UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

VICTOR BRANDÃO DE ANDRADE

EVALUATING THE EFFECT OF RETRIEVAL AUGMENTED GENERATION IN MISTRAL-7B-INSTRUCT-V0.2'S CLOJURE'S CODE REVIEW

CAMPINA GRANDE - PB

2024

VICTOR BRANDÃO DE ANDRADE

EVALUATING THE EFFECT OF RETRIEVAL AUGMENTED GENERATION IN MISTRAL-7B-INSTRUCT-V0.2'S CLOJURE'S CODE REVIEW

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador : João Arthur Brunet Monteiro

CAMPINA GRANDE - PB

2024

VICTOR BRANDÃO DE ANDRADE

EVALUATING THE EFFECT OF RETRIEVAL AUGMENTED GENERATION IN MISTRAL-7B-INSTRUCT-V0.2'S CLOJURE'S CODE REVIEW

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

BANCA EXAMINADORA:

João Arthur Brunet Monteiro
Orientador – UASC/CEEI/UFCG

Adalberto Cajueiro de Farias
Examinador – UASC/CEEI/UFCG

Francisco Vilar Brasileiro
Professor da Disciplina TCC – UASC/CEEI/UFCG

Trabalho aprovado em: 15 de Maio de 2024.

CAMPINA GRANDE - PB

RESUMO

Revisão de código é uma das atividades mais importantes da engenharia de software, visto que visa garantir a qualidade e confiabilidade do código, mas esse processo é feito majoritariamente de maneira manual, o que pode demandar tempo e tornar o processo oneroso e suscetível a falhas. O processo de revisão de código é um forte candidato para automação com objetivo de torná-lo mais eficiente e menos suscetível a falhas devido ao componente humano do processo. Neste trabalho, nós desejamos explorar a automação do processo de revisão de código através da aplicação de Grandes Modelos de Linguagem e uma técnica de otimização no contexto de revisão de código Clojure, que é uma linguagem de programação emergente. O Grande Modelo de Linguagem escolhido foi o Mistral-7B-Instruct-v0.2 e a técnica de otimização foi a Retrieval Augmented Generation (RAG), ambos os tópicos são discutidos nas seções seguintes deste trabalho. Nossos resultados mostram que o Mistral com e sem o uso da otimização com RAG pode revisar código como humanos, mas RAG não melhorou a revisão do modelo.

Evaluating the effect of Retrieval Augmented Generation in Mistral-7B-Instruct-v0.2's Clojure's code review

Victor Brandão De Andrade
Federal University of Campina Grande
Campina Grande, Brazil
victor.andrade@ccc.ufcg.edu.br

João Arthur Brunet Monteiro
Federal University of Campina Grande
Campina Grande, Brazil
joao.arthur@computacao.ufcg.edu.br

ABSTRACT

Code review is one of the most important activities in software engineering, since it intends to guarantee code's quality and reliability, but this process is done mostly manually, which can make it an onerous, time-consuming and a failure-susceptible task. The code review process is a strong candidate for automation in order to make it more efficient and less susceptible to failures due to its human component. In this work, we intend to explore the automation of the code review process by applying a LLM and an optimization technique in the context of Clojure's code review, which is an emergent programming language. The LLM chosen was Mistral-7B-Instruct-v0.2 and the optimization technique was Retrieval Augmented Generation (RAG), both topics are discussed in the following sections of this work. Our results show that Mistral with and without the RAG optimization can review code like humans, but RAG didn't improve the model's review.

KEYWORDS

Clojure, Code Review, RAG, LLMs.

1 INTRODUCTION

Large Language Models (LLMs) are Deep Learning models based on the Transformer architecture released by Google in 2017 with the paper "*Attention Is All You Need*"[26], which represents a major turning point in the Deep Learning study field. Since Transformer's release, the LLMs' studies have significantly evolved with the releases of *GPT-3*[3] by OpenAI, *Llama2*[24] by Meta, *Gemini*[23] by Google and *Mistral*[13] by MistralAI, models that have been getting popular. These models are pre-trained with a massive amount of data, acquiring general knowledge due to their ability of learning underlying patterns of the language, which makes them capable of learning and performing Natural Language Processing (NLP) tasks, such as text summarization, text generation, translation, etc.

Once the pre-trained models are generalists, sometimes they need to be fine-tuned, which is the process of further training the model on a smaller and task-specific dataset to specialize the model for the specific task that it's intended to solve. Training and fine-tuning LLMs for a number of tasks might be a challenge regarding the computing resources required, once they usually require GPUs and sometimes even TPUs, as it's discussed in the study "*Energy and Policy Considerations for Deep Learning in NLP*"[22]. With this challenge in mind, researchers and practitioners have been proposing approaches to improve these models' performance without the need to perform training or fine-tuning, like Prompt Engineering and *Retrieval Augmented Generation (RAG)*[14], that has been explored due to its potential to generate correct and credible responses, as it's discussed in the study "*Retrieval Augmented Generation for Large Language Models: A Survey*"[9]. There are several studies like "*Retrieval-Generation Synergy Augmented Large Language Models*"[8] that explore the application of RAG with LLMs and even studies that improved RAG like *RAG-Fusion*[19]. RAG was proposed by Meta's researchers, and consists of using non-parametric memory (an external knowledge source) along with the parametric memory (knowledge stored in the model's parameters) to help the model to generate more accurate responses. One of the greatest advantages of RAG is that its knowledge can be easily altered or supplemented on the fly without the need of fine-tuning.

The application of LLMs in several fields has grown in the past years. One of these fields is Software Engineering (SE) as it's discussed in "*Applications of LLMs to Software Engineering Tasks: Opportunities, Risks, and Implications*"[18]. LLMs have been used for a number of SE tasks, such as the generation of unit tests, language translation to modernize legacy software with new programming languages, code review and build code generation tools like *Codex*[4], which was the base model for *GitHub Copilot*[1]. One of the most important activities in Software Engineering is Code Review as it's discussed in "*Design and code inspections to reduce errors in program development*"[7], since it intends to guarantee the code's quality, consequently preventing bugs and troubles with code's maintainability. However, the code review process is still more of a manual process, and that makes it onerous and susceptible to failures, which is the reason to seek its automation as it's discussed in "*Automating Code Review Activities by Large-Scale Pre-Training*"[15] that proposed the CodeReviewer model and "*Using Pre-Trained Models to Boost Code Review Automation*"[25] that explored the T5 model, both studies in the context of applying LLMs in the task of code review.

As mentioned, there are studies that have explored the application of LLMs to review code[15, 25], but these aforementioned studies focus on popular programming languages, such as Java, Python, C, etc. However, as emergent as the LLMs is the programming language *Clojure*[12], a dialect of Lisp designed in 2005 and released in 2007, that has been getting popular and being adopted by great startups (e.g. Nubank, MercadoLibre, etc.), due to its pure functional characteristics like immutability and its tools to concurrency-safe state management. Since both, LLMs and Clojure are emergent, there is a lack of knowledge related to the application of LLMs to review Clojure code, and the goal of this work is to bridge this gap by analyzing the performance of the *Mistral-7B-Instruct-v0.2*[2] while reviewing 1636 diff hunk from Clojure pull requests extracted from 100 different open source projects. We conducted experiments with

[1] https://github.com/features/copilot
[2] https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2

and without the use of RAG to also shed light on the improvement of such technique when applied in this context. As code artifacts[3], we generate the code that runs our experiment and analyze its results and this paper.

This paper is organized as follows. In the next section, we provide the background to better understand this work. Then, in Section 3, we detail the experiment's methodology; in Section 4 discusses the results; Section 5 concludes our findings and proposes future works.

2 BACKGROUND AND RELATED WORKS

In this section, we present the key concepts for conducting this research. We explain the importance of code review, the pull based software development and the fundamentals behind RAG.

2.1 Large Language Models

Large Language Models (LLMs) are Deep Learning models based on the Transformer architecture released by Google in 2017 with the paper "*Attention Is All You Need*"[26]. These models have shown the ability to learn the underlying patterns of the language used in their pre-training. They are pre-trained in a huge amount of data and become able to generate human-like content, which makes them capable of performing human-like tasks such as read, write, code, etc.

2.2 Code Review

As discussed in "*Design and Code Inspections to Reduce Errors in Program Development*"[10] the process of code's inspection or code review is an important activity in the software development process, since it provides a way to find errors in design and code, resulting in more reliable and maintainable systems. The process of code review has been modernized through the years as it's discussed in "*An Exploratory Study of the Pull-Based Software Development Model*"[17], mostly because of version control tools like *Git*[4] and the *GitHub*[5] system, that makes it easier the collaboration between several developers through its Pull Request system, that provides a way to perform modern code reviews, allowing discussions between developers, code conflict verification and demanding the approval of the submitted change by the reviewers before being integrated with the code.

The work "*What makes a Code Review Trustworthy?*"[16] presents that the code review process depends on the reviewers' expertise and experience, and that even with several code standards being defined, not everything can be checked and validated. Since the process depends on the reviewers' expertise, it is performed mostly manually, which means that its workload can grow exponentially as the size of the project increases, which can easily make the process onerous and susceptible to failures. So the automation of this process is an urgent demand, as it's discussed in forementioned works that applied LLMs to automate this process[15, 25]. Another work that has done the same thing, but using Llama as model, was "*LLaMA-Reviewer: Advancing Code Review Automation with Large Language Models through Parameter-Efficient Fine-Tuning*"[11] where the researches fine-tune the Llama model in three following tasks: Review Necessity Prediction, Review Comment Generation and Code Refinement to produce the LLaMA-Reviewer.

2.3 Retrieval Augmented Generation

Retrieval Augmented Generation (RAG) is a technique that enhances the accuracy and reliability of LLMs responses by adding an external data source to help the model to generate more accurate responses. The idea consists of combining parametric memory of the generator component, which is the knowledge stored in the model's parameters after pre-train in a large dataset, with non-parametric memory, which is represented by an index of relevant documents accessed by a retriever component. In RAG systems, instead of passing the prompt right through the generator component, the prompt is first passed to the retriever component, which searches for the most relevant documents related to the received prompt and then concatenates the retrieved documents to the original prompt as context to help the generator component. The RAG's retriever component usually implements semantic search, which is a kind of information retrieval that interprets the meaning of the sentences and tries to retrieve the most relevant documents using the document's semantic and context.

Since the document index used by the retriever is a dynamic source, it can be easily updated with new information as the world's knowledge evolves and change, which makes it an easier approach to update the model's knowledge when compared with fine-tuning the model, a task that usually requires large computational resources and can even make the base model forgets some knowledge acquired during its pre-train, as it's discussed in "*Analyzing the Forgetting Problem in Pretrain-Finetuning of Open-domain Dialogue Response Models*"[21].

2.4 Prompt Engineering

Prompt Engineering as it is discussed in "*A Systematic Survey of Prompt Engineering*"[1] has emerged as an approach to extend the capabilities of LLMs. The approach consists of giving the model task-specific instructions, also known as prompt, to enhance the model's response without modifying the model's parameters. This technique has shown significant potential when it comes to enhancing the adaptability and applicability of the model, without the need for fine-tuning the model to the specific task, which was already mentioned in the previous section to be expensive in terms of computing resources.

In cases where the model does not have knowledge to perform the required task, it tries to use its parametric memory to answer, which can lead to hallucinations. But it is possible to give the model input-output examples as a workaround, so it tries to answer based on the examples. RAG, as mentioned before, takes advantage of that, but tries to retrieve the best examples based on the input, so the model gets the best information.

[3] https://drive.google.com/drive/folders/135piPPDe0jHowMVvipAbLWTtkC1zNfpP?usp=drive_link
[4] https://git-scm.com/
[5] https://github.com/

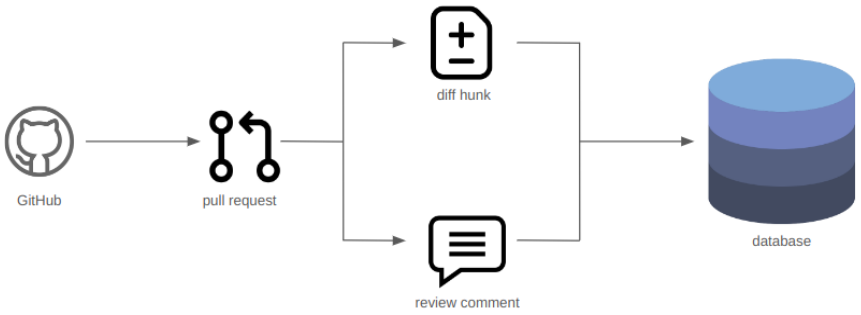

Figure 1: Dataset collection workflow

3 METHODOLOGY

In this section, we present how our code review dataset was built and how we design our experiment, which has as main goal, evaluating how the use of RAG improves the Mistral-7B-Instruct-v0.2 code review of Clojure code. For further reference, we named the model that ran without RAG as Mistral and the model that ran with RAG as Mistral-RAG.

3.1 Research Questions

Our research is guided by the following research questions:

RQ1: Is the model's review similar to the human's review?

RQ2: Considering the similarity between the model's review and the human's review, did RAG improve the model's review when compared to the results of Mistral?

3.2 Dataset Collection

Initially, we selected the Top 100 Clojure open-source projects in GitHub, which is a ranking that uses as a quality metric the number of stars that a project has. We decided to select the most popular projects, because they were most likely to have developer's comments and discussions in its pull requests. By doing so, we got the diff hunks from the pull request, which represent the submitted modifications and also the reviewer's comments written by a human reviewer about the submitted change, information that we can use later to evaluate the model's review.

As it was done in the study "*Automating Code Review Activities by Large-Scale Pre-Training*"[15], we choose to use diff hunk in our work. So, it was written, a Golang script, that was responsible for performing the requests to extract pull request data using GitHub's API. After the execution of the script, we retrieve over 12 thousands pull requests from the Top 100 Clojure projects to build our dataset[6]. After the retrieval stage, we performed some preprocessing over the extracted data to remove noise and unnecessary information, like the annotations about the number of the changed lines, lines of code that were removed in the diff hunk, code comments and filtering the data to have only examples where the number of lines were greater or equal than 3, because we wanted to remove examples with few lines of code and that consequently had less information to review, like imports. We ended with 6771 examples in our dataset and Figure 1 presents how the extraction process was done.

3.3 Tools

The Mistral-7B model released by MistralAI has shown high performance while maintaining an efficient inference. As it is discussed in the model's paper[13], Mistral-7B leverages GQA[5], which accelerates the inference speed and reduces the memory requirements during the inference, and SWA[2, 20], which was designed to handle longer sequences more effectively at a reduced computational cost. As a result, it outperformed the Llama2 13B across all tested benchmarks, and surpassed the Llama 34B in mathematics and code generation. Mistral-7B also approaches the coding performance of Code-Llama 7B. To perform this paper's experiment, we choose the Mistral-7B-Instruct-v0.2, which is a version of the base model that was fine-tuned in instruction datasets.

To apply RAG, we choose *Sentence-BERT*[7] [6] as the embedding model used to create the vector representations of the diff hunks, that were stored in the *Chromadb*[8] vector database, which was configured to use the cosine similarity in its embeddings search and used as the model's external source of knowledge when we ran Mistral along with RAG.

3.4 Prompt

3.4.1 *Mistral Prompt.*

As the prompt for the model, we used the following structure, where we start by telling the model that it is an experienced programmer, and ask it to review a given diff hunk. We decided to ask the review to be itemized instructions so it would be easier to compare with the human review, once the review is expected to contain the instructions for the developer that wrote the code. The <s> and [INST] are special tokens recommended[9] in order to get optimal outputs.

```
<s>[INST] Given that you are an experienced
```

[6]https://github.com/raiaiaia/llm-code-review-clj/blob/main/Dataset/data_filtered.csv

[7]https://huggingface.co/google-bert/bert-base-uncased

[8]https://www.trychroma.com/

[9]https://community.aws/content/2dFNOnLVQRhyrOrMsloofnW0ckZ/how-to-prompt-mistral-ai-models-and-why

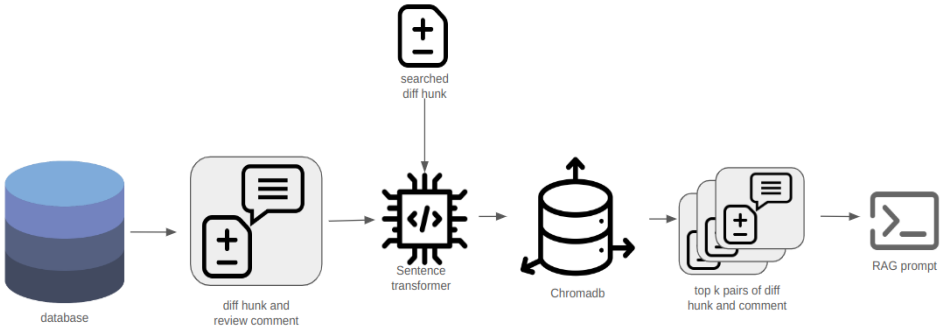

Figure 2: RAG's prompt building process

```
programmer, review the code snippet below:

{diff_hunk}

Format your review as text with itemized
concrete instructions to the author of
the code and do not add this prompt to
the answer. [/INST]
```

3.4.2 *Mistral-RAG Prompt.*

When running Mistral along with RAG, as it was discussed in RAG's section, where we explained how it works, we just concatenate the retrieved documents to the prompt to work as code review examples for the model to use as the base of its review. We formatted each of the retrieved documents as follows, presenting the diff hunk and the review comment for it.

```
<s>[INST] Given that you are an experienced
programmer, review the code snippet below:

{diff_hunk}

Review based on the following review examples:

The following code
{example_diff_hunk_1}
generate the following review comment
{example_review_comment_1}
...

Format your review as text with itemized
concrete instructions to the author of
the code and do not add this prompt to
the answer. [/INST]
```

3.5 Experiment

As discussed before, our experiment was divided in two phases, in the first phase we run the model Mistral-7B-Instruct-v0.2 in the code review task over our Clojure diff hunk dataset and collect the model's responses. In the second phase, we ran the model using RAG to pass some of our reviewed diff hunks as context for the model, in order to improve its review quality. We first have populated the Chromadb with the embedding representations generated using *Sentence-BERT*[6] and used the search system of Chromadb, which was configured to use cosine distance between the embedding representations of the diff hunks to retrieve the most similar hunks according to the one that is being searched, and then we pass as context the diff hunk followed by its human review comment, so the model has examples of similar code reviews. Figure 2 represents the two phases of our experiment, and Figure 3 represents how we created RAG's prompt.

Our independent variable is, whether we use RAG, and our dependent variable is the similarity between model's review and human's review that was collected. Then we formulate the two following hypotheses, based on our research questions:

For the Research Question 1 we have:

- The null hypothesis is "The model does not review code like a human reviewer" and its corresponding alternative hypothesis is "The model can review code like a human reviewer".

For the Research Question 2 we have:

- The null hypothesis is "There is no difference between the similarities found with Mistral and Mistral-RAG" and its corresponding alternative hypothesis is "Using Mistral-RAG, the model's reviews are more similar with the human reviews".

The environment where the experiment was executed was the *Google Colab*[10] platform, using the Colab Pro+ which gives us 500 computing units per month and has an *NVIDIA A100 40 GB PCIe GPU Accelerator*[11], 84 GB of RAM and 200 GB of disk space available. Figure 5 represents the overview of our experiment. Once we have

[10] https://colab.google/
[11] https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/A100-PCIE-Prduct-Brief.pdf

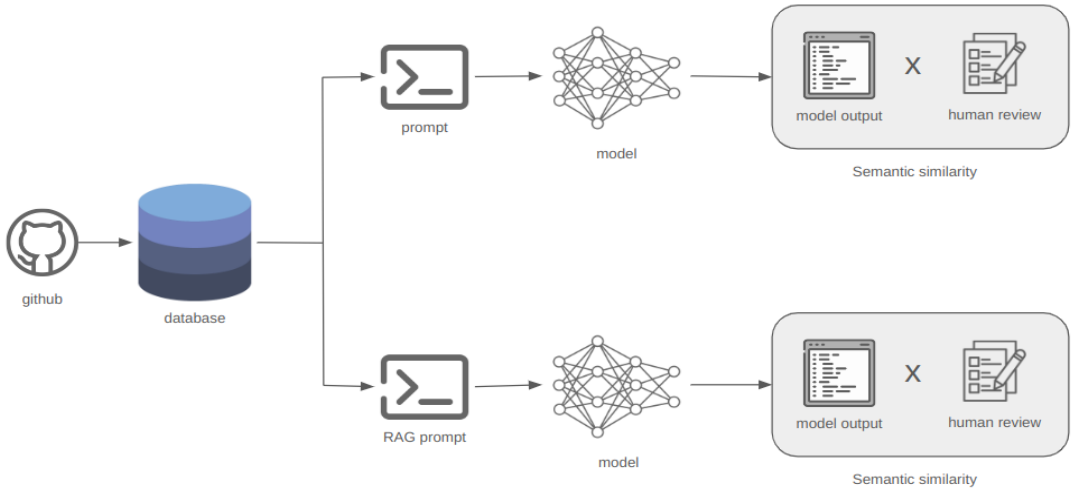

Figure 3: Experiment workflow

limited resources, we could only run our experiment over a limited subset of our dataset, which was 1636 examples from our 6771 examples.

4 RESULTS

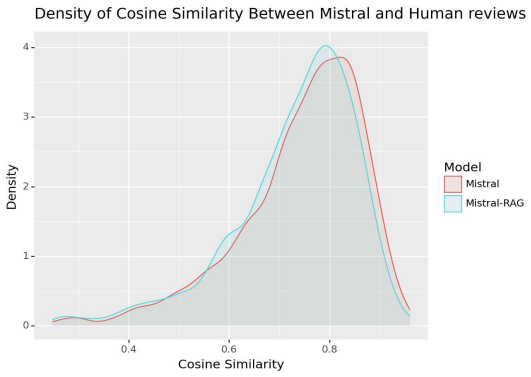

Figure 4: Density of Similarities

To evaluate the performance of our model we used the *Sentence-BERT*[6] to generate embeddings of the Mistral's review comment and the human review comment, and then we used the Cosine Similarity between the embeddings in order to verify if the model review code like a human reviewer. Table 1 shows the results of Mistral and Mistral-RAG executions over our dataset with 1636 examples.

As it's shown in Table 1, we can see that we have found high values of cosine similarity, once that the 50% percentile or median is over 75% in both cases, which means that 50% of our dataset is over 75% of cosine similarity, so we can conclude that the model's review is close to human reviews, which answers our Research Question 1. Figure 4 shows a density plot divided between Mistral and Mistral-RAG. We can see that both executions, with and without RAG generated similar results, with the RAG execution being slightly worse than when we ran without it, which contradicts our hypothesis of RAG improving the model's responses, that was formulated based on our Research Question 2.

	Mistral	Mistral-RAG
Count	1636	1636
Min	25%	25%
Max	96%	96%
Mean	74%	73%
Standard Deviation	12%	12%
25 percentile	68%	67%
50 percentile	77%	76%
75 percentile	83%	82%

Table 1: Similarity between Mistral and Human Review summary

By the Figure 6 we can see that our dataset concentrate around 80% of cosine similarity, which is reinforced by the standard deviation being 12%, what tells us that our data is clustered tightly around the mean that was 74% for Mistral and 73% for Mistral-RAG.

5 CONCLUSION AND FUTURE WORKS

As discussed in the results section, the hypothesis formulated based on our Research Question 1 was reinforced by our results, once that we can see the model's reviews had high similarity with the human reviews. However, we contradicted our hypothesis formulated based on the Research Question 2, but as result of that we could also formulate a hypothesis on why that happened. Among the possible reasons for that, we believe that the strongest one is the fact that we weren't able to run our experiment with our complete dataset, due to our resources' limitation. When working with LLMs, one of the most important factors is the amount of data available

to run the experiment, and we had an already limited dataset with 6771 examples that became smaller because we were able to run only 1636 examples. Due to our resources' limitations, we couldn't test different prompt settings, which we think is one of the most important settings of our experiment.

We think that the quality and the amount of reviews passed as examples within the prompt probably had a negative impact on the results. Our hypothesis over this outcome is that diff hunks probably confuse the model, once it didn't have the full context of the code. The main difference between our work and the previous works that found good results is the amount of data available, once the previous works were done with languages like Java and Python, which have a bigger amount of data available to use than Clojure, which is an emergent language.

For future work, we want to run our experiment with our complete dataset, and evaluate if the amount of data really has an effect when compared to our primary result that we showed in this paper. We also would like to test different prompt settings and other vector databases, different from Chromadb, to evaluate if it can generate better results when used with Mistral-RAG.

6 ACKNOWLEDGMENTS

Firstly, I would like to thank God for the opportunity of taking this bachelor's degree. I would like to thank my advisor João Arthur Brunet Monteiro for all the trust that he has put in me since the beginning of my journey in this course, and all the help and knowledge shared with me. I also would like to thank all the members of the study group that I joined in the beginning of this work, but I want to thank, specially Rayanne, who was a really important person through this research and without her help, this might not be possible, so I leave here my thanks for all her support.

I would like to thank my family for all the support they gave me through this entire degree, specially my cousin Nayara, who has always been an inspiration for me when it comes to academic things. Finally, through my years at UFCG I met a lot of incredible people, so many that I can't mention everyone, so I'll leave here my thanks to all of them, but specially to my friends - Eric, Gabriel, Ruan and Wellisson - that since the beginning of this degree made my days more funny and helped me to study.

REFERENCES

[1] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. 2023. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. *arXiv preprint arXiv:2305.13245* (2023).

[2] Iz Beltagy, Matthew E Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150* (2020).

[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[4] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* (2021).

[5] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509* (2019).

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[7] Michael E Fagan. 1999. Design and code inspections to reduce errors in program development. *IBM Systems Journal* 38, 2.3 (1999), 258–287.

[8] Zhangyin Feng, Xiaocheng Feng, Dezhi Zhao, Maojin Yang, and Bing Qin. 2024. Retrieval-generation synergy augmented large language models. In *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 11661–11665.

[9] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997* (2023).

[10] Georgios Gousios, Martin Pinzger, and Arie van Deursen. 2014. An exploratory study of the pull-based software development model. In *Proceedings of the 36th international conference on software engineering*. 345–355.

[11] Tianxing He, Jun Liu, Kyunghyun Cho, Myle Ott, Bing Liu, James Glass, and Fuchun Peng. 2021. Analyzing the forgetting problem in pretrain-finetuning of open-domain dialogue response models. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*. 1121–1133.

[12] Rich Hickey. 2020. A history of Clojure. *Proceedings of the ACM on programming languages* 4, HOPL (2020), 1–46.

[13] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7B. *arXiv preprint arXiv:2310.06825* (2023).

[14] 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.

[15] Zhiyu Li, Shuai Lu, Daya Guo, Nan Duan, Shailesh Jannu, Grant Jenks, Deep Majumder, Jared Green, Alexey Svyatkovskiy, Shengyu Fu, et al. 2022. Automating code review activities by large-scale pre-training. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1035–1047.

[16] Junyi Lu, Lei Yu, Xiaojia Li, Li Yang, and Chun Zuo. 2023. LLaMA-Reviewer: Advancing Code Review Automation with Large Language Models through Parameter-Efficient Fine-Tuning. In *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 647–658.

[17] Stacy Nelson and Johann Schumann. 2004. What makes a code review trustworthy?. In *37th Annual Hawaii International Conference on System Sciences, 2004. Proceedings of the*. IEEE, 10–pp.

[18] Ipek Ozkaya. 2023. Application of large language models to software engineering tasks: Opportunities, risks, and implications. *IEEE Software* 40, 3 (2023), 4–8.

[19] Zackary Rackauckas. 2024. RAG-Fusion: a New Take on Retrieval-Augmented Generation. *arXiv preprint arXiv:2402.03367* (2024).

[20] Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950* (2023).

[21] Pranab Sahoo, Ayush Kumar Singh, Sriparna Saha, Vinija Jain, Samrat Mondal, and Aman Chadha. 2024. A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications. *arXiv preprint arXiv:2402.07927* (2024).

[22] Emma Strubell, Ananya Ganesh, and Andrew McCallum. 2019. Energy and policy considerations for deep learning in NLP. *arXiv preprint arXiv:1906.02243* (2019).

[23] Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805* (2023).

[24] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288* (2023).

[25] Rosalia Tufano, Simone Masiero, Antonio Mastropaolo, Luca Pascarella, Denys Poshyvanyk, and Gabriele Bavota. 2022. Using pre-trained models to boost code review automation. In *Proceedings of the 44th international conference on software engineering*. 2291–2302.

[26] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).