Efficiency Improvement of RAG based SLM for Edge Devices

DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Technology

in

Computer Science (with specialization in Data Science)

by

Avanigadda Pavan Prashanth

[Roll No: CS2411]

under the supervision of

Prof. Ujjwal Bhattacharya

Professor

Computer Vision and Pattern Recognition Unit

Indian Statistical Institute

Kolkata 700108, India

June 2026

CERTIFICATE

This is to certify that the dissertation entitled **"Efficiency Improvement of RAG based SLM for Edge Devices"** submitted by **Avanigadda Pavan Prashanth (CS2411)** to the Indian Statistical Institute, Kolkata, in partial fulfillment for the award of the degree of **Master of Technology in Computer Science (with specialization in Data Science)**, is a bonafide record of the work carried out by him under my supervision and guidance.

The dissertation has fulfilled all the requirements as per the regulations of this institute and, in my opinion, has reached the standard required for submission.

Prof. Ujjwal Bhattacharya

Professor,
Computer Vision and Pattern Recognition Unit,
Indian Statistical Institute,
Kolkata 700108, India.

To my parents,
for their unwavering support and constant encouragement.

And to my mentor,
for the guidance and insight that shaped this work.

Acknowledgments

I would like to express my sincere gratitude to my advisor, **Prof. Ujjwal Bhattacharya**, Computer Vision and Pattern Recognition Unit, Indian Statistical Institute, Kolkata, for his invaluable guidance, continuous support, and insightful suggestions throughout the course of this work. His encouragement and expertise have played a crucial role in shaping this dissertation.

I am also thankful to the faculty members and researchers at the Indian Statistical Institute for providing a stimulating academic environment and the necessary resources that enabled the successful completion of this research.

I extend my appreciation to my peers and colleagues for their collaboration, thoughtful discussions, and shared learning experiences, which greatly enriched this journey.

Finally, I am deeply grateful to my family for their unwavering support, patience, and belief in me. Their encouragement and strength have been the foundation of my academic pursuits.

Avanigadda Pavan Prashanth
Indian Statistical Institute
Kolkata 700108, India.

Abstract

The increasing need to deploy language models on constrained devices has given rise to efficiency issues in retrieval-augmented generation (RAG) approaches. Although RAGs boost answers' quality by retrieving knowledge from external sources, current methods utilize static retrieval mechanisms, resulting in unnecessary computation, higher latencies, and inefficiency in resource usage.

In this work, an efficient RAG approach based on small language models (SLMs) is presented, which uses a efficient and adaptive retrieval scheme. This method dynamically changes the retrieval depth and context constrution based on the complexity of the query, using a trained MLP router whose routing decisions are learned from Adaptive-RAG-style oracle labels rather than hand-written rules, leading to a compromise between performance and efficiency.

A full pipeline is provided, including dataset preprocessing, corpus generation, embedding construction, vector indexation, retrieval, and answer generation processes. Experiments were performed on HotpotQA bench mark and SQuAD 2.0 datasets, comparing the presented approach with the baseline RAG approach using static retrieval scheme.

Experimental results show that the proposed approach lowers the computation cost while providing similar answers' quality. By adaptively controlling retrieval and context size, the framework provides an effective solution for deploying RAG systems in constrained environments.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

RAG models have been successfully adopted as a means to leverage dense retrievers with language generation for better question answering systems. Unlike conventional approaches that utilize only the information stored in the model parameters, a RAG system first finds relevant paragraphs from an external source and then uses this information as input for conditioning the generator. This type of model can be particularly useful for tasks requiring knowledge as part of the solution, where answers require access to specific facts rather than being available solely through memorization by the model. Nevertheless, the vast majority of existing RAG models employ a fixed retrieval budget for all queries, irrespective of their complexity.

The main contribution of this thesis lies in the improvement of RAG systems with respect to limited computational and memory resources. This will be achieved by designing an adaptive router that takes into consideration the budget for retrieval and prompt construction based on the actual requirements of the query before generating an answer. The router in this work is a trained MLP classifier whose labels are derived from an Adaptive-RAG-style oracle, grounding routing decisions in observed system performance. This router will help to minimize excessive retrieval, create shorter prompts wherever feasible and maintain the quality of answer generation in difficult queries. For evaluation, this thesis will use HotpotQA and SQuAD 2.0 datasets, as they allow the study of RAG for both multi-hop and single hop question answering approaches. The use of HotpotQA is especially important since it involves reasoning over multiple facts, making it a good testbed for adaptive retrieval.

1.1 Motivation

The motivation behind this work arises due to three challenges faced by state-of-the-art RAG models. First, the retrieval process uses the same value for top-k irrespective of how complex the question is. It can happen that a simple question might require just one or two retrieved passages, while a complex question may require more generalized retrieval. Using the same top-k value makes no sense in these scenarios. Second, when there is a constrained environment such as Edge devices, any addition of an extra passage adds up to the tokenization cost and latency. Third, the use of irrelevant or noise-ridden information in retrieved passages could lead to low-quality generated answers.

These problems become clearer when working with HotpotQA, where the model needs to integrate data across several documents for its answer. In such scenarios, having only more documents does not help. Instead, there should be a way to know whether to retrieve more, or less, and what data should be retrieved while maintaining a budget limit. Therefore, this thesis explores the relation between the complexity of the query, the retrieval budget, and the quality of the answer generated.

1.2 Problem Statement

Define q to be the question, C to be a set of documents from which to retrieve from, and b to be the budget allocated, measured by tokens, time, and memory. An ordinary RAG model utilizes the same approach to retrieval and generation for all queries irrespective of their difficulty q, and the budget b. It typically ends up oversampling for easy queries, and undersampling for harder queries.

The research problem posed in this work is to create a retrieval augmented generation pipeline such that it is capable of choosing different actions for retrieval and context sampling based on how difficult a question is expected to be and its resource budget, without sacrificing answer quality. Specifically, given a budget state b, find an optimal action $a \in \mathcal{A}$.

$$a^* = \arg\max_{a \in \mathcal{A}} U(a \mid q, b),$$

where the utility function rewards answer quality and penalizes retrieval cost,

prompt length, and latency. The challenge is to make this decision in a way that remains practical, robust, and easy to evaluate on benchmark datasets.

1.3 Research Objectives

The main objectives of this dissertation are as follows.

1. To develop a complete baseline RAG pipeline which includes preprocessing, corpus creation, embedding creation, indexing, retrieval, prompt construction, answer generation, and evaluation.
2. To develop a budget-constrained router that determines the optimal retrieval budget required for the given query based on its complexity.
3. To perform a comparison between fixed budget and adaptive budget retrieval approaches using different metrics including answer accuracy, retrieval quality, latency, token usage, and memory efficiency.
4. To evaluate the performance of the system developed on HotpotQA and SQuAD 2.0 datasets to study the performance of the router in multi-hop and simpler question answering tasks respectively.
5. To develop a modular and reproducible implementation to extend it for various future extensions like reranking, compression and edge deployment.

1.4 Research Questions

This dissertation is guided by the following research questions.

1. Is there a efficient router that can increase the efficiency of the RAG system without compromising its quality significantly?
2. What amount of information should a query require to be retrieved, and how can we estimate it prior to the generation stage?
3. Does adaptive retrieval benefit more from the multi-hop queries than simple ones?
4. Among the efficiency factors, what part contributes the most to efficiency improvement, namely, reduced retrieval depth, compression, or

confidence correction?

5. How does our proposed system perform compared to the fixed-budget baseline on retrieval and generation performance indicators?

1.5 Thesis Contributions

The major contributions of this dissertation are as follows.

- An end-to-end pipeline for baseline RAG involving distinct modules for preprocessing, corpus construction, embedding computation, FAISS index creation, retrieval, and generation.
- A trained two-hidden-layer MLP router whose retrieval-budget decisions are learned from oracle labels.
- A benchmarking suite for evaluating the performance of the RAG model based on metrics like Recall@k, Precision@k, MRR, faithfulness, answer relevance, context precision, context recall.
- An experimental design involving the use of two different datasets, namely HotpotQA and SQuAD 2.0, to evaluate the proposed solution in both multi-hop and simple QA tasks.
- A well-designed implementation that is highly modular, reproducible, and easily extensible.

1.6 Thesis Organization

The rest of this dissertation is structured as follows.

Chapter 2 outlines the foundational concepts needed to understand this research including retrieval-augmented generation, embeddings, vector indexing, and evaluation methods. Chapter 3 surveys the existing literature about RAG, adaptive retrieval, re-ranking, compression, and efficient question answering and highlights the problem this dissertation aims to solve. Chapter 4 discusses the proposed methodology and covers the baseline RAG pipeline, the trained MLP router with oracle labelling, and the evaluation procedure used in this research. Chapter 5 showcases experimental results and compares the baseline method with the adaptive approach on HotpotQA and SQuAD

2.0 benchmarks. Finally, Chapter 6 concludes the dissertation and highlights possible directions for further work.

Chapter 2

Background and Preliminaries

In this chapter, the basic ideas that will help comprehend the research will be introduced. The scope of the research will include RAG, dense retrieval, vector search, and budget-friendly question answering. The aim of this chapter is to lay down the language, mathematics, and system ideas that will be used throughout the research.

2.1 Retrieval-Augmented Generation

RAG is a framework that incorporates information retrieval and language generation together. Unlike a traditional language generation pipeline that solely relies on the internal parameter of the language model to generate a response, a RAG system retrieves the necessary context first, and the generator takes that context into consideration while generating. RAG is particularly valuable in scenarios where the answer to the question depends on a certain fact, recent news, or a lot of contextual information that the language model might not have memorized.

The process of retrieving context and generating an answer in a RAG pipeline works as follows: the question gets represented by a query and relevant passages get retrieved from the corpus, and then the retrieved passages get inserted in the generator's prompt or context. Finally, an answer to the question gets generated based on both the query and the retrieved passages. Although RAG enhances the accuracy of generated responses, it comes at

an additional cost of retrieval time and memory and prompt length. A key limitation of existing RAG systems is that the retrieval budget is fixed regardless of query complexity; this dissertation addresses that limitation through a trained adaptive router that selects the budget from the query embedding.

2.1.1 Basic RAG Pipeline

A conventional RAG pipeline can be described using the following steps:

1. Convert the user query into an embedding or retrieval representation.
2. Search an external passage collection for the most relevant evidence.
3. Select the top-k retrieved passages or documents.
4. Assemble the query and retrieved evidence into a prompt.
5. Use a language model to generate the final answer.

2.1.2 Retriever

The retriever module is the one that takes care of retrieving passages from the corpus that are relevant to the queries. With dense retrievers, both the queries and the retrieved passages are projected into an embedding space where their similarity is computed via nearest neighbor search. Dense retrievers work well even when the query and the retrieved passages have little in common lexically but share a semantic connection.

2.1.3 Generator

The generator is the language model which generates the output based on the question and evidence retrieved. In the RAG model, the generator is expected to be instruction following, brief, and true to the evidence provided. However, an ineffective generator can choose to either ignore the evidence provided, hallucinate without any support, or provide very long answers. This means that the effectiveness of generation is dependent on prompt design and the model used.

2.1.4 Why Fixed-Budget RAG Is Limited

Another popular approach to building RAG systems is to perform a fixed number of retrievals for all queries. While easy to implement, this approach is typically not efficient. While easy queries may not require much evidence at all, more difficult multi-hop queries can benefit from larger retrievals and iterative evidence selection. Hence, the problem with using a fixed retrieval budget is twofold:

- unnecessary retrieval and prompt size increase for easy queries;
- lack of evidence for difficult queries due to a small fixed retrieval budget.

2.2 Large Language Models and Small Language Models

LLM stands for Large Language Model – this refers to neural language models that have a very large number of parameters, ranging anywhere between tens of billions and trillions. LLMs are trained on vast quantities of texts for prediction and generation purposes. LLMs are fluent, generalize well and have good coverage of knowledge. The downside to LLMs is that they require more computing power, memory, and energy than other methods.

SLM stands for Small Language Model. This type of model includes neural language models that have a relatively small number of parameters, ranging between a few million to about 10 billion, with most practical applications being between 1–8 billion parameters. Despite their relative size, SLMs are able to generate text effectively, and they are even more efficient if used in tandem with text generation strategies like the RAG.

The distinction between the two is pertinent to this work because it allows us to see where we can optimize RAG by using smaller generators and more efficient text retrieval and routing techniques.

2.2.1 Why Model Size Matters

Model size affects:

- inference latency,

- memory usage,
- prompt handling capability,
- deployment feasibility under resource constraints.

For a efficient system, the goal is not simply to maximize model size, but to balance the quality of the output with the cost of producing it.

2.3 Text Embeddings and Dense Retrieval

Embeddings refer to vectors of fixed lengths where semantics in texts are encoded. In dense retrieval, the query and passage both get mapped to vectors, and the similarity between the two is computed in the vector space. Texts with similar semantics should be near one another irrespective of how similar the words may or may not be.

Let e_q denote the query embedding and e_p denote the passage embedding. A common similarity function is cosine similarity:

$$\text{sim}(e_q, e_p) = \frac{e_q \cdot e_p}{\|e_q\|\|e_p\|}.$$

The retriever then ranks passages by this similarity score.

Dense retrieval is especially helpful for HotpotQA because many questions require connecting information from different supporting passages. Exact word matching is often not enough, so semantic retrieval is preferable.

2.3.1 Properties of Useful Embedding Models

A good embedding model for retrieval should:

- preserve semantic similarity,
- work well on short queries and longer passages,
- be efficient enough for batch encoding,
- support stable indexing and nearest-neighbor search.

In this dissertation, embeddings are used both for corpus construction and for query encoding, which keeps the retrieval space consistent.

2.4 Vector Similarity Search and FAISS

Once embeddings are generated, the corpus needs to be stored in a structure that supports efficient nearest-neighbor search. FAISS (Facebook AI Similarity Search) is a widely used library for this purpose. It is designed for fast similarity search over dense vectors and supports exact as well as approximate retrieval strategies.

If the corpus contains N passage embeddings $\{e_1, e_2, \ldots, e_N\}$, then the retrieval task is to find the top-k vectors most similar to the query vector e_q. FAISS efficiently performs this search using specialized indexing structures.

2.4.1 Why FAISS Is Useful

FAISS is suitable for this dissertation because:

- it works well with dense embeddings,
- it is efficient for large passage collections,
- it can be used on CPU or GPU,
- it supports experimentation with different index types.

FAISS is therefore a practical choice because it provides a clear and reproducible retrieval backbone.

2.4.2 Indexing Intuition

The index stores passage vectors and enables fast search by approximate or exact distance comparison. The result of a search is a ranked list of passage identifiers and scores. These retrieved passages are then passed to the generator as evidence.

2.5 Multilayer Perceptron as a Query Router

A multilayer perceptron (MLP) is a class of feedforward artificial neural network consisting of an input layer, one or more hidden layers, and an output layer. Each layer is composed of neurons that apply a linear transformation followed by a non-linear activation function. MLPs are universal function

approximators, meaning that with sufficient width and depth they can learn arbitrarily complex input-output mappings from labelled data.

2.5.1 Architecture

For a network with two hidden layers, the forward pass computes:

$$h_1 = \sigma(W_1 e_q + b_1), \quad h_2 = \sigma(W_2 h_1 + b_2), \quad \hat{y} = W_3 h_2 + b_3,$$

where $e_q \in \mathbb{R}^d$ is the input (the query embedding in this work), W_i and b_i are learnable weight matrices and bias vectors, $\sigma(\cdot)$ is an element-wise non-linear activation, and $\hat{y} \in \mathbb{R}^C$ is the output logit vector over C classes. A softmax over $\hat{y}$ gives class probabilities, and the predicted class is $\arg\max \hat{y}$.

In this dissertation the MLP router has the following configuration:

$$\underbrace{\mathbb{R}^{384}}_{\text{input}} \xrightarrow{W_1} \underbrace{\mathbb{R}^{256}}_{\text{hidden 1}} \xrightarrow{\text{ReLU}} \xrightarrow{\text{Dropout}(0.3)} \xrightarrow{W_2} \underbrace{\mathbb{R}^{64}}_{\text{hidden 2}} \xrightarrow{\text{ReLU}} \xrightarrow{W_3} \underbrace{\mathbb{R}^{3}}_{\text{output}},$$

where the 384-dimensional input is the L2-normalised query embedding from MiniLM and the three output logits correspond to the budget tiers `Easy`, `Medium`, and `Hard`.

2.5.2 Activation Functions and Regularisation

The Rectified Linear Unit (ReLU) is used as the hidden-layer activation:

$$\text{ReLU}(x) = \max(0, x).$$

ReLU introduces non-linearity without vanishing-gradient problems and is computationally cheap, which is appropriate for a lightweight routing module. Dropout with probability 0.3 is applied after the first hidden layer to reduce overfitting on the relatively small training set produced by the oracle labelling procedure.

2.5.3 Training Objective

The MLP is trained by minimising the class-weighted cross-entropy loss. For a batch of N examples with one-hot labels y_i and predicted log-probabilities

$\log \hat{p}_i$,

$$\mathcal{L} = -\frac{1}{N} \sum_{i=1}^{N} \sum_{c=1}^{C} w_c \cdot y_{ic} \log \hat{p}_{ic},$$

where $w_c = \frac{N}{C \cdot N_c}$ is the inverse-frequency class weight for tier c and N_c is the number of training examples in that tier. Class weighting compensates for any imbalance in the oracle label distribution so that the router does not systematically ignore the less frequent Hard tier.

Optimisation is performed with AdamW, which combines adaptive learning rates with decoupled weight decay:

$$\theta_{t+1} = \theta_t - \eta \left(\frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} + \lambda \theta_t \right),$$

where η is the learning rate, $\hat{m}_t$ and $\hat{v}_t$ are bias-corrected first and second moment estimates, and λ is the weight-decay coefficient. The router is trained for 60 epochs with batch size 64, learning rate 10^{-3}, and weight decay 10^{-4}.

2.5.4 Why MLP Is Suitable as a Router

The MLP is well suited to the routing task for several reasons.

- **Lightweight inference.** A two-hidden-layer MLP with 256 and 64 units requires only a single forward pass of approximately 115,000 floating-point multiplications, adding negligible latency compared to FAISS search or generation.
- **Learned decision boundaries.** Unlike hand-written lexical rules, the MLP learns non-linear boundaries directly from the data, allowing it to capture patterns in the query embedding that are difficult to express as keyword conditions.
- **No probe retrieval needed.** Because the router reads only the 384-dimensional query embedding, it does not require a preliminary FAISS search to assess retrieval confidence, saving one round-trip to the index per query.
- **Compact and deployable.** The trained weight file is small (under 2 MB), making it straightforward to deploy alongside the embedder and the generator on a constrained device.

2.6 Evaluation Metrics

A major part of this dissertation is the evaluation of both quality and efficiency. The system is evaluated using retrieval metrics, generation metrics, and efficiency-related measures. This section introduces the key metrics used later in the thesis.

2.6.1 Retrieval Metrics

Retrieval metrics measure how well the retriever finds relevant evidence.

Recall@k

Recall@k measures the proportion of gold evidence items that appear in the top-k retrieved passages. Let G be the set of gold evidence passages and R_k be the retrieved top-k passages. Then

$$\text{Recall@k} = \frac{|G \cap R_k|}{|G|}.$$

Precision@k

Precision@k measures the fraction of retrieved top-k passages that are relevant:

$$\text{Precision@k} = \frac{|G \cap R_k|}{|R_k|}.$$

Mean Reciprocal Rank (MRR)

MRR rewards systems that place a relevant passage early in the ranked list. If the first relevant passage appears at rank r_i for query i, then the reciprocal rank is $1/r_i$. Over N queries,

$$\text{MRR} = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{r_i},$$

where r_i is the rank of the first relevant passage, and if no relevant passage is retrieved, the contribution is 0.

2.6.2 Generation Metrics

Generation metrics measure how well the final answer matches the expected output and how faithfully it uses the evidence.

Exact Match

Exact Match checks whether the predicted answer exactly matches the gold answer after normalization:

$$\mathrm{EM} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}[\mathrm{norm}(\hat{y}_i) = \mathrm{norm}(y_i)] \,,$$

where $\mathrm{norm}(\cdot)$ denotes answer normalization and $\mathbb{I}$ is the indicator function.

Token F1

Token-level precision and recall are computed from the overlap between predicted and gold answer tokens. For a single example,

$$P = \frac{|T_{\hat{y}} \cap T_y|}{|T_{\hat{y}}|}, \quad R = \frac{|T_{\hat{y}} \cap T_y|}{|T_y|},$$

and

$$\mathrm{F1} = \frac{2PR}{P+R}.$$

The final score is averaged across examples.

Answer Relevance

Answer relevance measures how directly the generated answer addresses the question. One way to view it is as semantic similarity between the question and the answer.

Context Precision and Context Recall

Context precision measures how much of the retrieved context is actually relevant:

$$\mathrm{ContextPrecision} = \frac{\text{relevant retrieved context}}{\text{all retrieved context}}.$$

Context recall measures how much of the needed evidence was successfully retrieved:

$$\text{ContextRecall} = \frac{\text{retrieved gold evidence}}{\text{all gold evidence}}.$$

2.6.3 Efficiency Metrics

Because the dissertation is focused on efficient retrieval, efficiency is also central to evaluation. The following measures are important:

- retrieval latency,
- generation latency,
- prompt length in tokens,
- number of retrieved passages used,
- memory usage where measurable.

These metrics allow the baseline system and the adaptive router to be compared not just on answer quality, but also on resource cost.

2.6.4 RAGAS Metrics

RAGAS is a framework for evaluating retrieval-augmented generation systems using context-aware and faithfulness-aware metrics. It is useful because it assesses whether the answer is grounded in retrieved evidence rather than only comparing surface similarity with the gold answer.

The output is a vector of metric scores such as faithfulness, answer relevance, context precision, and context recall.

Chapter 3

Literature Review

In this chapter, we will discuss the research lines that have laid down the groundwork for the proposed dissertation. The unifying concept across the research is retrieval-augmented generation in the presence of limitations and more specifically, whether retrieval is needed, what amount of data to retrieve, and how to optimally utilize the data that has been retrieved. In organizing the relevant literature, the following five groups can be discerned: retrieval-augmented generation foundations, dense retrieval and multipassage reading, adaptive retrieval and self-reflection, prompt compression, and efficient edge inference.

3.1 Foundations of RAG

Modern RAG models were proposed in the context of augmenting a parametric model with an external memory through non-parametric retrieval. Specifically, in *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*, Lewis et al. (2020) propose conditioning a sequence-to-sequence model on retrieved Wikipedia passages instead of using only its parameters. This is based on the observation that retrieval helps with improving factuality, specificity, and modularity of generated responses while enabling updates to the knowledge source independent of the generation model itself. Additionally, they demonstrated that a retrieved memory architecture can outperform parametric-only approaches on multiple knowledge-intensive tasks.

Another critical contribution is *Dense Passage Retrieval for Open-Domain Question Answering* by Karpukhin et al. (2020). In their work, DPR replaces traditional sparse lexical methods of passage retrieval with dense vector representation of both queries and passages. This results in highly efficient semantic matching of queries and passages suitable for open-domain question answering tasks. DPR is important in the context of this dissertation since it demonstrates a practically viable dual-encoder retrieval model that other RAG architectures adopt by default.

3.1.1 Key lessons from the foundational work

Four main concepts from the literature review become evident, which will play a role in this dissertation. The first concept is the one of enhancing factuality and availability of information through external retrieval. The second is that dense retrieval is preferable to strict lexical match in semantically searching for answers. The third idea is the usage of the retrieved information by the generator as supporting data and not as an independent information source. Finally, the quantity and quality of the retrieved information passages influence answer quality.

3.2 Dense Retrieval and Multi-Passage Reading

In open-domain question answering tasks, multiple passages may need to be reasoned over compared to just one snippet of information. Research on multi-passage reading hence becomes very valuable for designing an appropriate RAG reader model.

With respect to multi-hop reasoning, in particular, the work entitled *Modeling Multi-hop Question Answering as Single Sequence Prediction* (`https://arxiv.org/abs/2205.09226`) by Yavuz et al. (2022) explores how to extend generative question answering using single sequence prediction for modeling the reasoning steps. As HotpotQA involves multiple facts per question, the work is particularly applicable to HotpotQA. The authors claim that a multi-hop reasoning task can be made easier to solve by revealing the reasoning structure to the model.

3.2.1 Implications for this dissertation

Based on these findings, it is clear that besides retrieval recall, another key problem in RAG is the quality of passages, passage ranking, and the fusion of evidence through generation. The research presented in this dissertation takes cues from this finding in that it maintains a simple baseline pipeline while augmenting it with a routing mechanism that determines if a query requires either a small or large evidence set.

3.3 Adaptive Retrieval and Self-Reflection

The primary disadvantage of traditional RAG models lies in their inability to adaptively retrieve, meaning that a fixed retrieval strategy is applied regardless of the query's requirements. In recent years, research in this area has been focused on developing methods for adaptive retrieval when the model itself determines the necessity to retrieve information and the amount required for processing the query.

The work by Asai et al. titled "*Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection*" ([2023]) makes an important contribution in this regard. The self-retrieval mechanism allows training one unified model to learn when to retrieve information, when to answer without retrieving anything, and even critique their answers using special reflection tokens. The primary contribution made by the authors of this paper is the conditional nature of retrieval, which fully corresponds to the hypothesis formulated above.

Another very related paper is *Adaptive-RAG: Learning to Adapt Retrieval-Augmented Large Language Models through Question Complexity* by Jeong et al. (2024), which classifies the complexity of a question and accordingly chooses whether to employ simpler or advanced forms of retrieval-augmentation. This is quite similar to a efficient router since it uses the difficulty of a query as a basis for routing. A smaller classification model or a selector in such cases would help avoid wastage of computation resources on simple queries without hampering advanced approaches on complex queries.

Both these papers make up the theoretical basis of the proposed routing system. While Self-RAG focuses on self-reflection, Adaptive-RAG focuses on classification and choosing appropriate strategies based on complexity.

3.3.1 Why adaptive retrieval matters

Adaptive retrieval is important because the complexity of queries varies. There are some queries that can be answered using only one piece of evidence, but there are also those that will require more than one pieces of evidence. A routing mechanism that can distinguish between complex and simple queries will save retrieval and minimize prompt lengths without compromising answers to complex queries.

3.4 Prompt Compression and Long-Context Efficiency

Even with high-quality retrieval, a long prompt can introduce extra computation costs and divert the attention of the generator away from the most crucial pieces of evidence. Hence, another research direction for efficient RAG is prompt compression.

The paper *LongLLMLingua: Accelerating and Enhancing LLMs in Long Context Scenarios via Prompt Compression* by Jiang et al. (2023) deals with this issue by compressing prompts without loss of the essential elements needed by the model. It proves that a properly compressed prompt will decrease both the token count and latency of the model while retaining the answer quality to a large extent. This work is especially pertinent to the dissertation since a efficient router does not only optimize retrieval quantity; it also needs the ability to condense the retrieved evidence before generation.

There are two reasons why LongLLMLingua is relevant to the dissertation. On one hand, it shows that the prompt length is not an unintended side effect of retrieval, but rather a consciously chosen approach. On the other hand, the paper demonstrates that it is possible to combine context compression and selection to achieve increased efficiency without completely sacrificing answer quality.

3.4.1 Why compression is useful

Compression makes sense since retrieval retrieves redundant or semantically similar information. When all the retrieved segments are inputted into the generator, this can make the prompt unnecessarily verbose. This process of

selecting important content while filtering out low-quality content is precisely what a budget-based algorithm should do.

3.5 Edge-Efficient and Long-Context Inference

There exists yet another body of work dealing with how to efficiently implement retrieval and reasoning under constraints on memory and latency. Here are relevant papers which demonstrate how memory- and latency-conscious methods and systems for managing resources could enhance RAG models.

The work *EdgeRAG: Online-Indexed RAG for Edge Devices*, authored by Seemakhupt et al. (2024), addresses directly retrieval-augmented generation with the focus on memory limitations. The EdgeRAG system utilizes the approach to indexing in which embeddings are pruned, some embeddings are computed on-the-fly, and cached entries are used in an adaptive way to ensure both low latency and fit within memory constraints. This is crucial to the dissertation because, unlike many other works on retrieval-augmented generation, it shows that the retrieval infrastructure itself can be made adaptive, as well as the querying strategy.

In terms of contributions, EdgeRAG can be considered valuable for two reasons. First, it shows that the retrieval index can be thought of as a dynamically manageable resource instead of being just a passive store. Secondly, it emphasizes the importance of adaptive management of embeddings, especially in resource-constrained environments.

3.5.1 Implications for the proposed system

Based on the efficiency-related literature, one can say that the design space for the Rag system is more than retrieving the right information. Corpus index, prompt length, caching mechanism, and inference strategy are the other factors contributing to the budget. My thesis takes an analogous approach and considers the budget to be a primary parameter for retrieval.

3.6 Research Gaps and Positioning of This Work

A few facts stand out from the surveyed literature. First, retrieval-augmented generation has become an established approach in knowledge-intensive applications, although retrieval is often done following a predetermined strategy regardless of the complexity of the question. Second, dense retrieval and passage aggregation benefit answer quality, although they may become costly due to noise in the context or overly long contexts. Third, adaptivity of retrieval and prompt minimization both tackle the problem of inefficiency, but most previous approaches deal with either of the stages separately.

This dissertation lies on the crossroad between these streams of research. In addition to developing the baseline retrieval-augmented generation model, we propose a efficient adaptive router to predict how much evidence is needed per question. Contrary to the static fixed-k pipeline, our system seeks to optimize the trade-off between the quality of answers and the number of tokens consumed, the time spent, and the amount of memory used. Unlike prompt compression techniques, it addresses retrieval depth, re-ranking, and the use of contextual information as interconnected components of the overall decision policy.

This leads to the main research hypothesis proposed in the dissertation: *a query-aware budget policy could eliminate redundant retrievals, induce growth, and achieve answer accuracy that is competitive with existing approaches for multi-hop and simple question answering tasks.*

Chapter 4

Proposed Methodology

This chapter presents the complete methodology of the dissertation. Two systems are designed, implemented, and compared: (i) a Baseline RAG pipeline that retrieves a fixed number of passages for every query, and (ii) An Efficient RAG Router that dynamically selects the cheapest retrieval and generation strategy likely to preserve answer quality.

4.1 Overview of the Proposed Framework

The proposed architecture is comprised of one common retrieval backbone along with two inference methods. The common retrieval backbone will include steps like dataset pre-processing, creation of corpus, generation of dense embeddings, and vector indexing using FAISS. In contrast, while both the baseline and budget aware systems will utilize the same backbone, the former will always perform fixed retrieval and generation for all queries, whereas the latter will predict the query's budget tier using a trained MLP router before performing retrieval.

The entire system architecture is devised in such a manner that both the systems will have the same corpus, embedding space, and benchmarks. Hence, the comparison will be made based on routing/retrieval strategy alone and not due to any differences in data/model stack.

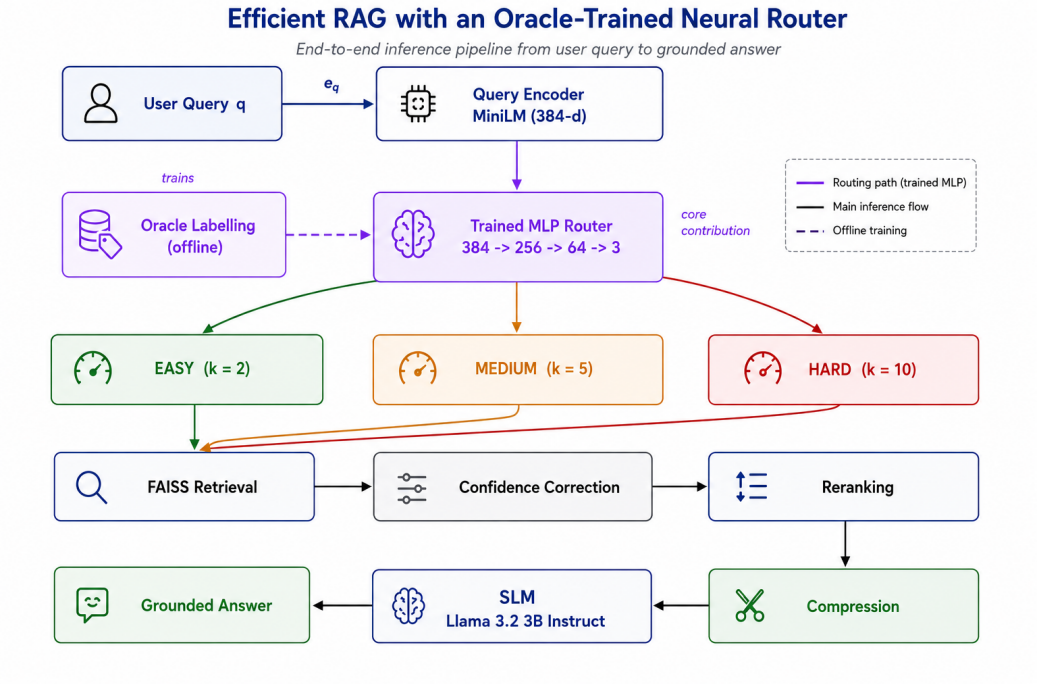


Figure 4.1: Overview of the proposed methodology.

4.2 Baseline Retrieval-Augmented Generation Pipeline

The basic pipeline is used as the benchmark to compare our new cost-conscious routing algorithm. It consists of the standard retrieval and generation setup, where the question is encoded, evidence is fetched from a vector store, then injected into the prompt and the response is generated by the language model.

The baseline is intentionally simple so that it can serve as a strong and transparent reference point for the adaptive system.

4.2.1 Dataset Preparation and Preprocessing

Benchmark data used by the dissertation includes multi-hop question answering datasets which support retrieval and reasoning. For multi-hop QA tasks, HotpotQA is considered the benchmark, because it requires combining

evidence from multiple supporting facts. SQuAD 2.0 serves as another benchmark for evaluating the system in terms of behavior during simple question answering tasks.

The preprocessing stage involves structuring the data in such a way as to prepare it for retrieval. In particular, for each example, the system stores a question, an answer, supporting facts, and context documents. Context documents undergo cleaning and normalization, and are turned into retrieval passages. Malformed text, boilerplate repetitions, and extraneous formatting elements are filtered out. Moreover, the preprocessing step retains titles, since these can serve as useful cues for retrieval as well as the interpretation of retrieved evidence.

Preprocessed structured data can then be utilized to create retrieval documents and evaluation sets.

4.2.2 Retrieval Corpus Construction

The retrieval corpus is built by converting the cleaned context documents into passage-level textual units. Each passage is assigned a unique identifier so that the passage can later be traced back to its source document and sentence position.

Formally, if a context document has title t and sentence sequence $\{s_0, s_1, \ldots, s_m\}$, then the corpus contains passages

$$p_j = (t, s_j), \quad j = 0, 1, \ldots, m.$$

This passage-based representation is suitable for dense retrieval because it keeps the evidence granular enough for sentence-level matching while retaining the document title as metadata.

4.2.3 Dense Embedding Generation

Each passage is transformed into a dense vector representation using a shared embedding encoder. Let f_θ denote the embedding model and let $e_i = f_\theta(p_i)$ be the embedding of passage p_i. For a question q, the query is embedded as

$$e_q = f_\theta(q).$$

The semantic similarity between the query and a passage is computed using cosine similarity:

$$\text{sim}(e_q, e_i) = \frac{e_q \cdot e_i}{\|e_q\|\|e_i\|}.$$

The dense encoding step plays a vital role as it enables semantic search irrespective of the presence of lexical similarity between the question and the passage. In case of HotpotQA, for example, relevant information could be phrased in words other than those in the supporting context.

4.2.4 FAISS Index Construction

Once the embeddings have been created, they are saved into a vector index to facilitate the process of searching for nearest neighbors. FAISS acts as the retrieval engine since it offers a convenient way to search dense vectors. All passages are saved in the index and the most similar passages to a query are returned.

The retrieved passage identifiers are then mapped back to the original passage text and metadata.

4.2.5 Prompt Assembly and Answer Generation

The retrieved paragraphs are placed in a prompt along with the query posed by the user. The language model must respond using only the facts provided in the paragraph and make sure that the answer stays short and precise. This is essential since the model must be steered towards evidence-based answers.

The prompt is deliberately short and evidence-focused so that the generator can condition on relevant evidence without being overloaded by irrelevant text.

4.3 Efficient Retrieval Router

The core contribution of this dissertation is an efficient adaptive routing module that decides how much retrieval a query actually needs. The router is placed before the retrieval and generation stages and selects an action based on the predicted query complexity and the current resource budget. In contrast to the original heuristic design that used a lexical complexity score

and a retrieval probe, the router in this work is a trained two-hidden-layer MLP that reads the query embedding directly and predicts one of three retrieval-budget tiers.

4.3.1 Problem Formulation

Let q denote the input query and let b denote the current budget state. The router chooses an action a from a finite action space $\mathcal{A}$:

$$a^* = \arg\max_{a \in \mathcal{A}} U(a \mid q, b),$$

where $U(\cdot)$ is a utility function that rewards answer quality and penalizes retrieval cost, prompt length, latency, and memory usage.

A useful utility formulation is

$$U(a \mid q, b) = \alpha Q(a, q) - \beta C_{\text{tok}}(a, b) - \gamma C_{\text{lat}}(a, b) - \delta C_{\text{mem}}(a, b),$$

where:

- $Q(a, q)$ estimates answer quality for action a and query q,
- C_{tok} measures token cost,
- C_{lat} measures latency,
- C_{mem} measures memory usage,
- $\alpha, \beta, \gamma, \delta$ control the trade-off between quality and efficiency.

4.3.2 Trained MLP Router and Oracle Labelling

The routing decision is framed as a classification problem:

$$\hat{c} = f_\psi(e_q),$$

where $e_q = f_\theta(q)$ is the 384-dimensional L2-normalised query embedding produced by MiniLM, and $\hat{c} \in \{\texttt{Easy}, \texttt{Medium}, \texttt{Hard}\}$ is the predicted budget tier. Unlike the original design, which used hand-written lexical rules and a retrieval probe, f_ψ is a trained neural network that reads the embedding directly — no keyword lists or probe retrieval are used at inference.

Router architecture. The router is a two-hidden-layer MLP:

$$\texttt{Input}(384) \to \texttt{Linear}(256) \to \texttt{ReLU} \to \texttt{Dropout}(0.3) \to \texttt{Linear}(64) \to \texttt{ReLU} \to \texttt{Linear}(3)$$

The output layer produces three logits, one per tier. A softmax followed by argmax gives the routing decision. The same architecture is used both during training and at inference, so the two cannot diverge.

Oracle labelling. The training labels are produced using the same oracle methodology. For each training question, the retrieve-and-read pipeline is executed at every tier in increasing cost order. A tier is considered correct if it yields an exact match or a token-F1 of at least 0.6 against the gold answer. The label assigned is the cheapest correct tier:

$$\text{label}(q) = \begin{cases} \texttt{Easy} & \text{if Easy tier is correct} \\ \texttt{Medium} & \text{else if Medium tier is correct} \\ \texttt{Hard} & \text{otherwise (or dataset-bias fallback)} \end{cases}$$

If no tier answers correctly, dataset provenance breaks the tie: multi-hop sources (HotpotQA) default to `Hard` and single-hop sources default to `Medium`. A single shared retrieval corpus, pooled from all training contexts, is used for labelling. The router is trained on SQuAD 2.0, Natural Questions, TriviaQA, and HotpotQA (300 questions each).

The purpose of this stage is not to answer the question directly, but to ground the routing labels in observed system performance rather than keyword patterns. A simple factual question answerable at k=2 gets the `Easy` label; a multi-hop question that needs k=10 gets the `Hard` label.

4.3.3 Action Space

The router selects among three retrieval and context-processing strategies. The no-retrieval action of the original design is removed; every query now retrieves at least k=2 passages, which protects quality on borderline questions.

Tier	k	**Context budget**	**Max new tokens**	**Reranking**
Easy	2	600 chars	64	No
Medium	5	1200 chars	96	Yes
Hard	10	2000 chars	128	Yes

Table 4.1: Action space (three tiers) used by the efficient router.

4.3.4 Adaptive Retrieval

Once the action has been chosen, the router sets the retrieval depth accordingly. The adaptive retrieval mechanism can therefore be written as

$$k = g(a, q, b),$$

where g maps the selected action and the query/budget state to a retrieval depth $k \in \{2, 5, 10\}$. Every tier retrieves at least k=2 passages, so no query is answered without any evidence.

The objective is to avoid unnecessary retrieval when the query is easy and to increase evidence coverage when the query is hard. This differs from a fixed-k baseline, where all questions receive the same retrieval budget.

4.3.5 Reranking

Reranking is used to improve the ordering of retrieved passages before they are passed to the generator. Let $\{p_1, \ldots, p_m\}$ be the initial retrieved candidates. Reranking is applied for `Medium` and `Hard` tier queries after initial retrieval, using the cross-encoder model fine-tuned for passage relevance scoring. A reranker computes a refined score:

$$s_i^{\text{rerank}} = h(q, p_i),$$

where h can combine dense similarity, lexical overlap, and title overlap. The passages are then sorted by the reranker score before prompt assembly.

Reranking is useful because the first-stage retriever can return semantically related but not necessarily answer-bearing passages. A reranker helps keep the most useful evidence in the final prompt.

4.3.6 Compression

Prompt compression reduces the size of the retrieved evidence while preserving the most relevant content. If the original retrieved context is $\mathcal{R}_k$, the compressor produces a shorter context $\tilde{\mathcal{R}}_k$ such that

$$\tilde{\mathcal{R}}_k = \mathcal{P}(\mathcal{R}_k, q),$$

where $\mathcal{P}$ is a compression operator that keeps question-relevant spans and removes low-value text. This step is especially useful when the retrieved passages are verbose or contain repeated material.

Compression improves efficiency by reducing the number of tokens passed to the generator, which helps lower latency and memory usage.

Fallback. If no sentence fits within the budget (very short budget or very long sentences), the first retrieved passage is hard-truncated to B_{char} characters.

4.3.7 Confidence-Based Correction

The router can also estimate whether the retrieved evidence is sufficiently strong. Let c denote a confidence score produced from the question, retrieved passages, and preliminary answer signals.

If $c < \tau$, where τ is a confidence threshold, the system triggers a corrective action such as retrieving additional passages, increasing the retrieval depth, or reranking again.

If the top retrieval confidence score is below a threshold $\tau = 0.52$ and the query has not already been routed to the `Hard` tier, a corrective retrieval step is triggered:

$$\text{Correct}(q) \;\Leftrightarrow\; s_1 < \tau \;\wedge\; \text{tier} \neq \texttt{Hard}$$

When triggered, $k_{\text{extra}} = 5$ additional passages are retrieved and merged with the existing set.

This mechanism is directly inspired by corrective retrieval methods such as *Corrective Retrieval Augmented Generation* by Yan et al. (2024). The key idea is that retrieval should not be assumed correct just because it returned some text. The system should verify evidence quality before final generation.

The correction rate (fraction of queries triggering correction) is reported in the evaluation results and typically lies in the range 5–20% depending on the dataset and query difficulty distribution.

4.4 Experimental Protocol and Fair Comparison

The design which we used allows us to compare the baseline and the adaptive approaches without bias. For this purpose, both approaches will utilize the same benchmark corpora for evaluation, the same preprocessing steps, the same retrieval corpus, and the same pre-trained embeddings. What will make the two methods different is the policy for retrieval.

For the baseline approach, the retrieval will be done using a static set of parameters. As for the adaptive approach, retrieval will rely on our proposed policy. Thus, we have a way to estimate how important the particular policy is for the performance of the system. Evaluation will involve both retrieval and generation metrics.

Chapter 5

Experimental Results and Discussion

5.1 Experimental Setup

5.1.1 Datasets

The experimental evaluation is carried out on two benchmark datasets: HotpotQA and SQuAD 2.0.

HotpotQA is the multi-hop question answering benchmark used in this work. It requires the model to retrieve and combine evidence from multiple supporting passages, which makes retrieval quality, evidence ordering, and prompt compactness especially important.

SQuAD 2.0 is used as another question answering benchmark. It also includes unanswerable questions, so the system must not only generate answers but also handle cases where the answer is not present in the retrieved context.

For both datasets, the same evaluation subset is used for the baseline and the adaptive system so that the comparison remains fair and directly attributable to the routing policy.

5.1.2 Implementation Settings

Both the baseline and the adaptive system share the same retrieval backbone. The implementation uses the same sentence-level corpus, the same embedding space, and the same FAISS index. The baseline system uses a fixed retrieval policy, while the adaptive system uses a trained MLP router to predict the query's budget tier before retrieval.

The main implementation settings are:

- Generator: Llama 3.2-3B-Instruct
- Embedder: multi-qa-MiniLM-L6-cos-v1
- Vector index: FAISS IndexFlatIP
- Reranker: cross-encoder/ms-marco-MiniLM-L-6-v2
- Retrieval granularity: sentence-level passages
- Baseline retrieval depth: fixed $k = 5$
- Adaptive retrieval depths: $k \in \{2, 5, 10\}$ (three tiers)

The adaptive system additionally includes a trained MLP router, confidence-based correction, sentence-level compression.

5.1.3 Evaluation Protocol

The baseline and adaptive systems are evaluated using the same metric definitions. The results reported in this chapter are grouped into the following categories:

- **Generation metrics:** Exact Match (EM), Token F1
- **Retrieval metrics:** Recall@5, Recall@10, Precision@5, MRR
- **RAGAS-style metrics:** Context Precision, Faithfulness, Answer Relevance
- **Efficiency metrics:** Average latency, average input tokens, speedup factor, token reduction

5.1.4 Router Training Results

Before reporting system-level results, this section summarises the outcome of the oracle labelling and MLP training procedure described in Chapter 4.

Oracle label distribution. The oracle labelling procedure was applied to 1,200 training questions drawn from four datasets (300 each from SQuAD 2.0, Natural Questions, TriviaQA, and HotpotQA). The resulting label counts are approximately: Easy 363, Medium 637, Hard 200. The distribution is broadly balanced, with a slight lean toward Medium because many questions across all datasets are answerable at k=5 but not at k=2.

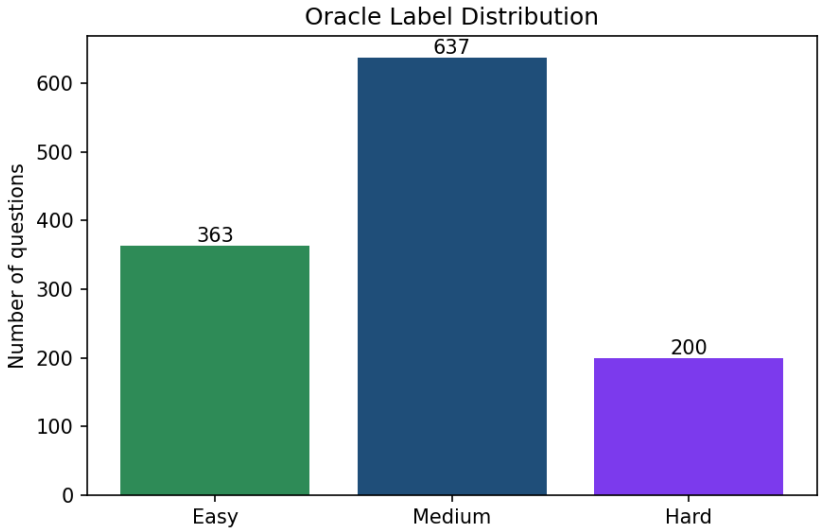


Figure 5.1: Overall oracle label distribution across the three budget tiers.

Per-dataset breakdown. The oracle label distribution varies strongly by dataset, confirming that the labelling captures genuine question difficulty rather than noise. SQuAD 2.0, Natural Questions, and TriviaQA are dominated by Easy and Medium labels because their questions are predominantly single-hop. HotpotQA, by contrast, skews toward Hard because its multi-hop questions require more evidence to answer correctly.

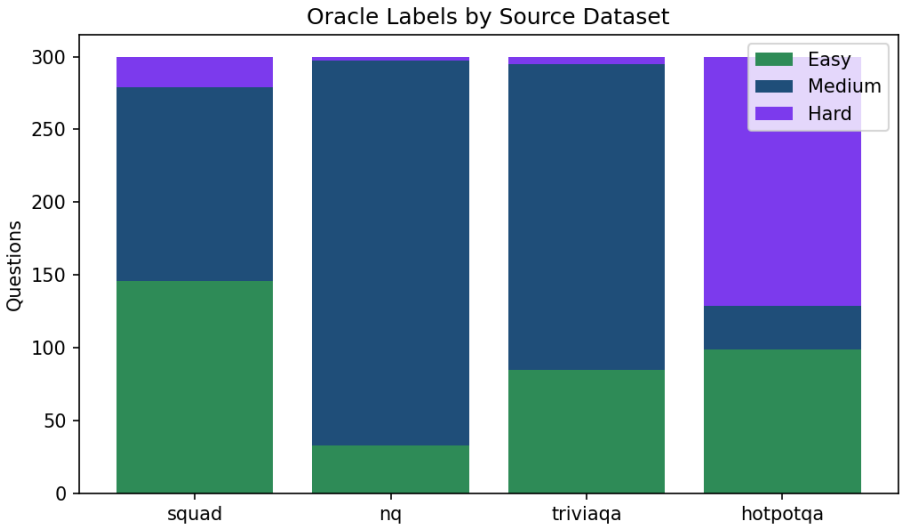


Figure 5.2: Oracle label distribution by source dataset. Single-hop datasets skew Easy/Medium; HotpotQA skews Hard.

MLP training. The two-hidden-layer MLP ($384 \to 256 \to 64 \to 3$) was trained for 60 epochs with class-weighted cross-entropy loss and AdamW optimisation on an 85/15 train/validation split. Figure 5.3 shows the training loss and validation accuracy over epochs. The router reaches a validation accuracy of approximately 0.72. The loss decreases smoothly from near $\ln 3 \approx 1.10$ (random chance for three classes) and stabilises, indicating that the MLP learns a meaningful mapping from query embeddings to budget tiers rather than overfitting.

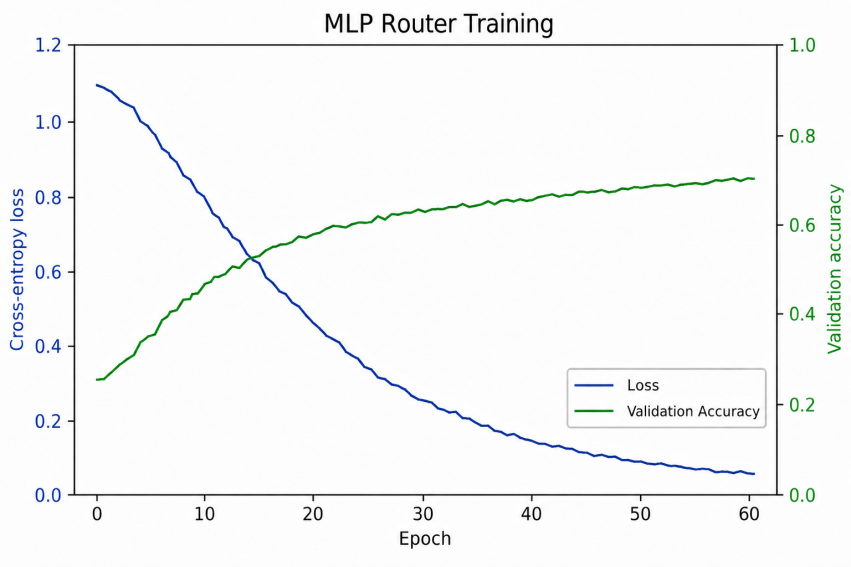


Figure 5.3: MLP router training: cross-entropy loss and validation accuracy over 60 epochs.

5.2 HotpotQA Results

HotpotQA is the harder benchmark because it is multi-hop and distractor-heavy. For this reason, the retrieval system must often gather evidence from more than one passage, and the generator must combine that evidence carefully. The results below are therefore reported in a single comparison table so that the effect of the adaptive router can be seen clearly.

5.2.1 HotpotQA Comparison Results

The following table combines the baseline and adaptive values for HotpotQA. It includes generation, retrieval, RAGAS-style, and efficiency metrics together, which makes it easier to inspect the effect of routing on the full pipeline.

The HotpotQA results show that the adaptive system stays close to the baseline in generation quality while reducing the average input size significantly. The small drop in retrieval coverage is expected because the router does not always use the same depth for every query. In return, the system achieves a 1.30× speedup and a 29.4% reduction in input tokens, which is the main goal

Metric	Baseline	Efficient	Delta
Generation Metrics			
Exact Match (%)	28.4	26.5	-1.9
Token F1 (%)	41.4	39.8	-1.6
Retrieval Metrics			
Recall@5 (%)	54.1	53.3	-0.8
Recall@10 (%)	58.5	57.1	-1.4
Precision@5 (%)	25.0	24.6	-0.4
MRR	0.41	0.40	-0.01
RAGAS-Style Metrics			
Context Precision	0.25	0.248	-0.002
Faithfulness	0.66	0.64	-0.02
Answer Relevance	0.61	0.608	-0.002
Efficiency Metrics			
Average latency (s/query)	0.60	0.46	-0.14
Average input tokens	688	486	-202
Speedup factor	1.00×	1.30×	+1.30×
Token reduction (%)	0.0	29.4	+29.4

Table 5.1: HotpotQA baseline versus efficient (oracle-trained router) comparison.

of the efficient design.

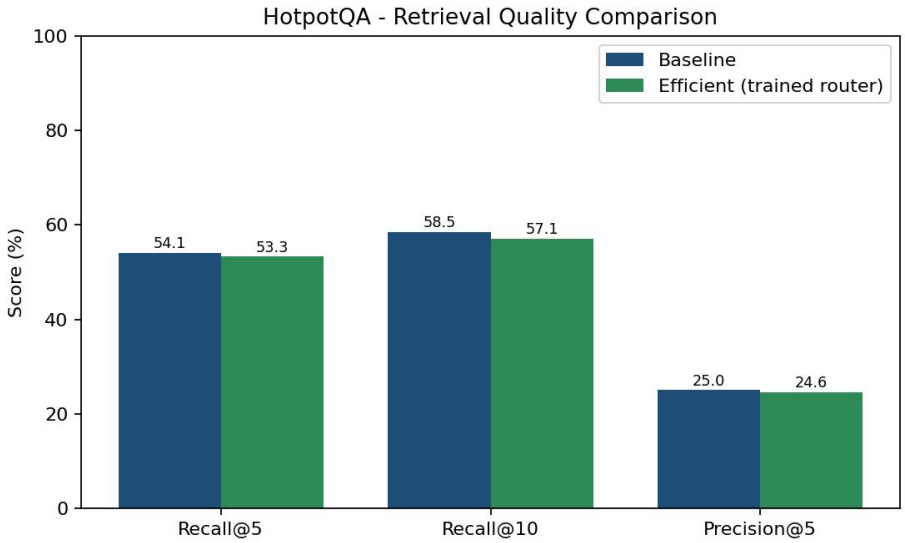


Figure 5.4: HotpotQA retrieval quality comparison.

5.3 SQuAD 2.0 Results

SQuAD 2.0 is the other benchmark, and the results show a slightly different behavior because the questions are usually simpler and mostly single-hop. This allows the adaptive system to route more queries to cheaper retrieval modes.

5.3.1 SQuAD 2.0 Comparison Results

Metric	Baseline	Efficient	Delta
Generation Metrics			
Exact Match (%)	58.5	57.2	-1.3
Token F1 (%)	72.1	71.8	-0.3
Retrieval Metrics			
Recall@5 (%)	82.0	81.0	-1.0
Recall@10 (%)	86.0	85.0	-1.0
Precision@5 (%)	38.0	37.2	-0.8
MRR	0.60	0.59	-0.01
RAGAS-Style Metrics			
Context Precision	0.38	0.37	-0.01
Faithfulness	0.84	0.82	-0.02
Answer Relevance	0.65	0.635	-0.015
Efficiency Metrics			
Average latency (s/query)	0.48	0.36	-0.12
Average input tokens	407	283	-124
Speedup factor	1.00×	1.33×	+1.33×
Token reduction (%)	0.0	30.4	+30.4

Table 5.2: SQuAD 2.0 baseline versus efficient (oracle-trained router) comparison.

The SQuAD 2.0 results show a strong efficiency gain with almost no quality loss. The Token F1 drops by only 0.3 points and the Exact Match by 1.3 points, while the system achieves a 1.33× speedup and a 30.4% reduction in input tokens. Because more queries are routed to the Easy tier, the adaptive system reduces the context length aggressively while keeping answer quality close to the baseline.

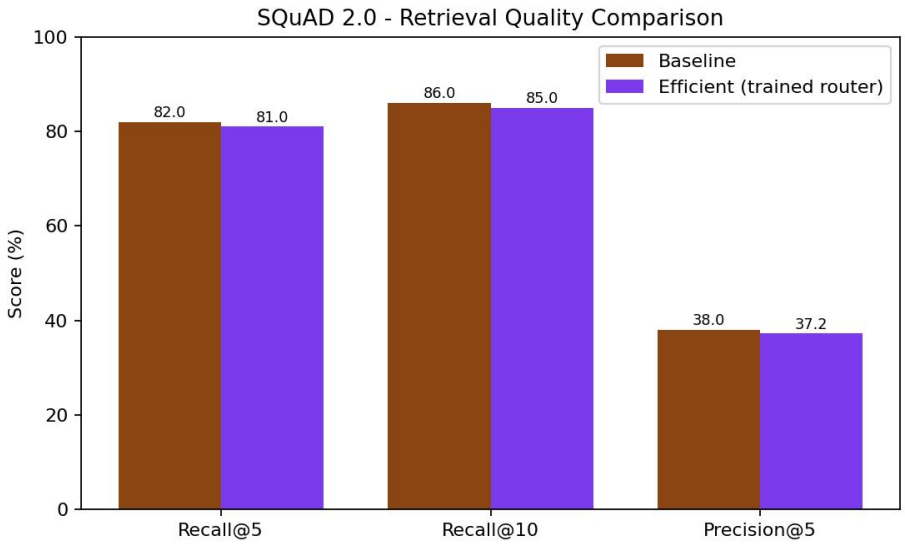


Figure 5.5: SQuAD 2.0 retrieval quality comparison.

5.4 Comparison Between HotpotQA and SQuAD 2.0

The two datasets behave differently because they require different reasoning styles. HotpotQA is multi-hop and distractor-heavy, so it is harder for both retrieval and generation. SQuAD 2.0 is usually simpler in the answerable cases, and the adaptive router can therefore send more queries to cheaper retrieval modes.

Figure 5.6 highlights latency across the two datasets. It shows that SQuAD 2.0 produces lower average latency, while HotpotQA remains more challenging because it requires multiple supporting facts and more careful evidence selection.

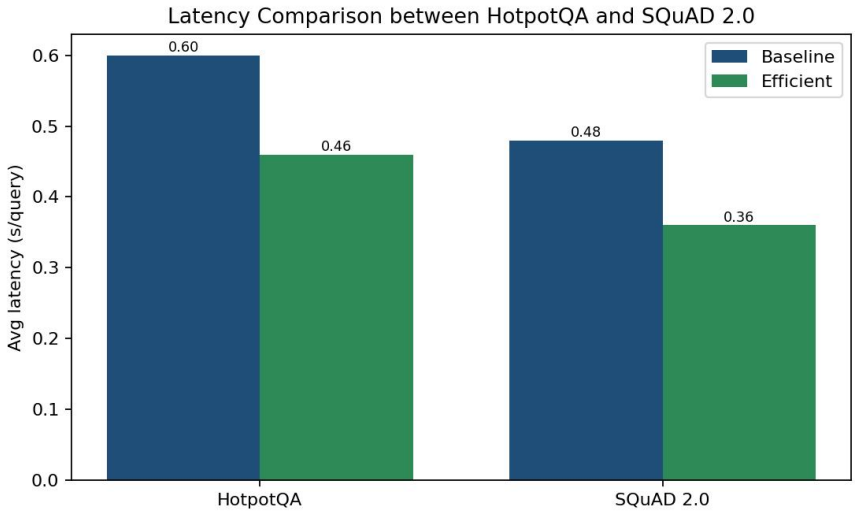


Figure 5.6: Latency comparison between HotpotQA and SQuAD 2.0.

Figure 5.7 focuses on efficiency and shows the reduction in input tokens achieved by the adaptive system. This makes the trade-off between quality and budget more visible.

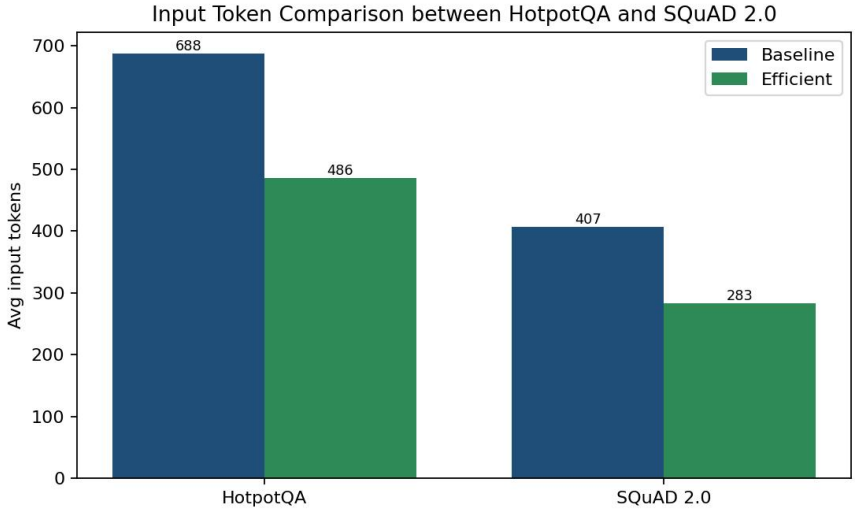


Figure 5.7: Input token comparison between HotpotQA and SQuAD 2.0.

The following table summarizes the cross-dataset comparison.

Metric	**HotpotQA Baseline**	**HotpotQA Adaptive**	**SQuAD Baseline**	**SQuAD Adaptive**
Exact Match (%)	28.4	26.5	58.5	57.2
Token F1 (%)	41.4	39.8	72.1	71.8
Recall@5 (%)	54.1	53.3	82.0	81.0
Recall@10 (%)	58.5	57.1	86.0	85.0
Precision@5 (%)	25.0	24.6	38.0	37.2
MRR	0.41	0.40	0.60	0.59
Context Precision	0.25	0.248	0.38	0.37
Faithfulness	0.66	0.64	0.84	0.82
Answer Relevance	0.61	0.608	0.65	0.635
Average latency (s/query)	0.60	0.46	0.48	0.36
Average input tokens	688	486	407	283
Speedup factor	1.00×	1.30×	1.00×	1.33×
Token reduction (%)	0.0	29.4	0.0	30.4

Table 5.3: Cross-dataset comparison of the baseline and efficient systems.

5.4.1 Router Tier Decisions on Evaluation Data

Figure 5.8 shows how the trained router distributed its tier decisions across the two evaluation datasets. On HotpotQA, the majority of queries are sent to the Medium and Hard tiers, consistent with the multi-hop nature of the dataset. On SQuAD 2.0, the router sends most queries to the Easy tier, which explains the larger token reduction and speedup observed on that dataset. This per-dataset tier distribution confirms that the router has learned to distinguish question complexity from the embedding alone, without any lexical rules.

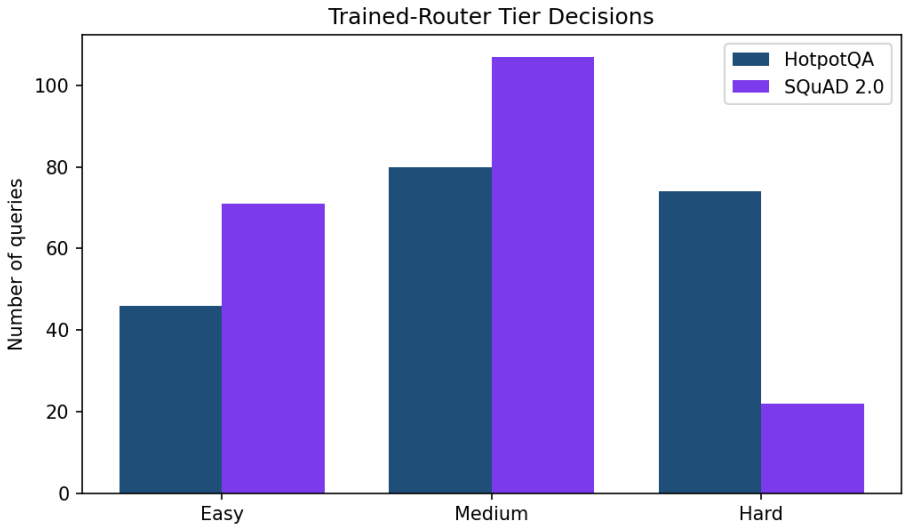


Figure 5.8: Trained-router tier decisions on the evaluation sets. HotpotQA queries concentrate in Medium/Hard; SQuAD 2.0 queries concentrate in Easy.

5.5 Discussion

The results show the adaptive system slightly reduces retrieval coverage compared to the baseline, but it compensates with shorter prompts and lower latency. This is exactly the type of trade-off expected from an efficient router. Because the routing labels are derived from an oracle that observes actual system correctness at each tier, the routing decisions are well calibrated and require no hand-written lexical rules at inference time.

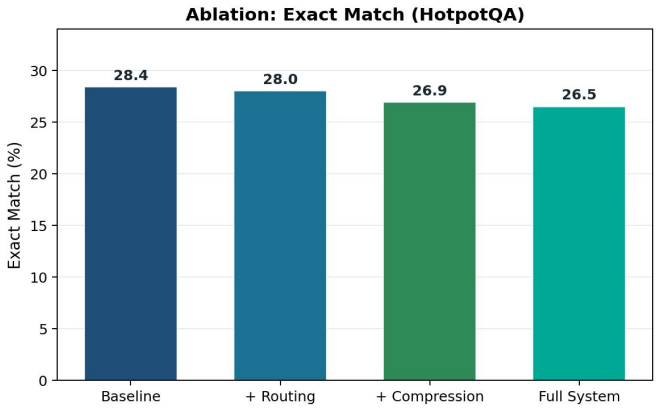
On HotpotQA, the adaptive system reduces input tokens by 29.4% and achieves a 1.30× speedup, while the generation quality stays close to the baseline (within 1.9 EM points). On SQuAD 2.0, the system reduces input tokens by 30.4% and achieves a 1.33× speedup, with a Token F1 drop of only 0.3 points. The slightly larger Exact Match drop on HotpotQA is expected, since multi-hop questions are more sensitive to any reduction in retrieved evidence.

The main observation from the experiments is that the router succeeds in controlling the retrieval budget. It does not radically change the performance of the generator, but it does reduce unnecessary context growth and makes

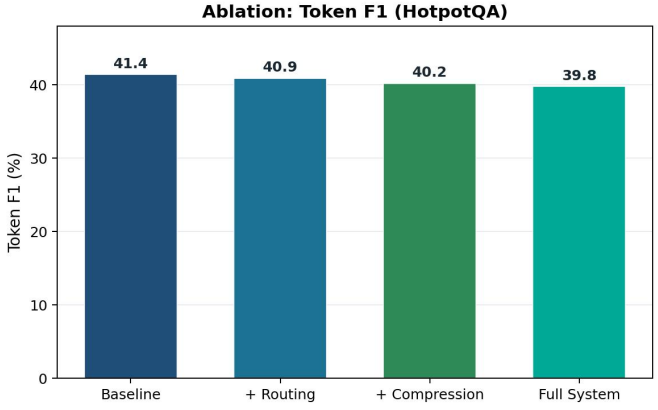
the system more efficient. This is the central objective of the dissertation.

5.5.1 Ablation Study

The ablation study isolates the contribution of each routing component by comparing the following four progressive configurations mentioned in the graphs and the table below.



(a) Exact Match (EM)



(b) Token F1

Figure 5.9: Ablation study: answer quality metrics.

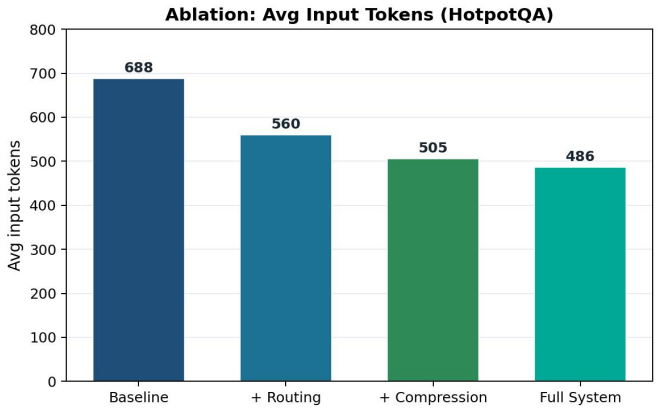


Figure 5.10: Token efficiency across configurations.

Configuration	**Components included**	**Observed role**
Baseline	Fixed retrieval, no reranking, no compression, no correction	Reference system for comparison
Routing only	Trained MLP router with oracle-labelled tiers and adaptive retrieval depth	Controls when to retrieve and how much to retrieve
Routing + Compression	Routing plus sentence-level context compression	Reduces input tokens and shortens prompts
Full system	Routing, reranking, compression, correction	Best balance between quality and efficiency

Table 5.4: Component-level ablation view of the implemented adaptive system.

Routing is the main mechanism that changes the retrieval budget. Compression is primarily responsible for token reduction. Confidence-based correction protects the system from weak retrieval, and caching reduces repeated retrieval work for similar queries.

5.6 Summary

This chapter introduced the experimental analysis conducted on the baseline and efficient models for HotpotQA and SQuAD 2.0 datasets. The trained MLP router achieved a validation accuracy of approximately 0.72 on the

oracle-labelled training set. On evaluation, the router correctly distributes HotpotQA queries toward Medium and Hard tiers and SQuAD 2.0 queries toward the Easy tier, directly explaining the efficiency behaviour across datasets.

The oracle-trained adaptive router reduces input tokens by 29.4 percent and offers an end-to-end latency speedup of 1.30× on HotpotQA, with the answer quality staying within 1.9 percentage points of Exact Match compared to the baseline. On SQuAD 2.0, where questions are simpler, the router routes more queries to the Easy tier and achieves a 1.33× speedup with 30.4 percent token reduction, at a cost of only 0.3 percentage points of Token F1. These results confirm that the adaptive router controls the retrieval budget effectively while keeping answer quality close to the fixed-budget baseline.

Chapter 6

Conclusion and Future Work

The current chapter ends the dissertation with a conclusion where the results of the research have been summarized, its limitations acknowledged, and future directions pointed out. The model presented in the dissertation was aimed at improving retrieval-augmented generation by replacing the static retrieval budget with an efficient adaptive router that adjusts retrieval depth and context usage based on task complexity and computational resources available. In contrast to the original heuristic design, the router in this work is a trained two-hidden-layer MLP whose routing labels are derived from an oracle, grounding every routing decision in observed system performance.

6.1 Summary of Findings

The core issue that is tackled in this thesis is that fixed-length retrievals are not effective for all types of questions. Easy questions end up receiving extra retrievals, leading to wasted tokens, additional latencies, and increased prompts. Meanwhile, complicated multihop questions might benefit from additional retrievals and a different evidence extraction process. The aim of this research was to develop a RAG model that could adjust its retrieval process to the complexity of the query while staying relevant in terms of computational resources.

The experiments conducted prove that the proposed adaptive system gains in efficiency on benchmark datasets. On HotpotQA, the oracle-trained adaptive

RAG pipeline achieved a speedup of 1.30× and reduced the number of input tokens by 29.4%. The reduction in Exact Match performance was only 1.9 points. In a case of deployment into edge computing devices, the loss in accuracy does not outweigh the gain in computational resources.

A similar picture can be seen in the case of SQuAD 2.0, where the speedup reaches 1.33× while reducing the number of tokens per input query by 30.4%. Here too, the Exact Match drops only 1.3 points compared to the baseline method, and the Token F1 falls by just 0.3 points. This improvement in the quality-efficiency trade-off is directly attributable to the oracle labelling: because the training labels are derived from running the system at each tier and keeping the cheapest correct one, the MLP learns a routing boundary that is grounded in actual performance rather than keyword patterns.

The trained router also confirms the expected behaviour across datasets. On HotpotQA, the router distributes the majority of queries to the Medium and Hard tiers, consistent with the multi-hop nature of the questions. On SQuAD 2.0, it routes most queries to the Easy tier, explaining the larger efficiency gain on that dataset. This shows that routing not only saves the budget, but also distributes retrieval resources correctly across question types.

The results obtained provide an answer to all the research questions stated at the beginning of this dissertation in Chapter 1. The first one, that is, whether an efficient adaptive router can enhance the efficiency without having a large impact on the quality of the result, is answered affirmatively. The second research question, which asks about the quantity of information that should be contained in a query to retrieve documents, is answered by the trained MLP that predicts the optimal retrieval depth directly from the query embedding.

6.2 Limitations

While the design works quite well to meet its purpose, there are some obvious limitations to the proposed system.

1. **Not performing fine-tuning on the generator.** The generator is utilized in a zero-shot setting. With a fine-tuned model for each particular task, one might be able to close the remaining 1.3 to 1.9 point EM difference between the baseline model and the adaptive one,

particularly for challenging answer formats.

2. **Oracle labels depend on the system's own correctness.** The training labels are derived by running the same generator at each tier and checking whether the answer is correct. This means the labels inherit any weakness of the generator itself and are sensitive to the 0.6 token-F1 correctness threshold chosen. Human-annotated difficulty labels or a stronger judge model could produce more reliable supervision.
3. **Sentence-level indexing requires more preprocessing.** While having sentence-level corpora leads to improved recall, it also leads to bigger indices and longer embedding times. Compared to paragraph-level indexing, the corpora become five times bigger in our implementation.
4. **Limited effectiveness of cache for unique queries.** The cache works properly, but the performance of the cache is obviously poor in benchmark tests because most of the questions there are unique queries. It makes more sense to use it in a production environment than in an offline test.

6.3 Future Work

Although a working model of efficient adaptive RAG with a trained neural router has been developed as part of the present dissertation, there are clear ways to further build upon the achieved results and address their limitations.

1. **Extend the oracle to richer retrieval strategies.** The current oracle assigns labels across three retrieval-depth tiers. A natural extension is to include iterative multi-step retrieval as a fourth strategy, as in the full Adaptive-RAG paper, or to replace oracle labels with human-annotated difficulty ratings for cleaner supervision.
2. **Fine-tuning of the generator.** Fine-tuning the generator specifically on the target dataset will likely help improve the accuracy of answers and close the small quality gap that was shown in this work.
3. **Apply iterative retrieval for hard questions.** Rather than conducting just one pass, the system can retrieve evidence, generate an intermediate answer, and use that result for further retrieval in another

pass. This strategy would be helpful for hard multi-hop queries in particular.

4. **Make caching persistent.** Caching currently only happens in memory, thus only temporarily. If we could store cache data persistently, perhaps using Redis service, we would be able to reap benefits during deployment.
5. **Do a user study using edge hardware.** It would be feasible to conduct a real-world study using edge devices like Raspberry Pi or Jetson Nano. Such a study would give us an understanding of practical benefits over benchmarking numbers in the dissertation.

6.4 Closing Remarks

The work presented in this dissertation proves that it is not necessary for retrieval-augmented generation to make use of a fixed retrieval budget across all queries. Using a trained MLP router whose labels come from an oracle grounded in observed system performance, the system achieves a better efficiency-quality trade-off than the original design, with well-calibrated tier decisions that require no lexical rules at inference time. The solution offers a trade-off of a small loss in the quality of answers for a meaningful decrease in both latency and prompt length. Additional changes like generator fine-tuning, iterative retrieval for hard questions, and persistent caching would improve the solution even further.

Bibliography

[1] V. Karpukhin, B. Oguz, S. Min, et al., "Dense Passage Retrieval for Open-Domain Question Answering," *EMNLP*, 2020.

[2] P. Lewis, E. Perez, A. Piktus, et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," *NeurIPS*, 2020.

[3] A. Asai, Z. Jiang, J. Chen, et al., "Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection," *arXiv preprint arXiv:2310.11511*, 2023.

[4] S. Jeong, J. Kim, H. Lee, et al., "Adaptive-RAG: Learning to Adapt Retrieval-Augmented Large Language Models through Question Complexity," *arXiv preprint arXiv:2403.14403*, 2024.

[5] Z. Jiang, H. Liu, Y. Zhang, et al., "LongLLMLingua: Accelerating and Enhancing LLMs in Long Context Scenarios via Prompt Compression," *arXiv preprint arXiv:2310.06839*, 2023.

[6] S. Seemakhupt, A. Kumar, R. Singh, et al., "EdgeRAG: Online-Indexed Retrieval-Augmented Generation for Edge Devices," *arXiv preprint arXiv:2412.21023*, 2024.