UPDF
WWW.UPDF.COM

Seven Failure Points When Engineering a Retrieval Augmented Generation System

Scott Barnett, Stefanus Kurniawan, Srikanth Thudumu, Zach Brannelly, Mohamed Abdelrazek
{scott.barnett,stefanus.kurniawan,srikanth.thudumu,zach.brannelly,mohamed.abdelrazek}@deakin.edu.au
Applied Artificial Intelligence Institute
Geelong, Australia

arXiv:2401.05856v1 [cs.SE] 11 Jan 2024

ABSTRACT

Software engineers are increasingly adding semantic search capabilities to applications using a strategy known as Retrieval Augmented Generation (RAG). A RAG system involves finding documents that semantically match a query and then passing the documents to a large language model (LLM) such as ChatGPT to extract the right answer using an LLM. RAG systems aim to: a) reduce the problem of hallucinated responses from LLMs, b) link sources/references to generated responses, and c) remove the need for annotating documents with meta-data. However, RAG systems suffer from limitations inherent to information retrieval systems and from reliance on LLMs. In this paper, we present an experience report on the failure points of RAG systems from three case studies from separate domains: research, education, and biomedical. We share the lessons learned and present 7 failure points to consider when designing a RAG system. The two key takeaways arising from our work are: 1) validation of a RAG system is only feasible during operation, and 2) the robustness of a RAG system evolves rather than designed in at the start. We conclude with a list of potential research directions on RAG systems for the software engineering community.

CCS CONCEPTS

• **Software and its engineering → Empirical software validation**.

KEYWORDS

Retrieval Augmented Generation, RAG, SE4AI, Case Study

ACM Reference Format:
Scott Barnett, Stefanus Kurniawan, Srikanth Thudumu, Zach Brannelly, Mohamed Abdelrazek . 2024. Seven Failure Points When Engineering a Retrieval Augmented Generation System. In *Proceedings of 3rd International Conference on AI Engineering — Software Engineering for AI (CAIN 2024).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The new advancements of Large Language Models (LLMs), including ChatGPT, have given software engineers new capabilities to build new HCI solutions, complete complex tasks, summarise documents, answer questions in a given artefact(s), and generate new content. However, LLMs suffer from limitations when it comes to up-to-date knowledge or domain-specific knowledge currently captured in company's repositories. Two options to address this problem are: a) Finetuning LLMs (continue training an LLM using domain specific artifacts) which requires managing or serving a fine-tuned LLM; or b) use Retrieval-Augmented Generation (RAG) Systems that rely on LLMs for generation of answers using existing (extensible) knowledge artifacts. Both options have pros and cons related to privacy/security of data, scalability, cost, skills required, etc. In this paper, we focus on the RAG option.

Retrieval-Augmented Generation (RAG) systems offer a compelling solution to this challenge. By integrating retrieval mechanisms with the generative capabilities of LLMs, RAG systems can synthesise contextually relevant, accurate, and up-to-date information. A Retrieval-Augmented Generation (RAG) system combines information retrieval capabilities, and generative prowess of LLMs. The retrieval component focuses on retrieving relevant information for a user query from a data store. The generation component focuses on using the retrieved information as a context to generate an answer for the user query. RAG systems are an important use case as all unstructured information can now be indexed and available to query reducing development time no knowledge graph creation and limited data curation and cleaning.

Software engineers building RAG systems are expected to preprocess domain knowledge captured as artifacts in different formats, store processed information in appropriate data store (vector database), implement or integrate the right query-artifact matching strategy, rank matched artifacts, and call the LLMs API passing in user queries and context documents. New advances for building RAG systems are constantly emerging [8, 12] but how they relate and perform for a specific application context has to be discovered.

In this work we present the lessons learned and 7 failure points arising from 3 case studies. The purpose of this paper is to provide 1) a reference to practitioners and 2) to present a research road map for RAG systems. To the best of our knowledge, we present the first empirical insight into the challenges with creating robust RAG systems. As advances in LLMs continue to take place, the software engineering community has a responsibility to provide knowledge on how to realise robust systems with LLMs. This work is an important step for robustness in building RAG systems.

Research questions for this work include:

- *What are the failure points that occur when engineering a RAG system?* (section 5) We present an empirical experiment using the BioASQ data set to report on potential failure points. The experiment involved 15,000 documents and 1000 question

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
CAIN 2024, April 2024, Lisbon, Portugal
© 2024 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

UPDF
WWW.UPDF.COM

and answer pairs. We indexed all documents then ran the queries and stored the generated responses using GPT-4. All question and answer pairs were then validated with OpenAI evals [1]. Manual inspection (all discrepancies, all flagged as incorrect, and a sample of correct labels) was analysed to identify the patterns.
- *What are the key considerations when engineering a RAG system?* (section 6) We present the lessons learned from three case studies involving the implementation of a RAG system. This presents the challenges faced and insights gained.

Contributions arising from this work include:

- A catalogue of failure points (FP) that occur in RAG systems.
- An experience report from 3 case studies of implementing a RAG system. Two currently running at Deakin University.
- A research direction for RAG systems based on the lessons learned from the 3 case studies.

2 RELATED WORK

Retrieval augmented generation encompasses using documents to augment large language models through pre-training and at inference time [7, 9, 12]. Due to the compute cost, data preparation time and required resources using RAG without training or fine-tuning is an attractive proposition. However, challenges arise when using large language models for information extraction such as performance with long text [8].

A recent survey [19] showed that large language models are used across the RAG pipeline including retriever, data generation, rewriter, and reader. Our work complements this survey by taking a software engineering perspective to shine a light on what issues engineers will face and what software engineering research is necessary to realise solutions with the current state-of-the-art RAG systems.

Emerging work has looked at benchmarking RAG systems [3] but not at the failures occurring during implementation. Software engineering research has investigated the use of RAG systems for code-related tasks [15]. However, the application of RAG systems is broader than software engineering tasks. This paper complements existing work by presenting challenges faced during the implementation of a RAG system with a focus on practitioners.

Errors and failures that arise from RAG systems overlap with other information retrieval systems including 1) no metrics for query rewriting, 2) document re-ranking, and 3) effective content summarisation [19]. Our results confirm this The unique aspects are related to the semantic and generative nature of the use of large language models including evaluating factual accuracy [16].

3 RETRIEVAL AUGMENTED GENERATION

With the explosion in popularity of large language model services such as ChatGPT[2], Claude[3], and Bard [4], people have explored their use as a question and answering systems. While the performance is impressive [16] there are two fundamental challenges: 1) hallucinations - where the LLM produces a response that looks right but is incorrect, and 2) unbounded - no way to direct or update the content of the output (other than through prompt engineering). A RAG system is an information retrieval approach designed to overcome the limitations of using a LLM directly.

RAG works by taking a natural language query is converted into an embedding which is used to semantically search a set of documents. Retrieved documents are then passed to a large language model to generate an answer. An overview of a RAG system is shown in Figure 1 as two separate processes, Index and Query. See this survey for more details [19]

3.1 Index Process

In a RAG system, the retrieval system works using embeddings that provide a compressed semantic representation of the document. An embedding is expressed as a vector of numbers. During the Index process each document is split into smaller chunks that are converted into an embedding using an embedding model. The original chunk and the embedding are then indexed in a database. Software engineers face design decisions around how best to chunk the document and how large a chunk should be. If chunks are too small certain questions cannot be answered, if the chunks are too long then the answers include generated noise.

Different types of documents require different chunking and processing stages. For example, video content requires a transcription pipeline to extract the audio and convert to text prior to encoding (see subsection 4.2. The choice of which embedding to use also matters as changing the embedding strategy requires re-indexing all chunks. An embedding should be chosen based on the ability to semantically retrieve correct responses. This process depends on the size of the chunks, the types of questions expected, the structure of the content and the application domain.

3.2 Query Process

The Query process takes place at run time. A question expressed as natural language is first converted into a general query. To generalise the query a large language model is used which enables additional context such as previous chat history to be included in the new query. An embedding is then calculated from the new query to use for locating relevant documents from the database. Top-k similar documents are retrieved using a similarity method such as cosine similarity (vector databases have techniques such as inverted indexes to speed up retrieval time). The intuition is that chunks that are semantically close to the query are likely to contain the answer.

Retrieved documents are then re-ranked to maximise the likelihood that the chunk with the answer is located near the top. The next stage is the Consolidator which is responsible for processing the chunks. This stage is needed to overcome the limitations of large language models 1) token limit and 2) rate limit. Services such as OpenAI have hard limits on the amount of text to include in a prompt. This restricts the number of chunks to include in a prompt to extract out an answer and a reduction strategy is needed to chain prompts to obtain an answer. These online services also restrict the number of tokens to use within a time frame restricting the latency of a system. Software engineers need to consider these tradeoffs when designing a RAG system.

[1] https://github.com/openai/evals
[2] https://chat.openai.com/
[3] https://claude.ai/
[4] https://bard.google.com/

UPDF
WWW.UPDF.COM

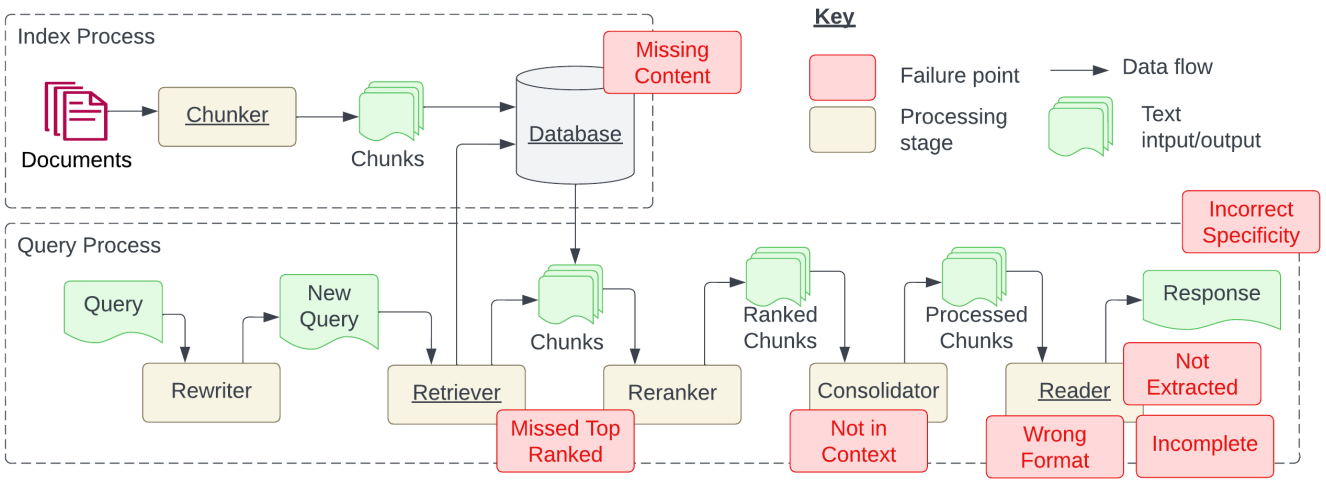

Figure 1: Indexing and Query processes required for creating a Retrieval Augmented Generation (RAG) system. The indexing process is typically done at development time and queries at runtime. Failure points identified in this study are shown in red boxes. All required stages are underlined. Figure expanded from [19].

The final stage of a RAG pipeline is when the answer is extracted from the generated text. Readers are responsible for filtering the noise from the prompt, adhering to formatting instructions (i.e. answer the question as a list of options), and producing the output to return for the query. Implementation of a RAG system requires customising multiple prompts to process questions and answers. This process ensures that questions relevant for the domain are returned. The use of large language models to answer real time questions from documents opens up new application domains where question and answering is new capability. Thus, RAG systems are difficult to test as no data exists and needs to be experimentally discovered through either a) synthetic data generation, or b) piloting the system with minimal testing.

4 CASE STUDIES

This study conducted three case studies to discover the challenges that arise when implementing RAG systems. A summary of each of the case studies is shown in Table 1. All scripts, data, and examples of each of the failure points for the BioASQ case study are available online [5]. The other two case studies have been excluded due to confidentiality concerns.

4.1 Cognitive Reviewer

Cognitive Reviewer is a RAG system designed to support researchers in analysing scientific documents. Researchers specify a research question or objective and then upload a collection of related research papers. All of the documents are then ranked in accordance with the stated objective for the researcher to manually review. The researcher can also ask questions directly against all of the documents. Cognitive Reviewer is currently used by PhD students from Deakin University to support their literature reviews. The Cognitive Reviewer does the Index process at run time and relies on a robust data processing pipeline to handle uploaded documents i.e. no quality control possible at development time. This system also uses a ranking algorithm to sort the uploaded documents.

4.2 AI Tutor

The AI Tutor is a RAG system where students ask questions about the unit and answers are sourced from the learning content. Students are able to verify the answers by accessing a sources list from where the answer came from. The AI Tutor works by integrating into Deakin's learning management system, indexing all of the content including PDF documents, videos, and text documents. As part of the Index process, videos are transcribed using the deep learning model Whisper [17] before being chunked. The AI Tutor was developed between August 2023 to November 2023 for a pilot in a unit with 200 students that commenced the 30th of October 2023. Our intention is to present the lessons learned during implementation and present a followup findings at the conclusion of the pilot. This RAG pipeline includes a rewriter to generalise queries. We implemented a chat interface where previous dialogue between the user and the AI Tutor was used as part of the context for each question. The rewriter considers this context and rewrites the query to resolve ambiguous requests such as 'Explain this concept further.'

4.3 Biomedical Question and Answer

The previous case studies focused on documents with smaller content sizes. To explore the issues at a larger scale we created a RAG system using the BioASQ [10] dataset comprised of questions, links to document, and answers. The answers to questions were one of yes/no, text summarisation, factoid, or list. This dataset was prepared by biomedical experts and contains domain specific question and answer pairs. We downloaded 4017 open access documents from the BioASQ dataset and had a total of 1000 questions. All documents were indexed and the questions asked against the RAG system. The generated questions were then evaluated using the

[5] https://figshare.com/s/fbf7805b5f20d7f7e356

UPDF
WWW.UPDF.COM

Case Study	Domain	Doc Types	Dataset Size	RAG Stages	Sample Questions
Cognitive Reviewer*	Research	PDFs	(Any size)	Chunker, Rewriter, Retriever, Reader	What are the key points covered in this paper?
AI Tutor*	Education	Videos, HTML, PDF	38	Chunker, Rewriter, Retriever, Reader	What were the topics covered in week 6?
BioASQ	Biomedical	Scientific PDFs	4017	Chunker, Retriever, Reader	Define pseudotumor cerebri. How is it treated?

Table 1: A summary of the RAG case studies presented in this paper. Case studies marked with a * are running systems currently in use.

OpenEvals technique implemented by OpenAI[6]. From the generated questions we manually inspected 40 issues and all issues that the OpenEvals flagged as inaccurate. We found that the automated evaluation was more pessimistic than a human rater for this domain. However, one threat to validity with this finding is that BioASQ is a domain specific dataset and the reviewers were not experts i.e. the large language model may know more than a non-expert.

5 FAILURE POINTS OF RAG SYSTEMS

From the case studies we identified a set of failure points presented below. The following section addresses the research question *What are the failure points that occur when engineering a RAG system?*

FP1 Missing Content The first fail case is when asking a question that cannot be answered from the available documents. In the happy case the RAG system will respond with something like "Sorry, I don't know". However, for questions that are related to the content but don't have answers the system could be fooled into giving a response.

FP2 Missed the Top Ranked Documents The answer to the question is in the document but did not rank highly enough to be returned to the user. In theory, all documents are ranked and used in the next steps. However, in practice the top K documents are returned where K is a value selected based on performance.

FP3 Not in Context - Consolidation strategy Limitations Documents with the answer were retrieved from the database but did not make it into the context for generating an answer. This occurs when many documents are returned from the database and a consolidation process takes place to retrieve the answer.

FP4 Not Extracted Here the answer is present in the context, but the large language model failed to extract out the correct answer. Typically, this occurs when there is too much noise or contradicting information in the context.

FP5 Wrong Format The question involved extracting information in a certain format such as a table or list and the large language model ignored the instruction.

FP6 Incorrect Specificity The answer is returned in the response but is not specific enough or is too specific to address the user's need. This occurs when the RAG system designers have a desired outcome for a given question such as teachers for students. In this case, specific educational content should be provided with answers not just the answer. Incorrect specificity also occurs when users are not sure how to ask a question and are too general.

[6]https://github.com/openai/evals

FP7 Incomplete Incomplete answers are not incorrect but miss some of the information even though that information was in the context and available for extraction. An example question such as "What are the key points covered in documents A, B and C?" A better approach is to ask these questions separately.

6 LESSONS AND FUTURE RESEARCH DIRECTIONS

The lessons learned from the three case studies are shown in Table 2. We present our findings for the research question: *What are the key considerations when engineering a RAG system?* Based on our takeaways we identified multiple potential research areas linked to RAG as follows:

6.1 Chunking and Embeddings

Chunking documents sounds trivial. However, the quality of chunking affects the retrieval process in many ways and in particular on the embeddings of the chunk then affects the similarity and matching of chunks to user queries. There are two ways of chunking: heuristics based (using punctuation, end of paragraph, etc.), and semantic chunking (using the semantics in the text to inform start-end of a chunk). Further research should explore the tradeoffs between these methods and their effects on critical downstream processes like embedding and similarity matching. A systematic evaluation framework comparing chunking techniques on metrics like query relevance and retrieval accuracy would benefit the field.

Embeddings represent another active research area, including generating embeddings for multimedia and multimodal chunks such as tables, figures, formulas, etc. Chunk embeddings are typically created once during system development or when a new document is indexed. Query preprocessing significantly impacts a RAG system's performance, particularly handling negative or ambiguous queries. Further research is needed on architectural patterns and approaches [5] to address the inherent limitations with embeddings (quality of a match is domain specific).

6.2 RAG vs Finetuning

LLMs are great world models due to the amount of training data, and finetuning tasks applied on the model before it's released. However, these models are general-purpose models (may not know the very specifics of your domain) and also not up to date (there is a cutoff date on their knowledge). Fine-tuning and RAG offer two potential customisation pathways, each with distinct tradeoffs. Finetuning requires curating internal datasets to adapt and train the LLM on. However, all your data are baked into the model and you need to

UPDF
WWW.UPDF.COM

FP	Lesson	Description	Case Studies
FP4	Larger context get better results (Context refers to a particular setting or situation in which the content occurs)	A larger context enabled more accurate responses (8K vs 4K). Contrary to prior work with GPT-3.5 [13]	AI Tutor
FP1	Semantic caching drives cost and latency down	RAG systems struggle with concurrent users due to rate limits and the cost of LLMs. Prepopulate the semantic cache with frequently asked questions [1].	AI Tutor
FP5-7	Jailbreaks bypass the RAG system and hit the safety training.	Research suggests fine-tuning LLMs reverses safety training [11], test all fine-tuned LLMs for RAG system.	AI Tutor
FP2, FP4	Adding meta-data improves retrieval.	Adding the file name and chunk number into the retrieved context helped the reader extract the required information. Useful for chat dialogue.	AI Tutor
FP2, FP4-7	Open source embedding models perform better for small text.	Opensource sentence embedding models performed as well as closed source alternatives on small text.	BioASQ, AI Tutor
FP2-7	RAG systems require continuous calibration.	RAG systems receive unknown input at runtime requiring constant monitoring.	AI Tutor, BioASQ
FP1, FP2	Implement a RAG pipeline for configuration.	A RAG system requires calibrating chunk size, embedding strategy, chunking strategy, retrieval strategy, consolidation strategy, context size, and prompts.	Cognitive Reviewer, AI Tutor, BioASQ
FP2, FP4	RAG pipelines created by assembling bespoke solutions are suboptima.	End-to-end training enhances domain adaptation in RAG systems [18].	BioASQ, AI Tutor
FP2-7	Testing performance characteristics are only possible at runtime.	Offline evaluation techniques such as G-Evals [14] look promising but are premised on having access to labelled question and answer pairs.	Cognitive Reviewer, AI Tutor

Table 2: The lessons learned from the three case studies with key takeaways for future RAG implementations

sort out the security/privacy (who can access what). Furthermore, as the foundation model itself evolves or you get new data to add to the model, you will need to run finetuning again. On the other side, RAG systems seem to offer a pragmatic solution allowing you to chunk your data as needed and only use relevant chunks into the context to ask the LLM to generate an answer from the included context. This facilitates continuously updating the knowledge with new documents and also gives the control over what chunks the user is able to access. However, optimal strategies for chunk embedding, retrieval, and contextual fusion remain active research. Further work should systematically compare finetuning and RAG paradigms across factors including accuracy, latency, operating costs, and robustness.

6.3 Testing and Monitoring RAG systems

Software engineering best practices are still emerging for RAG systems. Software testing and test case generation are one of the areas for refinement. RAG systems require questions and answers that are application specific often unavailable when indexing unstructured documents. Emerging work has considered using LLMs for generating questions from multiple documents [4]. How to generate realistic domain relevant questions and answers remains an open problem.

Once suitable test data is available quality metrics are also required to assist engineers in making quality tradeoffs. Using large language models is expensive, introduces latency concerns, and has performance characteristics that all change with each new release. This characteristic has previously been studied for machine learning systems [5, 6] but the required adaptations (if any) have yet to be applied to LLM based systems such as RAGs. Another idea is to incorporate ideas from self-adaptive systems to support monitoring and adapting RAG systems, preliminary work has started for other machine learning applications [2].

7 CONCLUSION

RAG systems are a new information retrieval that leverages LLMs. Software engineers increasingly interact with RAG systems a) through implementing semantic search, or b) through new code-dependent tasks. This paper presented the lessons learned from 3 case studies including an empirical investigation involving 15,000 documents and 1000 questions. Our findings provide a guide to practitioners by presenting the challenges faced when implementing RAG systems. We also included future research directions for RAG systems related to 1) chunking and embeddings, 2) RAG vs Finetuning, and 3) Testing and Monitoring. Large language models are going to continue to obtain new capabilities of interest to engineers and researchers. This paper presents the first investigation into RAG systems from a software engineering perspective.

ACKNOWLEDGMENTS

To Amanda Edgar, Rajesh Vasa, Kon Mouzakis, Matteo Vergani, Trish McCluskey, Kathryn Perus, Tara Draper, Joan Sutherland and Ruary Ross for their support and involvement in making the AI Tutor project possible.

UPDF
WWW.UPDF.COM

REFERENCES

[1] Fu Bang. 2023. GPTCache: An Open-Source Semantic Cache for LLM Applications Enabling Faster Answers and Cost Savings. In *3rd Workshop for Natural Language Processing Open Source Software*.

[2] Maria Casimiro, Paolo Romano, David Garlan, Gabriel Moreno, Eunsuk Kang, and Mark Klein. 2022. *Self-adaptive Machine Learning Systems: Research Challenges and Opportunities*. 133–155. https://doi.org/10.1007/978-3-031-15116-3_7

[3] Jiawei Chen, Hongyu Lin, Xianpei Han, and Le Sun. 2023. Benchmarking Large Language Models in Retrieval-Augmented Generation. *arXiv preprint arXiv:2309.01431* (2023).

[4] Mingda Chen, Xilun Chen, and Wen-tau Yih. 2023. Efficient Open Domain Multi-Hop Question Answering with Few-Shot Data Synthesis. *arXiv preprint arXiv:2305.13691* (2023).

[5] Alex Cummaudo, Scott Barnett, Rajesh Vasa, and John Grundy. 2020. Threshy: Supporting safe usage of intelligent web services. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1645–1649.

[6] Alex Cummaudo, Scott Barnett, Rajesh Vasa, John Grundy, and Mohamed Abdelrazek. 2020. Beware the evolving 'intelligent'web service! An integration architecture tactic to guard AI-first components. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 269–280.

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

[8] Sebastian Hofstätter, Jiecao Chen, Karthik Raman, and Hamed Zamani. 2023. Fid-light: Efficient and effective retrieval-augmented text generation. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1437–1447.

[9] Gautier Izacard and Edouard Grave. 2020. Leveraging passage retrieval with generative models for open domain question answering. *arXiv preprint arXiv:2007.01282* (2020).

[10] Anastasia Krithara, Anastasios Nentidis, Konstantinos Bougiatiotis, and Georgios Paliouras. 2023. BioASQ-QA: A manually curated corpus for biomedical question answering. *Scientific Data* 10 (2023), 170. Citation Key: 422.

[11] Simon Lermen, Charlie Rogers-Smith, and Jeffrey Ladish. 2023. LoRA Fine-tuning Efficiently Undoes Safety Training in Llama 2-Chat 70B. arXiv:2310.20624 [cs.LG]

[12] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems* 33 (2020), 9459–9474.

[13] Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2023. Lost in the middle: How language models use long contexts. *arXiv preprint arXiv:2307.03172* (2023).

[14] Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. 2023. G-eval: Nlg evaluation using gpt-4 with better human alignment, may 2023. *arXiv preprint arXiv:2303.16634* (2023).

[15] Noor Nashid, Mifta Sintaha, and Ali Mesbah. 2023. Retrieval-based prompt selection for code-related few-shot learning. In *Proceedings of the 45th International Conference on Software Engineering (ICSE'23)*.

[16] OpenAI. 2023. GPT-4 Technical Report. https://doi.org/10.48550/ARXIV.2303.08774

[17] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. 2023. Robust speech recognition via large-scale weak supervision. In *International Conference on Machine Learning*. PMLR, 28492–28518.

[18] Shamane Siriwardhana, Rivindu Weerasekera, Elliott Wen, Tharindu Kaluarachchi, Rajib Rana, and Suranga Nanayakkara. 2023. Improving the domain adaptation of retrieval augmented generation (RAG) models for open domain question answering. *Transactions of the Association for Computational Linguistics* 11 (2023), 1–17.

[19] Yutao Zhu, Huaying Yuan, Shuting Wang, Jiongnan Liu, Wenhan Liu, Chenlong Deng, Zhicheng Dou, and Ji-Rong Wen. 2023. Large language models for information retrieval: A survey. *arXiv preprint arXiv:2308.07107* (2023).