PicoRAG: A Lightweight Retrieval-Augmented Generation (RAG) Framework for Resource Constrained Devices

1. INTRODUCTION

The symbiotic relationship between AI and data is becoming increasingly crucial for businesses to thrive in the modern era [1]. By leveraging AI and data, businesses can gain a competitive edge by making data-driven decisions, automating processes, and delivering personalized experiences. In this regard, one particularly promising AI technology is generative AI, including large language models (LLMs), which can process and generate human-like text at scale. Even though, generative AI and LLMs are highly capable tools but their effectiveness is often limited by their limited context length [2].

To overcome this limitation, businesses can connect LLMs to their proprietary data through a process known as Retrieval Augmented Generation (RAG). RAG involves combining the generative capabilities of LLMs with the ability to retrieve relevant information from a company's specific knowledge base [3]. This allows the LLM to provide more accurate, informative, and contextually relevant responses, making it a valuable asset for tasks such as customer service, content creation, and decision-making.

Retrieval-Augmented Generation (RAG) has emerged as a powerful technique to enhance the capabilities of large language models (LLMs) by incorporating external knowledge [4, 5]. This approach addresses the limitations of LLMs, such as their tendency to hallucinate or provide incorrect information, by retrieving relevant information from external sources before generating a response.

Current RAG systems often necessitate significant computational resources, typically requiring cloud-based infrastructure and large language models (LLMs) like LLaMA 3.1 405b [6] or Claude 3.5 Sonnet [7]. This reliance on substantial computing power can limit the usability of RAG in local environments with constrained resources, making it challenging for smaller businesses or individuals to implement and benefit from this technology [8]. Lightweight RAG systems are essential for democratizing access to this powerful technology[9]. By operating efficiently on mid-tier laptops, they can empower businesses and individuals in diverse fields to leverage the benefits of RAG without the need for expensive infrastructure or subscription-based APIs.

The proposed lightweight RAG framework aims to address the need for more accessible and efficient systems capable of running locally on, mid-tier hardware without requiring cloud-based infrastructure. This solution will involve developing a locally-stored Small Language Model (SLM) integrated with optimized retrieval mechanisms, allowing it to augment its generative capabilities with relevant information from external or locally stored datasets. By reducing reliance on high computational resources, the framework will make RAG more feasible for small businesses, students, and individual users, empowering them to harness AI-driven insights for learning, decision-making, and task automation.

This proposal outlines an approach to develop a lightweight Retrieval-Augmented Generation (RAG) framework designed for mid-tier personal computers such as Laptops. The focus is on making RAG systems more efficient, and suitable for local deployments without relying on cloud infrastructure with optimizations throughout the pipeline.

2. LITERATURE REVIEW

2.1. RECENT SURVEYS ON RETRIEVAL-AUGMENTED GENERATION

Retrieval-Augmented Generation (RAG) has emerged as a significant advancement in enhancing the capabilities of Large Language Models (LLMs). Recent survey papers have explored various aspects of RAG, highlighting its potential and challenges across different applications.

Gao et al. (2024) provide a comprehensive overview of RAG for LLMs, discussing different paradigms and core components while emphasizing the need for improved robustness and efficiency in real-world applications [5]. Building on this, Zhao et al. (2024) focus specifically on RAG for AI-generated content, identifying limitations such as retrieval noise and computational overhead, and suggesting the development of flexible RAG pipelines and broader applications [10].

The trustworthiness of RAG systems is a crucial concern addressed by Zhao Y (2024), who propose a framework for assessment along six dimensions: factuality, robustness, fairness, transparency, accountability, and privacy [11]. This aligns with Fan et al.'s (2024) emphasis on developing trustworthy RAG-augmented LLMs that are reliable, and fair [12].

Huang et al. (2024) categorize RAG into pre-retrieval, retrieval, post-retrieval, and generation phases, highlighting challenges in retrieval quality and system efficiency [13]. Similarly, Wu et al. (2024) survey RAG techniques, covering retrievers, retrieval fusions, and training strategies, emphasizing the need for more efficient retrieval techniques and improved training algorithms [14].

Specialized applications of RAG have also been explored. Chen et al. (2024) focus on Retrieval-Augmented Knowledge Integration (RAKI), discussing challenges in determining the necessity of external knowledge and ensuring prediction consistency [4]. Procko et al. (2024) survey the use of Knowledge Graphs in RAG, suggesting the potential for applying Graph RAG to a wider range of domains [15].

Li et al. (2024) introduces the concept of Generative Information Retrieval (GenIR), a paradigm leveraging generative models for information retrieval, and discuss challenges such as scalability and handling dynamic corpora [16]. Hu et al. (2024) provide an overview of Retrieval-Augmented Language Models (RALMs), exploring both Retrieval-Augmented Generation (RAG) and Retrieval-Augmented Understanding (RAU) [17].

While these surveys offer valuable insights into various aspects of RAG, there is a noticeable emphasis in the literature regarding need for more efficient RAG systems. Local RAG, which focuses on retrieval and generation processes occurring entirely locally on hardware without relying on external services, presents unique challenges and opportunities. These include issues of efficiency, privacy, and scalability in resource-constrained environments. A thorough investigation of Local RAG systems could address critical questions about optimizing retrieval mechanisms for local datasets, managing the trade-offs between model size and performance, and ensuring data privacy and security in offline environments. Such a study would complement the existing body of research and provide valuable insights for developing more versatile and accessible RAG solutions. The tabular form of this Literature Review has been attached in Annexture A.

2.2. RECENT APPROAHES FOR EFFICIENT RAG:

The literature offers a variety of techniques for constructing efficient and effective Retrieval-Augmented Generation (RAG) systems. FiD-Light [18] compresses the encoded representations of retrieved passages before feeding them to the decoder, thereby substantially reducing decoding time and overall latency. Sparse RAG [19] employs a selective decoding mechanism where the model evaluates the relevance of each retrieved document and only attends to the most pertinent ones during decoding. While both methods seek to improve efficiency, FiD-Light concentrates on compressing the input to the decoder, whereas Sparse RAG introduces a dynamic selection process to focus on relevant information. A study [20] proposes a hierarchical RAG system for biomedical question answering that leverages both Parameter-Efficient Fine-Tuning (PEFT) and Hierarchical Retrieval methods to identify relevant documents and passages. Additionally, a study [9] demonstrates that smaller language models (SLMs) combined with RAG can achieve performance comparable to larger language models (LLMs), suggesting that efficient RAG systems can be constructed using less computationally demanding models. A study [21] has focused on optimizing various components of a RAG system, including the retriever, reranker, and reader, for efficient open retrieval question answering. RGR [22] leverages retrieved reference questions to guide the generation process, improving question diversity and human-likeness. RAGCache [23] addresses the performance bottleneck of RAG by introducing a multilevel dynamic caching system for intermediate states. RoCR [24] proposes a robust CiM-backed RAG framework for resource-constrained edge devices, incorporating contrastive learning and noise-aware training. SimplyRetrieve [25] offers a user-friendly tool for Retrieval-Centric Generation (RCG), enabling clear separation of roles between LLMs and retrievers. Superposition Prompting [26] addresses the limitations of LLMs in handling long contexts by introducing a parallel prompt path approach, improving efficiency and mitigating the "distraction phenomenon". These advancements collectively contribute to the development of more efficient, robust, and versatile RAG systems.

State of the art Retrieval-Augmented Generation (RAG) models, while promising, face challenges such as retrieval relevance, generation accuracy, computational cost, bias, and interpretability. To address these limitations and make RAG more accessible, there is a pressing need for more efficient approaches that can run on resource-constrained devices. Such advancements would enable a wider range of applications and improve the overall usability of RAG technology.

2.3. EVALUATION OF RETRIEVAL-AUGMENTED GENERATION

Researchers have employed a diverse range of datasets and metrics to evaluate RAG models. Popular datasets include KILT[27], PopQA[28], BioASQ-QA[29], TriviaQA[30], HotpotQA[31], MTEB[32], MS MARCO[33], NewsQA[34], SQuAD[35], MMLU[36], LaMP[37], and more.

Metrics used for evaluation include qualitative human evaluation, Recall@k, Precision@k, F1@k, Mean Average Precision (MAP), Query Latency, pass@k, ROUGE-1, ROUGE-L, BLEU-4, Q-BLEU, Ref-Q, Time-to-First-Token (TTFT), System Throughput, Mean Absolute Error (MAE), Retrieval time, Generation time, Coverage, Coleman-Liau Readability Index, and Hallucinations checked by human evaluation.

3. AIMS AND OBJECTIVES

3.1. PROBLEM STATEMENT:

The current landscape of Retrieval-Augmented Generation (RAG) systems is characterized by high computational demands, reliance on cloud infrastructure, and significant cost barriers, limiting accessibility for small businesses, educational institutions, and individual users. This creates a pressing need for a lightweight RAG framework capable of operating efficiently locally on, mid-tier hardware. Such a solution would democratize access to RAG technology, addressing challenges related to resource requirements, scalability, privacy, and deployment complexity. By developing a framework that combines a locally-stored Small Language Model with optimized retrieval mechanisms, we can empower a broader range of users to leverage AI-driven insights for learning, decision-making, and task automation without the need for expensive infrastructure or specialized technical expertise.

3.2. RESEARCH AIM

The primary aim of this research is to develop and evaluate a lightweight Retrieval-Augmented Generation (RAG) framework that can operate efficiently locally on, mid-tier personal computers, making RAG technology more accessible and practical for a wider range of users and applications.

3.3. OBJECTIVES

A. Design and implement a lightweight RAG framework using a locally-stored Small Language Model (SLM) and optimized retrieval mechanisms.
B. Explore efficient indexing and retrieval algorithms suitable for operation on mid-tier hardware with limited computational resources.
C. Evaluate the framework's effectiveness in terms of response quality, retrieval accuracy, and computational efficiency compared to existing cloud-based RAG systems.

3.4. RESEARCH QUESTIONS

RQ1. What architectural design choices and integration strategies between SLM and retrieval mechanisms yield optimal performance for a locally-deployed RAG framework while maintaining minimal resource consumption?

- *Measurable metrics: Framework latency (seconds), memory footprint (GB), throughput (queries/second), and system resource utilization (CPU/RAM %) across different architectural configurations*

RQ2. How do various combinations of embedding, indexing, retrieval algorithms and SLMs affect the efficiency-accuracy trade-off when operating under mid-tier hardware constraints?

- *Measurable metrics: Index size (MB), query latency (ms), memory usage (MB), retrieval accuracy (precision@k, recall@k) for different algorithm combinations and parameter settings*

RQ3. What are the quantifiable performance differences between cloud-based and lightweight local RAG systems across standardized benchmarks, and at what hardware specification threshold does local deployment become practically viable?

- *Measurable metrics: Comparative analysis of response quality (ROUGE/BLEU scores), retrieval accuracy (MRR/MAP), resource efficiency (queries/watt), and cost-effectiveness (queries/dollar) across different hardware tiers and deployment scenarios*

4. PROPOSED FRAMEWORK

4.1. OVERVIEW OF THE FRAMEWORK:

This section outlines the step-by-step process used to develop a RAG pipeline for interactive engagement with PDF documents. The Figure 1 illustrates the proposed framework for a simple local Retrieval Augmented Generation (RAG) system.

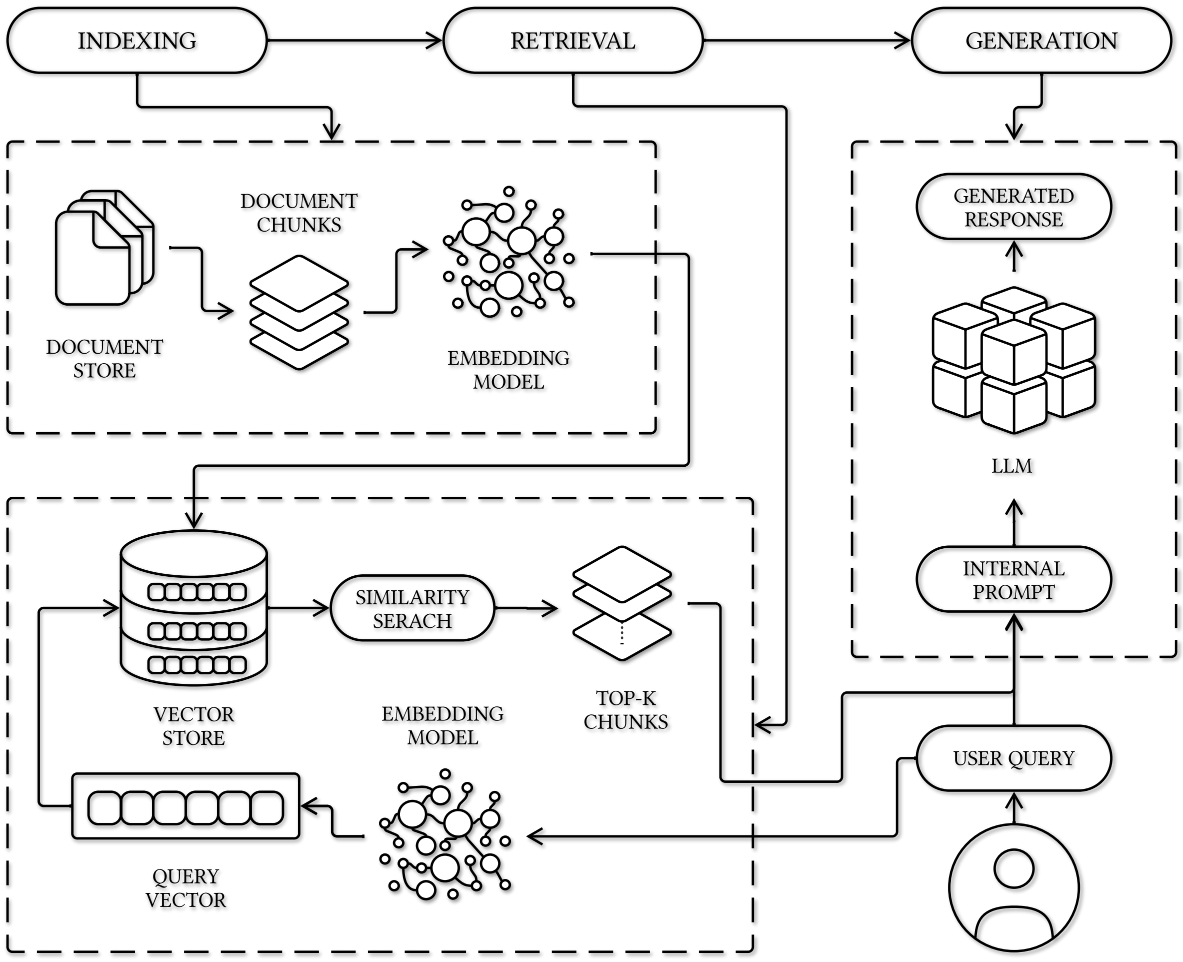

Figure 1. Proposed Framework for Simple Lightweight Local RAG

This framework is designed to generate responses based on a given user query by retrieving relevant information from a local document store. This proposed framework provides a basic foundation for building a local RAG system and, it is going to be further enhanced by incorporating techniques such as fine-tuning the embedding model, using more advanced similarity metrics, and optimizing the LLM for the specific task. Here's a breakdown of the key components and their functions:

Indexing:

- **Document Store:** Stores the documents that will be used for retrieval.
- **Document Chunks:** The documents are broken down into smaller chunks to improve retrieval efficiency.
- **Embedding Model:** This model is used to convert the document chunks into numerical representations (embeddings) that can be compared.

Retrieval:

- **Query Vector:** The user query is also converted into a numerical representation (query vector) using the same embedding model.
- **Similarity Search:** The query vector is compared to the embeddings of the document chunks using a similarity metric (e.g., cosine similarity) to find the most relevant chunks.
- **Top-K Chunks:** The top-k most relevant chunks are selected based on their similarity scores.

Generation:

- **Internal Prompt:** The selected chunks are combined into a single prompt that is fed to the LLM.
- **LLM:** The LLM generates a response based on the provided internal prompt.
- **Generated Response:** The final response is returned to the user.

4.2. DOCUMENT PREPROCESSING AND EMBEDDING

4.2.1. PDF Loading

The process begins by loading a PDF document using the PyMuPDF library [38], which extracts text from each page. Since text extraction can be inconsistent, we store the extracted text in a list of dictionaries, with each dictionary representing a page. This list is transformed into a DataFrame for further analysis.

An exploratory data analysis (EDA) has to be conducted to evaluate the average size of text per page, measured in both characters and tokens. Given that many embedding models have limits on the number of tokens they can process at once, this step is critical. For example, the all-mpnet-base-v2 model used in this project can process up to 384 tokens [39].

4.2.2. Text Splitting

Given the limitations on input text length in embedding models, the text was split into smaller chunks. Although there are several ways to split text, this project uses sentence-based chunking. Sentences are grouped into chunks, usually of 10 sentences, which fits well with the 384-token limit of the embedding model. Two methods for sentence splitting are considered:

1. **Simple Rule-Based Splitting**: Splitting text based on punctuation (e.g., using text.split(".")).
2. **Natural Language Processing (NLP) Libraries**: More advanced methods using libraries like spaCy[40] or NLTK[41] for accurate sentence segmentation.

SpaCy was chosen for sentence segmentation due to its robustness and scalability. After segmentation, sentences are chunked into groups of 10 sentences, which ensures manageable chunks and aligns with the embedding model's capacity. Alternative text-splitting methods were considered, including punctuation-based splitting and other techniques provided by libraries like LangChain[42]. However, the sentence-based approach was chosen for its balance between granularity and manageability. Chunks that contained fewer than 30 tokens (such as headers and footers) have to be filtered out to improve data quality. A DataFrame containing the filtered text chunks, along with metadata like page numbers, has to be saved for further processing.

4.2.3. Embedding

Once the text is split into manageable chunks, each chunk is converted into a numerical representation (embedding) using a pre-trained model from the sentence-transformers library, specifically *all-mpnet-base-v2.* This model generates embeddings as 768-dimensional vectors.

The embeddings capture the semantic essence of the text and are stored alongside the chunks for future use. To optimize performance, the embedding process is executed on a GPU. Batched embedding operations further enhance performance, achieving a fourfold increase in speed when processing multiple chunks simultaneously. The final embeddings and their corresponding chunks are stored in a PyTorch tensor for efficient reuse.

4.3. SEARCH AND ANSWER GENERATION

4.3.1. Retrieval

The retrieval process uses **semantic search**, where the similarity between the user's query and the pre-computed embeddings is measured to find the most relevant text chunks. The steps involved in retrieval are:

1. **Query Definition:** A user specifies a query.
2. **Query Embedding:** The query vector is generated using the using the same embedding model that was used for the document chunks.
3. **Similarity Calculation:** Similarity between the query embedding and document chunk embeddings is computed using cosine similarity as a base model.
4. **Result Sorting:** Chunks are ranked by relevance based on their similarity to the query.

The similarity between query embeddings and text embeddings was computed using *cosine similarity* for moderate-sized datasets, but larger datasets may require specialized libraries like HNSW[43] & FAISS[44] for efficient vector search. Alternative similarity measures, such as the dot product, were considered. However, cosine similarity was preferred for its emphasis on direction rather than magnitude, which is more suitable for text data.

4.3.2. Prompt Creation

The next step is constructing a prompt for a LLM to generate an answer. The prompt is composed of the user's query along with the top-ranked text chunks retrieved during the search process. For the purposes of this project, we are using the *gemma2-7b-it* model, a locally-executed instruction-following language model [45].

4.3.3. Answer Generation

Once the prompt is constructed, the LLM generates a response based on both the query and the relevant contextual information retrieved from the document. The generated response is designed to provide a coherent and informative answer to the user's query.

To enhance the quality of the generated answers, prompt engineering is applied. This involves augmenting the prompt with additional contextual information to ensure the LLM produces responses that are as accurate and detailed as possible. This workflow, which integrates retrieval, augmentation, and generation, is aligned with the methodologies outlined in NVIDIA's guide on RAG pipelines[46].

4.4. VALIDATION AND PERFORMANCE EVALUATION

To evaluate our lightweight RAG pipeline, we would employ a multi-faceted approach focusing on performance, accuracy, and efficiency. We'd measure response time, memory usage, and CPU utilization to assess computational efficiency on mid-tier hardware. For accuracy and relevance, we'd use metrics like BLEU, ROUGE, and human evaluation to compare generated responses against gold-standard answers. Retrieval quality would be evaluated using precision, recall, and mean reciprocal rank (MRR). A suitable dataset for this evaluation could be the KILT[27] dataset, which contains diverse question-answer pairs and associated passages. This dataset is particularly appropriate as it mimics real-world information seeking scenarios and has been widely used in evaluating both retrieval and question-answering systems.

5. DELIVERABLES

This research provides an empirical contribution by comprehensively quantifying the performance-resource trade-offs in local RAG systems with different configurations on multiple benchmarks - an understudied aspect in current literature that typically focuses on cloud infrastructure, while proposing a ready to deploy system on consumer-grade hardware, filling a critical gap in understanding the practical limitations and optimization potential of edge-deployed RAG systems. The key deliverables of this research will include:

In fulfilment of Objective A: A fully functional, open-source lightweight RAG framework optimized for local deployment on mid-tier hardware (≤16GB RAM, 10th gen Intel Core i5 processor, 4GB VRAM), incorporating the best-performing combination of embedding model, retrieval mechanism, and Small Language Model (SLM) as determined through comprehensive benchmarking;

In fulfilment of Objective B: A comparative analysis report documenting the performance evaluation of various embedding models (*all-mpnet-base-v2* as the base model), retrieval algorithms (e.g., HNSW, IVF-PQ, LSH), and Small LLMs (*gemma2-7b-it* as the base model)

In fulfilment of Objective C: A comprehensive evaluation across standard benchmarks such as KILT, SQuAD, PopQA and Doc2Dial , with metrics for retrieval accuracy (precision@k, recall@k, MRR), response quality (ROUGE, BLEU, etc.), and computational efficiency (latency, memory usage, storage requirements).

6. LIMITATIONS AND FUTURE WORK

While the proposed lightweight RAG framework aims to bring the power of Retrieval-Augmented Generation to local, mid-tier systems, there are inherent limitations to consider. First, the performance of locally-stored Small Language Models (SLMs) may not match the quality and fluency of responses generated by large, cloud-based LLMs due to the smaller model size and reduced training data. This can lead to lower accuracy or less nuanced responses in complex tasks. Additionally, the reliance on limited computational resources may restrict the scalability and speed of the system when handling large datasets or extensive queries. Finally, ensuring the effectiveness of retrieval mechanisms in diverse, dynamic, or domain-specific datasets could present challenges in maintaining high-quality, contextually relevant responses. Future work could explore optimizations to mitigate these limitations, including fine-tuning retrieval algorithms and improving the efficiency of local model training and execution.

7. SCOPE

This research focuses on developing a lightweight Retrieval-Augmented Generation (RAG) framework specifically designed for local, mid-tier personal computers. The scope encompasses the design, implementation, and evaluation of a RAG system that combines a locally-stored Small Language Model (SLM) with optimized retrieval mechanisms. The project will explore efficient indexing and retrieval algorithms suitable for resource-constrained environments, as well as techniques for effective integration of the SLM with the retrieval system. The research will be limited to text-based RAG applications and will not address multimodal or speech-based systems. The primary focus will be on general knowledge retrieval and question-answering tasks. The evaluation will compare the lightweight RAG framework's performance against existing cloud-based solutions, considering subjective factors such as response quality, retrieval accuracy. The project will not involve the development of new language models but will instead focus on optimizing existing SLMs for local deployment within the RAG framework.

8. CONCLUSION

This exploration of development of a lightweight Retrieval-Augmented Generation (RAG) framework for local systems marks our humble contribution toward democratizing access to advanced AI technologies. By integrating a locally-stored Small Language Model (SLM) with optimized retrieval mechanisms, this research addresses the limitations of current cloud-dependent RAG systems, enabling their use on mid-tier personal computers without reliance on costly APIs. This framework has the potential to empower small businesses, educational institutions, and individual users by offering a cost-effective, efficient solution without reliance on high-end infrastructure. The study aims to demonstrate that local RAG systems can achieve competitive performance, opening doors to broader applications in personal knowledge management, educational support, and business operations. As we move forward, the insights gained from this research will contribute to the ongoing evolution of efficient, locally-deployable AI systems, paving the way for further advancements in resource-efficient and user-friendly RAG technologies.