FrugalRAG: Learning to retrieve and reason for multi-hop QA

Abhinav Java [1] **Srivathsan Koundinyan** [1] **Nagarajan Natarajan** [1] **Amit Sharma** [1]

Abstract

We consider the problem of answering complex questions, given access to a large unstructured document corpus. The de facto approach to solving the problem is to leverage language models that (iteratively) retrieve and reason through the retrieved documents, until the model has sufficient information to generate an answer. Attempts at improving this approach focus on retrieval-augmented generation (RAG) metrics such as accuracy and recall and can be categorized into two types: (a) fine-tuning on large question answering (QA) datasets augmented with chain-of-thought traces, and (b) leveraging RL-based fine-tuning techniques that rely on question-document relevance signals. However, efficiency in the number of retrieval searches is an equally important metric, which has received less attention. In this work, we show that: (1) Large-scale fine-tuning is not needed to improve RAG metrics, contrary to popular claims in recent literature. Specifically, a standard ReAct pipeline with improved prompts can outperform state-of-the-art methods on benchmarks such as HotPotQA. (2) Supervised and RL-based fine-tuning can help RAG from the perspective of frugality, i.e., the latency due to number of searches at inference time. For example, we show that we can achieve competitive RAG metrics at nearly half the cost (in terms of number of searches) on popular RAG benchmarks, using the same base model, and at a small training cost (1000 examples).

1. Introduction

We study the problem of answering questions, such as "*Can a microwave melt Toyota Prius battery*?", given access to a large unstructured corpus like Wikipedia. The de facto approach to solving the problem is to use language models (LMs) coupled with the ability to retrieve relevant documents (e.g.,, Wiki passages) against queries, i.e.,, the retrieval-augmented generation (RAG) paradigm (Jeong et al., 2024; Jiang et al., 2023; Chan et al., 2024; Asai et al., 2023). However, answering complex questions often requires multi-hop reasoning and retrieval, i.e.,, the LM has to iteratively decompose the user utterance into sub-queries or search phrases (e.g.,, "*melting point of Toyota Prius battery*"), retrieve documents relevant to the sub-queries, and reason through the retrieved documents to issue further sub-queries, until the LM is able to generate an answer for the original query.

Most of recent work in this space focus exclusively on the accuracy of generated answers, using (a) supervised fine-tuning (SFT) techniques on large QA training datasets (Asai et al., 2023; Chan et al., 2024; Hsu et al., 2024), or (b) reinforcement-learning (RL) based techniques such as GRPO (Jin et al., 2025) and DPO (Hsu et al., 2024). In either case, they rely on tens or even hundreds of thousands of training data points, either in the form of <question, answer> demonstrations or as <question, documents> relevance signals. For instance, the most recent work SearchR1 (Jin et al., 2025) applies GRPO to fine-tune the Qwen-2.5-7B-Instruct (Yang et al., 2024) model using more than 100000 training examples on the popular HotPotQA (Yang et al., 2018) dataset. Similarly, another recent RAG framework called LeReT (Hsu et al., 2024) uses more than 90000 training examples on the same dataset, using Llama3.1-8B (Grattafiori et al., 2024). However, in real-world QA settings, ground-truth labelled examples are difficult to obtain and latency of the system is critical for user experience. In this work, we challenge the claims in the recent literature in multiple ways:

(1) Efficiency, i.e.,, number of hops or searches needed to accurately answer, is equally important;

(2) The oft-overlooked simple baseline strategy of ReAct for iterative retrieval and reasoning (Yao et al., 2023; Shao et al., 2023), with optimized few-shot prompting, already is quite competitive to several recent techniques; the prompt optimization needs tens of examples, in stark contrast to aforementioned RAG techniques like Self-RAG and SearchR1 that need several orders of magnitude more number of examples, and

[*]Equal contribution [1]Microsoft Research, India. Correspondence to: Abhinav Java <java.abhinav99@gmail.com>, Amit Sharma <amshar@microsoft.com>.

Proceedings of the 42^{nd} International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025.

(3) We can leverage RL-based techniques like GRPO to further improve such strong baselines, from the perspective of efficiency where they often underperform, using *just 1000 training examples*.

Our solution, **FRUGALRAG**, is a two-stage framework that removes the need for large-scale labels while still achieving effective and efficient inference-time search. In the *first stage*, the model is trained to maximize evidence coverage by generating diverse and informative search queries across multiple hops. In the *second stage*, we post-train the model to decide when to stop retrieving and generate an answer. This decision is modeled explicitly, allowing the model to weigh the cost of further retrievals against the confidence in the retrieved evidence.

Optimizing for coverage and efficiency in a single stage leads to unstable training; we find that models either over-retrieve or stop too early. Our key insight is that learning when to stop is more naturally learned through reinforcement learning (RL) signals, whereas better coverage can be obtained by repeatedly issuing high quality search queries using frameworks such as ReAct. By separating the exploration stage from decision making, FRUGALRAG better aligns the learning signal with the multi-hop RAG task resulting in high quality and efficient retrievals.

We evaluate FRUGALRAG on standard benchmarks such as HotPotQA (Yang et al., 2018), 2WikiMultiHopQA (Ho et al., 2020) and MuSiQue (Trivedi et al., 2022b), using both document retrieval metrics such as recall and answer quality metrics. Compared to baselines, we show that FRUGALRAG obtains the highest document recall and answer quality while incurring a low number of search queries per question. In particular, on HotPotQA, FRUGALRAG using a 3B model obtains higher document recall with only two searches per question compared to methods such as LeReT and SearchR1 that are based on 8B and 7B models and include finetuning on thousands of examples.

2. FrugalRAG: Two-stage training framework

In this section, we describe FRUGALRAG, a novel framework for enhancing retrieval augmented generation in LMs by decoupling the evidence exploration stage from answer generation. FRUGALRAG demonstrates several key advantages over contemporary RAG approaches – (1) *Requires only 1000 annotated training examples* which is a 100 times reduction in dataset size compared to existing works (Jin et al., 2025; Hsu et al., 2024; Chan et al., 2024), (2) *Dynamically adapts test-time compute* which results in low inference time latency and high retrieval recall, unlike existing fixed compute methods (Hsu et al., 2024).

Setup. Let Q denote the user utterance which is a complex question requiring multiple iterative retrievals, and f denote an LM, that, at each hop, inspects the current context and decides on a next action. At hop h (with $1 \le h \le B$, where B is the total budget of allowed hops), f generates a thought–action pair (T_h, A_h): the action A_h includes generating a search query S_h that is passed to the retriever $\mathcal{R}(\cdot)$, yielding a set of documents $\mathcal{D}_h = \mathcal{R}(S_h)$. We write $\mathcal{D}_0$ for any initial context (either empty or containing $\mathcal{R}(Q)$), and the available context to f at hop h is the union of Q with all previously retrieved documents $\{\mathcal{D}_0, \ldots, \mathcal{D}_{h-1}\}$, and thought-action pairs $\{(T_0, A_0), \ldots, (T_{h-1}, A_{h-1})\}$. When the model generates FINISH, say at h_{term} hops (alternatively at B), we invoke a generator LM denoted as g, conditioning it on the original query Q, and the available context $\{\mathcal{D}_h, T_h, A_h\}_{h=0}^{h_{\text{term}}}$ to produce the final answer. In this setting, the challenge lies in having f iteratively craft highly targeted queries S_h so that the standard retriever $\mathcal{R}$ can surface the minimal collection of relevant documents needed to answer Q within the allocated budget.

FRUGALRAG framework requires access to only the ground truth documents (Y) during both its *training phases* and does not leverage the final answer annotation. These ground truth documents are essential for providing fine-grained feedback to the model. During *inference*, FRUGALRAG only leverages the question (Q), the document index ($\mathcal{I}$), and a trained retriever model ($\mathcal{R}$).

Our key observation is that with sufficient test-time compute, even a base model is capable of generating multiple, different search queries to help answer a given question, following a ReAct framework (e.g., see Section 5, Table 1, with base models such as Qwen-2.5-7B-Instruct) . Therefore, learning is required not to *scale* the compute at test-time as argued in (Jin et al., 2025; Chen et al., 2025), but to control it adaptively based on a question's difficulty. In the following subsections we discuss the two stages of our learning algorithm: *Stage 1*: Generating a base policy aimed at maximizing evidence coverage through exploration (Sec. 2.1); and *Stage 2*: Finetuning the policy using RL to control test-time compute (Sec. 2.2). We present our overall framework in Algorithm. 1.

2.1. Stage 1: Coverage Maximization (Explore)

Gathering evidence plays a crucial role in answering multi-hop questions, which often require iterative retrieval and reasoning across multiple sources. Drawing on recent advances in test-time scaling, we observe that we can boost evidence coverage (i.e., recall) simply by letting the model f issue multiple search queries S_h at inference time. This approach sidesteps the need for massive supervised fine-tuning—instead, it harnesses the model's own generated rollouts to gather, and then integrate additional information. In the next section, we describe how we construct our training data and design our fine-tuning protocol to fully leverage

this capability.

Training Dataset Generation. We choose ReAct (Yao et al., 2023) for generating rollouts. Here, a rollout is a set of outputs generated by the model f. In the standard ReAct setup, an off-the-shelf model f generates a thought-action pair (T_h, A_h) at each hop $h \in [1, B]$, where A_h is either a call to the retriever $\mathcal{R}$ or FINISH indicating the end of rollout. At each hop h, we generate samples $\{(T_h^1, A_h^1, S_h^1) \dots (T_h^n, A_h^n, S_h^n)\}$ using n bootstrapped prompts (Khattab et al., 2023) (See Appendix B). For each search query $S_h^i, i \in [1, n]$ we retrieve corresponding documents $\mathcal{D}_h^i = \mathcal{R}(S_h^i)$, then discard any documents already present in the context. We then compute recall against ground-truth labels and add the sample i that achieves the highest recall to the context for the next hop $h+1$. This dataset generation strategy is simple and easily parallelizable. We conduct two separate runs – the standard ReAct framework where f is allowed to generate FINISH, and the second where f can only call the retriever. Although the former is more efficient and finishes before B search hops, we observe that the latter yields a significantly higher overall recall owing to a greater number of retrievals. Unlike previous work (Chan et al., 2024; Asai et al., 2023; Hsu et al., 2024; Jin et al., 2025) that generated orders of magnitude more data, we only use 1000 questions to generate our dataset during this step.

Supervised "Exploration" Finetuning (FRUGALRAG-Explore). Although the base model f without FINISH maximizes exploration, we cannot use it directly for reinforcement learning because it does not include FINISH. Consequently, during fine-tuning, we sample rollouts from both configuration described above, 90% without FINISH and 10% with it. We want to use supervised finetuning to build a strong base-policy for RL, that prioritizes exploration while ensuring that FINISH remains in the model's generation distribution. Hence, we finetune the model f to obtain our base policy f_S. At each iteration, the model predicts the next (T_h, A_h, S_h) tuple given the rollout which comprises interleaved thought-action-search tuples and retrieved documents represented as an ordered set $\{(\mathcal{D}_0, T_0, A_0, S_0) \dots (\mathcal{D}_{h-1}, T_{h-1}, A_{h-1}, S_{h-1})\}$ till $h-1$, using standard cross-entropy error as the objective function. f_S has several notable advantages – **(1)** *off-the-shelf model f does not explore.* Despite prompt optimization, f is generally over-confident and predicts the answer without sufficient exploration. **(2)** *removing* FINISH *results in over-retrievals.* We observe that simply removing FINISH from the ReAct loop yields high recall even with the base model f, however, the model is forced to utilize the full budget for every question and cannot be post-trained for efficiency as it never generates a rollout with FINISH.

2.2. Stage 2: Controlling test-time compute with RL

Given a finetuned base policy model f_S, we propose a strategy that enables the model to generate extended rollouts only when required. This mechanism is crucial for inference-time efficiency, as it allows the model to adaptively determine the appropriate rollout length. Since f_S generally prioritizes exploration, our only goal is to *learn when to sufficient evidence has been gathered*, thereby reducing overall search latency during inference. Below we show how this problem can be formulated as a reinforcement learning task, as it requires evaluating and comparing different rollouts. However, unlike recent work using RL for scaling the number of retrievals (Jin et al., 2025), here our focus is to use RL to reduce the number of retrievals per question.

Reward Design. Our reward function is designed to guide the model towards the optimal rollout length. To achieve that, we first generate the entire rollout using f_S and then compute the reward. Let h^* denote the optimal rollout length (or hop), such that the subsequent retrievals do not improve the overall recall, denoted by c. If the model chooses to terminate at hop $h_{\text{term}} > h^*$, we penalize the additional wasted steps. Conversely, we also penalize the model if $h_{\text{term}} < h^*$ making sure it never loses its ability to explore. This ensures that model learns to explore adequately for complex questions while avoiding redundant retrievals. The reward is as follows:

$$\mathbf{R} = \begin{cases} \max(-R_{\max}, \min(\log \frac{1-\Delta}{\Delta}, 0)) - \beta, \\ \qquad \text{if } \Delta < 0,\ c > \tau \\ \max(-R_{\max}, \min(\log \frac{1-\Delta}{\Delta}, R_{\max})), \\ \qquad \text{if } \Delta > 0,\ c > \tau \\ R_{\max} + \alpha \cdot \frac{h^*}{B}, \\ \qquad \text{if } \Delta = 0,\ c > \tau \\ \max(-R_{\max}, \min(\log \frac{1-\Delta}{\Delta}, 0)), \\ \qquad \text{if } c < \tau \end{cases} \tag{1}$$

where $R_{\max}$ is the maximum possible reward, Δ is the difference $h_{\text{term}} - h^*$ normalized by maximum budget B, and β, α, τ tunable hyper-parameters which control the early penalty, correct bonus, and minimum recall threshold respectively. The early penalty β ensures that the model never under-explores, and the correct bonus is scaled by h^*/B (when $\Delta = 0$) giving higher reward to difficult rollouts. Conceptually, our reward function penalizes the policy in proportion to the magnitude of $|\Delta|$; as $|\Delta|$ decreases, the reward approaches the upper bound $R_{\max}$, with the maximum attained when the policy emits FINISH exactly at the optimal location h^* based on the ground truth.

In addition to Eq. 1, we incorporate format reward R_f following (Shao et al., 2024) to ensure f_S consistently adheres to ReAct style formatting. If the generated output deviates

from the expected format and results in no retrievals, we assign a reward of -1 for each such instances and compute the average across hops. Finally, our overall reward is the sum of R_f and $\mathbf{R}$, with range $[-R_{\max} - \beta - 1, R_{\max} + \alpha + 1]$.

Optimization. Motivated by the recent success and memory efficiency of GRPO (Shao et al., 2024), we adopt it as our optimization algorithm. At each hop h, we sample v tuples $\{T_h^i, A_h^i, S_h^i\}_{i=1}^v$, and retrieve their corresponding documents $\mathcal{D}_h^i$, and de-duplicate. We repeat this until we reach the maximum budget $h = B$, collecting sample tuples and documents. For each rollout i, we then compute a cumulative reward using Eq. 1: $R^i \leftarrow \mathbf{R}(\{\mathcal{D}_h^i\}_{h=1}^m, Y^j)$, and backpropagate through every logit produce by the policy along that rollout and mask any further generations once f_S emits the FINISH.

3. Experiments

Details of the experimental setup comprising benchmarks, metrics, and baselines are provided in the Appendix A. Additional training details are provided in Appendix B. Additional ablations and results are in Appendices F- G.

Main results on HotPotQA. In Tables 1 and 2, we compare FRUGALRAG against baselines on HotPotQA, using Qwen2.5-3B-Instruct and Qwen2.5-7B-Instruct as the base models, respectively. For fair comparison, all baselines use the ColBERTv2 (Santhanam et al., 2021) retriever indexed on Wikipedia. The key takeaway is that FRUGALRAG *consistently outperforms the baselines on both answer and retrieval metrics, using competitive or significantly smaller number of searches on average.*

The *ReAct Few Shot (FS) baseline* (Yao et al., 2023; Khattab et al., 2023) outperforms vanilla Retrieval Augmented Generation and ReAct without few shot prompts consistently. The optimized prompts enable ReAct FS to generate more search queries, and as a result achieve very strong performance on HotPotQA; **71.47%** and **77.65%** recall with Qwen2.5-3B-Instruct and Qwen2.5-7B-Instruct respectively. Note that it obtains almost the same as the recall reported by LeReT (77.1% using Llama3.1-8B) after significant fine-tuning, and comparable Exact Match (42.1) to the highest obtained Exact Match reported by recently proposed methods such as Search-R1 (43.3); signifying the importance of building a strong baseline (see Table 4).

Table 1 demonstrates the *effectiveness of* FRUGALRAG using Qwen2.5-3B-Instruct, achieving the highest score on all metrics. Specifically, compared to ReAct FS, FRUGALRAG improves F1 score by **+5.9** and recall by **+10.22**, while reducing latency by **1.58** (**3.33** $\rightarrow$ **2.10**) times. Similarly, Table 2 presents the accuracy metrics and retrieval metrics using Qwen2.5-7B-Instruct. FRUGALRAG achieves substantial improvements over the strongest baseline ReAct FS with similar number of searches, of **+5.16**, **+5.03**, and **+1.20** on F1, EM, and Match Scores respectively, while introducing a negligible increase in latency of **0.09** average searches. Finally, we note that FRUGALRAG-Explore is either best or second best in terms of both answer and recall metrics but introduces a very high latency compared to FRUGALRAG. Overall, our results highlight that our method strikes a significantly better efficiency-accuracy tradeoff, compared to strong baselines.

Table 1. Performance of FRUGALRAG vs. baselines on HotPotQA with Qwen2.5-3B-Instruct. We report answer-level (F1, EM, Match), retrieval-level (Recall, Sup. F1), and average search count.

Method	F1	EM	Match	Recall	Sup. F1	Searches
Naive	8.33	1.24	22.41	0	0	0
CoT	13.90	6.54	22.23	0	0	0
RAG (3)	35.29	21.85	43.29	58.56	66.29	1.0
RAG (5)	36.56	25.37	36.63	63.20	69.94	1.0
RAG (10)	35.27	23.30	37.73	67.92	73.67	1.0
ReAct	25.35	10.84	42.91	65.13	73.96	1.88
ReAct FS	50.88	38.02	47.65	71.47	78.64	3.33
ReAct FS Explore (B=2)	50.53	37.40	47.70	66.55	75.23	2.00
ReAct FS Explore (B=3)	52.64	42.70	52.10	72.30	78.64	3.00
ReAct FS Explore (B=4)	51.01	36.80	46.00	73.45	80.65	4.00
ReAct FS Explore (B=5)	51.77	39.00	47.80	75.40	81.50	5.00
ReAct FS Explore (B=6)	53.73	41.80	51.10	76.75	82.46	6.00
FRUGALRAG-Explore	53.17	40.56	49.65	81.41	85.72	5.90
FRUGALRAG	**56.78**	**43.59**	**52.69**	**81.69**	**86.01**	2.10

Table 2. Performance of FRUGALRAG vs. baselines on HotPotQA with Qwen2.5-7B-Instruct. We report answer-level (F1, EM, Match), retrieval-level (Recall, Sup. F1), and average search count.

Method	F1	EM	Match	Recall	Sup. F1	Search
Naive	13.63	5.49	24.30	0	0	0
CoT	23.64	14.07	24.96	0	0	0
RAG (ndocs=3)	35.67	22.20	44.01	58.56	66.29	1.0
RAG (ndocs=5)	36.53	22.70	46.54	63.20	69.94	1.0
RAG (ndocs=10)	37.62	23.28	47.49	65.13	73.96	1.0
ReAct	29.75	11.85	56.28	73.45	80.03	2.07
ReAct FS	53.59	42.10	53.20	77.65	82.80	2.91
ReAct FS Explore (B=2)	57.70	44.30	55.00	74.40	80.87	2.00
ReAct FS Explore (B=3)	59.82	46.10	57.00	78.50	83.74	3.00
ReAct FS Explore (B=4)	59.78	46.50	56.10	81.75	85.89	4.00
ReAct FS Explore (B=5)	57.90	44.10	55.00	81.20	85.68	5.00
ReAct FS Explore (B=6)	57.59	44.70	56.30	80.75	85.36	6.00
FRUGALRAG-Explore	**61.84**	47.67	**57.01**	**84.19**	**87.63**	5.88
FRUGALRAG	61.22	**47.87**	54.77	79.88	84.80	2.90

4. Conclusion

In this work, we argue that efficiency is an equally important metric to study in RAG solutions, besides the traditional RAG metrics such as retrieval performance and accuracy of the generated answers. We demonstrate that simple ReAct baseline that iteratively retrieves (by invoking the search tool) and reasons (to decide what search call to issue next) is quite competitive, especially if we can optimize its few-shot prompt using just tens of training examples. We pro-

pose a two-stage framework FRUGALRAG that a) works with 1000 training examples, compared to state-of-the-art RAG techniques that use over 100,000 examples, and b) yet achieves competitive accuracies while also using far fewer search queries at inference time, on popular multi-hop QA datasets.

References

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. Self-rag: Learning to retrieve, generate, and critique through self-reflection. In *The Twelfth International Conference on Learning Representations*, 2023.

S Borgeaud, A Mensch, J Hoffmann, T Cai, E Rutherford, K Millican, G Driessche, JB Lespiau, B Damoc, A Clark, et al. Improving language models by retrieving from trillions of tokens. arxiv. *arXiv preprint arXiv:2112.04426*, 2021.

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

Mingyang Chen, Tianpeng Li, Haoze Sun, Yijie Zhou, Chenzheng Zhu, Haofen Wang, Jeff Z Pan, Wen Zhang, Huajun Chen, Fan Yang, et al. Learning to reason with search for llms via reinforcement learning. *arXiv preprint arXiv:2503.19470*, 2025.

Jacob Devlin. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

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

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps. In Donia Scott, Nuria Bel, and Chengqing Zong, editors, *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6609–6625, Barcelona, Spain (Online), December 2020. International Committee on Computational Linguistics. doi: 10.18653/v1/2020.coling-main.580. URL https://aclanthology.org/2020.coling-main.580/.

Sheryl Hsu, Omar Khattab, Chelsea Finn, and Archit Sharma. Grounding by trying: Llms with reinforcement learning-enhanced retrieval. *arXiv preprint arXiv:2410.23214*, 2024.

Soyeong Jeong, Jinheon Baek, Sukmin Cho, Sung Ju Hwang, and Jong C Park. Adaptive-rag: Learning to adapt retrieval-augmented large language models through question complexity. *arXiv preprint arXiv:2403.14403*, 2024.

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 *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7969–7992, 2023.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick SH Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *EMNLP (1)*, pages 6769–6781, 2020.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, et al. Dspy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*, 2023.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466, 2019.

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. Retrieval-augmented generation for knowledge-intensive nlp tasks. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Niklas Muennighoff, Nouamane Tazi, Loïc Magne, and Nils Reimers. Mteb: Massive text embedding benchmark. *arXiv preprint arXiv:2210.07316*, 2022.

Niklas Muennighoff, SU Hongjin, Liang Wang, Nan Yang, Furu Wei, Tao Yu, Amanpreet Singh, and Douwe Kiela. Generative representational instruction tuning. In *ICLR 2024 Workshop: How Far Are We From AGI*, 2024.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 3505–3506, 2020.

Keshav Santhanam, Omar Khattab, Jon Saad-Falcon, Christopher Potts, and Matei Zaharia. Colbertv2: Effective and efficient retrieval via lightweight late interaction. *arXiv preprint arXiv:2112.01488*, 2021.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36: 68539–68551, 2023.

Zhihong Shao, Yeyun Gong, Yelong Shen, Minlie Huang, Nan Duan, and Weizhu Chen. Enhancing retrieval-augmented large language models with iterative retrieval-generation synergy. *arXiv preprint arXiv:2305.15294*, 2023.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Weijia Shi, Sewon Min, Michihiro Yasunaga, Minjoon Seo, Rich James, Mike Lewis, Luke Zettlemoyer, and Wen-tau Yih. Replug: Retrieval-augmented black-box language models. *arXiv preprint arXiv:2301.12652*, 2023.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. *arXiv preprint arXiv:2212.10509*, 2022a.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. musique: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022b.

Leandro von Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, Nathan Lambert, Shengyi Huang, Kashif Rasul, and Quentin Gallouédec. Trl: Transformer reinforcement learning. https://github.com/huggingface/trl, 2020.

Liang Wang, Haonan Chen, Nan Yang, Xiaolong Huang, Zhicheng Dou, and Furu Wei. Chain-of-retrieval augmented generation. *arXiv preprint arXiv:2501.14342*, 2025.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

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.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Livia Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs. *Advances in Neural Information Processing Systems*, 37:62557–62583, 2024.

A. Experimental Setup

Benchmarks. We conduct evaluation of FRUGALRAG using three widely adopted Multi-Hop RAG benchmarks– HotPotQA (Yang et al., 2018), 2WikiMultiHopQA (Ho et al., 2020), and MuSiQue (Trivedi et al., 2022b), under their full-wiki setting using a ColBERT-v2 (Santhanam et al., 2021) retriever index over Wikipedia passages provided in official datasets. These datasets emphasize the need for models to perform reasoning across multiple documents to arrive at an answer. The **HotPotQA** benchmark comprises of of 7405 development examples requiring supporting evidence from Wikipedia abstracts. We report the results on the entire dev-set to access the models ability to perform end-to-end reasoning and retrieval. For the **2WikiMultiHopQA** benchmark, we utilize 12576 development examples, each coupled with its supporting evidence and document title. This evaluates a model's ability to hop between structured and unstructured wikipedia text. Finally, for **MuSiQue**, we test on its 2405 development examples of 2-4 hop questions derived from a composition of two-hop queries. We provide details of the datasets and the index in Appendix D.

Metrics. While past work tends to focus on either recall of documents (Hsu et al., 2024) or final answer accuracy (Jin et al., 2025), we evaluate our methods on all three metrics: final answer, recall of retrievals, and efficiency. To access the final answer fidelity we report F1, EM (Exact Match), and Match score. **F1** score is the token level harmonic mean of precision and recall between predicted and ground truth string. **EM** requires the predicted string to exactly match the ground truth. **Match** requires the ground truth to be a substring of the generated answer. We also evaluate the alignment of supporting evidence with the retrieved documents by reporting **Recall**, **Support-F1** (Trivedi et al., 2022b), and **Latency** (number of searches). During both training and testing, we set the number of retrieved documents per query to 3, 5, and 5 for HotPotQA, 2WikiMultiHopQA, and MuSiQue. A detailed overview of metrics is presented in the Appendix C

Baselines. We evaluate FRUGALRAG by comparing it against no-retrieval and retrieval-based baselines. The no-retrieval baselines include naive generation and chain-of-thought (CoT) prompting (Wei et al., 2022). The retrieval-based baselines are vanilla retrieval-augmented generation (RAG) with CoT, ReAct (Yao et al., 2023), and optimized few-shot prompting with ReAct (Khattab et al., 2023). We also compare FRUGALRAG to recently proposed approaches that leverage large-scale fine-tuning, including Search-R1 (Jin et al., 2025) and LeReT (Hsu et al., 2024), where applicable. However, we note that these recent methods do not report all metrics that we consider and their metrics may not be directly comparable (due to varying model sizes). Therefore, we conduct comparisons using the closest model scales and overlapping subsets of evaluation metrics. Furthermore, we compare FRUGALRAG with our version of Search-R1 (Jin et al., 2025) by replicating their reward function consistent with our setup.

B. Training Details

We train FRUGALRAG in two stages using Qwen2.5-3B-Instruct and Qwen2.5-7B-Instruct as our base models. In both the supervised finetuning and reinforcement learning (RL) stages, we leverage the TRL library (von Werra et al., 2020). Algorithm 1 shows the overall training framework of FRUGALRAG. We plan to publicly release our code soon and have attached a copy of the codebase for review in the meantime. Below, we discuss each step along with their implementation details.

Few-Shot Prompt Optimization Details. We leverage DSPy (Khattab et al., 2023) for automatic few-shot prompt generation following LeReT (Hsu et al., 2024). Specifically, we use 50 training examples ($\mathcal{L}_{\text{init}}$) with the BOOTSTRAPFEWSHOTWITHRANDOMSEARCH method, which uses the LM f to generate few-shot examples, selecting the best performing ones for subsequent prompting. We select 4 best performing few-shot prompts from a total of 15 candidate sets using the sum of answer EM and answer passage match. Answer EM checks for an exact string-match between the generated and actual answer, and passage match checks if the actual answer is present in the retrieved passages. This step is crucial because it facilitates dataset generation using diverse rollouts and ensures the answer format is followed by the model. For this step, we serve our model on one A100 80GB GPU using SGLang (Zheng et al., 2024). For all experiments involving Qwen2.5, we utilize the 7B-Instruct variant for prompt optimization. The optimized prompts are then reused without modification for the 3B variant.

Dataset Generation Details. For each few-shot prompt p_i, the model f generates a tuple (T_h^i, A_h^i, S_h^i) representing a candidate output for the next hop. As described in Sec. 2.1, we evaluate all candidate tuples at hop h and select one with the highest recall. This selected candidate is then used as the context for the next hop and the process is repeated till budget B (optionally till the selected candidate action A_h indicates FINISH). We set the budget $B = 6$, where the initial retrieval step is always $\mathcal{R}(Q^{(j)})$ with $Q^{(j)}$ denoting the original user utterance. The generated dataset is denoted by $\mathbf{D}$. For all experiments involving Qwen2.5, we utilize the 7B-Instruct variant along with its prompts to generate the dataset. For each dataset, we prepare the Stage 1 finetuning dataset with 1000 randomly sampled examples from the corresponding training split. For further improving results, we can repeat few shot prompt optimization and dataset generation using different base models.

Supervised "Explore" Finetuning Details. Stage 1 (Sec. 2.1) consists of full-parameter finetuning for a single epoch, using a learning rate of 2×10^{-5} and a weight decay of 0.01 for all models and datasets. We choose a maximum sequence length of 4096 during finetuning and use the standard next token prediction loss given by:

$$\max_f \mathbb{E}_{(x,y)\sim \mathbf{D}} \log p_f(x|y) \tag{2}$$

where $y = (T_h, A_h, S_h)$ and $x = Q^{(j)} \cup \{(T_k, A_k, S_k, \mathcal{D}_k)\}_{k=0}^{h-1}$ sampled from the generated dataset $\mathbf{D}$.

We train the model f for 1 epoch using a batch size of 4 and apply gradient accumulation of 2 steps, resulting in an effective batch size of 8. Optimization is performed using AdamW (Loshchilov and Hutter, 2017) with a learning rate of 2×10^{-5}. We use a linear learning rate scheduler with a warmup phase of 20 steps. The training is performed using 8 H100 80GB GPUs and takes about 15 minutes.

Controlling test-time compute with RL. Our RL step employs GRPO for fine-tuning the base policy f_S. Specifically, following the notation in DeepSeekMath (Shao et al., 2024), for each question $Q^{(j)}$, we sample a group of outputs $\{o_h^1, o_h^2, \ldots, o_h^v\}$ at hop h, where v is set to 4. We optimize our base policy f_S using the standard GRPO objective using the cumulative rollout reward as defined in Eq. 1. We use a KL divergence penalty with weight 0.001, set the maximum reward $R_{\max} = 2.0$, and apply an early penalty $\beta = 3$. We set τ to 60%, 40%, and 30% for HotPotQA, 2Wiki, and MuSiQue datasets respectively to improve the diversity of rollouts. Generation is limited to a maximum of 256 completion tokens and the maximum prompt size is 2048. Training is conducted using DeepSpeed-Zero2 (Rasley et al., 2020) and 8 H100 GPUs with a learning rate of 10^{-6}. Due to the long prompt (which includes retrieved documents from previous hops), we use a total batch size of 16 with gradient accumulation set to 2. Training the model for 500 steps takes between 8 and 10 hours. We select the best performing checkpoint before any collapse in reward, following (Jin et al., 2025) since RL training maybe unstable.

C. Metrics

F1. is the harmonic mean of the precision and recall measured at the word-level, and is given by

$$\text{F1} = 2 \cdot \frac{\text{TP}_{\text{ans}}}{\text{TP}_{\text{ans}} + 0.5 * (\text{FP}_{\text{ans}} + \text{FN}_{\text{ans}})} \tag{3}$$

where TP_{ans} denotes correctly predicted words, FP_{ans} represents the extra words, and FN_{ans} are the missing words in the generated answer.

Exact Match. is the exact match between the normalized

Algorithm 1: Our novel two-stage framework, FRUGALRAG consists of **(1)** *Dataset Generation* and *Supervised "Explore" Finetuning*, and **(2)** *Controlling test-time compute wth RL.*

Input: Labeled dataset $\mathcal{L} = \{(Q^{(j)}, Y^{(j)})\}_{j=1}^{1000}$, $\mathcal{L}_{\text{init}} = \{(Q^{(j)}, Y^{(j)})\}_{j=1}^{50}$, retriever $\mathcal{R}$, base LM f, budget B, max hops m, number of samples v

// Prompt Optimization
Generate few-shot prompts $\{p_1, \ldots, p_n\}$ using f and $\mathcal{L}_{\text{init}}$;
// Dataset Generation
Initialize finetuning dataset: $\mathbf{D} \leftarrow []$;
for $Q^{(j)}, Y^{(j)}$ *in* $\mathcal{L}$ **do**
  Initialize buffer: main_rollout $\leftarrow []$;
  Initialize $\mathcal{D}_0 \leftarrow \mathcal{R}(Q^{(j)})$ or $\emptyset$;
  Initialize T_0, A_0;
  $\mathcal{H}_0 \leftarrow \{Q^{(j)}, T_0, A_0, \mathcal{D}_0\}$;
  Append $\mathcal{H}_0$ to main_rollout;
  for $h = 1$ *to* m **do**
    for $i = 1$ *to* n **do**
      for $h = 1$ *to* B **do**
        $(T_h^i, A_h^i, S_h^i) \leftarrow f(\mathcal{H}_{h-1}^i; p_i)$;
        // occurs in 10% of calls
        if $A_h^i =$ FINISH **then**
          break
        $\mathcal{D}_h^i \leftarrow \mathcal{R}(S_h^i)$;
        Remove duplicate retrievals from $\mathcal{D}_h^i$;
        $\mathcal{H}_h^i \leftarrow \mathcal{H}_{h-1}^i \cup \{T_h^i, A_h^i, S_h^i, \mathcal{D}_h^i\}$;
      Evaluate all $\{\mathcal{D}_h^i\}_{i=1}^n$ (recall against ground truth $Y^{(j)}$);
      Select best-performing trajectory $\mathcal{H}^*$;
      Append $\mathcal{H}^*$ to main_rollout;
  Append each hop from main_rollout to $\mathbf{D}$;
// Stage 1 (Sec. 2.1)
$f_S \leftarrow$ Fine-tune f using $\mathbf{D}$ // See Eq. 2
// Stage 2 (Sec. 2.2)
for $Q^{(j)}, Y^{(j)}$ *in* $\mathcal{L}$ **do**
  for $h = 1$ *to* m **do**
    Generate v sample tuples $\{T_h^i, A_h^i, S_h^i, \mathcal{D}_h^i\}_{i=1}^v$;
  for $i = 1$ *to* v **do**
    Compute reward $R^i \leftarrow \mathbf{R}(\{\mathcal{D}_h^i\}_{h=1}^m, Y^{(j)}, f_S)$ // See Eq. 1
    Backpropagate loss on $\{T_h^i, A_h^i, S_h^i\}_{h=1}^m$ using R^i;

generated answer string and normalized ground truth answer. It is 100% if the two strings match exactly, and 0 otherwise.

Match Score. measures the accuracy of generated answer by checking if the ground truth answer string is present in the generated answer string. It is 100% if the ground truth string is in the generated answer, and 0 otherwise.

Recall. is a retrieval metric, that measures the percentage of ground truth documents retrieved by the model. It is given by the ratio of correctly retrieved document titles TP_{doc} and the total number of ground truth document titles $\text{TP}_{\text{doc}} + \text{FN}_{\text{doc}}$. We measure recall following LeReT (Hsu et al., 2024), using document titles. The document-level recall is given by –

$$\text{Recall} = \frac{\text{TP}_{\text{doc}}}{\text{TP}_{\text{doc}} + \text{FN}_{\text{doc}}} \tag{4}$$

Sup. F1. measures the word-level F1 score (3) between the ground truth evidence sentences and those retrieved from the documents. Following (Trivedi et al., 2022b), we compute the average F1 score across the ground truth evidence sentences, comparing them with corresponding retrieved documents. The evidence sentences are are provided in all three datasets. Supporting Document F1 or Sup. F1 is serves as a more reliable metric for retrieval with 2WikiMultiHopQA and MuSiQue since it considers fine-grained evidence rather than just the document titles.

D. Dataset and Retrieval Index

We use the pre-processed Wikipedia abstracts index [1] provided by ColBERTv2 (Santhanam et al., 2021) for all our experiments on HotPotQA (Yang et al., 2018). For each instance, we retrieve the top 3 documents and their titles and perform a maximum 6 retrievals. HotPotQA annotations consists of document title and evidence sentences which are used to compute the Recall and Supporting Document F1 respectively.

Since 2WikiMultiHopQA (Ho et al., 2020) and MuSiQue (Trivedi et al., 2022b) datasets are created using both the body and abstract of wikipedia articles we use the pre-processed dump of Wikipedia provided by (Karpukhin et al., 2020) and index it using ColBERTv2 (Santhanam et al., 2021). The generated index consists of 21M passages. For each instance, we retrieve top 5 documents and append it to our context.

[1] https://downloads.cs.stanford.edu/nlp/data/colbert/baleen/wiki.abstracts.2017.tar.gz

E. Related Work

Which metric to optimize. Multi-hop QA involves two subtasks: retrieving relevant documents, and then answering the question based on the documents. Some methods report document retrieval-specific metrics such as recall (Hsu et al., 2024) whereas others report final answer metrics such as exact match (Jin et al., 2025). Typically, a model is trained to optimize a particular metric (such as recall) and also evaluated on the same metric. For robustness, in this work we train on the recall metric and test on all metrics, including final answer metrics.

Finetuning-based techniques. A prevalent method for multi-hop QA using small LMs is supervised finetuning using reasoning traces from a large LM such as GPT-4 (Asai et al., 2023; Chan et al., 2024). Other methods are trained to predict the next query to be retrieved (Chan et al., 2024). Methods that scale the test-time compute that infer using multiple trajectories have also been proposed (Wang et al., 2025). Recently, reinforcement learning-based techniques have been proposed that develop a reward based on outputting the ground-truth answer (Jin et al., 2025). However, none of the techniques focus on efficiency of the solution. In fact, in Search-R1, the goal of RL is to increase the number of searches. Instead, we use RL to *decrease* the average number of searches done by our model. Next we discuss each sub-category in detail:

1. **Traditional RAG approaches** Early work in grounding generation with real world documents focused on end-to-end differentiable encoder-decoder pipeline REALM (Guu et al., 2020), which augments Masked-Language Modeling (MLM) with a latent retriever model, backpropagating through retrieval to learn both retriever and generator jointly. However, this approach incurs significant computational cost and has only been shown to work with relatively smaller models like T5 (Raffel et al., 2020). Building on this, (Lewis et al., 2020) proposed a general finetuning strategy, RAG-Token which demonstrated that join-training outperforms fixed dense retrieval and BM25.

2. **Prompting-based RAG approaches** With the recent advancements in the capabilities of large API-based LMs, some works explored prompting to call external search/retrievers at inference. Toolformer(Schick et al., 2023) uses a self-supervised objective to train an external model that decides to call tools (like Bing and Google search engines). ReAct (Yao et al., 2023) is another powerful prompting technique that allows the model to structure its outputs as thoughts, actions and observations, yielding significant improvements in the ability of LLMs to interact with external environments. (Trivedi et al., 2022a) proposed IRCoT, another

prompting strategy that alternates between chain-of-thought (Wei et al., 2022) steps and gathering evidence through retrievals. By using the intermediate traces, the IRCoT is able to decide what to retrieve by issuing the right search queries. Iter-RetGen (Shao et al., 2023) improves evidence gathering in multi-hop scenarios by combining retrieval and generation iteratively, such that a model's response is incorporated in the reasoning trace. However, both IRCoT (Trivedi et al., 2022a) and Iter-RetGen (Shao et al., 2023) rely on a fixed or predefined number of retrieval loops at inference, offering limited control over latency.

3. **Large-scale training** Some efforts have also been made to improve retriever-generator architectures, coupled with large scale training for better grounding. (Muennighoff et al., 2024) introduced a 7B scale model, GRITLM, that can perform both embedding tasks (retrieval) and generation through instruction tuning. GRITLM demonstrated improved performance on MTEB (Muennighoff et al., 2022) while outperforming existing models of its scale on generative tasks. REPLUG (Shi et al., 2023) aligns its dense retriever with the generator by treating generator models as API-based black box models. Specifically, (Shi et al., 2023) it minimizes the KL divergence between the retriever's document score distribution and the generator's log-likelihoods over retrieved passages, yielding tighter coupling between retriever and generator. RETRO (Borgeaud et al., 2021) pretrains an autoregressive model augmented with a frozen BERT (Devlin, 2018) based retriever by cross-attending the retrieved document chunks with preceding tokens. In contrast, our method avoids retriever pretraining: it trains only the search-query generator on a small supervised dataset, yet still achieves high recall and generation quality. FRUGALRAG is independent of the retriever model and can be augmented with the aforementioned existing retrievers.

4. **RL-based Retrieval Augmented Generation** Recently, framing search query as an RL problem has received attention. LeReT (Hsu et al., 2024) performs preference optimization using diverse few shot prompts leveraging hundred-thousands of ground truth annotated documents. However, LeReT utilizes a fixed amount of compute per instance during inference and cannot be readily generalized to variable-hop scenarios. Similarly, concurrent works, (Jin et al., 2025) and (Chen et al., 2025) propose end-to-end RL-based optimization that only leverages the final answer annotation. These methods show that RL can effectively be used to teach the search query generator model *to issue more search queries* for multi-hop problems *without considering latency*. Our two-stage RL framework, by contrast, first explores without RL to maximize recall and then learns to stop at test time using RL.

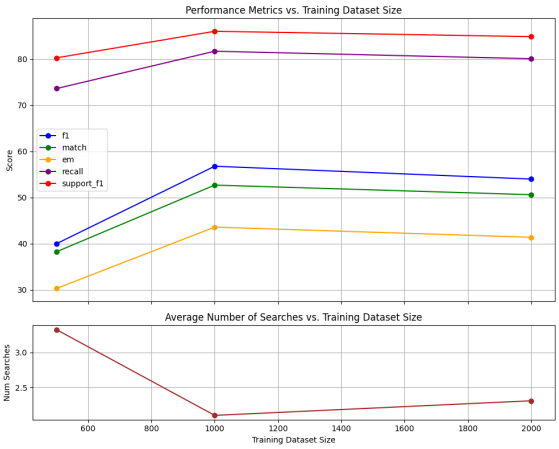

Figure 1. Effect of scaling dataset size on FRUGALRAG (Qwen2.5-3B-Instruct) on HotPotQA dataset. We see a sharp decrease in number of searches when we increase the dataset size from 500 to 1000 examples.

F. Ablations

Adaptive vs fixed budget searches. A simple way to make ReAct baselines more efficient is to limit their number of searches to a fixed budget. *Does variable compute (searches) really help in handling questions of varying difficulty?* To get an estimate of the query difficulty and search requirements, we look at the histogram of number of searches in the trajectories of FRUGALRAG-Explore up until when 100% Recall is attained. Fig. 2 illustrates that most additional searches beyond a certain point are redundant (Estimated Optimal Searches), and the utility of each additional query diminishes exponentially, from the perspective of Recall. So, we can expect ReAct baselines with fixed (small) search budget to attain competitive performance.

We investigate the utility of variable compute in Tables 1- 2 with Qwen3.5-3B-Instruct and Qwen3.5-7B-Instruct respectively. We consider the best baseline (ReAct FS), and give it a search budget B, and ensure that it does not generate FINISH till the entire budget B is consumed. This allows us to directly compare the accuracy of FRUGALRAG that uses variable budget per query with fixed-budget baselines. With Qwen2.5-3B-Instruct (Table 1), we find that FRUGALRAG is superior to all budgets $B = \{2, 3, 4, 5, 6\}$ in F1, Recall, and Sup. F1 while also being more efficient (**2.10** avg. searches). This trend also holds true for Qwen2.5-7B-Instruct – FRUGALRAG outperforms its close counterpart in terms of efficiency ($B = 3.0$) by **+1.4**, **+1.38**, and **+1.06** on F1, Recall and Sup. F1, while having a slightly smaller latency (0.10 fewer searches on average). The benefits of

Table 3. Comparison of FRUGALRAG with baselines on 2WikiMultiHopQA and MuSiQue. We show that FRUGALRAG is better on both answer-level (F1, EM, Match) and retrieval-level (Recall, Sup. F1, Search latency) metrics.

Method	2Wiki						MuSiQue					
	F1	EM	Match	Recall	Sup. F1	Search	F1	EM	Match	Recall	Sup. F1	Search
Qwen2.5-3B-Instruct												
Naive	9.84	0.62	37.64	0.00	0.00	0	4.36	0.20	7.36	0.00	0.00	0
CoT	15.31	7.07	25.38	0.00	0.00	0	6.62	1.94	5.95	0.00	0.00	0
RAG (ndocs=3)	18.15	12.19	21.54	32.87	46.16	1.00	9.15	4.26	8.27	19.95	23.61	1.00
RAG (ndocs=5)	20.11	13.78	24.33	36.38	49.46	1.00	9.63	3.97	8.02	22.73	25.52	1.00
RAG (ndocs=10)	22.02	14.86	27.05	40.61	53.16	1.00	10.73	4.84	10.50	27.13	27.85	1.00
ReAct	15.87	2.64	**34.84**	39.76	52.67	2.30	7.46	1.44	11.75	25.85	26.81	2.68
ReAct FS	23.83	16.23	24.18	45.77	57.67	3.99	13.30	7.73	11.66	30.44	29.09	4.32
FRUGALRAG-Explore	25.97	17.08	28.19	**54.13**	**64.17**	5.91	**18.29**	**11.21**	**16.13**	**37.10**	**32.03**	5.87
FRUGALRAG	**29.94**	**19.27**	33.06	43.15	55.49	2.32	15.02	7.65	10.09	29.46	28.46	3.07

Table 4. Comparison of FRUGALRAG with baselines on 2WikiMultiHopQA and MuSiQue. We show that FRUGALRAG is better on both answer-level (F1, EM, Match) and retrieval-level (Recall, Sup. F1, Search latency) metrics.

Method	2Wiki						MuSiQue					
	F1	EM	Match	Recall	Sup. F1	Search	F1	EM	Match	Recall	Sup. F1	Search
Qwen2.5-7B-Instruct												
Naive	16.43	9.88	32.47	0.00	0.00	0	6.40	0.70	8.00	0.00	0.00	0
CoT	24.24	18.30	29.60	0.00	0.00	0	10.80	3.59	8.06	0.00	0.00	0
RAG (ndocs=3)	14.95	6.34	30.38	32.87	46.16	1.00	10.52	4.00	12.53	19.95	23.61	1.00
RAG (ndocs=5)	16.82	7.28	33.27	36.38	49.46	1.00	11.51	4.79	12.53	22.73	25.52	1.00
RAG (ndocs=10)	20.32	10.13	37.03	40.61	53.16	1.00	11.59	4.55	14.06	27.13	27.85	1.00
ReAct	21.28	6.80	43.80	45.80	58.62	2.78	10.69	3.56	20.39	30.94	29.11	3.13
ReAct FS	41.74	32.90	42.50	49.25	61.03	3.48	20.26	11.91	18.99	33.90	30.50	4.16
FRUGALRAG-Explore	**42.55**	33.07	**45.61**	**55.01**	**64.97**	5.89	22.09	13.23	**22.67**	**37.11**	**31.88**	4.32
FRUGALRAG	44.22	**34.20**	43.90	52.05	62.03	4.16	**24.24**	**13.94**	20.68	33.46	30.04	2.63

FRUGALRAG are more pronounced at the 3B scale, likely because the dataset poses a greater challenge for smaller models, amplifying the need for variable compute.

Impact of Exploration Finetuning. We investigate the importance of exploration finetuning in Table 5. Exploration finetuning enables us to build a better base policy f_S for GRPO, since it has the ability to both explore (issue large number of searches), and generate FINISH. This property makes it the ideal choice for GRPO, since GRPO requires access to diverse and high-quality rollouts. To validate the effectiveness of f_S (alternatively, FRUGALRAG-Explore) as a base policy, we perform GRPO with our reward function Eq. 1 using the base model Qwen2.5-3B-Instruct on HotPotQA dataset and present the results in Table 5. As hypothesized, FRUGALRAG (w/o Explore-SFT) achieves a recall of **75.30** (2.77 searches) compared to FRUGALRAG, which outperforms it with a recall of **81.69** (2.10 searches).

Impact of Controlling test-time compute. Table 5 evaluates the contribution of GRPO within the FRUGALRAG framework. GRPO's role is to learn the optimal stopping point h^* at which to terminate the search. One natural question is: *Can h^* be learned through supervised finetuning on labeled data?* To answer this, we compare our reinforcement-learning–based approach (FRUGALRAG) against a supervised finetuning variant (FRUGALRAG-SFT w/ FINISH). Across both answer-level and retrieval-level metrics, FRUGALRAG significantly outperforms FRUGALRAG-SFT: it improves F1 by **+5.01** and Recall by **+3.85**. Although FRUGALRAG-SFT attains a high Recall of 77.85%, it does so at the cost of efficiency, on average performing 2.92 searches. This ablation confirms that learning the stopping policy via GRPO is both more accurate and more efficient.

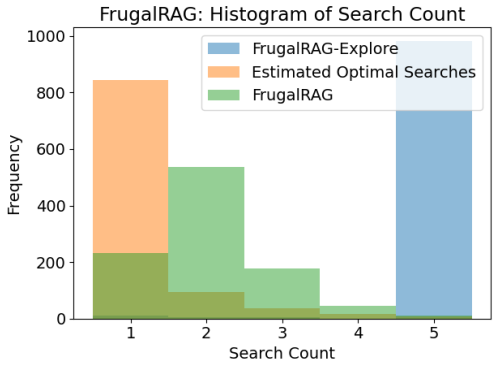

Figure 2. Histogram of number of searches over 1,000 random examples from the HotPotQA dataset.

Table 5. Ablation results on the HotPotQA dataset using Qwen2.5-3B-Instruct demonstrate that both exploration fine-tuning and test-time compute control are essential for effective Multi-Hop RAG.

Method	F1	EM	Match	Recall	Sup. F1	Search
ReAct FS	45.81	33.54	42.12	71.47	78.64	3.33
FRUGALRAG-SFT (w/o GRPO, w FINISH)	51.08	38.46	47.58	77.85	83.27	2.92
FRUGALRAG (w/o Explore-SFT)	53.33	39	49.8	75.30	81.49	2.77
FRUGALRAG-Explore (w/o GRPO, w/o FINISH)	53.17	41.80	51.10	81.41	85.72	5.90
FRUGALRAG	**56.78**	**43.59**	**52.69**	**81.69**	**86.01**	**2.10**

Table 6. Performance comparison of FRUGALRAG on HotPotQA using different generator models g and common Qwen2.5-3B-Instruct base model f.

Method	F1	EM	Match	Recall	Sup. F1	Search
ReAct (Generator 3B)	25.35	10.84	42.91	65.13	73.96	1.88
ReAct (Generator 7B)	25.45	9.99	46.77	64.88	73.80	1.90
ReAct FS (Generator 3B)	50.88	38.02	47.65	71.47	78.64	3.33
ReAct FS (Generator 7B)	51.26	38.69	47.73	72.16	79.21	3.43
FRUGALRAG (Generator 3B)	56.78	43.59	52.69	81.69	86.01	2.10
FRUGALRAG (Generator 7B)	61.37	46.98	57.05	81.45	85.84	2.12

G. Additional Results

Comparison Against RL-based Techniques. In Table 7, we compare FRUGALRAG with the recent RL-based techniques including LeReT (Llama3.1-8B), Search-R1 (Qwen2.5-3B, Qwen2.5-7B), and Search-R1-Instruct (Qwen2.5-3B-Instruct, Qwen2.5-7B-Instruct). It is noteworthy that LeReT leverages the entire HotPotQA (Yang et al., 2018) train-set (90k examples) with its ground truth evidence annotation for preference learning. Similarly, Search-R1 combines Natural Questions (Kwiatkowski et al., 2019) and HotPotQA (Yang et al., 2018) training datasets with their final answer annotations, resulting in over 100000 training examples. Unlike existing methods, FRUGALRAG only requires 1000 evidence annotated examples for training. We note that a direct comparison with these baselines is non-trivial due differences in underlying base models and retrievers. Consequently, we directly report the metrics provided by the authors in their official papers. To enable a fairer comparison, we replicate Search-R1 (Jin et al., 2025) using a final answer F1 score as its reward, and train Qwen2.5-7B-Instruct on 1000 HotPotQA examples with ColBERTv2 (Santhanam et al., 2021) retriever, similar to the setup in FRUGALRAG. Table 7 demonstrates that both the 7B and 3B version of FRUGALRAG outperform Search-R1-7B-Instruct trained with 1000 examples by a large margin of **+22.92** and **+17.26** F1 respectively. We find the FRUGALRAG is clearly better than Search-R1 because it leverages fine-grained feedback from the ground truth documents, demonstrating the effectiveness of FRUGALRAG with limited data.

Performance on 2WikiMultiHopQA and MuSiQue. In Table 4, we analyze the performance of FRUGALRAG on 2WikiMultiHopQA (2Wiki) and MuSiQue datasets with Qwen2.5-7B-Instruct. Notably, our framework excels on the particularly challenging MuSiQue (Trivedi et al., 2022b) dataset — improving F1 score by **+3.98**, and reducing the latency by **1.58** times, compared to ReAct FS. FRUGALRAG increases the average number of searches by **+0.68** on 2Wiki (Ho et al., 2020). Despite the slight latency increase, FRUGALRAG still yields a notable improvement of **+2.8** in F1 score, underscoring its effectiveness even in more heterogeneous (2Wiki) retrieval settings. Overall, across the datasets, the accuracy metrics of FRUGALRAG are competitive wrt FRUGALRAG-Explore (which is the best or the second best method in terms of accuracy), and its number of searches is competitive wrt ReAct FS baseline (which is the most frugal on 2 out of 3 datasets). This emphasizes FRUGALRAG's ability to generate high quality answers with remarkable efficiency.

Table 3 presents a comparison between FRUGALRAG and ReAct FS on the 2WikiMultiHopQA and MuSiQue datasets using Qwen2.5-3B-Instruct. We observe a similar trend as Table 4 – FRUGALRAG significantly decreases the average number of searches for both 2WikiMultiHopQA (from 3.99 to 2.32) and MuSiQue (from 4.32 to 3.07).On 2WikiMultiHopQA, FRUGALRAG achieves the highest F1 score (29.94), which is **+6.11** higher than ReAct FS, while requiring 0.68 fewer searches on average. Similarly, on MuSiQue, FRUGALRAG outperforms ReAct FS in F1 score and achieves comparable EM and Match scores, while issuing **1.25** fewer searches on average. These experiments highlight the effectiveness of FRUGALRAG across various datasets and model scales.

Dataset Scaling Results. In Fig. 1, we observe that as the training dataset size increases from 500 to 1000, there is a significant improvement in answer metrics, for example F1 score improves from 40 to 57. However, at 2000 examples, the F1 score drops slightly to 54. Interestingly, the average

Table 7. Comparison with recent techniques that use large-scale Reinforcement Learning. FRUGALRAG demonstrates superior performance on all metrics compared to the state-of-the-art on HotPotQA dataset using both Qwen2.5-3B and Qwen2.5-7B.

Method	F1	EM	Match	Recall	Sup. F1	Search
LeReT (Llama3.1-8B)	-	**52.5**	-	77.1	-	2.0
SearchR1-3B	-	28.4	-	-	-	-
SearchR1-3B-Instruct	-	32.4	-	-	-	-
SearchR1-7B	-	43.3	-	-	-	-
SearchR1-7B-Instruct	-	37.0	-	-	-	-
SearchR1-7B-Instruct* (1000 examples)	39.52	25.61	53.49	76.19	82.34	5.97
FRUGALRAG-Explore (3B)	53.17	40.56	49.65	81.41	85.72	5.90
FRUGALRAG (3B)	56.78	43.59	52.69	81.69	86.01	2.10
FRUGALRAG-Explore (7B)	61.22	47.87	**57.31**	**84.19**	**87.63**	5.88
FRUGALRAG (7B)	**61.84**	47.60	54.77	79.88	84.80	2.90

number of searches decreases significantly from 3.33 at 500 examples to approximately 2 at larger sizes, indicating more efficient retrieval as the model becomes better trained with diverse data. We plan on incorporating a comprehensive study of different data mixtures and sizes in our future work.

Generator Model Scaling. Table 6 presents the performance of FRUGALRAG across different generator models g on the HotPotQA dataset. In all experiments, we use Qwen2.5-3B-Instruct as the search query model f and vary the answer generation model g to evaluate its impact. As expected, FRUGALRAG demonstrates improved performance on standard answer quality metrics (F1, EM, and Match) when equipped with a stronger generator model (7B) compared to a smaller 3B variant.

The relative gains in performance brought about by improved retrieval are noteworthy. While both generator sizes benefit from enhanced retrieval quality, the magnitude of improvement is significantly larger for the stronger 7B model. Specifically, the F1 score increases from 51.26 to 61.37 when using the larger model, **+10.11** point gain. In contrast, the smaller 3B model improves from 50.88 to 56.78. This disparity highlights a key insight; larger models are more capable of leveraging improvements in retrieval to generate better answers. Consequently, improving the retrieval quality disproportionately benefit more capable generator models, emphasizing the importance of optimizing the search query model f.

H. Limitations and Future Work

Even though our method uses a small number of examples for training, it has some limitations in the analysis, and leaves room for future work in the following aspects: (a) in terms of generalization to new domains with no access to training data, and (b) in terms of coping with different types of retrievers (e.g., can we strike better efficiency-accuracy trade-off with a more powerful retriever?).