The object of this study is Retrieval-Augmented Generation (RAG) systems used to improve the quality of responses by large language models (LLMs). The task addressed is to improve the efficiency of the semantic text segmentation stage in such systems, which directly affects the accuracy of extracting relevant fragments.

The work reports a method of semantic text segmentation for RAG systems, based on the sliding window technique with a dynamically changing size. The method devised uses embedding models and makes it possible to take into account the semantic context of the text. The adjustable value of the cosine similarity threshold used in semantic splitting makes it possible to additionally increase the relevance of query formation to LLM. The developed algorithm for setting this threshold value makes it possible to more fully take into account the specificity of the query subject. Compared to advanced methods of semantic text segmentation, the method devised provides the following gains depending on the maximum document size parameter: IoU from 0.2 % to 2.8 %, precision from 0.4 % to 3.1 %, omega precision from 1.4 % to 14.8 %. The gains are primarily associated with text processing at the level of semantically complete units in the form of sentences, rather than tokens. In addition, the dynamic sliding window technique allowed for better adaptation to the text structure. The results are valid within the framework of the used evaluation, which covers heterogeneous text datasets, and could be applied in practice when building RAG systems in industries with high requirements for preserving the semantic integrity of the text, for example, in law, science, or technology. The algorithms that implement the proposed method are posted on GitHub as Python libraries

***Keywords:** RAG, sliding-window, semantic, chunking, embeddings, binary search, evaluation, tuning, AI, LLM*

UDC 681.3.07: 004.8
DOI: 10.15587/1729-4061.2025.326177

SEMANTIC TEXT SPLITTING METHOD DEVELOPMENT FOR RAG SYSTEMS WITH CONTROLLED THRESHOLD AND SLIDING WINDOW SIZE

Oleg Galchonkov
Corresponding author
PhD, Associate Professor*
E-mail: o.n.galchenkov@gmail.com
Oleksii Horchynskyi*
Svetlana Antoshchuk
Doctor of Technical Sciences, Professor*
Volodymyr Nareznoy
Head of Department
Department of IT
LLC "Provectus IT"
Sutter St Ste str., 447, San Francisco, CA, USA, 94108
*Department of Information Systems
Institute of Computer Systems
Odesa Polytechnic National University
Shevchenka ave., 1, Odesa, Ukraine, 65044

Received 24.01.2025
Received in revised form 07.03.2025
Accepted date 31.03.2025
Published date 22.04.2025

***How to Cite:** Galchonkov, O., Horchynskyi, O., Antoshchuk, S., Nareznoy, V. (2025). Semantic text splitting method development for rag systems with controlled threshold and sliding window size. Eastern-European Journal of Enterprise Technologies, 2 (2 (134)), 47–57. https://doi.org/10.15587/1729-4061.2025.326177*

1. Introduction

Large language models (LLMs) are a powerful tool that makes it possible to solve a wide range of problems related to natural language processing. Their advantages include the ability to effectively process large volumes of text information, extract semantic relationships, and generate meaningful responses. Due to these characteristics, large language models find application in various fields.

Despite their effectiveness and rapid development, the following problems remain relevant for large language models:

– hallucinations and distortions of facts in responses [1];

– difficulties in perceiving large context windows by language models, especially information in the middle of the window [2].

These problems are directly generated by LLMs because they are limited exclusively by the information that was used during training. The volume of this information is very large and heterogeneous, and the algorithms for generating responses are probabilistic, which leads to diverse and not entirely accurate responses to the same queries. In addition, the "attention" mechanism is used when generating responses. However, "attention" is formed based on the initial huge amount of information used in training. It does not take into account the importance of this or that information for a specific application. Moreover, LLM training is carried out by large companies using supercomputers, and, naturally, they do not have time to train LLMs on the most recent data and do not have access to some volumes of commercial and technological information.

The problem of hallucinations can be solved by retraining the model on a data set specific to a particular application. However, retraining is an expensive and computationally intensive process.

The problem of perceiving large contextual windows can be solved by various methods of text summarization, or isolating part of sentences from paragraphs [3].

One of the most promising methods for solving the above problems is the use of RAG (Retrieval Augmented Generation – generating a request to LLM and a response to the user taking into account additionally found relevant information). RAG makes it possible to expand the response of a large language model based on the context of the top-k documents retrieved for the request [4]. Thus, RAG makes it possible to expand the model's knowledge without additional training and reduces hallucinations in responses, while maintaining a limited size of the context window.

Copyright © 2025 Authors. This is an open access article under the Creative Commons CC BY license

Given the importance for practical use of obtaining the most relevant responses from LLMs and expanding their use, improving the algorithms of RAG is extremely relevant.

2. Literature review and problem statement

Paper [5] reports an improvement to the RAG system by using a rewriting model. The architecture proposed in the paper first improves the search query using LLM and only then uses it to obtain an answer. This makes it possible to eliminate the gap between the initial query and the required information. However, this does not use additional capabilities for highlighting the most important accents in the primary query.

Study [6] proposes an improvement to RAG systems using hypothetical documents. Based on the user's query, a hypothetical document is generated using a language model, which is then encoded into a vector using a contrastive encoder. Based on this vector, relevant documents are extracted, and hallucinations are filtered out. Thus, the method requires generating hypothetical documents from a large language model for each query to the system.

Paper [7] suggests an improvement to RAG systems by using atomic facts as a search unit instead of traditional paragraphs or sentences. Facts are created using a model distilled from GPT-4, which decomposes text into minimal self-contained statements containing a single fact and the necessary context. This approach increases the density of relevant information in the retrieved documents, which improves search accuracy. Despite the improvements, this approach requires a large amount of computing resources to run the distilled model.

Paper [8] describes an improvement to RAG systems through the Advanced RAG paradigm. Advanced RAG implements pre-search and post-search strategies. Pre-search optimizes indexing by adding metadata and applying query optimization techniques such as query reformulation and expansion. Post-search uses techniques for re-ranking extracted text fragments and context compression to highlight the most relevant information. However, the proposed algorithm is based on predicting future content, which may lead LLM away from the most relevant answer.

Study [9] reports an improvement to RAG systems through the use of the GraphRAG framework, which uses graph data structures to improve search and answer generation. GraphRAG includes three key stages:

– graph indexing (creation and organization of graph databases);

– graph searching (extraction of relevant nodes, paths, or subgraphs);

– graph generation (synthesis of answers using the extracted data).

This approach makes it possible to take into account the relationships between documents, which increases the accuracy of the answers. However, it requires the organization of graph databases, which take up additional memory.

Increasing the efficiency of segmentation can be achieved by embedding text and structured knowledge into low-dimensional spaces with subsequent detection of similarities between them, similar to how this is done on a large scale in LLMs [10]. However, at present, this approach requires further research.

The technology of using both a direct query to LLM and a reverse one in meaning in a pair is proposed in [11]. This allows for further division of the response into subspaces, thus reducing the likelihood of hallucinations. Experimental studies confirm the effectiveness of this approach. However, methods for integrating this approach with others require further development.

One of the possible developments of the method for contrasting query pairs [11] is proposed in [12]. A special self-tuning algorithm has been developed for better division of subspaces. However, the approach shows high performance only for large LLMs, and for a smaller number of parameters, typical for RAG, the performance is significantly reduced.

In [13], an approach is proposed that uses two languaga es to represent LLM queries. Subsequent matching and alignment makes it possible to form clarifying queries that increase the relevance of subsequent responses.

Despite all possible options for improving RAG systems, the stage of text segmentation into documents is always an important and relevant part of the system. The quality of the text segmentation stage depends solely on the methods used to divide the text into documents. The following main methods of text division can be distinguished:

– division of text into documents of a fixed size of characters or tokens;

– recursive division of text based on separators in the form of punctuation marks;

– semantic division of text.

Paper [14] considers improving text segmentation in RAG systems using the above-described methods. It is noted that semantic division methods, in contrast to non-semantic ones, achieve greater efficiency due to the use of the meaning and context of the text. This approach makes it possible to obtain more complete documents as a result of division, not taken out of context. The efficiency of semantic methods in the field of law, where the text is often intertwined and has a complex structure, is also noted. Despite the achieved results, this area is still underdeveloped. This allows us to state that further research on improving semantic division methods is advisable.

Study [15] reports an improvement in the algorithm for recursive text division using a more extended list of separators. The list of separators is expanded and ordered in such a way that more semantically significant separators are at the beginning. For example, the first symbol is a period, and the last is a hyphen for a link of words. In this way, the algorithm divides the text into more semantically related chunks without using an embedding model (a technique for converting text data into numerical vectors) [16]. The disadvantages of this approach include a rigid architecture of semantic division, which does not adapt to the specificity of the content.

Paper [17] presents a percentile method of semantic division. The text is divided into sentences using regular expressions on punctuation marks ("?", "!", ".") Then, pieces of text are formed using a window of size n sentences. Between the formed pieces, vectors are calculated using the embedding model. Cosine distances between vectors that lie above the 95th percentile are recognized as boundaries for dividing the text into documents. However, this approach does not provide for satisfying the restrictions on the number of tokens used. Study [18] reports an improvement of the percentile method from [17] by finding, using binary search, such a percentile value that gives the size of the largest document no more than a given number of tokens.

Work [18] also proposes a cluster method of semantic division of text based on dynamic programming. The original text is divided into pieces no larger than 50 tokens using a recursive text segmentation algorithm. For each obtained chunk, a vector is calculated using the embedding model. Then, the dynamic programming method is applied to maximize the pairwise cosine similarity between all pairs of chunks within each document. However, the use of division into clusters does not imply the formation of a package of different queries to increase the relevance of the resulting LLM answer.

In [19], a method for semantic division of text based on finding a local minimum is reported. Initially, the method forms chunks of text by *n* sentences of dimension using a sliding window and vectorizes them. Then, the method again uses the sliding window technique between vectors and compares the similarity between the formed windows. As soon as the similarity begins to decrease, and thus a local minimum is found, this place is recognized as the boundary for dividing the text into a document.

Thus, the algorithm from [15] is not exactly an algorithm for semantic text division, but rather an improvement of the recursive one, with an extended list of separators.

The percentile algorithms from [17, 18], as well as the algorithm for finding a local minimum from [19], are processed by a static predefined window. The window size is an additional non-intuitive parameter that should be adjusted for text processing. However, the above methods do not provide ways to adjust this parameter. In addition, a static window adapts worse to the structure of the text: in some cases, it can capture too small a fragment, losing context, and in others, it can be too large, combining semantically unrelated parts.

The dynamic algorithm from [18] operates at the level of tokens, not sentences. This approach may prove ineffective since it can split initially semantically complete pieces of text into sentences.

Summarizing the above-described methods for dividing text, we can highlight the following shortcomings that are important to consider when devising a semantic method for dividing text:

– missing the meaning and context of the text without using an embedding model;

– using techniques that are poorly adapted to the structure of the text in the form of a sliding window of static size;

– using non-intuitive parameters without a setting technique;

– fragmentation of an initially semantically complete text because of operating at the token level.

Thus, existing methods of semantic text division do not allow intuitive adjustment of algorithm parameters, they use techniques that are poorly adapted to the text structure, and often operate at the token level, which leads to artificial fragmentation of semantically integral parts. This creates problems with preserving the logical and semantic coherence of the text, as well as with adjusting the algorithm. All this emphasizes the feasibility of devising a method that would make it possible to flexibly adjust parameters and work not at the token level but with larger semantic units.

3. The aim and objectives of the study

The aim of our work is to devise a method for semantic division of text based on the technique of a sliding window with a dynamic size, which makes it possible to adjust the parameter responsible for the size of the window formed during the operation of the algorithm. Such a method could make it possible to better divide text into documents for RAG systems and provide a way to adjust the parameters.

To achieve the goal, the following tasks were set:

– to develop an algorithm for semantic division of text based on the technique of a sliding window with a dynamic change in size;

– to develop an algorithm for adjusting the parameter responsible for the size of the window formed during the operation of the algorithm;

– to evaluate the proposed method in comparison with other methods of text division.

4. The study materials and methods

The object of research in our paper is RAG systems that improve the quality of LLM responses. The subject of research is the methods of semantic division of text used in RAG.

The basic hypothesis of the study assumes that the use of the dynamic size window technique in the methods of semantic division of text allows for better adaptation to the structure of the text, which in turn improves the quality of text segmentation into documents.

Several main configurations of RAG systems can be distinguished: naive, with a preliminary search stage, with a post-search stage, and combining post and pre-search stages in one system.

In the naive configuration, three components of the system can be distinguished: document indexing, document extraction, and generation of a language model response (Fig. 1). With this configuration, the quality of responses depends solely on the quality of document indexing and the embedding model for their retrieval (Retrieval Augmented – search and extraction of relevant information and supplementation of the user's request with this information).

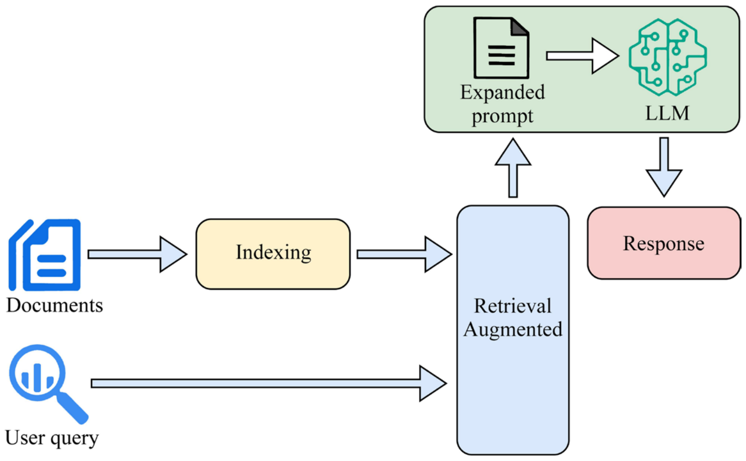

Fig. 1. Operational scheme of a naive RAG

In the configuration with the pre-search stage, a pre-search is added to the 3 components from the naive configuration. The pre-search component can perform various actions to improve the system's responses before the retrieval stage. For example: optimizing or expanding the query using the language model, adding metadata to indexed documents, etc. (Fig. 2). Thus, the quality of the system's responses depends not only on retrieval and indexing of documents but also on the pre-search stage.

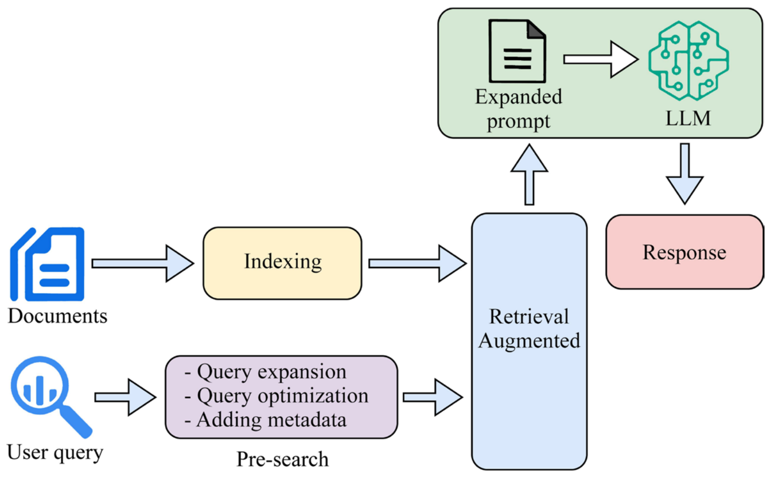

Fig. 2. RAG operation scheme with preliminary search

In the configuration with the post-search stage, the post-search is added to the 3 components from the naive configuration. The post-search component can perform various actions to improve the system's responses after the retrieval stage. For example: re-ranking documents using a language model or using keywords, summarizing or highlighting only the main information from the retrieved documents (Fig. 3). Thus, the quality of the system's responses depends not only on retrieval and indexing of documents but also on the post-search stage.

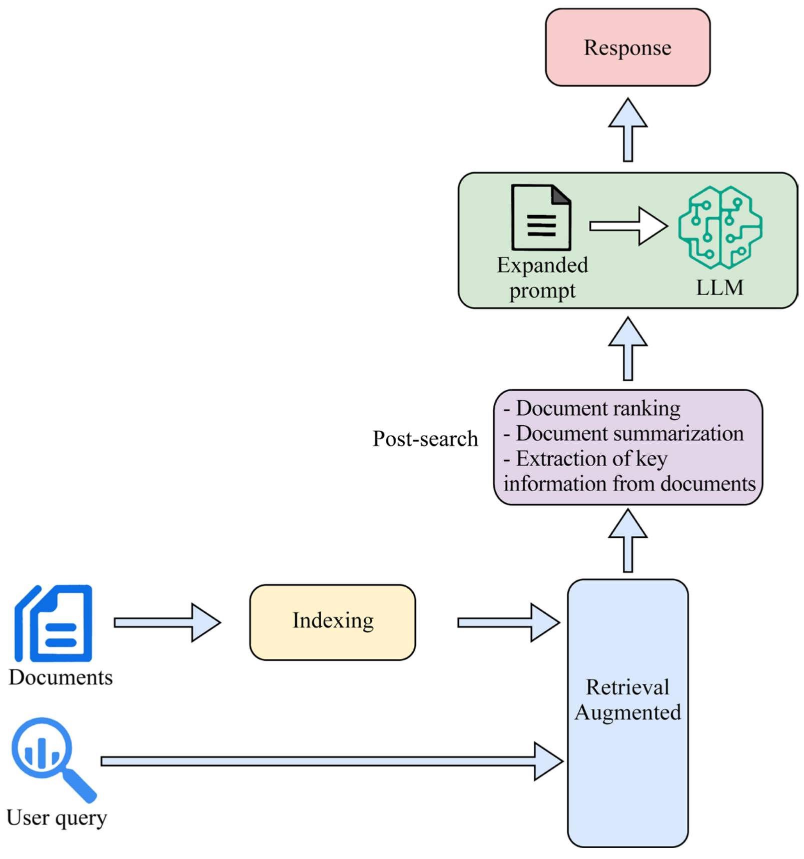

Fig. 3. Scheme of RAG operation with post search

In the combined configuration, the pre- and post-search stages are added to the 3 components from the naive configuration. Thus, the quality of the system's responses depends not only on retrieval and document indexing but also on the post- and pre-search stages together.

Since one of the tasks is to evaluate the devised method of semantic division of text, it is necessary to limit the influence of additional factors on the system's operation as much as possible. Therefore, the naive RAG configuration, limited to the retrieval stage, was used in the evaluation. Thus, the quality of the system is affected exclusively by document indexing and the retrieval model.

To evaluate the method devised, the framework for evaluating the division of text into documents for RAG systems reported in [18] was used. It implements the above-described naive configuration, limited to retrieval.

The framework employs a set of public data from various fields: medical, financial, Wikipedia articles, US President's speeches to Congress, user dialogs [20]. The total data set size is 328,208 tokens. Tokens were measured using the cl100k_base encoding used in gpt-4.

The framework uses 472 queries with relevant text fragments generated from the dataset using gpt-4 [21]. Based on the queries, top-k documents obtained using the text division method under evaluation are extracted. Afterwards, the tokens of the extracted documents are compared with the tokens of the relevant fragments.

To evaluate the extracted documents, various metrics are reported in [18]. Our work applies precision, which is calculated using the following formula:

$$\text{Precision}_q(C) = \frac{|t_e \cap t_r|}{|t_r|}, \tag{1}$$

where t_e are tokens of relevant documents,

t_r are tokens of extracted documents,

C is the divided dataset,

q is the query.

Our work also uses recall, which is calculated using the following formula:

$$\text{Recall}_q(C) = \frac{|t_e \cap t_r|}{|t_e|}, \tag{2}$$

where t_e are tokens of relevant documents,

t_r are tokens of extracted documents,

C is the divided dataset,

q is the query.

For the case when the recall is 100 %, the precision is denoted as "precision Ω".

In addition, token-wise Intersection over Union (IoU) is used, which is calculated using the following formula:

$$IoU_q(C) = \frac{|t_e \cap t_r|}{|t_e| + |t_r| - |t_e \cap t_r|}, \tag{3}$$

where t_e are tokens of relevant documents,

t_r are tokens of extracted documents,

C is the divided dataset,

q is the query.

Some applications for assessing precision and recall calculate the function:

$$F_1 = 2\times\frac{\text{Precision}\times\text{Recall}}{\text{Precision}+\text{Recall}} = 2\times\frac{|t_e \cap t_r|}{|t_e|+|t_r|}. \quad (4)$$

However, in practice, the IoU metric usually performs the function of F$_1$ evaluation and provides a more intuitive understanding [18].

During the evaluation, the devised method used the all-MiniLM-L6-v2 model [22] to obtain semantic information about the text. Thus, the evaluation process of the devised method was as follows (Fig. 4).

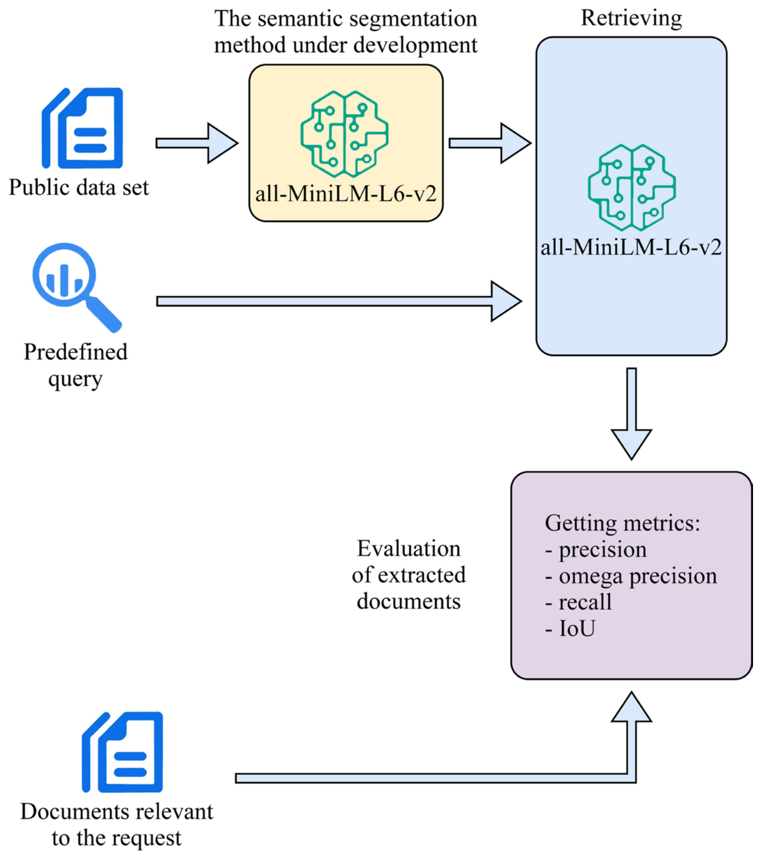

Fig. 4. Scheme of the process for evaluating the method devised

During the evaluation, the all-MiniLM-L6-v2 model [22] was used to obtain embeddings at the retrieval stage. This model is a retrained version of the embedding model based on MiniLM-L6-H384-uncased [23], which was trained on a dataset of 1 billion sentence pairs.

The all-MiniLM-L6-v2 model transforms sentences and paragraphs into a vector of dimensionality 384 and is used for clustering and semantic search tasks.

5. Devising and experimentally investigating the method of semantic division of text with the ability to adjust parameters

5. 1. Development of an algorithm for semantic division of text

The original text T at the input is divided into a set of paragraphs $P=\{p_1,p_2,\ldots,p_n\}$ using the following expression:

$$P = split(T, \backslash n+), \quad (5)$$

where \n+ is a regular expression for finding newline characters.

Then each paragraph p_i is divided into a set of sentences, also using regular expressions:

$$S_i = \{s_{i1}, s_{i2}, \ldots, s_{ij}\}, \quad (6)$$

where i is the paragraph number from set P,

j is the number of sentences in the i-th paragraph, using the following expression:

$$S_i = split(p_i, [?!.]), \quad (7)$$

where [?!.] is a regular expression for finding punctuation symbols.

A finite set of sentences S can be expressed as:

$$S = \bigcup_{i=1}^{n} S_i, \quad (8)$$

where S_i is the set of sentences for the i-th paragraph;

n is the number of paragraphs in set P.

The resulting finite set of sentences S is passed to the algorithm to start working.

The algorithm's logic is based on the sliding window technique with a dynamically changing size [24]. The algorithm sets the pointer l to the first sentence, the pointer r to the second, and then iteratively moves the pointer r, expanding the window $W[l, r]$. The sentence at the pointer l and the window $W[l, r]$ are transformed into vectors using the embedding model E:

$$\begin{cases} v_l = E(S_l), \\ v_w = E(W[l,r]), \end{cases} \quad (9)$$

where v_l is the vector of sentence at pointer l,

v_w is the vector of proposals of window $W[l, r]$.

Then distance d between vectors v_l and v_w is calculated using the cosine similarity formula [25]:

$$d = \frac{v_l \cdot v_w}{\|v_l\| \cdot \|v_w\|}, \quad (10)$$

where $v_l \bullet v_w$ is the scalar product of vectors $v_l \bullet v_w$,

$\|v_l\|$, $\|v_w\|$ are the norms of vectors v_l and v_w.

The result is the distance between sentence S_l and window $W[l, r]$ (Fig. 5).

If distance d is less than the specified threshold t: the document $W[l, r)$ is generated (Fig. 6).

Then the l and r pointers are updated as follows:

$$\begin{cases} l = r, \\ r = r+1. \end{cases} \quad (11)$$

The algorithm continues to work until:

$$r \le i, \quad (12)$$

where i is the number of sentences in set S.

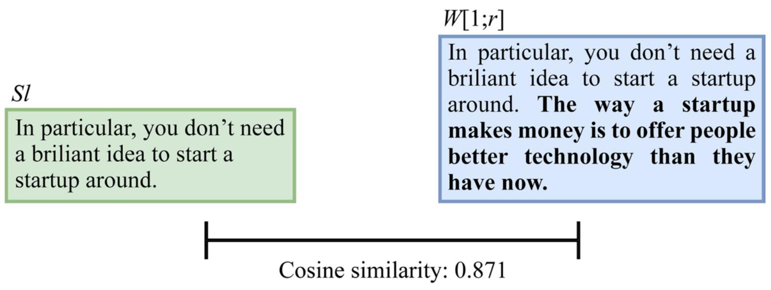
Sl
In particular, you don't need a briliant idea to start a startup around.
W[1;r]
In particular, you don't need a briliant idea to start a startup around. The way a startup makes money is to offer people better technology than they have now.
Cosine similarity: 0.871

Fig. 5. Finding the cosine similarity between S_l and the window $W[l,r]$

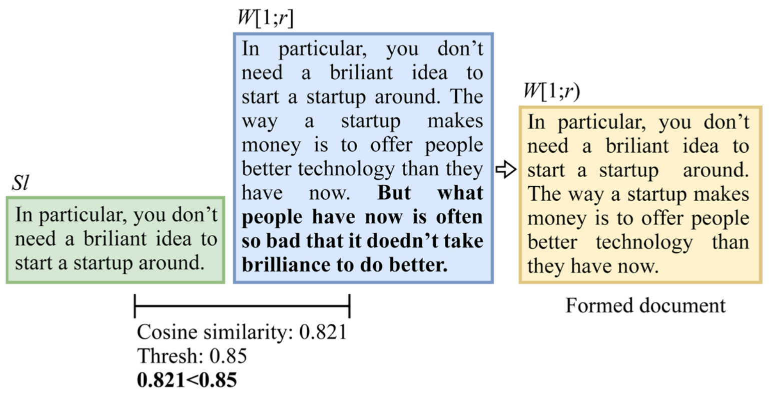
W[1;r]
In particular, you don't need a briliant idea to start a startup around. The way a startup makes money is to offer people better technology than they have now. But what people have now is often so bad that it doedn't take brilliance to do better.
W[1;r)
In particular, you don't need a briliant idea to start a startup around. The way a startup makes money is to offer people better technology than they have now.
Sl
In particular, you don't need a briliant idea to start a startup around.
Cosine similarity: 0.821
Thresh: 0.85
0.821<0.85
Formed document

Fig. 6. Formation of document $W[l, r]$ when passing the set threshold t=0.85 by distance d

When expanding window $W[l, r]$, it is clear how the semantic meaning in the form of vectors obtained using the embedding model gradually "drifts" from sentence S_l to the sequentially formed windows $W[l, r]$, $W[l, r+i]$ (Fig. 7).

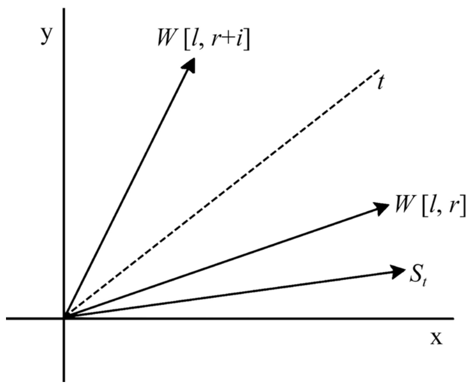

Fig. 7. Drift of semantic meaning when the window expands

Adding a new sentence moves the window $W[l, r]$ away from sentence S_l. If the new sentence S_r is very semantically distant from S_l, the cosine similarity will immediately move away. If it is close, the similarity will not change much.

Using a high threshold value of t=0.95, depending on the text, will lead to smaller documents with 1–2 sentences, while a low threshold of t=0.72 – to larger ones.

For additional control over the documents being created, the α parameter is used. It controls the maximum number of sentences in a document to prevent the creation of excessively large documents with semantically similar sentences. Such situations can occur when processing lists or tables with similar values.

Below is a diagram of the algorithm (Fig. 8).

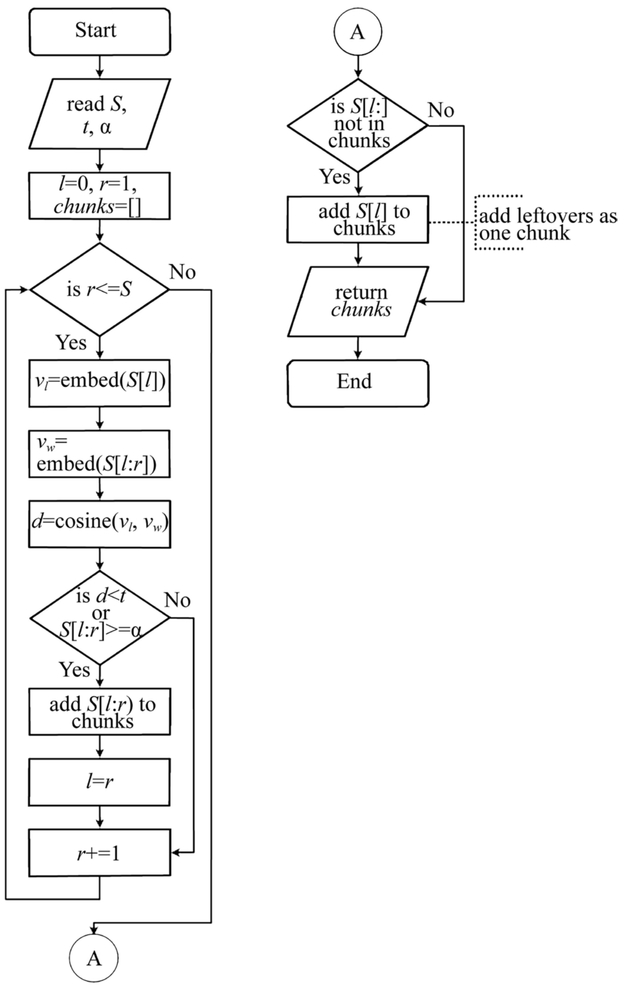

Fig. 8. Flowchart of the semantic division algorithm

The time complexity of the algorithm is of the following order:

$$O(n), \tag{13}$$

where n is the number of sentences.

The developed algorithm is available as a python library hosted on GitHub [26].

5. 2. Development of an algorithm for adjusting the cosine similarity threshold parameter

One of the main parameters of the algorithm is the cosine similarity threshold t.

In RAG systems, at the first stage there is always access to the data that needs to be divided. Thus, it is always possible to adjust the algorithm based on this data.

The developed adjustment algorithm is based on the binary search algorithm [27]. The source text is divided into paragraphs using operation (5) and then divided into sentences using operation (7).

Next, based on the specified size of the static window m, sentences $S=\{s_1, s_2, ..., s_j\}$ are sequentially processed and chunks of text are formed from – sentences:

$$C_i=\{s_i, s_{i+1}, ..., s_{i+m-1}\}, \tag{14}$$

where i is the number of a sentence from set S.

For each obtained chunk C_i, the first sentence C_{i1} and the entire chunk C_i are transformed into vectors using the embedding model E:

$$\begin{cases} v_1 = E(C_{i1}), \\ v_c = E(C_i). \end{cases} \tag{15}$$

Next, distance d_i between v_1 and v_c is found using cosine similarity (10).

A dictionary is formed from the obtained distances and chunks:

$$D=\{(d_1, C_1), (d_2, C_2), ..., (d_i, C_i)\}, \tag{16}$$

where d_i is the key;

C_i – value.

The created dictionary D is sorted in ascending order based on distances d_i (Fig. 9).

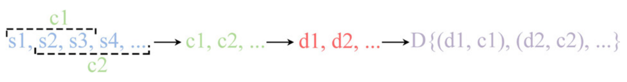

Fig. 9. The process of forming dictionary *D*

After receiving the sorted dictionary D, the binary search algorithm is launched with a human evaluation. The evaluation is performed by entering a command in the terminal to increase or decrease the threshold value. If the generated chunk with a given distance is semantically complete in the human opinion, the threshold is decreased. If the generated chunk is semantically different, it is increased. After the evaluation is complete, the found distance is returned, the number is limited to two digits after the decimal point. The found value is the threshold t (Fig. 10).

Using the binary search-based tuning, the threshold value t of the cosine similarity can be adjusted based on the data that will be split. In addition, since the static window size of m sentences is set in the process of forming dictionary D, the tuning finds the minimum threshold value for forming semantically complete documents of m sentences or more.

The algorithm for forming dictionary D is shown below (Fig. 11).

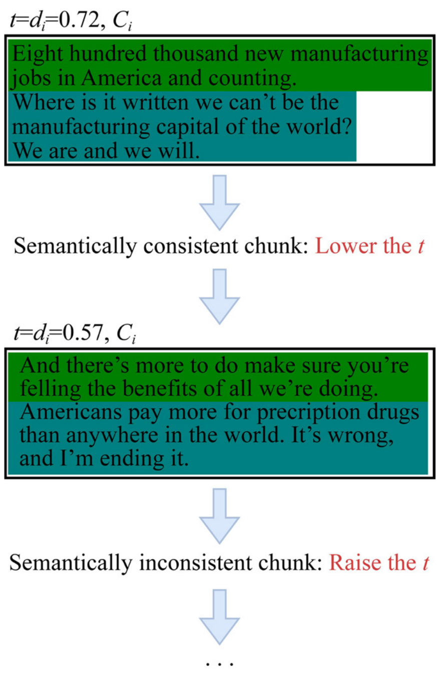
t=d_i=0.72, C_i
Eight hundred thousand new manufacturing jobs in America and counting.
Where is it written we can't be the manufacturing capital of the world? We are and we will.
Semantically consistent chunk: Lower the t
t=d_i=0.57, C_i
And there's more to do make sure you're felling the benefits of all we're doing.
Americans pay more for precription drugs than anywhere in the world. It's wrong, and I'm ending it.
Semantically inconsistent chunk: Raise the t
. . .

Fig. 10. The process of finding the threshold *t* using an assessment from a person

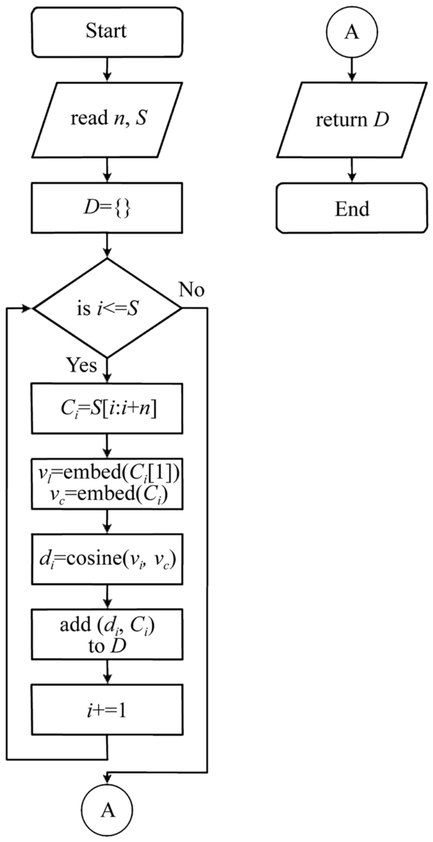

Fig. 11. Flowchart of the algorithm that forms a dictionary

Thus, the time complexity of creating a dictionary is of the following order:

$$O(n), \qquad (17)$$

where n is the number of sentences.

The space complexity of creating a dictionary, which characterizes the amount of memory used, is also of the following order:

$$O(n), \qquad (18)$$

where n is the number of sentences.

Below is a flowchart of the tuning algorithm based on binary search with human evaluation (Fig. 12).

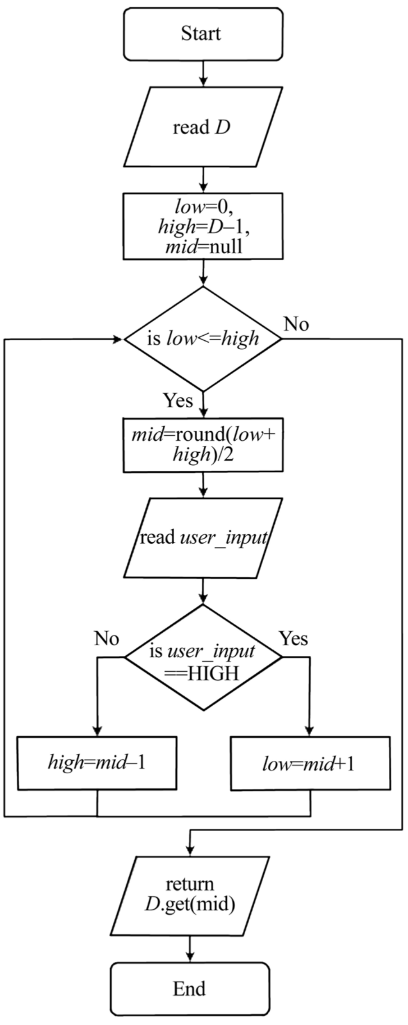

Fig. 12. Flowchart of the tuning algorithm based on binary search with human evaluation

The time complexity of tuning with human assessment is of the following order:

$$O(\log(k)), \qquad (19)$$

where k is the number of elements in the generated dictionary.

The developed algorithm for adjusting the cosine similarity threshold parameter is part of the python library posted on GitHub [26].

5. 3. Experimental comparison of the proposed method with other methods of semantic text division

The framework from [18] was used to evaluate the method devised. Some of the results for other methods were also taken from [18]. The number of documents that were selected for subsequent processing was the same for all methods and was 5. The results of applying the devised method are given with a threshold value of t=0.72 and with the values of the maximum size of the divided documents in the form of α=3 and α=6 sentences.

For other division methods, the size parameter of the divided documents in tokens is set. Also, for some of the methods, a non-zero overlap parameter is set, which is responsible for the size of the text in tokens that intersects between the divided documents.

To obtain the threshold value t, tuning was performed using the proposed algorithm based on binary search, with a human assessment. During tuning, the window size was equal to m=3 sentences and the datasets from [18] were used. The threshold values found for individual datasets were averaged.

All results are given in Table 1. The best indicators for each metric are highlighted in bold. The method devised is marked with an asterisk. The metric values are given as percentages, ± indicates the standard deviation value.

From Table 1 it is evident that in comparison with the best-proven cluster semantic method of dividing text [18], the use of the devised method gives an increase in metrics depending on the size of α: IoU from 0.2 % to 2.8 %, precision from 0.4 % to 3.1 %, precision Ω from 1.4 % to 14.8 %.

In comparison with the modified percentile semantic method [18], in addition to an increase in precision, precision Ω, IoU, there is also an increase in the recall metric: IoU from 3.9 % to 6.5 %, precision from 4.1 % to 6.8 %, precision Ω from 19 % to 32.4 %, recall from 0.8 % to 7.0 %.

There is also an increase in precision, precision Ω, IoU of the proposed method in comparison with non-semantic methods of text division [28, 29]:

– in comparison with recursive [28]: IoU from 0.9 % to 3.5 %, precision from 1.1 % to 3.8 %, precision Ω from 3.8 % to 17.2 %;

– in comparison with token [29]: IoU from 3.3 % to 5.9 %, precision from 3.5 % to 6.2 %, precision Ω from 15.9 % to 29.3 %.

Table 1

Results of the evaluation of text division methods

Division algorithm	Parameter	Recall, mean, (deviation) (%)	Precision, mean (deviation) (%)	Precision Ω, mean (deviation) (%)	IoU, mean (deviation) (%)
1	2	3	4	5	6
Recursive [28]	size: 250 tokens; overlap: 125 tokens	78.7 (+21.3…–39.2)	4.9 (±4.5)	21.9 (±14.9)	4.9 (±4.4)
TokenText [29]	size: 250 tokens; overlap: 125 tokens	82.4 (+17.6…–36.2)	3.6 (±3.1)	11.4 (±6.6)	3.5 (±3.1)
Reursive [28]	size: 250 tokens; overlap: 0 tokens	78.5 (+21.5…–39.5)	5.4 (±4.9)	26.7 (±18.3)	5.4 (±4.9)

Continuation of Table 1

1	2	3	4	5	6
TokenText [29]	size: 250 tokens; overlap: 0 tokens	77.1 (+22.9…–39.3)	3.3 (±3.0)	16.4 (±10.3)	3.3 (±3.0)
Recursive [28]	size: 200 tokens; overlap: 0 tokens	75.7 (+24.3…–40.7)	6.5 (±6.2)	31.2 (±18.4)	6.5 (±6.1)
TokenText [29]	size: 200 tokens; overlap: 0 tokens	76.6 (+23.4…–38.8)	4.1 (±3.7)	19.1 (±11.0)	4.1 (±3.6)
KamradtMod [18]	size: 250 tokens; overlap: 0 tokens	63.1 (+36.9…–46.9)	2.7 (+3.8…–2.7)	13.5 (±13.3)	2.7 (+3.8…–2.7)
KamradtMod [18]	size: 200 tokens; overlap: 0 tokens	67.9 (+32.1…–44.9)	3.5 (+4.1…–3.5)	16.0 (±14.9)	3.5 (+4.1…–3.5)
Cluster [18]	size: 250 tokens; overlap: 0 tokens	77.3 (+22.7…–38.6)	6.1 (±5.1)	28.6 (±16.7)	6.0 (±5.1)
Cluster [18]	size: 200 tokens; overlap: 0 tokens	75.2 (+24.8…–39.9)	7.2 ±6.1)	33.6 (±20.0)	7.2 (±6.0)
DynWindow*	: 0.5; α: 6	76.4 (+23.6...–39.7)	6.3 (±5.4)	29.3 (±15.5)	6.3 (±5.4)
DynWindow*	: 0.5; α: 3	66.7 (+33.3...–42.3)	9.5 (±8.0)	46.0 (±19.5)	9.2 (±7.7)
DynWindow*	: 0.72; α: 6	74.9 (+25.1…–40.0)	7.6 (±6.5)	35.0 (±18.3)	7.4 (±6.3)
DynWindow*	: 0.72; α: 3	68.7 (+31.3…–41.3)	10.3 (±8.5)	48.4 (±19.8)	10.0 (±8.3)

6. Discussion of results based on investigating the devised method of semantic division of text

A feature of the devised method of semantic division of text is the joint use of the algorithm of direct text division (Fig. 8) and the algorithm of adjustment of the parameter of the threshold value of cosine similarity (Fig. 11, 12). Adjustment of the threshold value by a person allows the most adequate consideration of the semantic completeness and semantic features of the processed documents. The division of text by the developed algorithm of semantic division uses the adjusted threshold value. This ensures the best size of the variable proce.

ssing window from the point of view of the semantic meaning of the document formed as a result of division. The use of the threshold in the formation of the document is illustrated in Fig. 6. And the effect of an excessive window size on the semantic meaning of the entire document when adding the next sentence, semantically distant from the previous one, is shown in Fig. 7.

The application of the devised method based on the dynamic size sliding window technique and adjustable cosine similarity threshold provides a significant increase in comparison with both semantic and non-semantic methods of text division by the precision, precision Ω, and IoU metrics (Table 1). In the case of non-semantic methods [28, 29], the improvement is primarily due to the fact that the devised method takes into account the semantic context of the text. In the case of semantic methods, the increase in metrics is primarily due to the use of the dynamic size window technique. However, it is also worth highlighting the tendency that semantic methods of text division, in particular the devised one, have a lower recall value in comparison with non-semantic ones: recursive 78.7 %, token 82.4 %. This behavior is explained by the use of a wider token window by non-semantic methods (200–250 tokens) and the overlap strategy, while the devised method operates at the sentence level, not tokens. Table 1 also shows that the recall values of the devised method vary from 68.7 % at α=3 to 74.9 % at α=6, which is due to the fact that α=6 expands the maximum document size to 6 sentences, thus increasing the number of tokens in it. However, with a larger value of α, the precision, precision Ω, and IoU metrics naturally decrease. The value α=3 gives better indicators of the above metrics.

The use of the tuning algorithm for the threshold value t based on binary search directly affected the indicators of the precision and recall metrics since it determines the key parameter of the algorithm. With the obtained value t=0.72, the method gave a noticeable increase in the metrics of precision, precision Ω, IoU (Table 1).

The method should be primarily used in cases where it is necessary to segment a large text corpus for subsequent indexing and searching for relevant fragments when generating responses based on the information found.

The application of the proposed method will improve the quality of segmentation by forming more semantically complete fragments. Such an improvement, in turn, will have a positive effect on the quality and relevance of the generated responses.

Of the limitations, it is important to note that the method was evaluated using one framework with a specific and limited data set. However, the versatility of the framework used, and the heterogeneity of the data set allow us to hope that similar results will be observed for other data sets. It is also important to note that the value of the metrics is affected by the embedding model used at the retrieval stage during the evaluation.

Among the disadvantages of the devised method, one can highlight a decrease in the recall metric compared to the best-proven cluster semantic method of dividing the text [18]: from 0.3 % to 6.5 % depending on the size of α. However, the devised method demonstrates an increase in the IoU metric from 0.2 % to 2.8 %, which indicates a greater correspondence between the divided documents and those relevant to the query.

The devised method is especially effective in spheres where texts have a complex structure and preserving the semantic integrity of the obtained fragments is of particular importance, specifically in the legal, scientific, or technical fields.

A possible improvement of the method would be the use of more complex regular expressions with an expanded number of separators to divide the text not only into sentences but into smaller, semantically complete units of text.

7. Conclusions

1. An algorithm for semantic text division based on the sliding window technique with a dynamically changing size has been developed. Text division, unlike analogs, is performed at the sentence level, not at the token level.

2. An algorithm for adjusting the threshold value parameter for use in semantic text division based on binary search with human evaluation has been developed. Using the developed algorithm, the best threshold value of 0.72 was obtained for the used data set.

3. The devised method using the proposed algorithms for semantic text division and threshold value parameter

adjustment was evaluated in comparison with other text division methods. In this case, a typical framework for evaluating text division in RAG systems and a data set were used. The evaluation was based on 472 queries related to a heterogeneous data set of 328,208 tokens. In comparison with advanced methods of semantic division of text, the devised method provided an increase in metrics depending on the maximum document size parameter: IoU from 0.2 % to 2.8 %, precision from 0.4 % to 3.1 %, precision Ω from 1.4 % to 14.8 %.

Conflicts of interest

The authors declare that they have no conflicts of interest in relation to the current study, including financial, personal, authorship, or any other, that could affect the study, as well as the results reported in this paper.

Funding

The research was carried out with financial support from the National Research Foundation of Ukraine, grant No. 131/0161.

Data availability

The manuscript has related data in the data repository. References are provided in the text of the paper.

Use of artificial intelligence

The authors used artificial intelligence technologies within acceptable limits to provide their own verified data, which is described in the research methodology section.

References

1. Huang, L., Yu, W., Ma, W., Zhong, W., Feng, Z., Wang, H. et al. (2025). A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions. ACM Transactions on Information Systems, 43 (2), 1–55. https://doi.org/10.1145/3703155
2. Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., Liang, P. (2024). Lost in the Middle: How Language Models Use Long Contexts. Transactions of the Association for Computational Linguistics, 12, 157–173. https://doi.org/10.1162/tacl_a_00638
3. Hosseini, P., Castro, I., Ghinassi, I., Purver, M. (2025). Efficient Solutions For An Intriguing Failure of LLMs: Long Context Window Does Not Mean LLMs Can Analyze Long Sequences Flawlessly. arXiv. https://doi.org/10.48550/arXiv.2408.01866
4. Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N. et al. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. arXiv. https://doi.org/10.48550/arXiv.2005.11401
5. Ma, X., Gong, Y., He, P., Zhao, H., Duan, N. (2023). Query Rewriting in Retrieval-Augmented Large Language Models. Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing. https://doi.org/10.18653/v1/2023.emnlp-main.322
6. Gao, L., Ma, X., Lin, J., Callan, J. (2023). Precise Zero-Shot Dense Retrieval without Relevance Labels. Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). https://doi.org/10.18653/v1/2023.acl-long.99
7. Chen, T., Wang, H., Chen, S., Yu, W., Ma, K., Zhao, X. et al. (2024). Dense X Retrieval: What Retrieval Granularity Should We Use? Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, 15159–15177. https://doi.org/10.18653/v1/2024.emnlp-main.845
8. Jiang, Z., Xu, F., Gao, L., Sun, Z., Liu, Q., Dwivedi-Yu, J. et al. (2023). Active Retrieval Augmented Generation. Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing. https://doi.org/10.18653/v1/2023.emnlp-main.495
9. Procko, T. T., Ochoa, O. (2024). Graph Retrieval-Augmented Generation for Large Language Models: A Survey. 2024 Conference on AI, Science, Engineering, and Technology (AIxSET), 166–169. https://doi.org/10.1109/aixset62544.2024.00030
10. Kalinowski, A., An, Y. (2021). Exploring Sentence Embedding Structures for Semantic Relation Extraction. 2021 International Joint Conference on Neural Networks (IJCNN), 1–7. https://doi.org/10.1109/ijcnn52387.2021.9534215
11. Gao, T., Yao, X., Chen, D. (2021). SimCSE: Simple Contrastive Learning of Sentence Embeddings. Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing. https://doi.org/10.18653/v1/2021.emnlp-main.552
12. Limkonchotiwat, P., Ponwitayarat, W., Lowphansirikul, L., Udomcharoenchaikit, C., Chuangsuwanich, E., Nutanong, S. (2023). An Efficient Self-Supervised Cross-View Training For Sentence Embedding. Transactions of the Association for Computational Linguistics, 11, 1572–1587. https://doi.org/10.1162/tacl_a_00620
13. Miao, Z., Wu, Q., Zhao, K., Wu, Z., Tsuruoka, Y. (2024). Enhancing Cross-lingual Sentence Embedding for Low-resource Languages with Word Alignment. Findings of the Association for Computational Linguistics: NAACL 2024, 3225–3236. https://doi.org/10.18653/v1/2024.findings-naacl.204
14. Kshirsagar, A. (2024). Enhancing RAG Performance Through Chunking and Text Splitting Techniques. International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 10 (5), 151–158. https://doi.org/10.32628/cseit2410593
15. Semchunk. Available at: https://github.com/isaacus-dev/semchunk
16. Vector embeddings. Available at: https://platform.openai.com/docs/guides/embeddings
17. Kamradt, G. 5 Levels of text splitting. Available at: https://github.com/FullStackRetrieval-com/RetrievalTutorials/blob/main/tutorials/LevelsOfTextSplitting/5_Levels_Of_Text_Splitting.ipynb
18. Smith, B., Troynikov, A. (2024). Evaluating Chunking Strategies for Retrieval. Chroma Technical Report. Available at: https://research.trychroma.com/evaluating-chunking

19. Research Chunking Strategies. Available at: https://github.com/nesbyte/ResearchChunkingStrategies/blob/main/main.ipynb
20. Corpora. Available at: https://github.com/brandonstarxel/chunking_evaluation/tree/main/chunking_evaluation/evaluation_framework/general_evaluation_data/corpora
21. Questions. Available at: https://github.com/brandonstarxel/chunking_evaluation/blob/main/chunking_evaluation/evaluation_framework/general_evaluation_data/questions_df.csv
22. all-MiniLM-L6-v2. Available at: https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2
23. MiniLM-L6-H384-uncased. Available at: https://huggingface.co/nreimers/MiniLM-L6-H384-uncased
24. Berrier, J. (2021). Move along. Dynamic-Size Sliding Window Pattern/Technique. Available at: https://jamie-berrier.medium.com/move-along-c09d59bea473
25. Cosine similarity. Available at: https://en.wikipedia.org/wiki/Cosine_similarity
26. Horchunk. Available at: https://github.com/panalexeu/horchunk.git
27. Binary search. Available at: https://en.wikipedia.org/wiki/Binary_search
28. Recursive character text splitter. Langchain documentation. Available at: https://api.python.langchain.com/en/latest/character/langchain_text_splitters.character.RecursiveCharacterTextSplitter.html
29. Token text splitter. Langchain documentation. Available at: https://python.langchain.com/api_reference/text_splitters/base/langchain_text_splitters.base.TokenTextSplitter.html