 6 RAG algorithms across 10 benchmarks(section 3).

RAGLAB is available on GitHub under the MIT license.

2 RAGLAB

The overall architecture of RAGLAB is illustrated in Figure 1. We first introduce the core classes and concepts, then demonstrate an experimental case using a concise 5-line code snippet.

2.1 Classes and Concepts

2.1.1 Retriever

RAGLAB integrates two high-performing BERT-based models, Contriever(Izacard et al., 2021) and ColBERT(Santhanam et al., 2022). Furthermore, RAGLAB unifies the query interfaces across different retriever classes, making it possible for users to seamlessly switch between various retrievers. During the evaluation phase, researchers need to benchmark multiple RAG algorithms in parallel. In this context, repeatedly loading and querying retriever models and knowledge databases consumes a amount of time.

To address this issue, RAGLAB designs a retriever server and client architecture, enabling high-concurrency access to retrievers. Additionally, RAGLAB implements a retrieval caching mechanism. This mechanism stores the results of initial queries and their retrieved knowledge. Consequently, when queried with an identical input, the retriever will return the cached results directly without recomputation. Based on RAGLAB, users only need to load the retriever model and knowledge database once, facilitating retrieval services with latencies of less than 0.1 seconds across multiple parallel evaluation experiments.

2.1.2 Corpus

The external knowledge database has a significant impact on the performance of RAG systems. Wikipedia collects all kinds of knowledge and is broadly used in research areas. However, raw web data must undergo complex preprocessing before it can be directly utilized by RAG systems.

RAGLAB provides preprocessed Wikipedia corpora in two versions: the first version is based on the 2018 Wikipedia data open-sourced by the DPR project(Karpukhin et al., 2020); the second version utilizes the 2023 Wikipedia data open-sourced by the FactScore(Min et al., 2023). RAGLAB also prebuilt indices and embeddings for both the ColBERT and Contriever models, based on the Wikipedia 2018 and Wikipedia 2023 corpora. Additionally, RAGLAB open-sources all processing scripts, enabling researchers to directly download the prepro-

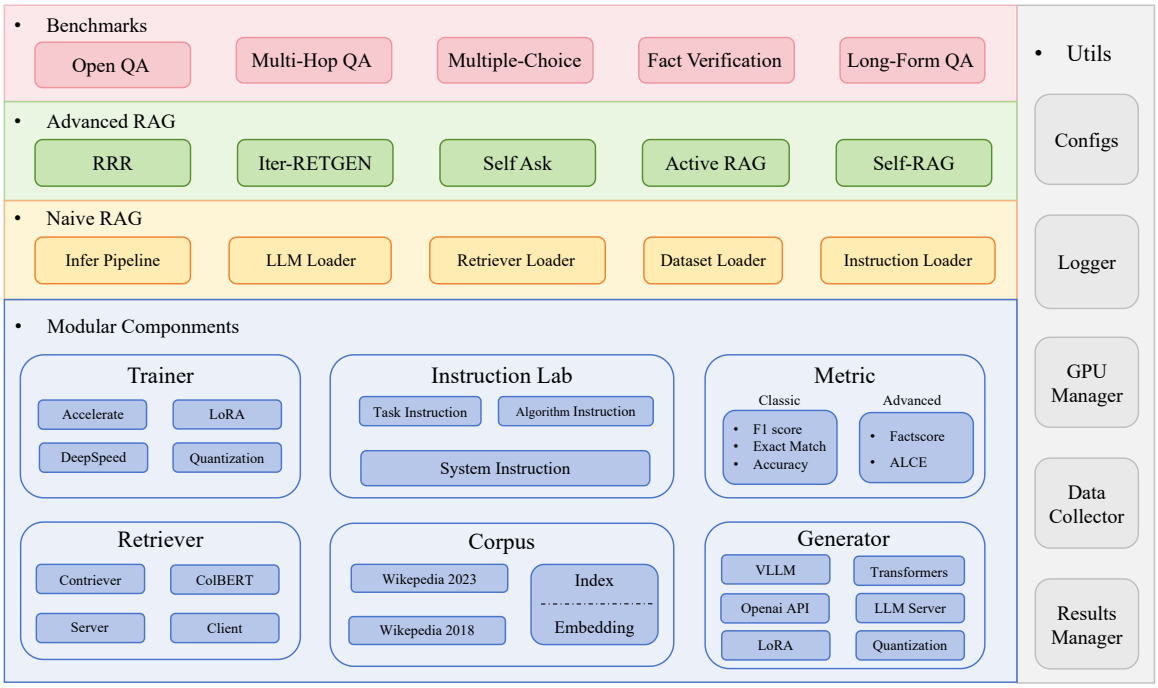

Figure 1: Architecture and Components of the RAGLAB Framework.

cessed Wikipedia corpora along with their corresponding indices and embeddings.

2.1.3 Generator

The generator is the core component of the RAG system. We integrate Huggingface Transformers(Wolf et al., 2020) and VLLM(Kwon et al., 2023), thereby enabling RAGLAB to be compatible with a wide range of open-source models while providing stable and efficient inference performance. RAGLAB also incorporates quantization and Low-Rank Adaptation (LoRA) (Hu et al., 2022) features, enabling users to employ models with 70 billion parameters or more as generators, even with limited computational resources.

Furthermore, anticipating the potential need for users to simultaneously load multiple generators within a single RAG algorithm, we develope a GPU management module. This module enables users to precisely allocate multiple generators across specified GPUs through parameter configurations.

In addition to open-source models, Generator modular includes `OpenaiModel`, supporting closed-source LLMs such as OpenAI. Future developments will extend support to other closed-source LLMs including Claude, Gemini and Azure.

2.1.4 Instruction Lab

Instruction has a significant impact on the quality of output generated by LLMs(Schulhoff et al., 2024). However, in frameworks such as LlamaIndex and LangChain, many key instructions lack transparency, being encapsulated at lower levels of the architecture. This encapsulation makes it challenging for users to modify. We find that the majority of published works developing their RAG algorithms utilize unaligned instructions, rendering experimental results across different studies incomparable.

To address these issues, RAGLAB designs the Instruction Lab module, which includes three key components: System Instruction, Task Instruction, and Algorithm Instruction. This module allows users to efficiently import and combine desired prompts from 3 instruction pools. Furthermore, users can adjust parameters within the configuration settings, facilitating comparative experiments using different instructions.

2.1.5 Trainer

RAGLAB integrates Accelerate(Gugger et al., 2022) and DeepSpeed libraries to provide comprehensive and efficient fine-tuning capabilities. Additionally, the Trainer module supports Low-Rank Adaptation (LoRA) and Quantized LoRA (QLoRA) (Dettmers et al., 2023) techniques, enabling users to fine-tune models with 70 billion parameters or more with limited computational resources.

We find that recent studies explore a novel

```python
from RAGLAB.rag.infer_alg import SelfRag_Reproduction
from utils import get_config()

args = get_config()
query = "What is Henry Feilden's occupation?" # query for interaction mode
Rag = SelfRag_Reproduction(args)

# interact mode
inference_result, generation_track = rag.inference(query, mode = 'interact')
print(inference_result)
# evaluation mode
evaluation_result = rag.inference(mode = 'evaluation')
print(evaluation_result)
```

Figure 2: A script that uses RAGLAB for reproducing Self-RAG algorithm.

method: adding special tokens during the generator training process to enhance performance. To facilitate the reproduction of these published works, the Trainer module supports adding special tokens during the fine-tuning phase.

2.1.6 Dataset and Metric

As shown in Table 2, following a comprehensive investigation, RAGLAB collects 10 widely used benchmarks encompassing five distinct tasks.

Table 2: Tasks and Datasets.

Task Type	Benchmarks
OpenQA	PopQA (Mallen et al., 2023) TriviaQA (Joshi et al., 2017)
Multi-HopQA	HotpotQA (Yang et al., 2018) 2WikiMultiHopQA (Ho et al., 2020)
Multiple-Choice	ArcChallenge (Clark et al., 2018) MMLU (Hendrycks et al., 2021)
Fact Verification	PubHealth (Zhang et al., 2023) StrategyQA (Geva et al., 2021) FactScore (Min et al., 2023)
Long-Form QA	ASQA (Stelmakh et al., 2022) FactScore (Min et al., 2023)

RAGLab implements a flexible data adaptation mechanism by individually mapping keys for each dataset, addressing the variability in raw data structures across different datasets. This approach enables users to easily extend new datasets by inheriting from existing dataset classes.

RAGLAB provides 3 classic metrics and 2 advanced metrics. Classic metrics include accuracy, exact match, and F1 score. Advanced Metrics include Factscore(Min et al., 2023) and ALCE(Gao et al., 2023). More specifically, FactScore represents an advanced metric evaluating the factual accuracy of long-form generation, while ALCE serves as a benchmark for assessing the citation accuracy and recall of RAG systems. Additionally, ALCE integrates other metrics, including ROUGE-L, MAUVE, str-em, and str-hit.

2.2 Architecture and Development Guide

RAGLAB reproduces six published RAG algorithms, encompassing Naive RAG, RRR, ITER-RETGEN, Self-ASK, Active RAG, and Self-RAG.These algorithms share numerous similarities, and each advanced RAG algorithm essentially represents an improvement upon Naive RAG.

```python
from raglab.rag.infer_alg import NaiveRag

class NewAlgorithm(NaiveRag):

    def __init__(self, args):
        super().__init__(args)

    def init(self,args):
        # customized  parameters if need
        # customized components if need

    def infer(self, query:str):
        #  pre build components:
        '''
        self.find_instruction() -> instructions
        self.retrieval.search() -> query
        self.llm.generate() -> output
        ect.
        '''

         # algorithm inference logic

        return output_text
```

Figure 3: Demostriction of developing new RAG algorithms in RAGALB.

The design philosophy of RAGALB draws inspiration from the HuggingFace Transformer library. Users only need to define their model from the Transformer library, after which they can employ the `generate()` method for inference. RAGALB implements each RAG algorithm as a distinct class. Two critical methods in each algorithm class are `init()` and `infer()`. The `init()` method serves to set parameters and load Generators, while the

Table 3: Performance comparison of various RAG algorithms using Llama3-8B as base language model.

Method	PopQA		TriviaQA		HopPopQA		WikiMultiHop		ARC		MMLU		PubHealth		StrategyQA		Factscore	ASQA
	ACC	F1	ACC	F1	ACC	F1	ACC	F1	ACC	F1	ACC	F1	ACC	F1	ACC	F1	factscore	str-em
Direct	25.6	16.3	68	59.8	20.6	23.8	24.8	25.7	77.4	57.2	58	42.8	76	76	56.4	56.4	55.1	25.4
NaiveRag	38.8	22.2	64.8	50.7	28.2	25.9	18	22.8	69.2	50.9	50.8	37.1	66.2	66.2	58.2	58.2	83.7	23.8
RRR	38.4	22.9	58.4	45.5	21.6	20.3	21.4	22.4	69.6	50.8	50	38	65.4	65.4	57.6	57.6	84.0	23.6
ITER-RETGEN	34.8	26.5	65.4	50.4	28.8	26.9	19.8	24.6	68.8	52.4	52.6	39.3	44.2	44.2	57	57	81.4	23.1
Active Rag	34.6	24.2	64.2	48.8	28.8	26.6	16	22.9	66.2	49.6	52	39.2	41.8	41.8	56.8	56.8	82.7	23.5
Self Ask	11.8	10.7	35.6	27.3	16.2	19.6	15.2	19.4	55.6	38.8	45.4	32.7	37.6	37.07	50.4	50.4	49.3	13.4
Self-RAG																		
always retrieval	38	12.4	61.8	29	23	15.8	18.6	11.6	58.6	41.6	39.2	25	67.8	67.8	48	45.8	70.6	32.3
adaptive retrieval	35.6	11.2	56.4	27.1	21	14.4	20.4	13.5	58	42.6	39.2	26.2	67.8	67.8	46.4	41	67.0	23.6
no retrieval	14.8	6.7	31.4	13.2	11.2	6.4	21	13.33	58	42.6	39.6	26.2	68.4	68.4	47.2	10.2	29.0	7.6

`infer()` method implements the algorithm's inference process. Based on this design framework, users can develop new algorithms through a few simple steps, as shown in Figure 3: (1) Define a `NewMethod()` class that inherits from `NaiveRAG`. (2) Add necessary parameters and components for the new algorithm by overriding the `init()` method. (3) Implement the new algorithm's inference logic by overriding the `infer()` method, utilizing the framework's provided components.

Algorithms inheriting from NaiveRAG can reuse the `inference()` method and all utility functions. Notably, the `inference()` method already provides automatic evaluation and interaction functionalities. This design enables researchers to focus solely on designing the `infer()` method to develop new algorithms. Section 2.3 will provide a detailed explanation of how to utilize the developed algorithm with just five lines of code.

2.3 Example Script

RAGLAB provides a user-friendly interface, allowing users to reproduce RAG algorithms for interaction or evaluation with just five lines of code. In Figure 2, we present an example script for reproducing the Self-RAG algorithm in both interaction and evaluation modes. The implementation process is as follows: (1) The `get_config()` function reads parameters from a YAML file and defines the args object. (2) The `SelfRag_Reproduction` class is defined to prepare all settings for the Self-RAG algorithm, based on the args parameters. (3) The `inference()` method in line 9 is called for the interaction mode. (4) The `inference()` method in line 12 is called again for the evaluation mode.

3 Experiment

One main aim of RAGLAB is to facilitate fair comparisons among various advanced RAG algorithms. To this end, we conducted comprehensive experiments by employing three distinct base models as generators while maintaining consistency across other fundamental components.

Experimental 1 Generator: In Experiment 1 , we select Llama3-8B as the base language model. We utilize the open-source data provided by Self-RAG as training data. The resulting fine-tuned model is designated as selfrag-llama3-8B, which serves as the generator for the Self-RAG algorithm. To ensure a fair comparison, we removed all special tokens from the training data, then full-weighted fine-tuned another model named Llama3-8B-baseline as the generator for other algorithms. For detailed training parameters, please refer to Appendix A.

Experimental 2 Generator: In Experiment 2, we selected Llama3-70B as the base language model. We select the QLoRA(Dettmers et al., 2023) method to fine-tune selfrag-llama3-70B and Llama3-70B-baseline. We use the same training data as in Experiment 1. For detailed training parameters, please refer to Appendix C.

Experimental 3 Generator: In Experiment 3, we selected GPT3.5 as base model. Additionally, we excluded the Self-RAG algorithm. Because closed-source models are not allowed to add special tokens during the training phase.

Additional Experimental Settings: We employ ColBERT as the retriever, utilizing Wikipedia 2018 as the external knowledge database. Local models are loaded with float16 precision, and during inference, we fix the random seed and use greedy sampling. The number of retrieved passages and the maximum generation length vary for each benchmark, please refer to Appendix B. We strive to maintain consistent instructions across all algorithms. For specific instructions and parameter settings, please refer to Appendix E and D, respectively. We select 10 comprehensive benchmarks for evaluation experiments, as detailed in Table 2.

Table 4: Performance comparison of various RAG algorithms using Llama3-70B as base language model.

Method	PopQA		TriviaQA		HopPopQA		WikiMultiHop		ARC		MMLU		PubHealth		StrategyQA		Factscore	ASQA
	ACC	F1	ACC	F1	ACC	F1	ACC	F1	ACC	F1	ACC	F1	ACC	F1	ACC	F1	factscore	str-em
Direct	25.6	24.7	76.4	75.6	27.8	38.3	28.2	34.8	90.4	68.8	73.4	55.6	77.2	77.2	70.6	70.6	70.5	31.99
NaiveRag	39.6	39	73.6	74.2	33.8	44	28.2	38.6	89.4	67.2	70.6	52.8	75.2	75.2	63.6	64	84.8	27.6
RRR	39	39.4	72.8	73.9	31.4	41.1	27.6	38.6	88.4	66.6	72.6	54.6	74.4	74.4	62.6	64	85.2	26.91
ITER-RETGEN	36.2	40.6	74.4	75.4	33.6	46.5	26.4	35.9	89.4	67.8	72.4	54.2	62.6	62.6	59.2	59.4	83.9	25.32
Active Rag	37	40	73.6	74.7	33.2	43.6	26.6	36.7	89.2	67.4	71.8	54.6	58	58	61	61	83.7	25.96
Self Ask	20.8	23.6	65.8	66.6	33.4	42.9	35	37	80.4	57.4	67.4	48.5	60.4	59.1	49.6	55.1	73.6	24.24
Self-RAG																		
always retrieval	45.2	16.8	77.6	43.4	40.6	26	38	22.8	89.4	68	72.8	55.6	79.4	79.4	68	71.4	84	45.96
adaptive retrieval	48.8	17.1	77.4	43.1	40.6	26	38.2	22.5	90	68.4	72.4	55	79.4	79.4	68	71.2	77.1	39.84
no retrieval	30	11.6	76.6	31.9	30.8	15.5	31	17.2	90	68.4	72.6	55	80.4	80.4	69.4	69.8	65.0	29.96

Table 5: Performance comparison of various RAG algorithms using GPT3.5 as base language model.

Method	PopQA		TriviaQA		HopPopQA		WikiMultiHop		ARC		MMLU		PubHealth		StrategyQA		Factscore	ASQA
	ACC	F1	ACC	F1	ACC	F1	ACC	F1	ACC	F1	ACC	F1	ACC	F1	ACC	F1	factscore	str-em
Direct	26.6	13.22	77	52.86	33.8	24.04	38	21.31	79.6	21.15	63.6	17.49	78	78	68.2	68.2	79.3	37.5
NaiveRag	45	17.16	72.8	26.47	41.6	17.74	33.2	16.44	67.4	15.94	54.4	10.83	53.8	53.8	61.8	61.8	84.5	39.1
RRR	46.2	17.71	73.6	27.45	37.2	16.34	33	16.43	68.4	16.02	54.4	11.01	54.8	54.8	63.6	63.6	84.5	39
ITER-RETGEN	44.2	16.75	73	26.02	44.8	18.92	34.6	16.31	69.8	16.95	55	11.32	39.2	39.2	56.2	56.2	84.2	39.6
Active Rag	44.2	17.34	72.8	27.45	43.8	18.76	34.2	16.94	70	21.57	55.8	13.38	50	50	61.2	61.2	83.7	33.7
Self Ask	38.2	18.89	68.6	38.12	36.4	25.52	41.6	23.52	63.4	14.71	48.6	10.16	45.2	45.01	39.6	39.43	86.7	30.2

Due to limited computation resources, we sequentially sampled 500 data points from each dataset. For evaluation, we employ a range of metrics, including Factscore, ACLE, accuracy (ACC), and F1 score across different datasets. The task-specific instructions for each dataset are detailed in Appendix F. The results of Experiments 1, 2, and 3 are presented in Tables 3, 4, and 5, respectively.

4 Experimental Result and Discussion

After analyzing the results from Experiments 1, Experiments 2, and Experiments 3, we find several valuable insights. When utilizing selfrag-llama3-8B as the generator for the Self-RAG algorithm, its performance across 10 benchmarks did not significantly surpass other RAG algorithms. However, when employing selfrag-llama3-70B as the generator, the Self-RAG algorithm significantly outperformed others in 10 benchmarks. We also find that Naive RAG, RRR, Iter-RETGEN, and Active RAG demonstrate comparable performance across 10 datasets. Notably, the ITER-RETGEN algorithm exhibits superior performance in MultiHopQA tasks. Furthermore, our findings indicate that RAG systems underperform compared to direct LLMs in multiple-choice question tasks. This conclusion aligns with experimental results from other studies(Chan et al., 2024; Asai et al., 2024; Wang et al., 2024). A possible explanation for this phenomenon is that multiple-choice questions include both the question and candidate answers. Additional retrieved information may mislead the generator.

5 Human Evaluation of RAGLAB

To comprehensively evaluate the user experience of the RAGLAB library, we implemented a user study. We developed a questionnaire comprising 12 questions, as shown in Figure 6. The study participants consisted of 20 NLP researchers, each having utilized RAGLAB at least three days. The questionnaire was administered offline, achieving a 100% response rate. The results indicated that 85% of respondents perceived RAGLAB as significantly enhancing their research efficiency, and 90% expressed willingness to recommend RABLAB to other researchers. Additionally, we gathered valuable suggestions for improvement, which will guide future system development.

6 Conclusion

We introduced RAGLAB, an efficient and user-friendly library for the fair comparison of RAG algorithms. With RAGLAB, researchers can easily conduct fair comparisons of existing RAG algorithms and develop new algorithms. We also conducted a fair comparison of 6 widely used RAG algorithms across 10 benchmarks, finding several valuable insights. We believe RAGLAB will become an essential research tool for the NLP community.

Limitations

RAGLAB provides a comprehensive and fair comparison of various RAG algorithms performance. However, there remain potential improvement in future work.

Due to limited computational resources, RAGLAB currently encompasses only 6 algorithms and 10 widely used benchmarks. However, there remains a need to include more algorithms and datasets. In future work, we will continue to follow the latest research developments and incorporate novel algorithms for fair comparison.

During the implementation process, we found that different retriever models and external knowledge databases significantly impact the performance of RAG algorithms. Due to limited computational resources, we only processed Wikipedia 2018 and Wikipedia 2023. In future work, we plan to include a wider variety of knowledge databases and conduct experiments on how different retriever models and external knowledge databases influence the performance of RAG algorithms.

Current research primarily focuses on improving the performance of algorithms, lacking a comprehensive evaluation of resource consumption and inference latency. At present, RAGLAB incorporates only 3 classic metrics and 2 advanced metrics. In future work, we aim to expand our evaluation framework to include a more diverse range of metrics.

We encourage the open-source community to address these limitations together. Our objective is to continually refine the RAGLAB framework, aiming to provide the most efficient and reliable evaluation platform and development tools.

References

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2024. Self-RAG: Learning to retrieve, generate, and critique through self-reflection. In *The Twelfth International Conference on Learning Representations*.

Chi-Min Chan, Chunpu Xu, Ruibin Yuan, Hongyin Luo, Wei Xue, Yike Guo, and Jie Fu. 2024. Rq-rag: Learning to refine queries for retrieval augmented generation. *arXiv preprint arXiv:2404.00610*.

Harrison Chase. 2022. Langchain. https://github.com/langchain-ai/langchain.

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. 2017. Reading Wikipedia to answer open-domain questions. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1870–1879, Vancouver, Canada. Association for Computational Linguistics.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: Efficient finetuning of quantized LLMs. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Tianyu Gao, Howard Yen, Jiatong Yu, and Danqi Chen. 2023. Enabling large language models to generate text with citations. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 6465–6488, Singapore. Association for Computational Linguistics.

Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. 2021. Did Aristotle Use a Laptop? A Question Answering Benchmark with Implicit Reasoning Strategies. *Transactions of the Association for Computational Linguistics (TACL)*.

Sylvain Gugger, Lysandre Debut, Thomas Wolf, Philipp Schmid, Zachary Mueller, Sourab Mangrulkar, Marc Sun, and Benjamin Bossan. 2022. Accelerate: Training and inference at scale made simple, efficient and adaptable. https://github.com/huggingface/accelerate.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. 2020. Retrieval augmented language model pre-training. In *International conference on machine learning*, pages 3929–3938. PMLR.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. In *International Conference on Learning Representations*.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. 2020. Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6609–6625, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Yasuto Hoshi, Daisuke Miyashita, Youyang Ng, Kento Tatsuno, Yasuhiro Morioka, Osamu Torii, and Jun Deguchi. 2023. Ralle: A framework for developing and evaluating retrieval-augmented large language models.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. 2021. Unsupervised dense information retrieval with contrastive learning.

Peter Izsak, Moshe Berchansky, Daniel Fleischer, and Ronen Laperdon. 2023. fastrag: Efficient retrieval augmentation and generation framework. `https://github.com/IntelLabs/fastRAG`.

Bwook Kim Jeffrey Kim. 2024. Autorag. `https://github.com/Marker-Inc-Korea/AutoRAG`.

Zhengbao Jiang, Frank Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Active retrieval augmented generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7969–7992, Singapore. Association for Computational Linguistics.

Jiajie Jin, Yutao Zhu, Xinyu Yang, Chenghao Zhang, and Zhicheng Dou. 2024. Flashrag: A modular toolkit for efficient retrieval-augmented generation research. *CoRR*, abs/2405.13576.

Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. 2017. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1601–1611, Vancouver, Canada. Association for Computational Linguistics.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781, Online. Association for Computational Linguistics.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474. Curran Associates, Inc.

Jerry Liu. 2022. LlamaIndex.

Xinbei Ma, Yeyun Gong, Pengcheng He, hai zhao, and Nan Duan. 2023. Query rewriting in retrieval-augmented large language models. In *The 2023 Conference on Empirical Methods in Natural Language Processing*.

Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Daniel Khashabi, and Hannaneh Hajishirzi. 2023. When not to trust language models: Investigating effectiveness of parametric and non-parametric memories. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9802–9822, Toronto, Canada. Association for Computational Linguistics.

Sewon Min, Kalpesh Krishna, Xinxi Lyu, Mike Lewis, Wen-tau Yih, Pang Koh, Mohit Iyyer, Luke Zettlemoyer, and Hannaneh Hajishirzi. 2023. FActScore: Fine-grained atomic evaluation of factual precision in long form text generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 12076–12100, Singapore. Association for Computational Linguistics.

Malte Pietsch, Timo Möller, Bogdan Kostic, Julian Risch, Massimiliano Pippi, Mayank Jobanputra, Sara Zanzottera, Silvano Cerza, Vladimir Blagojevic, Thomas Stadelmann, et al. 2019. Haystack: the end-to-end nlp framework for pragmatic builders.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah Smith, and Mike Lewis. 2023. Measuring and narrowing the compositionality gap in language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 5687–5711, Singapore. Association for Computational Linguistics.

Keshav Santhanam, Omar Khattab, Jon Saad-Falcon, Christopher Potts, and Matei Zaharia. 2022. ColBERTv2: Effective and efficient retrieval via lightweight late interaction. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3715–3734, Seattle, United States. Association for Computational Linguistics.

Sander Schulhoff, Michael Ilie, Nishant Balepur, Konstantine Kahadze, Amanda Liu, Chenglei Si, Yinheng Li, Aayush Gupta, HyoJung Han, Sevien Schulhoff, Pranav Sandeep Dulepet, Saurav Vidyadhara,

Dayeon Ki, Sweta Agrawal, Chau Pham, Gerson Kroiz, Feileen Li, Hudson Tao, Ashay Srivastava, Hevander Da Costa, Saloni Gupta, Megan L. Rogers, Inna Goncearenco, Giuseppe Sarli, Igor Galynker, Denis Peskoff, Marine Carpuat, Jules White, Shyamal Anadkat, Alexander Hoyle, and Philip Resnik. 2024. The prompt report: A systematic survey of prompting techniques.

Zhihong Shao, Yeyun Gong, Yelong Shen, Minlie Huang, Nan Duan, and Weizhu Chen. 2023. Enhancing retrieval-augmented large language models with iterative retrieval-generation synergy. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 9248–9274, Singapore. Association for Computational Linguistics.

Ivan Stelmakh, Yi Luan, Bhuwan Dhingra, and Ming-Wei Chang. 2022. ASQA: Factoid questions meet long-form answers. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 8273–8288, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Haoyu Wang, Tuo Zhao, and Jing Gao. 2024. Blendfilter: Advancing retrieval-augmented large language models via query generation blending and knowledge filtering. *arXiv preprint arXiv:2402.11129*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Max Woolf. 2023. The problem with langchain.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380, Brussels, Belgium. Association for Computational Linguistics.

Xiao Yu, Yunan Lu, and Zhou Yu. 2024. Localrqa: From generating data to locally training, testing, and deploying retrieval-augmented qa systems. *arXiv preprint arXiv:2403.00982*.

Tianhua Zhang, Hongyin Luo, Yung-Sung Chuang, Wei Fang, Luc Gaitskell, Thomas Hartvigsen, Xixin Wu, Danny Fox, Helen Meng, and James Glass. 2023. Interpretable unified language checking. *arXiv preprint arXiv:2304.03728*.

A Training Parameters for Llama3-8B

This appendix outlines the key training parameters used for fine-tuning the Llama3-8B model in our experiments. We employed full-weight fine-tuning on the Llama3-8B base model. The maximum sequence length was set to 4096 tokens, with a learning rate of 2e-5 and training conducted for 1 epoch. For a comprehensive list of training parameters, including computational resources and optimization settings, please refer to Table 6.

Table 6: Training Parameters for Llama3-8B.

Parameter	Value
Model	Llama3-8B
Fine-tuning method	Full weight
Number of GPUs	8
Total batch size	32
Batch size per GPU	1
Gradient accumulation steps	4
Mixed precision	bf16
Maximum sequence length	4096
Learning rate	2e-5
Learning rate scheduler	Linear
Warmup ratio	0.03
Weight decay	0
Number of epochs	1
DeepSpeed ZeRO stage	3

B Inference Parameters for Different Benchmarks

The number of retrieved passages and the maximum generation length were adjusted for each benchmark to accommodate their specific requirements. Table 7 presents a comprehensive overview of these parameters across various benchmarks.

Table 7: Inference Parameters for Different Benchmarks.

Benchmark	Precision	Max Length	N Docs
PopQA	float16	300	10
TriviaQA	float16	300	10
HotpotQA	float16	300	10
2WikiMultiHopQA	float16	300	10
Arc	float16	50	10
PubHealth	float16	50	10
MMLU	float16	50	10
StrategyQA	float16	300	10
Factscore	float16	300	5
ASQA	float16	300	5

C Training Parameters for Llama3-70B

We employed QLoRA fine-tuning on the Llama3-70B base model with a 4-bit quantization. The maximum sequence length was set to 4096 tokens, with a learning rate of 2e-5 and training conducted for 1 epoch. For the LoRA configuration, we used a rank of 64, an alpha of 16, and a dropout of 0.1. These parameters were consistently applied for both the self-rag-llama3-70B and Llama3-70B-baseline models. The training data remained the same as in Experiment 1. For a comprehensive list of training parameters, please refer to Table 8.

Table 8: Training Parameters for Llama3-70B using QLoRA.

Parameter	Value
Model	Llama3-70B
Fine-tuning method	QLoRA
Total batch size	32
Batch size per GPU	1
Gradient accumulation steps	4
Mixed precision	bf16
Maximum sequence length	4096
Learning rate	2e-5
Learning rate scheduler	Linear
Warmup ratio	0.03
Weight decay	0
Number of epochs	1
Quantization	4-bit
Quantization type	fp4
LoRA rank	64
LoRA alpha	16
LoRA dropout	0.1

D Configuration details for RAG methods

This appendix provides detailed configuration information for the various Retrieval-Augmented Generation (RAG) methods employed in our experiments. All algorithm parameters were set according to the optimal values reported in their respective original papers to ensure fair comparison and optimal performance.

Table 9: Configuration details for RAG methods.

Method	Parameter	Value
Self-RAG	beam_width	2
	max_depth	7
	w_rel	1.0
	w_sup	1.0
	w_use	0.5
	threshold	0.2
Active RAG	filter_prob	0.8
	masked_prob	0.4
	Query formulation	Implicit
ITER-RETGEN	max_iteration	3
Self Ask	max_iteration	5

E Algorithm Instructions

Naive RAG
read process insruction
"### Instruction:\n {task_instrucion} \n## Input:\n\n{query}\n\n Now, based on the following passages and your knowledge, please answer the question more succinctly and professionally. ### Background Knowledge:\n {passages} \n\n### Response:\n"

RRR
rewrite process instruction
"Provide a better search query for Wikipedia to answer the given question, end the query with '**'. \n\n Question: Ezzard Charles was a world champion in which sport? \n\n Query: Ezzard Charles champion** \n\n Question: What is the correct name of laughing gas? \n\n Query: laughing gas name** \n\n Question: {query} \n\n Query: "
read process insruction
"### Instruction:\n {task_instrucion} \n## Input:\n\n{query}\n\n Now, based on the following passages and your knowledge, please answer the question more succinctly and professionally. ### Background Knowledge:\n {passages} \n\n### Response:\n"

ITER-RETGEN
read process insruction
"### Instruction:\n {task_instrucion} \n## Input:\n\n{query}\n\n Now, based on the following passages and your knowledge, please answer the question more succinctly and professionally. ### Background Knowledge:\n {passages} \n\n### Response:\n"

Self ASK
follow up question instruction
"Question: When does monsoon season end in the state the area code 575 is located? Are follow up questions needed here: Yes. Follow up: Which state is the area code 575 located in? Intermediate answer: The area code 575 is located in New Mexico. Follow up: When does monsoon season end in New Mexico? Intermediate answer: Monsoon season in New Mexico typically ends in mid-September. So the final answer is: mid-September. \n{query} Are follow up questions needed here:"
read process insruction
"### Instruction:\n {task_instrucion} \n## Input:\n\n{query}\n\n Now, based on the following passages and your knowledge, please answer the question more succinctly and professionally. ### Background Knowledge:\n {passages} \n\n### Response:\n"

Active RAG
read process insruction
"### Instruction:\n {task_instrucion} \n## Input:\n\n{query}\n\n Now, based on the following passages and your knowledge, please answer the question more succinctly and professionally. ### Background Knowledge:\n {passages} \n\n### Response:\n"

Self-RAG
read process instruction
"### Instruction:\n{task_instruction}\n\n## Input:\n\n{query}\n\n### Response:\n"

Figure 4: Algorithm Instructions.

F Datasets Instructions.

Dataset-specific Instructions

PopQA
No special instruction

TriviaQA
No special instruction

HotPotQA
No special instruction

2WikiMultihopQA
No special instruction

ArcChallenge
Given four answer candidates, A, B, C and D, choose the best answer choice.

MMLU
Given four answer candidates, A, B, C and D, choose the best answer choice.

PubHealth
Is the following statement correct or not? Say true if it's correct; otherwise say false.

StrategyQA
You are only allowed to answer True or False, and generating other types of responses is prohibited.

Factscore
No special instruction

ASQA
Answer the following question. The question may be ambiguous and have multiple correct answers, and in that case, you have to provide a long-form answer including all correct answers.

Figure 5: Datasets Instructions.

G User Evaluation Questionnaire

RAGLAB System User Evaluation Questionnaire

1. What is your primary purpose for using the RAGLAB system?
A. Reproducing existing RAG methods B. Developing new RAG algorithms
C. Fair comparison platform D. Other (please specify): _____

2. Which of the following features do you find most useful when using the RAGLAB system? (Multiple selections allowed)
A. Modular RAG framework B. Pre-implemented advanced RAG algorithms
C. Comprehensive benchmark datasets D. Auxiliary preprocessing scripts
E. Standardized evaluation metrics

3. Please rate the ease of use of the RAGLAB system (1-5 points, 1 being the lowest, 5 being the highest):
A. 1 B. 2 C. 3 D. 4 E. 5

4. To what extent do you think the RAGLAB system has improved your research efficiency?
A. Significantly improved B. Slightly improved C. No change D. Slightly decreased
E. Significantly decreased

5. What challenges did you encounter when using RAGLAB to reproduce existing RAG methods?

6. Which components of the RAGLAB system were most helpful for your research?
A. Generator B. Retriever C. Instruction lab D. Trainer E. Corpus F. Metric
7. Did you use the preprocessed datasets and corpora provided by RAGLAB? If so, how did they help your research?

8. Did you encounter any performance issues or bugs while using the RAGLAB system? Please describe.

9. Compared to other RAG toolkits (such as LangChain, LlamaIndex, etc.), what advantages do you think RAGLAB has?

10. What suggestions do you have for improving the RAGLAB system?

11. Would you be willing to recommend the RAGLAB system to other researchers?
A. Very willing B. Might C. Unsure D. Probably not E. Definitely not

12. Overall, how satisfied are you with the RAGLAB system? (1-10 points, 1 being the lowest, 10 being the highest)
A. 1 B. 2 C. 3 D. 4 E. 5 F. 6 G. 7 H. 8 I. 9 J. 10

Figure 6: RAGLAB System User Evaluation Questionnaire