FRUGALRAG: LESS IS MORE IN RL FINETUNING FOR MULTI-HOP QUESTION ANSWERING

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement learning (RL) based on the final answer's reward has driven recent progress in small language models (SLMs) on reasoning-heavy tasks such as math and code. However, applying the same techniques to retrieval-augmented generation (RAG) benchmarks like multi-hop QA has yielded limited gains—often trailing supervised or prompting-only baselines. Instead, we argue that a viable path for RL in multi-hop QA is to use test-time scaling judiciously, for optimizing both the final answer accuracy and the efficiency in reaching that answer. We propose FrugalRAG, a two-stage finetuning framework that adaptively reduces the number of retrieval steps based on a question's difficulty. First, we train an SLM with supervised finetuning on a full-exploration policy that generates broad sub-queries. Then, we apply RL to adaptively prune search depth based on question difficulty, directly rewarding policies that balance correctness with frugality. Unlike prior approaches requiring 100× more data, our method achieves competitive performance with only 1,000 examples. On HotPotQA and other multi-hop QA benchmarks, FrugalRAG attains state-of-the-art efficiency–accuracy tradeoffs, cutting retrieval cost nearly in half. Moreover, on the challenging BrowseCompPlus benchmark, it generalizes zero-shot and surpasses SLM-based and other baselines. These results demonstrate the use of RL—not to increase reasoning steps but to optimize them—as an effective solution for scalable, efficient RAG.

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.

Based on the success of reinforcement learning (RL) in math and code applications (DeepSeek-AI, 2025), recent work applies RL-based finetuning to optimize retriever tool calling and generation of the right queries to answer a given question. However, the accuracy gains in multi-hop QA benchmarks are less impressive. For instance, on the HotPotQA benchmark (Yang et al., 2018), a simple ReAct-based Yao et al. (2023) strategy of letting a model generate up to 10 subqueries leads to higher document recall (63%) than state-of-the-art RL methods such as Search-R1 (see Table 2). Therefore, the key question is not how RL can be used to improve accuracy, rather how RL can be used to optimize the search process and make a RAG system more efficient (e.g., for HotPotQA, most questions can ideally be solved in 2-3 search queries). Ideally, we want a system that can adapt the number of search queries based on a question's difficulty level.

On efficiency, a second problem is the availability of labelled training data that maps a question to its correct answer. Outside of general web question-answering, it may be difficult to obtain ground-truth labels for questions when adapting a RAG system to work in a real-world application domain (with potentially private documents), Typical solutions in the literature (Jin et al., 2025a; Hsu et al., 2024),

however, train on 90,000-1,00,000 examples from existing benchmarks such as HotPotQA (Yang et al., 2018). Therefore, a second key problem is finetuning a language model for retrieval when labelled data is scarce. For concreteness, we set the number of training examples as 1000, an order of magnitude lower than existing efforts.

Under these considerations, we ask the question: *How can we train a model to answer questions using as few search calls as necessary? And can we do it using only 1000 training examples?* Our solution, **FRUGALRAG**, is a two-stage framework that uses exploration to generate a large number of search queries per question (Stage 1), and uses RL to optimize the number of search queries per question (stage 2). Specifically, 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 using RL 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.

We evaluate FRUGALRAG on standard multi-hop datasets 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, our models obtain state-of-the-art accuracy on the recently released FlashRAG index benchmark for HotPotQA and the MuSiQue, despite being trained on only 1000 examples.

We also find that the same FRUGALRAG model generalizes to harder information search tasks. A recently released benchmark from OpenAI tests models' abilities on challenging needle-in-a-haystack problems, wherein the reasoning steps (and hence the search queries) to find the answer can be much larger. The same FRUGALRAG 7B model described above generalizes zero-shot on this out-of-domain task; obtaining an accuracy of 20%, which is higher than that of bigger models such as DeepSeek-R1 and Search-R1-32B. Remarkably, it adapts to issue a larger number of queries for the BrowseComp data compared to earlier datasets, unlike other RL methods such as Search-R1 (Jin et al., 2025a).

2 RELATED WORK

In this section, we present a detailed overview of the relevant literature comprising representative multi-hop RAG methods, recent work leveraging reinforcement learning for search, and finally standard metrics for evaluation.

Multi-Hop RAG. Retrieval Augmented Generation is an active area of research that aims to ground the responses generated by LLMs in real world information, tackling fundamental limitations like hallucinations and trustworthiness. Several works (Guu et al., 2020; Lewis et al., 2021; Shao et al., 2023; Jeong et al., 2024; Jiang et al., 2023; Asai et al., 2023; Chan et al., 2024; Hsu et al., 2024; Li et al., 2025; Wang et al., 2025) have been proposed in recent literature that explore this problem. Guu et al. (2020) and Lewis et al. (2021) showed that augmenting the inputs of language models with retrieved information significantly improved their performance on knowledge intensive tasks like question answering. Subsequently, Trivedi et al. (2022a) leveraged few-shot Chain-of-Thought (Wei et al., 2022) prompts and iteratively retrieved information for complex questions that required multiple reasoning and retrieval steps (i.e. multi-hop settings). Shao et al. (2023) also incorporate a similar iterative retrieval approach. 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. 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) uses a general prompting framework by interleaving thoughts, actions, and tool calls

in order to perform complex tasks. Khattab et al. (2023) invoked LLMs through declarative calls, allowing a more structured programmatic way of calling LLMs. Further Khattab et al. (2023) also design example-data efficient compilers to optimize a given metric.

Beyond prompting, SelfRAG (Asai et al., 2023) trains small language models (SLMs) with dense supervision from stronger models to decide when to retrieve external information during question answering. RQRAG (Chan et al., 2024) leverages LLMs to curate datasets that teach smaller models essential skills such as query decomposition and backtracking. In addition, Chan et al. (2024) highlight the effectiveness of advanced decoding strategies such as tree-of-thought sampling for identifying high-quality reasoning trajectories. More recently, approaches like CoRAG (Wang et al., 2025) have scaled test-time computation by jointly reasoning over multiple trajectories and training models through rejection sampling. However, these methods (SelfRAG, RQRAG, CoRAG) demand substantial supervision (100k+ annotated examples) and are constrained either by inference inefficiency (RQRAG (Chan et al., 2024)) or by limited flexibility (CoRAG (Wang et al., 2025)), which relies on predefined number of search). Other works, such as DRAGIN (Su et al., 2024) and Adaptive RAG (Jeong et al., 2024), train multiple modules with large-scale supervised data to enable dynamic retrieval. SimpleDeepResearcher (Sun et al., 2025) propose data-engineering strategies to mitigate the problem of domain shift between training and testing environment showing that supervised finetuning on a small number of representative samples can yield strong improvements. In contrast to these approaches, we introduce the first method that jointly accounts for both training and inference costs. Our key intuition is that reinforcement learning can be leveraged to scale down search effectively, thereby reducing supervision requirements while maintaining competitive performance.

Reinforcement Learning for Search. Recently, framing search query as an RL problem has received attention. LeReT (Hsu et al., 2024) is an early work that adopted RL by performing preference optimization using diverse few shot prompts leveraging hundred-thousands of ground truth annotated documents. However, this finetuning process is computationally expensive and cannot be cheaply and readily generalized to variable-hop scenarios. As a result, LeReT utilizes a fixed amount of compute (i.e. search calls) per instance during inference. Similarly, concurrent works, Jin et al. (2025a) and Chen et al. (2025a) propose end-to-end RL-based optimization that only leverages the final answer annotation. R1-Seacher (Song et al., 2025) proposes an RL-only strategy for finetuning demonstrating strong performance on multihop RAG benchmarks by utilizing a curated balanced dataset with outcome based rewards. 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.

Metrics for Evaluation. Multi-hop QA involves two sub-tasks: 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., 2025a). 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. Unlike Wang et al. (2025); Li et al. (2025); Jin et al. (2025a); Song et al. (2025) we train on the reasoner module which performs reasoning and issues search calls keeping the final answer generator fixed.

3 FRUGALRAG: EFFICIENT TWO-STAGE FINETUNING FOR RAG

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., 2025a; 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). Algorithm 1 summarizes our framework.

Problem Formulation. Let Q denote a complex user question that requires multiple iterative retrievals to answer. Let f denote a language model (LM) that, at each reasoning hop, examines the current context and determines the next action. At hop h (where $1 \leq h \leq B$, and B is the maximum allowed number of hops), the model f produces a *thought–action-search query* triplet (T_h, A_h, S_h).

The search query S_h is passed to a retriever $\mathcal{R}(\cdot)$, which returns a set of documents: $\mathcal{D}_h = \mathcal{R}(S_h)$ from the document index $\mathcal{I}$.

Let $\mathcal{D}_0$ denote the initial context which is either empty or initialized as $\mathcal{R}(Q)$. At hop h, the model has access to the context: $\{Q\} \cup \{(\mathcal{D}_h, T_h, A_h, S_h)\}_0^{h-1}$. This includes the original query, all previously retrieved documents, and the previously generated thought, action, search query triplets. The process continues until the model outputs a special FINISH action, terminating after h_{term} hops (or at the budget limit B). At this point, a separate generator LM g is invoked to produce the final answer, conditioned on the original question Q and the full context accumulated up to termination.

The central challenge for f is to iteratively construct highly targeted queries S_h such that the retriever $\mathcal{R}$ can surface a minimal yet sufficient set of documents to answer Q within the hop budget B.

Our key observation is that, with sufficient test-time compute, even base models can generate multiple, diverse search queries to address a question (e.g., see Sec 4, Table 2, ReAct+Dspy). Therefore, the goal of our work is not to *scale up* test-time computation, as argued in prior work (Jin et al., 2025a; Chen et al., 2025a), but rather to *adaptively control* it based on the difficulty of each question.

Our framework, FRUGALRAG requires access only to ground truth documents Y during training. It does **not** require supervision in the form of final answer annotations. These documents are used to provide fine-grained feedback signals. At inference time, FRUGALRAG relies solely on the input question Q, the document index $\mathcal{I}$, and a trained retriever $\mathcal{R}$.

In the following subsections, we describe the two stages of our learning algorithm: (1) **Stage 1** (Sec. 3.1): Generating a base policy that maximizes evidence coverage through exploration, and (2) **Stage 2** (Sec. 3.2): Finetuning this policy with reinforcement learning to control test-time compute.

3.1 STAGE 1: EVIDENCE 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 refer to a rollout as a set of outputs generated by the model f. Each rollout comprises 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. This setup is similar to the standard ReAct (Yao et al., 2023) framework which provides a general prompting template for interleaving external (retrieved) information and model generated outputs. 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 – a standard run where f is allowed terminate generation by generating FINISH, and the second where f can only call the retriever till maximum budget is reached. Although the former is more efficient and finishes before B search hops, we observe that the latter yields a 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., 2025a) that generated orders of magnitude more data, we only generate 1000 examples 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 the configurations described above, i.e., 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 $\{(\mathcal{D}_0, T_0, A_0, S_0), (\mathcal{D}_1, T_1, A_1, S_1) \dots (\mathcal{D}_{(h-1)}, T_{(h-1)}, A_{(h-1)}, S_{h-1})\}$ 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 action space 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.

3.2 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 allows the model to adaptively determine the appropriate rollout length based on the complexity of the question, rather than scale the number of retrievals as in recent RL techniques (Jin et al., 2025a). Since f_S generally prioritizes exploration, our only goal is to learn when to sufficient evidence has been gathered. In turn, we can also reduce the overall search latency during inference whenever possible. Below we show how this problem can be formulated as a reinforcement learning task, as it requires evaluating and comparing different rollouts.

Reward Design. Our reward function is designed to guide the model toward discovering the optimal rollout length. To compute the reward, we first generate the complete rollout using the policy f_S and then evaluate it with ground truth evidence. Let h^* denote the optimal number of retrieval steps (or hops), defined as the point beyond which further retrievals do not improve a predefined metric, say c. If the model terminates at a hop $h_{\text{term}} > h^*$, it incurs a penalty for redundant steps. Conversely, stopping at $h_{\text{term}} < h^*$ is also penalized to encourage sufficient exploration. This reward structure enables the model to explore adequately for complex queries while avoiding unnecessary computation for simpler ones. Mathematically,

$$\mathbf{R} = \begin{cases} \max\left(-R_{\max}, \min\left(\log\left(\frac{1-\Delta}{\Delta}\right), R_{\max}\right)\right), & \text{if } \Delta > 0 \wedge c \geq \tau \quad \text{(late stop)} \\ R_{\max} + \alpha \cdot \left(\frac{h^*}{B}\right), & \text{if } \Delta = 0 \wedge c \geq \tau \quad \text{(perfect stop)} \\ \max\left(-R_{\max}, \min\left(\log\left(\frac{1-\Delta}{\Delta}\right), 0\right)\right), & \text{if } c < \tau \quad \text{(early stop)} \end{cases} \tag{1}$$

Here, Δ is the normalized difference $(h_{\text{term}} - h)/B$, where B is the maximum hop budget. $R_{\max}$ is the upper bound on the reward, and α is a tunable hyperparameter that scales the bonus for stopping exactly at h^*. This bonus is proportional to h^*/B, thereby assigning higher reward to correct terminations on more complex (i.e., longer) rollouts. A rollout is considered answerable if it performs better than a threshold $c \geq \tau$.

Intuitively, the reward function penalizes deviations from the optimal stopping point in proportion to $|\Delta|$. The closer the model stops to h^*, the higher the reward, with the maximum attained when $h_{\text{term}} = h^*$.

In addition to $\mathbf{R}$, we define a format reward $\mathbf{R_f}$ to enforce adherence to the format. If the output deviates from the expected format and results in failed retrievals, we assign a reward of -0.5; conversely, if the retrievals succeed, the model receives a reward of $+0.5$. This reward is averaged across all hops. The final reward is the mean of the main reward $\mathbf{R}$ and the format reward $\mathbf{R_f}$, yielding an overall reward in the range $[-R_{\max} - 0.50,\ R_{\max} + \alpha + 0.50]$.

Optimal Rollout Length. We define the optimal rollout length h^* as the minimum number of retrieval steps required to answer a query Q effectively. Any additional retrievals beyond h^* are considered redundant. Conversely, if the rollout has not yet gathered sufficient evidence to answer the query, it should continue retrieving. To balance retrieval efficiency and performance, we use FRUGALRAG-Explore (f_S) as a reference policy, assuming it represents the best achievable performance within a fixed budget B. Similarly, we determine the threshold τ based on the performance of the base policy FRUGALRAG-Explore (f_S).

Table 1: Results on HotPotQA, 2Wiki, and MuSiQue with ColBERTv2 retriever. FRUGALRAG demonstrates superior performance on Model Based Evaluation (MBE) and Gold Evidence Recall (%) compared to standard baselines.

Method	HotPotQA			2Wiki			MuSiQue		
	MBE	Recall	Searches	MBE	Recall	Searches	MBE	Recall	Searches
Zero-Shot	28.1	NA	NA	28.4	NA	NA	10.6	NA	NA
Zero-Shot CoT	29.10	NA	NA	29.8	NA	10.7	NA	NA	NA
Zero-Shot RAG	51.0	59.05	1	28.8	34.75	1	0.185	23.96	1
ReAct + DsPy	64.2	79.60	2.76	45.60	58.5	3.37	29.20	52.1	3.37
FrugalRAG-Explore 7B	67.70	**86.40**	5.99	47.6	**68.90**	5.99	30.10	**59.10**	5.93
FrugalRAG 7B	**68.47**	82.80	2.05	**48.93**	63.50	2.95	**33.72**	54.00	3.02

Table 2: Comparison of FRUGALRAG with state-of-the art methods on three datasets (HotPotQA, 2WIki, MuSiQue) using Answer (MBE) and Retrieval (Recall) metrics using E5-base-v2 retriever model. FRUGALRAG achieves a consistently competitive performance despite only being trained on 1000 training examples.

Method	#Train	HotPotQA			2Wiki			MuSiQue		
		MBE	Recall	Searches	MBE	Recall	Searches	MBE	Recall	Searches
Zero-Shot	-	28.10	NA	NA	28.4	NA	NA	10.60	NA	NA
Zero-Shot CoT	–	29.10	NA	NA	29.8	NA	10.7	NA	NA	NA
Zero-Shot RAG	-	45.50	52.49	1	28.80	35.00	1	18.00	22.01	1
ReAct + Dspy	100	49.30	0.634	3.23	35.65	60.03	3.27	27.30	49.8	3.74
IRCOT	-	45.60	66.90	3.31	30.53	57.80	3.41	20.62	44.10	3.19
ITER-RETGEN	36k	46.10	55.40	3.00	32.10	45.30	3.00	19.10	34.20	3.00
Ret-Robust	1000	39.38	35.90	4.02	45.17	31.50	4.23	24.24	18.80	4.23
Self RAG 7B	150k	37.60	52.60	1.00	30.10	41.30	1.00	15.00	26.70	1.00
Self RAG 13B	150k	39.70	52.60	1.00	31.50	41.30	1.00	15.00	26.70	1.00
O2 Searcher	-	42.70	50.10	1.77	45.70	55.20	2.42	26.40	37.00	1.95
SimpleDeepSearcher	871	50.40	64.80	2.75	49.30	60.50	3.64	34.30	50.40	2.86
R1-Searcher	8k	57.66	69.10	2.22	52.00	60.40	2.36	39.78	**57.70**	2.31
Search-R1	100k+	46.20	48.20	1.28	36.20	47.70	1.89	24.80	38.10	1.36
CoRAG	100k+	58.20	64.30	4.00	**59.00**	**65.40**	4.00	40.50	54.00	4.00
FrugalRAG-7B	1000	**69.50**	**70.40**	2.89	53.00	58.80	3.03	**41.50**	53.30	3.30

Optimization. We adopt GRPO (Shao et al., 2024) optimization algorithm because of its memory efficiency. At each hop h, we sample v tuples $\{T_h^i, A_h^i, S_h^i\}_{i=1}^v$, and retrieve non-duplicate documents $\mathcal{D}_h^i$. We collect sample tuples and documents through this process until FINISH or till maximum budget is exhausted. For each rollout i, we then compute a cumulative reward using Eq. 1, and backpropagate through every logit produced by the policy along that rollout.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Benchmarks and Evaluation. We evaluate FRUGALRAG on three widely adopted multi-hop retrieval-augmented generation (RAG) benchmarks—HotPotQA (Yang et al., 2018), 2WikiMultiHopQA (Ho et al., 2020) (2Wiki), and MuSiQue (Trivedi et al., 2022b)—under their full-wiki setting using the development splits, following the protocol of FlashRAG (Jin et al., 2025b). Beyond multi-hop RAG, we also assess FRUGALRAG in a highly challenging deep research scenario using BrowseCompPlus (Chen et al., 2025b). To demonstrate the effectiveness of FRUGALRAG, we experiment with three different retrievers: ColBERT-v2 (Santhanam et al., 2021), Qwen3-8B-Embedding (Zhang et al., 2025), and E5-base-v2 (Wang et al., 2022). Additional details on the

datasets and indexing are provided in Appendix C. Following prior work, we adopt a robust LLM judge-based evaluation metric (Chen et al., 2025b) to assess the accuracy of the final answers (MBE - Model Based Evaluation). We choose Qwen3-32B for answer evaluation. For retrieval-level evaluation, we follow FlashRAG (Jin et al., 2025b) and use recall as the metric, which measures whether the ground truth answer is present in the retrieved documents, assigning a score of 1 if it is found. Finally, to evaluate efficiency, we report the number of searches performed as a proxy for latency across different methods.

Baselines. We perform extensive experiments comparing FRUGALRAG with several strong baselines and state-of-the-art models. Our evaluation covers: (1) zero-shot performance; (2) prompting-based methods such as ReAct (Yao et al., 2023), IRCOT (Trivedi et al., 2022a), and DsPy (Khattab et al., 2023); (3) finetuned approaches including Iter-RetGen (Shao et al., 2023), Ret-Robust (Yoran et al., 2024), and Self-RAG (Asai et al., 2023). We further benchmark FRUGALRAG against recent reasoning-focused models, including CoRAG (Wang et al., 2025), O2 Searcher (Mei et al., 2025), Simple Deep Searcher (Sun et al., 2025), and R1-Searcher (Song et al., 2025).

Training. FRUGALRAG is model-agnostic; we train Qwen2.5-7B-Instruct using our two-stage framework. Both supervised finetuning and reinforcement learning (RL) stages leverage the TRL library (von Werra et al., 2020), with prompt bootstrapping based on DsPy (Khattab et al., 2023). For each dataset, Stage 1 finetuning uses 1000 randomly sampled training examples, performing full-parameter training for one epoch with a learning rate of 2×10^{-5}, weight decay of 0.01, and a maximum sequence length of 4096 (Sec. 3.1). Stage 2 applies GRPO (Shao et al., 2024) to further train the models (Sec. 3.2). Full hyperparameter details are in Appendix B.

4.2 MAIN RESULTS: EFFECTIVENESS OF FRUGALRAG

Table 1 compares FRUGALRAG with standard zero-retrieval and prompt-based baseline (ReAct). For fair comparison, all baselines use the ColBERTv2 (Santhanam et al., 2021) retriever indexed on Wikipedia (See Appendix B) and Qwen2.5-7B-Instruct as the base model. 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. The optimized prompts enable ReAct FS to generate more search queries, and as a result achieve very strong performance. Strikingly, on HotPotQA, ReAct FS achieves (a) **79.60%**, which is greater than the recall achieved by LeReT (Hsu et al., 2024) (77.1% using Llama3.1-8B) after significant finetuning; signifying the importance of building a strong baseline.

Table 1 demonstrate the overall *effectiveness of* FRUGALRAG-*Explore* achieving the highest Recall and MBE scores compared to all baselines. However, 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. Stage-2 finetuning significantly reduces the number of searches required while retaining the performance across three datasets. For instance, the search count on HotPotQA reduces on average by ≈50% across the three datasets while the MBE scores improve. Overall, our results highlight that FRUGALRAG strikes a significantly better efficiency-accuracy tradeoff.

4.3 COMPREHENSIVE BASELINE COMPARISONS

Table 2 demonstrates the performance of FRUGALRAG against 14 baselines and state-of-the-art RAG methods. For fair comparison, we setup our experiment using FlashRAG (Jin et al., 2025b) with a common retriever backend, E5-Base-V2 and an index of 21 million wikipedia passages.

FRUGALRAG establishes a strong performance on HotPotQA achieving a MBE score of 69.50%, significantly higher than the next best baseline at 58%. Notably, FrugalRAG is trained on only 1100 examples, yet outperforms state-of-the-art approches on HotPotQA. The second-best method, CoRAG (Wang et al., 2025) is a "fixed" inference-time compute approach, i.e., it issues a predefined number of searches (here, 4) regardless of the question. This makes CoRAG less generalizable to more complex datasets requiring different number of searches or inefficient for much easier tasks. On the contrary, FRUGALRAG adaptively assigns test time compute. Similarly, FRUGALRAG ranks either best or second-best on MBE score on 2Wiki and MuSiQue demonstrating strong QA capabilities.

Table 3: BrowseCompPlus. FrugalRAG, trained on datasets such as HotPotQA generalizes to this harder task and outperforms significantly larger models such as DeepSeek-R1.

Method	Model Size	Accuracy (%)	Recall (%)	Avg. Searches
Sonnet 4	-	37.35	47.33	9.03
Opus 4	-	36.75	50.84	10.24
kimi-k2-0711-preview	-	35.42	38.38	11.22
Gemini-2.5-Flash	-	34.58	40.19	9.77
Gemini-2.5-Pro	-	29.52	35.31	6.04
oss-120b-low	120B	25.54	22.50	2.21
oss-20b-high	20B	35.06	49.29	23.87
oss-20b-medium	20B	30.48	41.31	13.64
oss-20b-low	20B	14.10	17.37	1.87
DeepSeek-R1-0528	-	16.39	16.32	2.72
Qwen3-32B	32B	10.72	7.80	0.94
Search-R1-32B	32B	11.08	10.17	1.69
FrugalRAG-ColBERTv2 (HotPotQA)	7B	20.46	23.57	7.95
FrugalRAG-ColBERTv2 (2Wiki)	7B	21.53	22.93	10.96
FrugalRAG-ColBERTv2 (MuSiQue)	7B	21.14	23.73	8.39

Table 4: Comparison of SFT baselines (Stage 1 and SFT Finish) vs. Ours across datasets.

Method	HotpotQA			2Wiki			Musique		
	Recall	MBE	Searches	Recall	MBE	Searches	Recall	MBE	Searches
E5									
SFT (with FINISH)	68.9	57.8	2.84	52.6	43.8	3.48	52.5	31.8	3.43
FRUGALRAG-Explore	77.0	59.6	10.88	64.3	46.8	10.79	51.7	35.3	10.85
FRUGALRAG	70.4	69.5	2.89	58.8	53.0	3.03	53.3	41.5	3.30
ColBERTv2									
SFT (with FINISH)	79.9	67.7	2.07	63.7	48.5	3.13	51.70	29.80	3.25
FRUGALRAG-Explore	86.4	67.7	5.99	68.9	47.6	5.99	59.1	30.1	5.93
FRUGALRAG	82.8	68.5	2.05	63.5	48.9	2.95	54.0	33.7	3.02

4.4 RESULTS ON DEEP RESEARCH

We evaluate FRUGALRAG on BrowseComp-Plus, a recent, challenging deep research dataset that demands massively multihop search and reasoning. Strikingly, our models trained HotPotQA, MuSiQue, and 2Wiki readily generalize to harder datasets like BrowseComp-Plus. In Table 3 we show that FRUGALRAG issues between 7-10 queries on average despite being trained with a budget of 5 queries. We show that FRUGALRAG achieves superior performance compared to many larger models like Search-R1 32B, Qwen3-32B (Yang et al., 2025), and DeepSeek-R1 (DeepSeek-AI, 2025) on both answer accuracy (MBE) and recall.

4.5 ANALYSIS

FrugalRAG is Adaptive. Table 5 shows that FRUGALRAG adaptively increases test-time compute, measured as the number of searches, with the *difficulty* of the question. To quantify difficulty, we use MuSiQue, which provides explicit labels for the expected number of hops, and 2Wiki, where we use the number of ground-truth evidence documents as a proxy. As shown in Fig. 2, there is a strong correlation between question hardness and the number of queries issued by FRUGALRAG: $r = 0.82$ on 2Wiki and $r = 0.95$ on MuSiQue, demonstrating that FRUGALRAG effectively adapts to multi-hop reasoning challenges.

FrugalRAG is robust. In Table 7, we report results for models trained on HotPotQA, 2Wiki, and MuSiQue and evaluated across all datasets, testing FRUGALRAG's ability to generalize to unseen scenarios. Despite the distribution shift, FRUGALRAG maintains strong performance without compromising efficiency. This robustness holds for both E5-base-v2 and ColBERT-v2 retrievers.

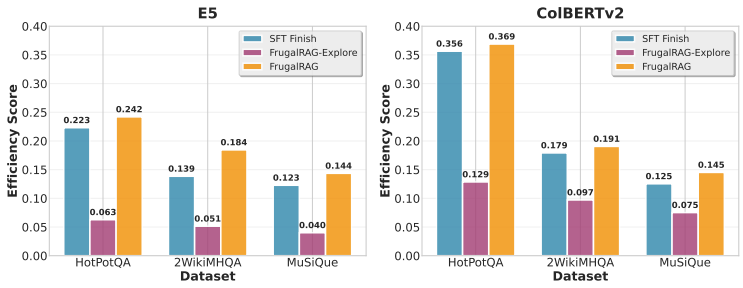

Figure 1: FRUGALRAG on average outperforms fixed budget and SFT based baselines, demonstrating the effectiveness of both Stage-1 finetuning and Stage-2 learning to control test time compute. The plots show the Tradeoff metrics (See Table 4 for detailed results).

FrugalRAG outperforms SFT. To capture the trade-off between answer quality and retrieval cost, we define a simple metric: Efficiency Tradeoff: $\frac{\text{MBE}+\text{Recall}}{2*\text{Searches}}$. This metrics allow us to jointly evaluate answer accuracy and evidence quality relative to the number of retrieval steps, enabling a comparison of efficiency across models and budgets. To assess the impact of the RL-finetuning (Stage 2), we conduct an ablation study by comparing FRUGALRAG against a simple baseline, called SFT (with FINISH). This baseline is trained for 1 epoch on the dataset generated during Stage 1, but with one key modification. Instead of sample 90% rollouts without FINISH, we sample all traces where the model outputs FINISH on its own. In Fig. 1, we demonstrate that FRUGALRAG *on average across all models* is better on both Efficiency Tradeoff. FRUGALRAG outperforms both FRUGALRAG-SFT and FRUGALRAG-Explore by a significant margin using Qwen2.5-7B-Instruct on all the datasets as illustrated in Fig. 1. We provide the metrics in Table 4.

Table 5: FRUGALRAG issues more queries for harder questions. We see a clear increasing trend in num. of search queries with amount of ground truth evidence (2Wiki) and ground truth num. of hops.

2WikiMultiHopQA			**MuSiQue**		
Num GT Evidence	Number of Questions	Searches	Actual Hops	Number of Questions	Searches
2	9,595	2.665 ± 1.430	2	1,252	3.054 ± 1.433
3	88	2.875 ± 1.483	3	760	3.924 ± 1.464
4	2,806	3.909 ± 1.502	4	405	4.205 ± 1.368

5 CONCLUSIONS, LIMITATIONS, AND FUTURE WORK

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 propose 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.

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*, 2025a.

Zijian Chen, Xueguang Ma, Shengyao Zhuang, Ping Nie, Kai Zou, Andrew Liu, Joshua Green, Kshama Patel, Ruoxi Meng, Mingyi Su, Sahel Sharifymoghaddam, Yanxi Li, Haoran Hong, Xinyu Shi, Xuye Liu, Nandan Thakur, Crystina Zhang, Luyu Gao, Wenhu Chen, and Jimmy Lin. Browsecomp-plus: A more fair and transparent evaluation benchmark of deep-research agent. *arXiv preprint arXiv:2508.06600*, 2025b.

DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. Realm: Retrieval-augmented language model pre-training, 2020. URL https://arxiv.org/abs/2002.08909.

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*, 2025a.

Jiajie Jin, Yutao Zhu, Zhicheng Dou, Guanting Dong, Xinyu Yang, Chenghao Zhang, Tong Zhao, Zhao Yang, and Ji-Rong Wen. Flashrag: A modular toolkit for efficient retrieval-augmented generation research. In Guodong Long, Michale Blumestein, Yi Chang, Liane Lewin-Eytan, Zi Helen Huang, and Elad Yom-Tov, editors, *Companion Proceedings of the ACM on Web Conference 2025, WWW 2025, Sydney, NSW, Australia, 28 April 2025 - 2 May 2025*, pages 737–740. ACM, 2025b. doi: 10.1145/3701716.3715313. URL https://doi.org/10.1145/3701716.3715313.

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.

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

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, 2021. URL https://arxiv.org/abs/2005.11401.

Xiaoxi Li, Guanting Dong, Jiajie Jin, Yuyao Zhang, Yujia Zhou, Yutao Zhu, Peitian Zhang, and Zhicheng Dou. Search-o1: Agentic search-enhanced large reasoning models. *arXiv preprint arXiv:2501.05366*, 2025.

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

Jianbiao Mei, Tao Hu, Daocheng Fu, Licheng Wen, Xuemeng Yang, Rong Wu, Pinlong Cai, Xinyu Cai, Xing Gao, Yu Yang, Chengjun Xie, Botian Shi, Yong Liu, and Yu Qiao. O^2-searcher: A searching-based agent model for open-domain open-ended question answering, 2025.

Fabio Petroni, Aleksandra Piktus, Angela Fan, Patrick Lewis, Majid Yazdani, Nicola De Cao, James Thorne, Yacine Jernite, Vladimir Karpukhin, Jean Maillard, Vassilis Plachouras, Tim Rocktäschel, and Sebastian Riedel. Kilt: a benchmark for knowledge intensive language tasks, 2021. URL https://arxiv.org/abs/2009.02252.

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.

Huatong Song, Jinhao Jiang, Yingqian Min, Jie Chen, Zhipeng Chen, Wayne Xin Zhao, Lei Fang, and Ji-Rong Wen. R1-searcher: Incentivizing the search capability in llms via reinforcement learning. *arXiv preprint arXiv:2503.05592*, 2025.

Weihang Su, Yichen Tang, Qingyao Ai, Zhijing Wu, and Yiqun Liu. Dragin: Dynamic retrieval augmented generation based on the information needs of large language models, 2024. URL https://arxiv.org/abs/2403.10081.

Shuang Sun, Huatong Song, Yuhao Wang, Ruiyang Ren, Jinhao Jiang, Junjie Zhang, Fei Bai, Jia Deng, Wayne Xin Zhao, Zheng Liu, Lei Fang, Zhongyuan Wang, and Ji-Rong Wen. Simpledeepsearcher: Deep information seeking via web-powered reasoning trajectory synthesis. *ArXiv*, abs/2505.16834, 2025. URL https://api.semanticscholar.org/CorpusID:278789130.

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, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. Text embeddings by weakly-supervised contrastive pre-training. *arXiv preprint arXiv:2212.03533*, 2022.

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, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

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.

Ori Yoran, Tomer Wolfson, Ori Ram, and Jonathan Berant. Making retrieval-augmented language models robust to irrelevant context, 2024. URL https://arxiv.org/abs/2310.01558.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*, 2025.

Table 6: FrugalRAG is modular: We demonstrate on HotPotQA using the E5-base retriever that FrugalRAG can be used with any answer generator in a plug and play fashion without sacrificing performance.

	HotPotQA		
Method	Recall	MBE	Searches
CoRAG	0.6430	**0.5820**	4.00
CoRAG + FrugalRAG	**0.7040**	0.5750	2.89

Table 7: FrugalRAG demonstrates robust performance: adapted using 1,000 training examples from each of HotPotQA, 2Wiki, and MuSiQue, and evaluated on all datasets. Despite testing on unseen data, FrugalRAG maintains high answer quality and evidence recall while requiring a low number of searches.

	HotPotQA			2Wiki			MuSiQue		
Method Variant	Recall	MBE	Searches	Recall	MBE	Searches	Recall	MBE	Searches
FrugalRAG + E5 (HotPotQA)	70.40	69.50	2.89	60.40	49.66	3.623	53.70	34.10	3.47
FrugalRAG + E5 (2Wiki)	71.00	54.96	2.85	58.80	53.00	3.03	52.22	29.41	3.38
FrugalRAG + E5 (MuSiQue)	69.10	53.95	2.72	59.70	52.20	3.33	53.30	41.50	3.30
FrugalRAG + ColBERTv2 (HotPotQA)	82.80	68.47	2.05	64.20	48.47	3.07	53.80	36.27	2.75
FrugalRAG + ColBERTv2 (2Wiki)	81.90	68.34	2.56	63.50	48.93	2.95	53.80	34.30	3.10
FrugalRAG + ColBERTv2 (MuSiQue)	83.10	61.11	2.53	60.80	46.41	3.13	51.70	29.80	3.02

A Additional Results

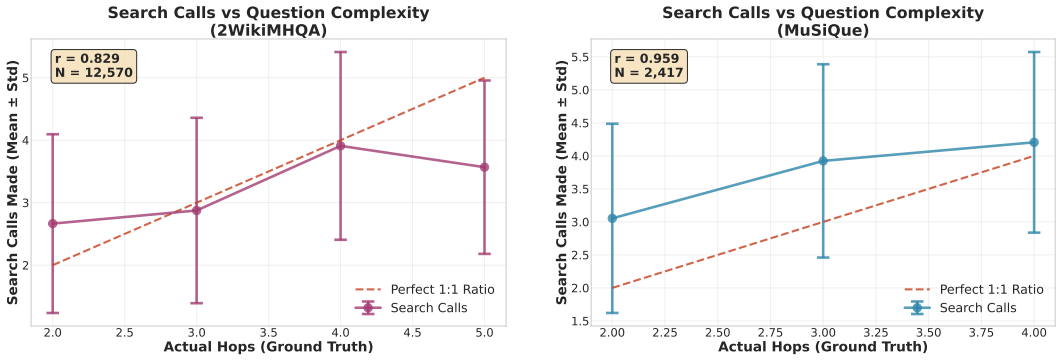

Figure 2: Question complexity and performance

B Training Details

Algorithm 1 shows the overall training framework of FrugalRAG. We plan to publicly release our code soon. 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 GPU using VLLM (Kwon et al., 2023). 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. 3.1, 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 further improving results, we can repeat few shot prompt optimization and dataset generation using different base models.

Supervised "Explore" Finetuning Details. We 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.

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 8. 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 0.1 since we have a trained base policy, and set the maximum reward $R_{\max} = 2.0$ for stability. Generation is limited to a maximum of 256 completion tokens and the maximum prompt size is 1024. Training is conducted using DeepSpeed-Zero2 (Rasley et al., 2020) and 7 H100 GPUs (where 1 is exclusively reserved for sampling). We set the learning rate to 10^{-6}. Due to the long prompt (which includes retrieved documents from previous hops), we use a total batch size of 48. We train FRUGALRAG for 400 steps across datasets and models, and report the performance using the final checkpoint.

C 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. For experiments in Table 2, we use E5-base provided by FlashRAG Jin et al. (2025b) indexed on Wikipedia Petroni et al. (2021) which consists of 21 million passages, and retrieve top 5 documents for all datasets.

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

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`

Perform prompt optimization on f using $\mathcal{L}_{\text{init}}$ to obtain few-shot prompts $\{p_1, \ldots, p_n\}$;

`// 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;

  $\mathcal{H}_0 \leftarrow \{Q^{(j)}, T_0, A_0, \mathcal{D}_0\}$ `// stores previous context`

  Append $\mathcal{H}_0$ to `main_rollout`;

  for $h = 1$ *to* m **do**

    for i *in* $1 \ldots n$ **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: Supervised "Explore" Finetuning`

$f_S \leftarrow$ Fine-tune f on $\mathbf{D}$ using standard next-token prediction `// See Eq. 2`

`// Stage 2: Controlling test-time compute with RL`

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 hop h ;

  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;