Enhancing RAG Performance through Chunking and Text Splitting Techniques

Abstract:

In the world of Generative Artificial Intelligence (GenAI) and Large Language Models (LLM), Retrieval-Augmented Generation (RAG) has transformed the way we interact with data. Using RAG, these models can leverage new data contexts to respond to user queries and gain valuable insights. Behind the outstanding capabilities of RAG, a fundamental pre-processing step is present known as chunking. This step plays a crucial role in the effectiveness of these RAG enhanced models. Chunking involves the breaking down of large text or documents into smaller segments of a fixed size. This allows the retriever to focus on smaller units at a time, making it easier to process and analyse the text. Finding the ideal chunking strategy can be a challenging task. Experimenting and analysis plays a decisive role here, as different chunking strategies cater to different use cases. This paper, mainly targeted for a for audience that is exploring RAG tuning techniques for higher accuracy, explores the various chunking techniques and their practical implementation using code snippets. After analysing the results for various use cases, the paper also suggests the best use cases for the different chunking strategies. Finally, it concludes by discussing the future potential and extending scope of RAG enhanced applications.

Introduction:

Traditional Large Language Models utilize a technique called pre-training on huge datasets which contain a large amount of information. This allows the models to generate text, answer user questions and even translate languages without any data specific to a task. Nevertheless, their knowledge is still limited. RAG helps these models overcome the limitation of entirely relying on the patterns and data used for model training. RAG combines the power of information retrieval with generative capabilities, allowing it to produce more contextually grounded and accurate responses. This approach also helps RAG to handle a wide range of tasks with greater flexibility and accuracy.

In RAG, data is first converted into vector form. Then, using similarity search algorithms, the system identifies suitable answers based on the user query. These chosen responses are fed into the LLM, which then generates a final response based on the input received. RAG is a methodology used to create use-case-specific GenAI applications.

Chunking strategies can be employed in RAG systems at multiple levels. First, the knowledge base itself can be chunked and indexed in a vector database, enabling efficient retrieval of relevant information based on the prompt. Moreover, the retrieved chunks can further be processed as

input to an LLM, ensuring that the model can effectively integrate the relevant knowledge while also following its maximum sequence length limitations.

Methods and Material:

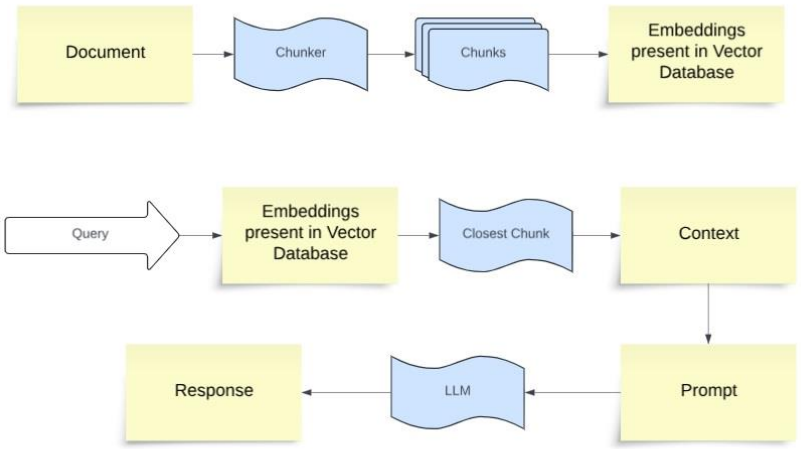

Fig 1: Methodology

Several factors play a crucial role in determining the best chunking strategy, and they vary depending on the use case. Here are some factors which we must keep in mind when selecting the best strategy:

Expectations for the complexity of user queries: The user queries can be short and precise or long and complex. This can decide which way you choose to chunk your content so that there is a closer relation between the embedded query and chunks.

Nature of the content being indexed: The content can be of long format like documents and books or short format like text messages. This factor can decide which model would be appropriate for your goal and which chunking strategy to apply.

Use case of the retrieved results: The results can be used for various use cases like question answering, summarization or semantic search. This also is a crucial factor in deciding the best chunking strategy to use.

Performance metrics of the model being used: Different models have varying performance with different chunking strategies. Therefore, the performance metrics of the model in use is a important factor when choosing a chunking strategy.

Now we will delve into a comparison between various rule and semantic based chunking strategies. Essentially rule based methods rely on definite separators like spaces and libraries like NLTK. Semantic methods can utilize various machine learning algorithms along with a few python libraries to get the context of the text and use its actual meaning for chunking.

A) **Fixed Size Chunking**.

It is the most widely used and straightforward rule-based approach to chunking. We use text fragmentation where text is divided into equal size chunks based on a predetermined number of characters or words. We also have an option whether to allow overlap between adjacent chunks. This makes sure that the context does not get lost between the chunks. This method is the best way to use chunking in most common cases.

It is also very beneficial in in scenarios where the chunk size is more important than the context. Compared to the other forms of chunking, this type is also computationally cheap and easier to use since it does not require any Natural Language Processing (NLP) libraries. It can also be used to process large volumes of text where uniform blocks can be created.

Example of how we can perform Fixed Size Chunking with LangChain:

```
text = "..." # your text
from langchain.text_splitter import
CharacterTextSplitter text_splitter =
CharacterTextSplitter(
# Using new line as a separator
separator = "\n\n",
chunk_size = 256,
chunk_overlap = 20
)
docs = text_splitter.create_documents([text])
```

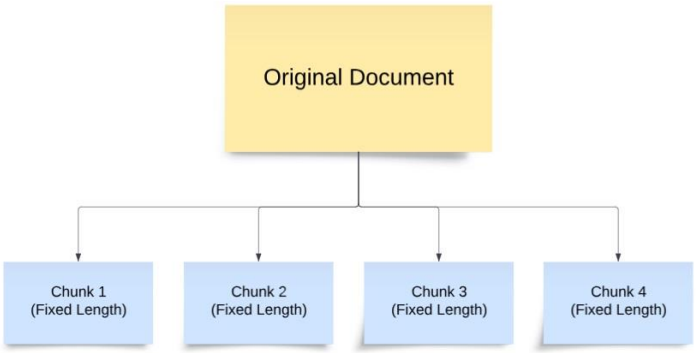

Fig 2: Fixed Size Chunking

B) Content Aware Chunking

1) Recursive Chunking

It is most widely used for processing generic text. Recursive chunking usually operates bases on a list of characters. It attempts to split up the text until the chunks reach an optimal size. If the initial attempt of splitting up the text does not produce chunks of a desired size or structure, the method recursively calls itself on the resulting chunks with a different criterion until the desired chunk size and structure is achieved.

This type of chunking gives priority to split up the text first at paragraph breaks, followed by individual lines, spaces and then finally empty strings. This mainly helps to preserve the semantic relationship of the text as much as possible. Chunks in recursive chunking are not always the same size, but they are of similar size, with the context and semantic relationships intact between them.

Example of how we can perform Recursive Chunking with LangChain:

```
text = "..." # your text
from langchain.text_splitter import RecursiveCharacterTextSplitter
text_splitter = RecursiveCharacterTextSplitter(
# chunk size and overlap
 chunk_size = 256,
chunk_overlap  = 20
)
```

docs = text_splitter.create_documents([text])

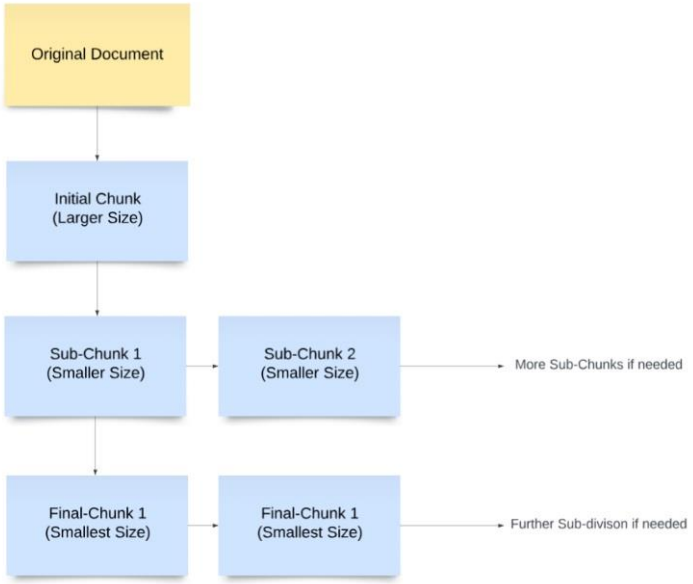

Fig 3: Recursive Chunking

2) **Sentence-Aware Chunking**

There are various LLMs which are specifically designed and optimized for embedding sentence level content. To maximise the capability of these models, sentence-aware chunking is used. This type of chunking segments the text based on sentence boundaries while also following the maximum length allotted to each chunk during the chunking process. This process, makes sure that each segment of text contains a full sentence and the grammatical and contextual flow of the original text is also maintained. This is very helpful, where the understanding of each sentence in the text is crucial.

There are a few ways where we can carry out Sentence-aware chunking. The two most used python libraries for this use case are NLTK and spaCy.

Natural language Toolkit (NLTK) is a widely used python library for dealing with human language data. It offers a sentence tokenizer that can split up text into sentences, which in turn helps to create meaningful chunks.

Example of how we can perform Sentence-Aware Chunking using NLTK with LangChain:

```
text = "..." # your text

from langchain.text_splitter import NLTKTextSplitter
text_splitter = NLTKTextSplitter()

docs = text_splitter.split_text(text)
```

spaCy is another python library which can be used for sentence aware chunking. It contains a sentence segmentation feature that can effectively divide the text into sentences while also preserving the context of the resulting chunks.

Example of how we can perform Sentence-Aware Chunking using spaCy with LangChain:

```
text = "..." # your text

from langchain.text_splitter import SpacyTextSplitter
text_splitter = SpaCyTextSplitter()

docs = text_splitter.split_text(text)
```

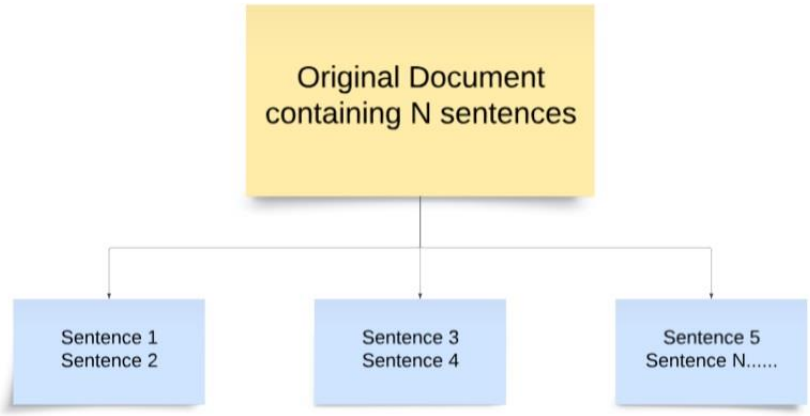

Fig 4: Sentence-Aware Chunking

3) **Semantic Chunking**

This type of chunking uses various advanced NLP tools to segment the text based on the semantic boundaries by using the meaning and context. With the help of LLMs, the ability to create embeddings of the text can be utilized. These embeddings are used to extract the semantic meaning present in the document which in turn creates chunks that contain sentences that have the same context.

To carry out Semantic chunking in an effective way, the document is first split up into sentences. Sentence groups are created for each sentence in the document, with each group containing the sentences that appear before and after the base sentence. After the generation of embeddings for each of these sentence groups, they are associated with the base sentence. Distance between the embeddings for each group is then calculated. If the sentences are of the same meaning or context, the distance between the groups of these sentences will be low. Higher distance between groups signifies that the context of the sentences has been changed.

This process can effectively help the model distinguish each chunk from the next, whether they share the same context or not.

Example of how we can perform Semantic Chunking with LangChain:

```
from langchain_experimental.text_splitter import SemanticChunker
from langchain_openai.embeddings import OpenAIEmbeddings

# specify the path of the document

with open("../../....") as doc:

document content = doc.read()

text_splitter = SemanticChunker(OpenAIEmbeddings())

docs = text_splitter.create_documents([content])
print(docs)
```

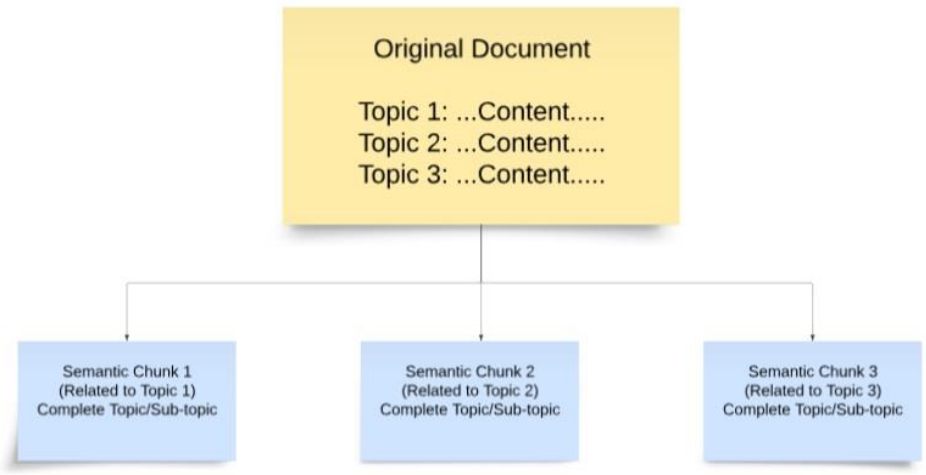

Fig 4: Semantic Chunking

Results and Discussion:

The chunking strategies discussed above fundamentally carry out the same process: splitting up text into multiple chunks for the LLM to use. However, the accuracy of the results can vary according to the use case.

Fixed-size chunking, when used, gives the best results when the data is well-structured. The uniform and specifically structured sequences of text in such data yield the best accuracy for forming relevant chunks. The LLM performs best when the chunks fed to it are connected and admissible, and fixed-size chunking is particularly effective for this kind of data. A few real-life applications where fixed-size chunking can be used include structured surveys which contain standardized data entries and the analysis of genetic sciences, where data is uniform and organized into specific lengths and sequences.

There can be situations where the text is generic and the initial level of chunking does not produce chunks of the desired size and structure; this is where recursive chunking comes into play. It ensures that the desired size and structure of the chunks are achieved, regardless of the number of times text splitting needs to be carried out. Although, it can be more time and resource consuming that fixed size chunking, the accuracy of output of the LLM fed with chunks using this method is better as it gives more contextually relevant information. A few real-life applications include summarizing long documents and customer support systems, where recursive chunking can be used to handle large volumes of support documents and FAQs.

Sentence-aware chunking is the most useful in cases where the understanding of each sentence present in the text is utterly important. The priority is that each chunk should contain a complete sentence even if the chunks are not of equal size. Analysis of articles and textbooks for education is where this method can be used to break down complex concepts into small manageable units. Another use case is in medical science and guidelines, where sentence-aware chunking can ensure that medical instructions do not get divided into multiple chunks which can confuse the users who might not have any prior knowledge of the field.

Semantic chunking is by far the best method to chunk the text based on its meaning and context rather than fixed size or structure. When the chunks are fed to the LLM using this method, it reduces ambiguity and misinterpretation of the data that can arise from splitting. It also helps the LLM to handle complex data easily and produce responses that are contextually coherent and relevant. The best use case where semantic chunking can really help is in legal document analysis. These documents are highly interconnected due to their complex structure and the need to reference various clauses and principles. Semantic chunking helps the model to retrieve all this relevant information and generate the most accurate results. It is also widely used in information

retrieval systems, such as search engines, where it enhances the process by grouping related pieces of information together and increases the accuracy of search results.

With the help of all these chunking strategies, the concept of RAG has brought significant advancements in the field of Information Retrieval. It combines the capabilities of retrieval systems with generative models, addressing the limitations of various traditional models. The future of RAG is promising with numerous opportunities for innovation and improvement. Advanced retrieval techniques are a major area of focus for the future of RAG. Dense retrieval using transformer-based models and advanced indexing mechanisms are areas which can increase the accuracy and efficiency of RAG applications. Another future scope is Multimodal RAG models, which will integrate multimodal data sources including images, audio, and video. This can significantly expand the capabilities of these RAG models. Future RAG models could also incorporate real-time adaptation. This could allow them to continuously update their knowledge base and improve their retrieval and generation capabilities based on new information and user feedback. These various NLP and GenAI innovations have given immense potential to the future of RAG applications.

Chunking Method	**Applied Use Cases**
Fixed Size Chunking.	Structured Documents and Surveys, Data analysis of genetic sciences
Recursive Chunking	Documents with large amount of data like logs of Customer Support Systems
Sentence-Aware Chunking	Analysis of Textbooks used for Education and study of Medical Guidelines
Semantic Chunking	Information Retrieval Systems and Legal Document Analysis.

Conclusion:

In conclusion, chunking your content and feeding it to an LLM is usually a straightforward task, but it can present challenges when generalized for all use cases and solutions. Selecting the right chunking technique is crucial for optimizing the performance of a RAG application. The chunking technique directly impacts the performance of the model, as it helps it to understand, retrieve and generate relevant information. The method of chunking which works for one use case and gives accurate results might not work for another, which highlights the importance of selecting chunking strategies to the specific use case of the application. By considering all factors such as context, structure of the

data and desired output, selecting the suitable chunking strategy is extremely essential for the model to deliver the most accurate and meaningful results.

Looking ahead, the future scope of RAG applications can also be enhanced by scalable architectures, multimodal integration, real-time adaptation, and ethical AI practices.

References:

1) [online] Available: https://python.langchain.com/v0.2/docs/introduction/

2) P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Kuttler, M. Lewis, W.-t. Yih, T. Rockt ¨ aschel ¨ et al., "Retrievalaugmented generation for knowledge-intensive nlp tasks," Advances in Neural Information Processing Systems, vol. 33, pp. 9459–9474, 2020

3) S. Borgeaud, A. Mensch, J. Hoffmann, T. Cai, E. Rutherford, K. Millican, G. B. Van Den Driessche, J.-B. Lespiau, B. Damoc, A. Clark et al., "Improving language models by retrieving from trillions of tokens," in International conference on machine learning. PMLR, 2022, pp. 2206–2240

4) Yunfan Gaoa , Yun Xiongb , Xinyu Gaob , Kangxiang Jiab , Jinliu Panb , Yuxi Bic , Yi Daia , Jiawei Suna , Meng Wangc , and Haofen Wang, "Retrieval-Augmented Generation for Large Language Models: A Survey"

5) I. ILIN, "Advanced rag techniques: an illustrated overview," https://pub.towardsai.net/advanced-rag-techniques-an-illustrated-overview-04d193d8fec6, 2023.

6) W. Yu, D. Iter, S. Wang, Y. Xu, M. Ju, S. Sanyal, C. Zhu, M. Zeng, and M. Jiang, "Generate rather than retrieve: Large language models are strong context generators," arXiv preprint arXiv:2209.10063, 2022.

7) S. Wang, Y. Xu, Y. Fang, Y. Liu, S. Sun, R. Xu, C. Zhu, and M. Zeng, "Training data is more valuable than you think: A simple and effective method by retrieving from training data," arXiv preprint arXiv:2203.08773, 2022.

8) Z. Jiang, F. F. Xu, L. Gao, Z. Sun, Q. Liu, J. Dwivedi-Yu, Y. Yang, J. Callan, and G. Neubig, "Active retrieval augmented generation," arXiv preprint arXiv:2305.06983, 2023.

9) L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray et al., "Training language models to follow instructions with human feedback," Advances in neural information processing systems, vol. 35, pp. 27 730–27 744, 2022

10) X. Wang, Q. Yang, Y. Qiu, J. Liang, Q. He, Z. Gu, Y. Xiao, and W. Wang, “Knowledgpt: Enhancing large language models with retrieval and storage access on knowledge bases,” arXiv preprint arXiv:2308.11761, 2023.

11) Barnett, S., Kurniawan, S., Thudumu, S., Brannelly, Z., Abdelrazek, M.: Seven Failure Points When Engineering a Retrieval Augmented Generation System , 2024

12) Chen, H., Jiao, F., Li, X., Qin, C., Ravaut, M., Zhao, R., Xiong, C., Joty, S.: ChatGPT’s One-year Anniversary: Are Open-Source Large Language Models Catching up? arXiv preprint arXiv:2311.16989 , 2023

13) El-Haj, M., Rayson, P., Young, S., Walker, M.: Detecting document structure in a very large corpus of UK financial reports. European Language Resources Association (ELRA) , 2014

14) Hada, R., Gumma, V., de Wynter, A., Diddee, H., Ahmed, M., Choudhury, M., Bali, K., Sitaram, S.: Are large language model-based evaluators the solution to scaling up multilingual evaluation? arXiv preprint arXiv:2309.07462 , 2023

15) Anantha, R., Bethi, T., Vodianik, D., Chappidi, S.: Context Tuning for Retrieval Augmented Generation , 2023

16) Kaddour, J., Harris, J., Mozes, M., Bradley, H., Raileanu, R., McHardy, R.: Challenges and applications of large language models. arXiv preprint arXiv:2307.10169 , 2023

17) Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., K¨uttler, H., Lewis, M., Yih, W.t., Rockt¨aschel, T., et al.: Retrieval-augmented generation for knowledge-intensive NLP tasks. Advances in Neural Information Processing Systems 33, 9459–9474 , 2020

18) Liu, N.F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., Liang, P.: Lost in the middle: How language models use long contexts. arXiv preprint arXiv:2307.03172 , 2023

19) Papineni, K., Roukos, S., Ward, T., Zhu, W.J.: Bleu: a method for automatic evaluation of machine translation. In: Proceedings of the 40th annual meeting of the Association for Computational Linguistics. pp. 311–318 , 2002

20) Wu, S., Irsoy, O., Lu, S., Dabravolski, V., Dredze, M., Gehrmann, S., Kambadur, P., Rosenberg, D., Mann, G.: BloombergGPT: A Large Language Model for Finance , 2023

21) Ye, H., Liu, T., Zhang, A., Hua, W., Jia, W.: Cognitive Mirage: A Review of Hallucinations in Large Language Models , 2023