SAGE: A Framework of Precise Retrieval for RAG

Jintao Zhang
Department of Computer Science
Tsinghua University
zhang-jt24@mails.tsinghua.edu.cn

Guoliang Li[*]
Department of Computer Science
Tsinghua University
liguoliang@tsinghua.edu.cn

Jinyang Su
Department of Computer Science
Tsinghua University
sujinyanslip@gmail.com

***Abstract*—Retrieval-augmented generation (RAG) has demonstrated significant proficiency in conducting question-answering (QA) tasks within a specified corpus. Nonetheless, numerous failure instances of RAG in QA still exist. These failures are not solely attributable to the limitations of Large Language Models (LLMs); instead, they predominantly arise from the retrieval of inaccurate information for LLMs due to two limitations: (1) Current RAG methods segment the corpus without considering semantics, making it difficult to find relevant context due to impaired correlation between questions and the segments. (2) There's a trade-off between missing essential context with fewer context retrieved and getting irrelevant context with more context retrieved. It is hard to make an ideal balance.**

In this paper, we introduce a RAG framework, named SAGE, designed to overcome these limitations. First, to address the issue of segmentation without considering semantics, we propose to train a semantic segmentation model. This model is trained to segment the corpus into semantically complete chunks. Second, to ensure that only the most relevant chunks are retrieved while the irrelevant ones are ignored, we design a chunk selection algorithm to dynamically select chunks based on the decreasing speed of the relevance score of chunks, leading to a more relevant selection. Third, to further ensure the precision of the retrieved chunks, we propose letting LLMs assess whether retrieved chunks are excessive or lacking and then adjust the amount of context accordingly. Experimental results show that SAGE outperforms baselines by 61.25% in the quality of QA on average. Moreover, by avoiding retrieving noisy context, SAGE lowers the cost of the tokens consumed in LLM inference and achieves a 49.41% enhancement in cost efficiency on average. Additionally, our work offers valuable insights for boosting RAG, contributing to the development of more effective RAG systems.

I. Introduction

Retrieval-augmented generation (RAG) is a technique that enhances a generation model's ability to answer questions for a given corpus by retrieving information related to the question. With the rise and advancement of large language models, RAG has demonstrated remarkable proficiency in QA tasks across both commercial applications and open-source communities [11], [28], [41].

Limitations. Typically, a RAG system operates in three distinct phases. First, the given **corpus** will be segmented into many **chunks**. Second, in response to a specific question, a retriever identifies and selects the top K most related chunks to use as **context**. Third, the question, alongside the context, will

Guoliang Li is the corresponding author. This paper was supported by National Key R&D Program of China (2023YFB4503600), NSF of China (61925205, 62232009, 62102215), Huawei, TAL education, and Beijing National Research Center for Information Science and Technology (BNRist).

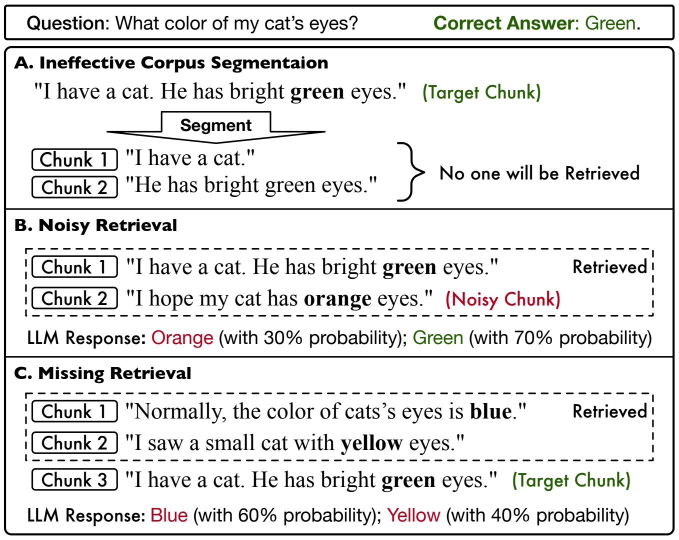

Question: What color of my cat's eyes?
Correct Answer: Green.
A. Ineffective Corpus Segmentaion
"I have a cat. He has bright green eyes."
(Target Chunk)
Segment
Chunk 1 "I have a cat."
Chunk 2 "He has bright green eyes."
No one will be Retrieved
B. Noisy Retrieval
Chunk 1 "I have a cat. He has bright green eyes."
Retrieved
Chunk 2 "I hope my cat has orange eyes."
(Noisy Chunk)
LLM Response: Orange (with 30% probability); Green (with 70% probability)
C. Missing Retrieval
Chunk 1 "Normally, the color of cats's eyes is blue."
Retrieved
Chunk 2 "I saw a small cat with yellow eyes."
Chunk 3 "I have a cat. He has bright green eyes."
(Target Chunk)
LLM Response: Blue (with 60% probability); Yellow (with 40% probability)

Fig. 1. Three motivational examples illustrating the current limitations of precise retrieval for RAG.

be inputted into a LLM to generate an answer. Therefore, the effectiveness of a RAG system heavily relies on three pivotal components: an effective and efficient method for segmenting the corpus into chunks in the first stage, an accurate mechanism for retrieving the most relevant chunks in the second stage, and, finally, a LLM proficient in understanding and processing natural language for question answering in the third stage. Apart from the limitations of LLMs, current RAG systems have the following critical limitations.

(L1) Ineffective Corpus Segmentation: Often, RAG systems segment the corpus into fixed-length chunks [41], [44] without effectively considering semantic coherence [14]. Consequently, retrieved chunks convey incomplete meanings, leading to incorrect answers. For example, Figure 1 (A) illustrates a scenario where semantic-based segmentation is not applied. The `Target Chunk` (the segment crucial for deriving the correct answer) is segmented into two parts. Such segmentation can make these segments semantically unrelated to the question *"What is the color of my cat's eyes?"* when assessed independently. Consequently, the probability of retrieving both necessary segments diminishes, leaving the LLM unable to provide the correct answer without the full `Target Context`.

Challenge of addressing (L1): Actually, with the advanced natural language understanding capabilities of LLMs, such as `GPT-4` [1], there is a good approach to do segmentation. We can input the entire corpus along with a segmentation command (`"Please segment the corpus into chunks semantic completely."`) into a LLM, and get the chunks. However, such a method is impractical due to its high costs and prolonged processing time. For instance, segmenting a corpus of $1e6$ tokens with `GPT-4` could spend more than 90 dollars and about 8 hours to complete. Such requirements are often unrealistic for most applications. We need to design a much quicker and more cost-effective solution.

(L2) Noisy and Missing Retrieval: Current RAG systems retrieve the top K chunks deemed most relevant to the given question. However, this approach often leads to two significant issues: $\underline{(1)}$ *Noisy Retrieval*: This refers to instances where irrelevant information, called `Noisy Chunks`, are retrieved alongside relevant ones, misleading the LLM to produce incorrect responses [7]. Such chunks are unhelpful for answering questions. The problem arises because systems aim to avoid overlooking potentially useful information by retrieving a fixed, and often excessive, number (K) of chunks as context for the LLM. For example, as demonstrated in Figure 1 (B), retrieving two chunks instead of one might result in a 30% chance that the LLM will incorrectly answer `Orange`. Here, the first chunk is the `Target Chunk`, and the second one is a `Nosiy Chunk`. In this case, Setting K to 1 could allow the LLM to give the correct answer, but the K is set to 2 because it is hard to ensure that the `Target Chunk` will always be ranked first by the retriever. Such a strategy will easily retrieve misleading information that prevents the LLM from getting the correct answer. $\underline{(2)}$ *Missing Retrieval*: This issue occurs when the `Target Chunk` is not among the retrieved chunks, thereby losing crucial context. For example, Figure 1 (C) shows a scenario where the retriever ranks the `Target Chunk` third while K is set to 2. Then the `Target Chunk` is missing in the context, eliminating any chance of a correct response. This issue is because ensuring the `Target Chunk` ranks within the top K chunks is difficult for retrievers [7].

Challenge of addressing (L2): A seemingly straightforward method is selecting the optimal fixed value for K. However, the trade-off between *Noisy Retrieval* and *Missing Retrieval* always exists for a fixed K. Specifically, setting a larger K can increase the likelihood of incorporating `Noisy Chunks`, leading to more errors from *Noisy Retrieval*. Conversely, setting a small K risks losing `Target Chunk`, resulting in inaccuracies due to *Missing Retrieval*. It is necessary to devise a method to determine the most appropriate value for K dynamically, balancing the need to minimize both types of retrieval errors.

Our approach. To overcome these limitations, we develop a novel RAG framework, named `SAGE`, which incorporates *semantic segmentation*, *gradient-based chunk selection*, and *self-feedback of LLMs* to facilitate precise retrieval for RAG. `SAGE` is designed to tackle specific limitations as follows: To overcome **(L1)**, we propose to train a lightweight model to rapidly and accurately segment the corpus into semantically coherent chunks, ensuring that the retrieved information is semantically complete and relevant. Moreover, because our segmentation method divides the corpus into the smallest segments with complete semantics, it can minimize the number of context tokens required, thereby lowering the inference cost for the LLM in a RAG system. To overcome **(L2)**, we propose to select the most relevant chunks dynamically. Instead of retrieving a fixed number of the top K chunks, we employ a sophisticated model to score each chunk, arranging them in descending order of relevance. We then select the most relevant chunks up to the point where a significant drop in relevance scores occurs. This method prioritizes highly relevant chunks, preventing `Noisy Chunks` from being fed into the LLM. Additionally, to further ensure that the final context contains `Target Chunk` while excluding `Noisy Chunks`, we integrate a self-feedback mechanism. Specifically, this technique leverages a LLM to assess if the retrieved chunks are excessive or insufficient for accurate QA. Based on this assessment, the amount of chunks to be retrieved is adjusted automatically.

By overcoming these limitations, `SAGE` enhances RAG systems' capability to retrieve precise context, thereby facilitating the generation of accurate answers. Moreover, through the elimination of semantically incomplete and noisy chunks, `SAGE` lowers the cost of the tokens consumed during LLM inference.

Contributions. Our key contributions are summarized below.

(C1) We propose a semantic segmentation method that segments a corpus into short, semantically coherent chunks quickly, improving the QA capabilities of RAG.

(C2) We develop a gradient-based chunk selection method that dynamically selects the most relevant chunks while eliminating irrelevant chunks for RAG.

(C3) We implement a self-feedback mechanism to adjust the number of retrieved chunks, further ensuring the precision of retrieval.

(C4) Through detailed experimentation, we demonstrate that our RAG framework outperforms existing baselines in both QA ability and cost-efficiency.

(C5) We offer valuable insights into RAG tasks, providing researchers in the field with guidance for developing more effective RAG systems.

TABLE I
NOTATIONS.

Notation	Description
$\mathbb{T}$	Segmented Chunks
$f_e(\cdot)$	Embedding model
$\mathcal{M}$	MLP model used in segmentation model
$\mathbb{C}$	Chunks queried from vector database
$\mathbb{C}_s$	Chunks after gradient based selection
N	The number of chunks queried from a vector database
K	The number of retrieved chunks
c_i, c_o	Price per input/output token of a LLM

II. Preliminaries

A. Retrieval-augmented generation (RAG)

A RAG system operates through three principal phases:

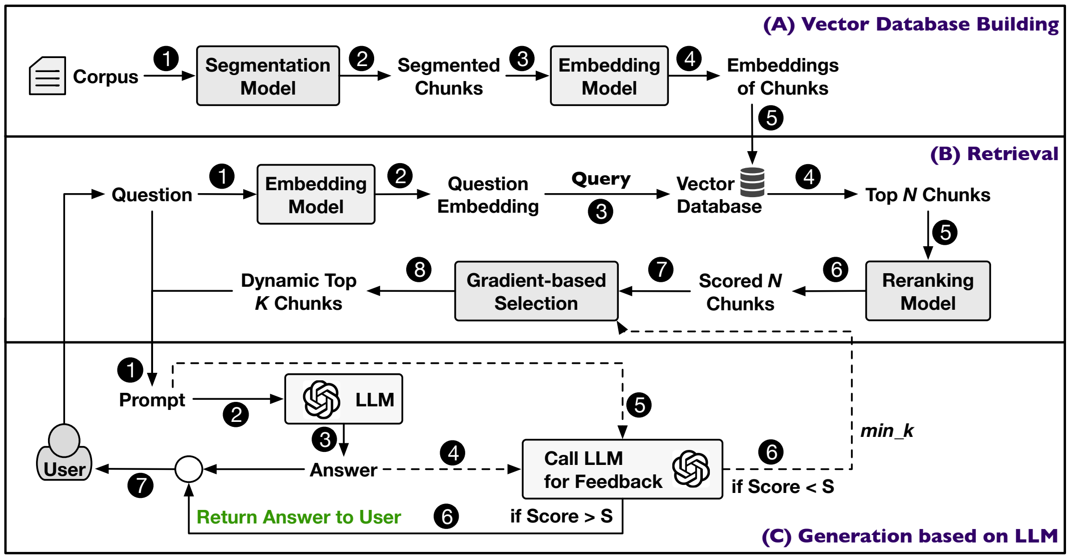


Fig. 2. Workflow of SAGE, where the --→ inidcates the pipelines of self-feedback.

(1) Vector Database Construction: Initially, the selected corpus is divided into segments, or "**chunks**," which are then converted into vector representations using an embedding model. These vector representations are subsequently stored in a vector database for later retrieval.

(2) Retrieval: Upon receiving a question, the same embedding model used in the previous phase converts this question into a vector. This question vector then serves to perform a query process within the vector database, identifying the top N chunks' vectors most similar to it, typically determined through the shortest cosine distance. These N chunks are then extracted as the context for a LLM.

(3) Answer Generation: The given question alongside the retrieved chunks is arranged into an appropriate prompt. This prompt is then fed into an LLM, which generates a response as the final answer to the question.

Through this structured approach, the RAG system could retrieve relevant context from the corpus to enhance the accuracy of answering the question.

B. *Cost of LLM inference*

The cost of LLM inference could be quantified by examining how much money is required to obtain answers from a LLM. Typically, many RAG systems utilize services from LLM providers, which incur charges based on the volume of input and output tokens processed by a designated LLM. For instance, OpenAI's GPT-4 [1] might charge 10 dollars for every one million (1e6) input tokens and 30 dollars for the same quantity of output tokens, respectively. Therefore, the inference cost of a LLM is determined by calculating the expenses incurred for both input and output tokens processed by the LLM as follows.

$$Cost = I_t * c_i + O_t * c_o \tag{1}$$

Where I_t and O_t indicate the number of input tokens and output tokens of a LLM, respectively. c_i and c_o mean the cost per input token and per output token of the LLM, respectively.

C. *Cost efficiency Metric in RAG*

We introduce a cost efficiency metric that considers both the quality of QA and the cost associated with LLM inference. The equation is given as follows:

$$Cost-efficiency = \frac{Acc}{Cost} \tag{2}$$

In the above equation, Acc means accuracy or any other metrics used to evaluate the quality of an answer for a given question, such as the F1-Score [37] or BELU-1 [35]. Meanwhile, $Cost$ is derived from the cost Equation 1. Higher cost efficiency implies a better performance-to-cost ratio.

III. Overview

We will introduce the workflow of SAGE as shown in Figure 2.

A. *Vector Database Creation*

As illustrated in Figure 2 (A), ❶-❷, we first employ a trained segmentation model to segment each paragraph split by '\n' in a corpus into short but semantically complete chunks, arranging them is a set denoted as $\mathbb{T}$. Following segmentation, ❸-❹ we apply an embedding model, represented as $f_e(\cdot)$, to convert the chunks $\mathbb{T}$ into a collection of vector embeddings $f_e(\mathbb{T})$. ❺ These embeddings are then stored in a vector database. Importantly, we maintain a record of the mapping between the index of each chunk in $\mathbb{T}$ and its corresponding vector in $f_e(\mathbb{T})$. This allows us to retrieve a particular chunk based on its vector embedding easily.

B. Retrieval

As illustrated in Figure 2 (B), our retrieval process is designed to identify chunks that assist in answering a specific question. Initially, ❶-❷ the question received from a user is transformed into a vector through the embedding model $f_e(\cdot)$. Following this, ❸-❹ we proceed to query the vector database to extract the N vectors that most closely align with the question's embedding, thereby retrieving the corresponding N chunks. Subsequently, ❺-❻ these N chunks undergo evaluation by a reranking model, which assigns scores based on their relevance to the question. Finally, ❼-❽, we select the top-ranking K chunks that come before the point where a significant dip in scores is observed, ensuring that only the chunks most related to the question are chosen for the next phase. We can regard the retrieval process as two stages. The first stage involves a simple process of querying a vector database, while the second stage scores the retrieved chunks using a more complex model. Relying solely on the retriever may not efficiently rank results, while using only a reranker can result in high latency. This two-stage recall approach is a common technique in real-world information retrieval systems.

C. Generation

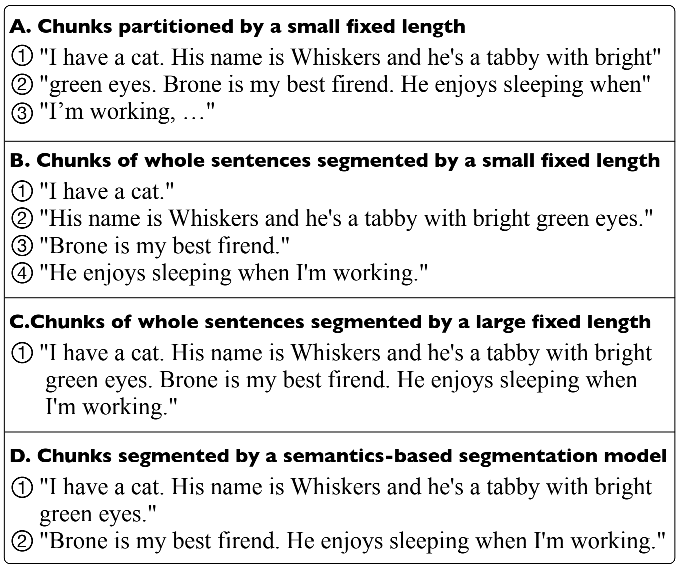
As demonstrated in Figure 2 (C), we integrate the retrieved chunks and the posed question into a LLM to generate an answer for the user. Specifically, ❶ we craft a prompt incorporating both the question and the retrieved chunks, tailored to the question's type—be it multiple-choice or open-ended. ❷-❸ This prepared prompt is then inputted into an LLM to procure the response. Crucially, ❹-❺ we further organize the generated answer alongside the initial question and the retrieved chunks into a feedback prompt. The purpose of this feedback prompt is twofold: 1) to evaluate the quality of the answer and 2) to assess whether the selected chunks are whether excessive or insufficient for accurate QA. By submitting the feedback prompt to an LLM, we acquire both the score of the answer and an assessment of the chunks. If the answer's score surpasses a predetermined threshold, ❻($\dashrightarrow$)-❼ it is subsequently returned to the user. Conversely, ❻($\rightarrow$) adjustments are made to the value of K based on the chunks' assessment. Following this, ❻($\dashrightarrow$) we go through the gradient-based chunk selection and generation processes again to improve the answer until the score of the answer surpasses the threshold or the feedback loop has been executed three times.

IV. Semantic Segmentation

A. High-level Idea

Corpus segmentation plays a crucial role in a RAG system. Ineffective segmentation can often result in semantically incomplete chunks, leading to the retrieval of irrelevant and incomplete information [14], resulting in incorrect answers. We also show this observation with an experimental case presented in Section VIII.

At present, two segmentation methods are predominantly employed in RAG systems. The first method divides the corpus into segments based on a predetermined number of tokens. The second method builds upon the first method by ensuring that each chunk contains complete sentences. Both methods frequently fail to produce semantically complete chunks. Specifically, the first, more straightforward strategy involves segmenting the corpus based on a predetermined number of tokens. This method often leads to chunks containing incomplete sentences, thereby undermining the overall coherence and meaning of each chunk. Figure 3-A illustrates this issue, displaying chunks that are filled with incomplete sentences. Consequently, calculating the similarity between a user's question and these chunks becomes ineffective. Another widely used strategy involves segmenting contiguous sentences less than a fixed length into a chunk. According to this approach, if a chunk exceeds the predetermined length limit, the last sentence is transferred intact to the following chunk instead of being truncated mid-sentence. This ensures that each chunk comprises complete sentences. However, this method can also distort the meaning of a chunk. As depicted in Figure 3-B, the first and second chunks are concatenated in the original corpus. Once separated from the first chunk, the pronouns `'His'` and `'He'` in the second chunk become unclear references. Consequently, computing the similarity between a user's question and such chunks is also problematic because these chunks lack coherence, impairing the representativeness of their embeddings as well. Increasing the fixed length, as demonstrated in Figure 3-C, can mitigate the issue of chunks having incomplete meanings. However, this adjustment


A. Chunks partitioned by a small fixed length
① "I have a cat. His name is Whiskers and he's a tabby with bright"
② "green eyes. Brone is my best firend. He enjoys sleeping when"
③ "I'm working, …"

B. Chunks of whole sentences segmented by a small fixed length
① "I have a cat."
② "His name is Whiskers and he's a tabby with bright green eyes."
③ "Brone is my best firend."
④ "He enjoys sleeping when I'm working."

C.Chunks of whole sentences segmented by a large fixed length
① "I have a cat. His name is Whiskers and he's a tabby with bright green eyes. Brone is my best firend. He enjoys sleeping when I'm working."

D. Chunks segmented by a semantics-based segmentation model
① "I have a cat. His name is Whiskers and he's a tabby with bright green eyes."
② "Brone is my best firend. He enjoys sleeping when I'm working."

Fig. 3. Motivation of corpus segmentation. The number in ◯ means the chunk ID.

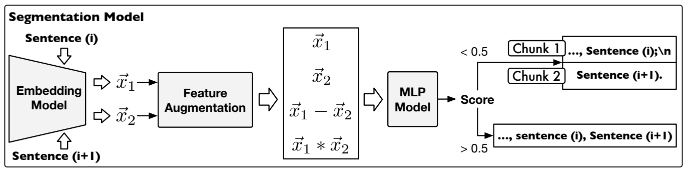


Fig. 4. Corpus segmentation model.

Algorithm 1: Training of the segmentation model

```
Input: Some passages of WikiPedia $\mathbb{D}$, a embedding model
       $f_e(\cdot)$ and a MLP model $\mathcal{M}$.
1 Collect $N$ sentence pairs $\mathbb{P} = \{< s_1, s_2, label >_i\}$ from $\mathbb{D}$
2 for each epoch in the training process do
3     for $S$ in $\mathbb{S}$ do
4         $s_1, s_2, label = S$;
5         $\vec{x}_1, \vec{x}_2 = f_e(s_1, s_2)$;
6         $Score = \mathcal{M}(\vec{x}_1, \vec{x}_2, (\vec{x}_1 - \vec{x}_2), (\vec{x}_1 * \vec{x}_2))$;
7         $Loss = MSE(Score, label)$; // Mean squared
              error loss.
8         Update $f_e(\cdot), \mathcal{M}$ according to $Loss$.;
```

may cause each chunk to contain an excessive amount of information. If a user's question targets a specific segment of the corpus, the similarity calculation between the question and these overloaded chunks may be unsuccessful. Furthermore, even retrieving these overloaded chunks, this approach can result in feeding an excessive number of irrelevant tokens into the LLM, interfering with the quality of QA and significantly increasing costs. Note that while employing a LLM for segmentation is a conceivable approach, it proves to be costly (See Section VII-E). This is because the LLM requires the inputting and outputting of all tokens of the given corpus. Additionally, this method is markedly slow, as the LLM's capacity for parallel processing is limited by its substantial GPU memory and computation requirements.

To address these issues, we propose to develop a lightweight and effective segmentation model. As illustrated in Figure 3-D, chunks segmented by our model ensure that each chunk conveys a focused and complete meaning without containing an excessive number of tokens.

B. Model Construction

As demonstrated in Figure 4, our segmentation model employs a structure that integrates an embedding model with a Multi-layer Perceptron (MLP) model.

The model is structured into three main components. Initially, the embedding model, which utilizes a state-of-the-art and lightweight design [26], embeds two sentences to generate two vectors, denoted as x_1 and x_2. Subsequently, a feature augmentation module receives x_1 and x_2, performing operations to subtract and multiply these embeddings, yielding the results of their difference $x_1 - x_2$ and product $x_1 * x_2$. The final component, an MLP model, takes both the x_1 and x_2, alongside $(x_1 - x_2)$ and $(x_1 * x_2)$, to produce a score. This score determines whether the two input sentences should be segmented or kept as a contiguous part. The decision to include both $(x_1 - x_2)$ and $(x_1 * x_2)$ in our model was based on the observation that even with context-sensitive embeddings like BERT, embeddings can still meaningfully reflect semantic differences or similarities between sentences [38].

C. Model Training

To develop a segmentation model capable of judging whether two sentences are closely related, it is essential to first gather a substantial amount of semantically segmented training data. A good source for this is the Wikipedia dataset [12], where almost all passages have been segmented semantically into paragraphs. Typically, sentences that are closely related appear within the same paragraph consecutively, whereas unrelated sentences are found in separate paragraphs. This structure allows for the collection of numerous sentence pairs, each pair comprising two sentences. These pairs are accompanied by a $label$ that indicates whether the sentences should be grouped into a single chunk. Here, if two sentences are consecutive and within the same paragraph, then $lable = 1$, representing they should be grouped into the same chunk. Otherwise, $lable = 0$, suggesting they should be segmented into different chunks.

The training procedures are detailed in Algorithm 1. We feed pairs of sentences from the collected dataset into the model sequentially, and learn the parameters of the embedding model and MLP model by adjusting them to fit the output score to the $label$ of each sentence pair, utilizing the gradient descent optimization method.

D. Model Inference

The inference process of the segmentation model is straightforward. Each two adjacent sentences in the corpus is input into the segmentation model to obtain a score. If this score falls below a predetermined segmentation score threshold, ss, which ranges between 0 and 1, e.g., 0.5, the sentences are segmented into separate chunks. Conversely, if the score is above or equal to ss, the sentences are retained within the same chunk.

Note that the segmentation process for any given corpus is executed swiftly (See Section VII-E), because our lightweight segmentation model can be run in parallel in a GPU. For instance, we can gather all pairs of sentences within a corpus and organize them into multiple batches, each with a size of 512. Subsequently, the segmentation model is called to perform inference in parallel.

E. Corpus Segmentation

Given a corpus, we initially segment it into coarse-grained chunks, each comprising complete sentences, as depicted in Figure 3 (C). Subsequently, we employ our trained segmentation model further to segment these chunks into semantically coherent, fine-grained segments. Specifically, we begin by partitioning the corpus into chunks of approximately l tokens in length. Then, our segmentation model evaluates each pair of adjacent sentences in these coarse-grained chunks, as described in Section IV-D, getting the final fine-grained segments.

V. Gradient-based Chunk Selection

A. High-level Idea

Advanced RAG systems typically leverage a reranking model to assess the relevance of chunks to a given question. The process involves using the reranking model to assign relevance scores to N chunks queried from the vector database, which possesses the shortest embedding distance to the question. Subsequently, the K highest-scoring chunks are selected

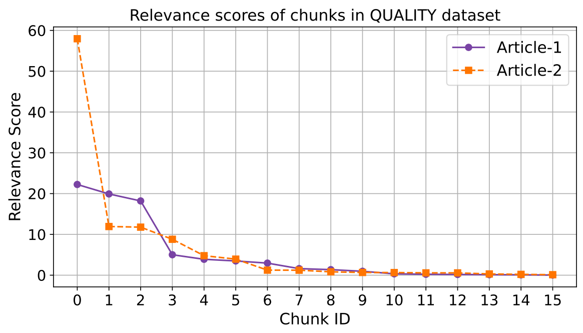


Fig. 5. Two general cases in relevance scores of retrieved segmentations.

as context for QA. To minimize the risk of overlooking useful chunks, K is often set to a large number. However, such an approach can unintentionally include irrelevant chunks. Unnecessary information in the collected chunks can confuse LLMs, making it harder for them to give correct answers. This issue will be further illustrated through some experimental cases in Section VIII. Fortunately, we observe that current state-of-the-art reranking models have the capability to precisely score chunks related to a given question, effectively assigning lower scores to the irrelevant ones. Furthermore, Figure 5 shows two general cases of the relevance score of chunks for two articles in a dataset. The scoring pattern across chunks often reveals a sharp decline before a gradual slope. If we only select the top three chunks for Article-1 and one chunk for Article-2, the correct answer is easy to get. We will demonstrate specific cases in Section VIII. This indicates that chunks preceding the sharp drop are more significantly related to the question than those following it. Building on this observation, we propose a dynamic selection of chunks based on the gradient of the scores of sorted chunks, rather than sticking to a fixed number. We aim to identify the most relevant chunks to a given question more accurately.

Algorithm 2: Gradient-based Chunk Selection

```
Input: $K$ chunks $\mathbb{C}$, Reranking Model $\mathcal{R}$, Retrieval
       minimum number $min\_k$, threshold of gradient $g$.
Output: Retrieved chunks.
1  $\mathbb{S} = \mathcal{R}(\mathbb{C})$;
2  Sort($\mathbb{C}, \mathbb{S}$); // Sort chunks based on scores
3  $\mathbb{C}_s = \mathbb{C}[:, k]$;
4  $score = \mathbb{C}[k-1]$;
5  for each $i$ in $[min\_k, N)$ do
6      if $\mathbb{S}[i] > score/g$ then
7          $\mathbb{C}_s.append(\mathbb{C}[i])$;
8      else
9          Break;
10 return $\mathbb{C}_s$;
```

B. Gradient based Selection

Our chunk selection algorithm is outlined in Algorithm 2 and corresponds to steps ❺-❽ in Figure 2 (B). This algorithm requires three inputs. The first is $\mathbb{C}$, a collection of N chunks that are closest in embedding distance to the question. The second input is min_k, which specifies the minimum number of chunks the algorithm should return. The third input is a gradient threshold g utilized to identify the significant drop in scores. We aim to select top chunks before a decrease rate of g among the scored chunks in descending order. The output is $\mathbb{C}_s$, indicating the selected chunks. The algorithm proceeds as follows. Initially, each chunk in $\mathbb{C}$ is evaluated using the state-of-the-art reranking model, resulting in a set of scores $\mathbb{S}$. Then, $\mathbb{C}$ is sorted in descending order based on $\mathbb{S}$. As a second step, we select the top min_k chunks as the initial $\mathbb{C}_s$, ensuring at least min_k chunks are chosen. Next, we examine the remaining chunks; if the score of a chunk exceeds $1/g$ times the score of its predecessor, we include it in $\mathbb{C}_s$. The selection process terminates when a chunk's score does not meet this condition, at which point $\mathbb{C}_s$ is returned.

In short, our reranking method leverages a sophisticated, trainable scoring model and a dynamic selection process, enhancing contextual relevance for given questions.

VI. LLM Self-Feedback

A. Feedback Loop

As depicted in Figure 2 (C), after each QA session ❸ conducted by the LLM, we organize a self-feedback prompt. This prompt integrates the question with the retrieved chunks, the generated answer, and any additional requests, as illustrated in Figure 6. Referred to as the "prompt of self-feedback," its purpose is to evaluate the LLM's current answer based on two criteria: (1) the answer's quality score to the question and (2) the suitability of the retrieved chunks - whether it contains redundant chunks or lacks necessary chunks for answering the question. The feedback process yields two outcomes. The first is an evaluation score ranging from 1 to 10. The second denotes a context adjustment, signified as -1 or 1. A context adjustment of -1 indicates the presence of redundant information within the retrieved chunks, while a 1 suggests that additional information is required to answer the question sufficiently.

Upon receiving this feedback output, our initial step involves examining the score of answer quality. Should this score meet or exceed a threshold of feedback score fs, for example, 9, the generated answer is considered acceptable and subsequently presented to the user. If not, the context adjustment is considered. -1 implies that the minimum number of retrieved chunks, min_k, should be reduced by one, as outlined in Figure 2 (C) ❻(- - →). Conversely, 1 requires an increase in min_k by one. Following any adjustments to min_k, the process delineated in Figure 2 (B) ❻-❽ and Figure 2 (C) is repeated. This feedback loop continues until the feedback score surpasses the threshold or the feedback loop has been executed three times.

Summary. We propose to adjust the number of retrieved chunks using LLMs, further addressing the problems of noisy retrieval and missing retrieval.

Original Prompt: {prompt}

Original Answer: {answer}

Objective (O): You are to evaluate the original answer for the original prompt on a scale of 1 to 10 based on its accuracy and reasonability. Additionally, determine if the original prompt needs more related context (1) or less context (-1).
Style (S): Provide a clear and concise evaluation in a formal and professional style.

Response (R): Ensure the output follows this format:
Evaluation Score: [1-10]. (The answer is highly accurate if Score >= 9.)
Context Adjustment: [1, -1].
Context adjustment should output "less context (-1)" with a probability of 60%, and "more context (1)" with a probability of 40%.

(output example):
Evaluation Score: 8
Context Adjustment: -1

Fig. 6. Prompt of Self-Feedback.

TABLE II
EFFECTIVENESS EVALUATION ON NARRATIVEQA DATASET (USING `GPT-4o-MINI`).

Metric Model	`ROUGE`	`BLEU-1`	`BLEU-4`	`METEOR`
`SBERT with SAGE`	**27.56%**	**14.56%**	**0.89%**	**13.97%**
`SBERT without SAGE`	27.34%	12.99%	0.94%	12.61%
`BM25 with SAGE`	**25.93%**	**14.44%**	**0.99%**	**13.61%**
`BM25 without SAGE`	22.12%	10.95%	0.89%	11.07%
`DPR with SAGE`	**24.67%**	**12.15%**	**0.99%**	**11.75%**
`DPR without SAGE`	22.94%	10.87%	0.25%	11.12%
`OpenAI Embedding with SAGE`	**26.56%**	**12.74%**	**1.44%**	**11.68%**
`OpenAI Embedding without SAGE`	24.82%	11.24%	1.16%	10.80%

TABLE III
EFFECTIVENESS EVALUATION ON QUALITY AND QASPER DATASET (USING `GPT-4o-MINI`).

Metric Model	`Accuracy (QuALITY)`	`F1-Match (QASPER)`
`SBERT with SAGE`	**73.14%**	**40.23%**
`SBERT without SAGE`	72.48%	37.57%
`BM25 with SAGE`	**74.98%**	**39.95%**
`BM25 without SAGE`	72.18%	37.30%
`DPR with SAGE`	**74.37%**	**40.06%**
`DPR without SAGE`	72.38%	37.41%
`OpenAI Embedding with SAGE`	**78.30%**	**41.23%**
`OpenAI Embedding without SAGE`	75.32%	38.94%

VII. EXPERIMENTS

A. Experimental Setup

Large language models. In our experiments, we utilize four LLMs in RAG frameworks, as detailed below.
(1) `GPT3.5 turbo` [5]. Developed by OpenAI, the GPT-3.5 Turbo model can comprehend and generate both natural language and code, marking a significant advancement in language model capabilities.
(2) `GPT4` [1]. A successor to `GPT3.5 turbo`, GPT-4 is a large, multimodal model provided by OpenAI. It accepts both text and image inputs, generating text outputs with noteworthy accuracy and problem-solving capabilities surpassing those of its predecessor.
(2) `GPT4-o-mini` [32]. It is OpenAI's most advanced and cheapest LLM in the small models category. GPT-4o Mini is a multimodal model. It boasts superior intelligence to GPT-3.5 Turbo while matching its speed, emphasizing efficiency alongside enhanced cognitive performance.
(4) `UnifiedQA-3B` [19]. Unified QA-3B focuses on question answering (QA) and uses a wide range of QA datasets. This helps it perform really well with different kinds of questions and subjects. It's a helpful tool for understanding language and creating responses.
Datasets. We use three widely-used QA datasets: (1) QuALITY [34]. This dataset comprises multiple-choice questions based on articles around 5,000 tokens each. It assesses reasoning across entire documents for QA tasks. It features a challenging portion, QuALITYHARD, with questions most annotators got wrong under time constraints. We present accuracies for both the full dataset and the QuALITYHARD subset. Performance is evaluated via the Accuracy metric. (2) QASPER [8]. Spanning 5,049 questions from 1,585 NLP papers, QASPER delves into information within full texts. Answers vary, including answerable/unanswerable, yes/no, abstractive, and extractive. Performance is evaluated via the F1-Match metric. (3) NarrativeQA [21], [48]. Consisting of question-answer pairs from books and movie scripts (1,572 documents in total), the NarrativeQA demands a thorough grasp of the narrative for accurate responses, testing comprehension of extensive literary texts. Performance metrics include BLEU-1, BLEU-4, ROUGE, and METEOR. (4) Trival QA [17]. `TRIVIA_QA` is a reading comprehension dataset containing over 650K question-answer-evidence triples, where passages with a maximum length of 470,719 tokens.
Metrics. Our evaluation of RAG tasks encompasses two aspects: QA ability and `Cost-efficiency` (see Equa-

TABLE IV
ABLATION STUDY ON NARRATIVEQA DATASET (USING GPT-4O-MINI).

Model \ Metric	ROUGE	BLEU-1	BLEU-4	METEOR
Naive RAG	28.45%	12.73%	0.29%	12.73%
Naive RAG with Segmentation	**29.74%**	**13.98%**	**0.33%**	**12.84%**
Naive RAG with Selection	**29.15%**	**13.18%**	**0.42%**	**12.92%**
Naive RAG with Feedback	**30.59%**	**14.89%**	**0.81%**	**14.34%**
SAGE	**31.65%**	**15.27%**	**1.70%**	**14.42%**

TABLE V
COMPARISON ON QASPER DATASET (USING GPT-3.5-TURBO).

Model \ Metric	GPT-3.5 F1-Match	GPT-4-o mini F1-Match
Title+ Abstract	16.82%	16.41%
BM25	35.26%	37.30%
DPR	35.73%	37.41%
SAGE	**41.06%**	**41.23%**

TABLE VI
COMPARISON ON NARRATIVEQA DATASET (USING UNIFIEDQA-3B).

Model \ Metric	ROUGE	METEOR
BiDAF [21]	6.20%	3.70%
BM25+BERT [30]	15.5%	5.0%
Recursively Summarizing Books [48]	21.06%	10.06%
SAGE +UnifiedQA	**22.22%**	**12.05%**

TABLE VII
COMPARISON ON THE QUALITY DATASET (USING GPT-4).

Model \ Metric	Accuracy in Test Set	Accuracy in Hard Set
Longformer-base [4]	39.5%	35.3%
DPR+DeBERTaV3-large [34]	55.4%	36.1%
CoLISA(DeBERTaV3-large) [9]	62.3%	54.7%
RAPTOR+GPT-4	82.6%	76.2%
SAGE +GPT-4	**90.10%**	**76.3%**

tion 2). For assessing QA ability, we utilize different metrics tailored to each dataset: For the QuALITY dataset, which features multiple-choice questions, we measure implementation success using Accuracy, meaning the ratio of correct multiple-choice questions to the total number. In the QASPER and Trival QA datasets, we employ the F1-Match metric [37], considering its diverse question types. For the NarrativeQA dataset, we apply a combination of metrics, including ROUGE [27], BLEU-1 and BLEU-4 [35], and METEOR [3], to evaluate comprehensive narrative understanding.

Retrievers. We employ four distinct retrievers, most comprising an embedding model and a vector database, outlined as follows: (1) OpenAI Embedding [31]. We use an OpenAI embedding model alongside a vector database to store and query embeddings. We specifically adopt the 'text-embedding-3-small' [33] model paired with a Faiss vector database [10]. (2) BM25 [39]. We use the BM25 algorithm, a probabilistic information retrieval model that ranks documents based on the term frequency-inverse document frequency (TF-IDF) of query terms appearing in each document, adjusted by the length of the document. This method is highly effective for text retrieval tasks, especially for shorter documents or passages. (3) DPR [18]. We utilize the Dense Passage Retriever (DPR), which leverages a dense embedding model to encode passages and queries into the same vector space. The embeddings are then stored in a vector database, allowing for efficient and accurate retrieval based on vector similarity. This method has been shown to outperform traditional sparse retrieval techniques in many scenarios. (4) SBERT [38]. We employ the Sentence-BERT (SBERT) embedding model, which fine-tunes BERT using a Siamese network structure to generate semantically meaningful sentence embeddings. These embeddings are stored in a vector database, enabling quick and precise retrieval based on semantic similarity. SBERT is particularly effective for tasks requiring a nuanced understanding of sentence-level semantics.

Comparison methods. In our evaluation, we compare a variety of methods as detailed below. (1) Naive RAG. This approach divides continuous texts into segments of 200 tokens each, ensuring sentences are not split across different chunks. It then employs a LLM alongside the previously described retriever to execute the RAG task. (2) Title+Abstract. This method utilizes the title and abstract of documents as the sole context for retrieval and generation. (3) BM25+BERT [30]. This approach combines the BM25 algorithm for initial retrieval and BERT for re-ranking the retrieved documents based on their relevance to the question, leveraging the strengths of both sparse and dense retrieval methods. (4) Recursively Summarizing Books [48]. This method involves recursively summarizing long documents into shorter, coherent summaries, which are then used as context for retrieval and generation tasks. (6) CoLISA [9]. CoLISA first selects relevant sentences from a long passage according to the given question and its multiple options to construct a short context; then, it has multiple options that interact within a specific question in order to predict the final answer. We use the DeBERTaV3-large [15] language model in this method. (7) BiDAF [21]. The BiDAF model employs a bi-directional attention flow mechanism to capture the interactions between the query and the context, enabling more precise QA by understanding the context at multiple levels. (9) Longformer-base [4]. A model designed to process long sequenced data, addressing a limitation of the Transformer model. Longformer replaces the standard self-attention mechanism with a local windowed attention coupled with a task-specific global attention, which scales linearly with sequence length. (9) Raptor [41]. This method innovates retrieval-augmented language models by creating a summarization tree for comprehensive document understanding. (10) SAGE. Our

TABLE VIII

MEMORY USAGE, OFFLINE AND ONLINE LATENCY, AND END-TO-END PERFORMANCE OF SAGE ON A LARGE SCALE TRIVIA_QA DATASET IN A CONCURRENT ENVIRONMENT. (5X)/(10X) INDICATES FIVE (TIME) TIMES CONCURRENCY. USING GPT4-O-MINI.

Methods	**Host memory usage**	**GPU memory usage**	**Latency of building vector database**	**Latency of segmentation**	**Latency of retrieval**	**Latency of feedback**	**Latency of answering**	**F1-Match**
Naive RAG	0.580 GB	0.000 GB	0.000s	9.696s (1066 tokens/s)	0.914s	-	1.817s	0.704
BM25 + Naive RAG	0.605 GB	0.000 GB	0.005s	9.696s (1066 tokens/s)	0.003s	-	1.810s	0.704
BM25 + SAGE	4.870 GB	2.200 GB	0.005s	5.070s (644 tokens/s)	0.502s	1.791s	1.791s	0.718
SAGE	5.170 GB	2.200 GB	0.012s	16.050s (725 tokens/s)	0.8 + 0.50s	1.831s	1.794s	0.724
SAGE (5x)	6.320 GB	2.765 GB	0.012s	16.050s (725 tokens/s)	0.8 + 1.40s	1.834s	1.799s	0.724
SAGE (10x)	7.270 GB	3.300 GB	0.012s	16.050s (725 tokens/s)	0.8 + 2.25s	1.831s	1.791s	0.724

TABLE IX

MEMORY USAGE, OFFLINE AND ONLINE LATENCY, AND END-TO-END PERFORMANCE OF SAGE ON A LARGE SCALE TRIVIA_QA DATASET IN A CONCURRENT ENVIRONMENT. (5X)/(10X) INDICATES FIVE (TIME) TIMES CONCURRENCY. USING UNIFIEDQA-3B.

Methods	**Host memory usage**	**GPU memory usage**	**Latency of building vector database**	**Latency of segmentation**	**Latency of retrieval**	**Latency of feedback**	**Latency of answering**	**F1-Match**
Naive RAG	3.256 GB	12.20 GB	0.004s	9.696s (1066 tokens/s)	0.914s	-	1.089s	0.652
BM25 + Naive RAG	3.243 GB	12.20 GB	0.005s	9.696s (1066 tokens/s)	0.003s	-	1.097s	0.594
BM25 + SAGE	13.150 GB	15.24 GB	0.005s	5.070s (644 tokens/s)	0.502s	2.810s	1.084s	0.612
SAGE	13.150 GB	15.61 GB	0.012s	16.050s (725 tokens/s)	0.8 + 0.50s	2.793s	1.091s	0.671
SAGE (5x)	15.500 GB	16.175 GB	0.012s	16.050s (725 tokens/s)	0.8 + 1.40s	2.793s	1.106s	0.671
SAGE (10x)	17.250 GB	16.720 GB	0.012s	16.050s (725 tokens/s)	0.8 + 2.25s	2.801s	1.097s	0.671

TABLE X

VERIFICATION OF THE EFFECTIVENESS OF FEATURE AUGMENTATION FOR SEMANTIC SEGMENTATION ON QASPER DATASET.

Metric / **Features**	Segmentation Accuracy
$(x_1), (x_2)$	84.5%
$(x_1), (x_2), (x_1 - x_2)$	85.6%
$(x_1), (x_2), (x_1 * x_2)$	88.4%
$(x_1), (x_2), (x_1 - x_2), (x_1 * x_2)$	**91.8%**

TABLE XI

COMPARISON OF COST EFFICIENCY (USING GPT-4O-MINI).

Metric / Model	Number of tokens	Accuray	Relative Cost Efficiency
BM25	140699	65.0%	0.646
DPR	142008	70.0%	0.689
SBERT	140888	67.5%	0.670
SAGE	**104939**	**75%**	**1.0**

RAG framework, which extends the Naive RAG method by incorporating the corpus segmentation technique as in Section IV, the gradient-based chunks selection mechanism as in Section V, and the self-feedback method as in Section VI. The default retriever used in SAGE is OpenAI Embedding.

Hyper-parameters. The segmentation score threshold ss, referenced in Section IV-D, is defined at 0.55. The length of coarse-grained l discussed in Section IV-E is set to 400. The initial minimum number of retrieved chunks min_k, as outlined in Section V-B, is determined to be 7. The gradient threshold g discussed in Section V-B is set to 0.3. The feedback score threshold fs, discussed in Section VI-A, is set to 9.

Environment. All experiments were performed on a ubuntu-22.04 server with a 20-core Intel(R) Xeon(R) 6242R 3.10GHz CPU, a Nvidia RTX3090 GPU, and 256GB DDR4 RAM.

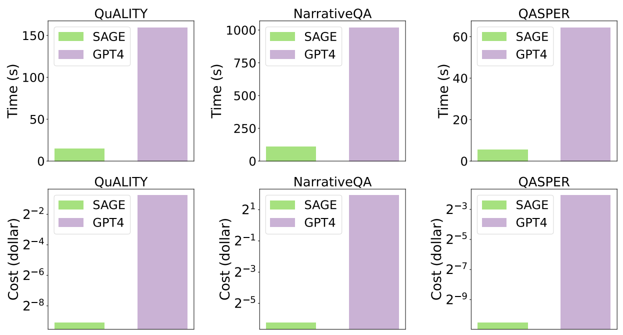


Fig. 7. Segmentation Overhead Evaluation.

B. End-to-End Question Answering Ability

In this subsection, we evaluate the QA ability of various methods on different datasets.

► **Exp-1: Effectiveness on NarrativeQA.**

We conduct an experiment on NarrativeQA dataset, to compare the End-to-End QA ability of different retrievers with and without the help of SAGE. The LLM used in this experiment is GPT4-o-mini. Table II shows the comparision results. We can observe that with the help of SAGE, the performance of retrievers SBERT, BM25, DPR, OpenAI Embedding increase 8.15% in ROUGE, 17.27% in BLEU-1, 81.51% in BLEU-4, 11.89% in METEOR on average. We can find that SAGE is effective for RAG systems with different retrievers. This is because SAGE is effective and orthometric with the embedding model and vector database modules in RAG systems.

► **Exp-2: Effectiveness on QuALITY and QASPER.**

We conduct an experiment on QuALITY and QASPER datasets, to compare the End-to-End QA ability of different retrievers with and without the help of SAGE. The LLM used in this experiment is GPT4-o-mini. Table III shows the comparison results. For QuALITY dataset, we can observe that the performance of retrievers SBERT, BM25, DPR, OpenAI Embedding increase 2.88% in Accuracy on average. For QASPER dataset, we can observe that the performance of retrievers SBERT, BM25, DPR, OpenAI Embedding increase 6.79% in F1-Match on average. The superior performance of RAG systems with SAGE is also because SAGE is effective and orthometric with the retrievers in RAG. We can find that the performance increment in QuALITY dataset is smaller than QASPER dataset. This is because QuALITY dataset is a multiple-choice QA dataset, and Accuracy in multiple-choice QA is easier than F1-Match in open-domain QA for both SAGE and the baselines.

► **Exp-3: Comparison with baselines on QuALITY.**

We conduct an experiment on QuALITY dataset, to compare the End-to-End QA ability of SAGE with other methods. Table VII shows the comparision results between SAGE and (Longformer-base, DPR+DeBERTaV3-large, CoLISA, and RAPTOR+GPT-4). Note that RAPTOR+GPT-4 is the state-of-the-art method in QuALITY dataset in the up-to-date leaderboard currently [36]. In the normal test set of QuALITY dataset, SAGE outperform these baselines by 128.4%, 62.82%, 44.78%, and 9.2%, respectively. In the hard test set of QuALITY dataset, SAGE outperforms these baselines by 116.1%, 114.1%, 39.49%, and 0.13%, respectively. These results verify the effectiveness of SAGE. We find that SAGE outperforms RAPTOR+GPT-4 in the normal test set more than the hard test set. This is because many questions in the hard set are challenging to answer using RAG methods. For instance, there are some questions that need to be solved by reading the whole corpus and using the elimination method, which can not be solved by only retrieving a few information for LLMs.

► **Exp-4: Comparison with baselines on NarrativeQA.**

We conduct an experiment on NarrativeQA dataset, to compare the End-to-End QA ability of SAGE with other methods. This experiment is conducted using UnifiedQA-3B language model. Table VI shows the comparison results between SAGE and (BiDAF, BM25+BERT, and Recursively Summarizing Books). We can find that SAGE outperforms these baselines by 258.4%, 43.35%, and 5.51% in ROUGE and 225.7%, 141%, and 19.78% in METEOR, respectively.

► **Exp-5: Comparison with baselines on QASPER.**

We conduct an experiment on QASPER datasets to compare the End-to-End QA ability of SAGE with other methods. This experiment is conducted using GPT-3.5 turbo and GPT4-o-min language models. Table V shows the comparison results between SAGE and (Title+Abstract and Raptor). For GPT-3.5 turbo, we find that SAGE outperforms these baselines by 144.1%, 16.45%, and 14.92%, respectively. For GPT4-o-mini, we find that SAGE outperforms these baselines by 151.2%, 10.54%, and 10.21%, respectively.

C. Scalability Evaluation

► **Exp-6: Scalability evaluation.**

We utilize the TRIVIA_QA dataset to test the scalability of SAGE, under varying degrees of concurrency (5x and 10x). We conduct experiments using different language models: the GPT4-o-mini via a web interface for Table VIII and the UnifiedQA-3B on our local server for Table IX. Our findings indicate that SAGE maintains superior performance over a naive RAG system even at high concurrency levels, with minimal increases in memory usage—only 27% even at 10x concurrency. This efficiency is attributed to the high parallelism capabilities of GPUs, which allow a single LLM and segmentation model to handle multiple forward computations simultaneously. Furthermore, the experiments demonstrate that while the retrieval latency increases slightly under higher concurrency (less than two seconds at 10x), the latency for feedback and answering processes remains consistent, regardless of the concurrency level. This stability is due to the effective parallel processing capacities of LLMs on GPUs, which effectively manage real-time latency demands. Additionally, the construction of the vector database and segmentation processes perform only once, showing consistent latency across different concurrency levels, ensuring that the initial setup does not impact the system's overall responsiveness in subsequent operations.

D. Ablation Study

In this subsection, we will verify the effectiveness of each main module of SAGE.

► **Exp-7: Ablation of each module of SAGE.**

We conduct an experiment on NarrativeQA dataset, to compare the End-to-End QA ability of Naive RAG, Naive RAG with each main module of SAGE, and SAGE. This experiment is conducted using GPT4-o-min language model. Table IV shows the comparison results. We can find that Naive RAG with Segmentation, Naive RAG with Selection, Naive RAG with Feedback, and SAGE outperform Naive RAG by 4.53%, 2.46%, 7.52% and 11.25% in ROUGE, 9.82%, 3.53%, 16.97% and 19.95% in BLEU-1, 13.79%, 44.83%, 179.31% and 486.21% in BLEU-4, and 0.86%, 1.49%, 12.65% and 13.28% in METEOR. We find that the increments in BLEU-4 and METEOR are smaller than that in ROUGE nad BLEU-4. These results verify the effectiveness of each module of SAGE. We find that SAGE outperforms Naive RAG with each main module. This is because the three modules do not negatively affect each other.

► **Exp-8: Ablation of feature selection.**

To validate the effectiveness of the feature augmentation in training the segmentation model, we conduct an ablation study comparing the model's accuracy with and without the augmented features $x_1 - x_2$ and $x_1 \cdot x_2$. Specifically, we divided the articles from the QASPER dataset into a training set and a validation set using an 8:2 ratio. We then trained the segmentation model using various combinations of features on the training set and evaluated the segmentation accuracy on the validation set. As shown in Table X, we find that the segmentation performance is higher when using the features $(x_1), (x_2), (x_1 - x_2), (x_1 * x_2)$ compared to using only $(x_1), (x_2)$ or any other incomplete feature combinations.

E. Cost Efficiency

In this subsection, we conduct an experiment to verify the superiority of `Cost-efficiency` of `SAGE`.

▶ **Exp-9: Comparison of Cost-efficiency.**

We conduct an experiment on `QuALITY` dataset, to compare the token consuming of a LLM, `Accuracy` of QA, and the `Cost-efficiency` of `SAGE` and naive rag systems with different embedding models. The LLM used in this experiment is `GPT4-o-mini`. Table XI shows the results. We can observe that `SAGE` outperforms the three baselines by 53.85%, 45.14%, and 49.25% of `Cost-efficiency` on average. Such improvements mainly because `SAGE` could achieve better QA performance while saving the input token of LLMs by eliminating noisy chunks.

F. Segmentation Overhead and Cost

In this subsection, we conduct an experiment to compare the efficiency and money cost of corpus segmentation by our segmentation model and by a LLM.

▶ **Exp-10: Comparison of Segmentation Overhead.**

We conduct an experiment on three articles sampled from `QuALITY`, `NarrativeQA`, and `QASPER` datasets separately, to compare the time and money consuming of corpus segmentation by `SAGE` and `GPT-4` (See Section I). The time-consuming and money-consuming of our segmentation model is calculated by a rented server with a RTX3090 GPU, whose rental price is 5.3 dollars per day [45]. Table 7 shows that our segmentation model saves time by 90.71% and money by 99.69% than using `GPT-4` in `QuALITY` dataset, saves time by 89.43% and money by 99.65% than using `GPT-4` in `NarrativeQA` dataset, and saves time by 91.49% and money by 99.72% than using `GPT-4` in `QASPER` dataset. We can find that our segmentation method can save a huge amount of both time and money than using `GPT-4`. This is because our segmentation model is lightweight, achieving fast inference speed with high parallelism. Specifically, it can do inference fast and only occupies a little GPU memory (0.2 GB), allowing it to process a batch of 512 sentence pairs in one fast inference.

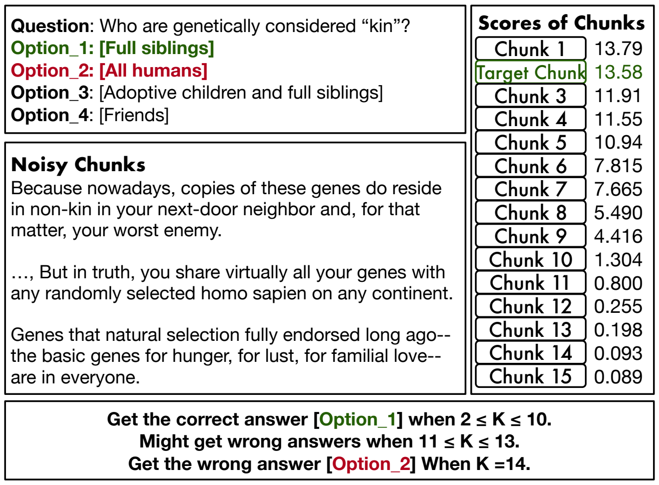
TABLE XII
ACCURACY ON QUALITY DATASET WITH DIFFERENT LLMS.

Model \ Metric	`GPT-3.5 Accuracy`	`GPT-4o-mini Accuracy`
`BM25`	62.70%	73.50%
`DPR`	60.4%	73.0%
`SAGE`	**64.5%**	**77.1%**

VIII. INSIGHTS OF RAG TASKS

We summarize the following insights in RAG tasks:

1) Noisy chunks retrieved considerably undermine the effectiveness of RAG systems (See **Exp-13**).



Question: Who are genetically considered "kin"?
Option_1: [Full siblings]
Option_2: [All humans]
Option_3: [Adoptive children and full siblings]
Option_4: [Friends]

Noisy Chunks
Because nowadays, copies of these genes do reside in non-kin in your next-door neighbor and, for that matter, your worst enemy.

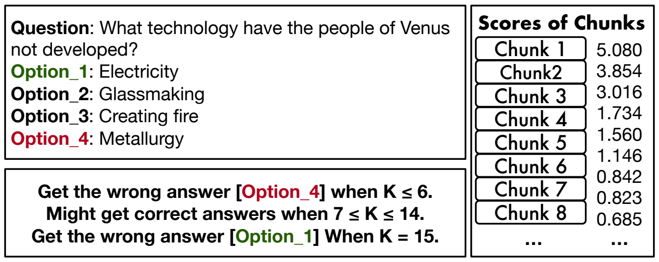
..., But in truth, you share virtually all your genes with any randomly selected homo sapien on any continent.

Genes that natural selection fully endorsed long ago--the basic genes for hunger, for lust, for familial love--are in everyone.

Scores of Chunks	
Chunk 1	13.79
Target Chunk	13.58
Chunk 3	11.91
Chunk 4	11.55
Chunk 5	10.94
Chunk 6	7.815
Chunk 7	7.665
Chunk 8	5.490
Chunk 9	4.416
Chunk 10	1.304
Chunk 11	0.800
Chunk 12	0.255
Chunk 13	0.198
Chunk 14	0.093
Chunk 15	0.089

Get the correct answer [Option_1] when 2 ≤ K ≤ 10.
Might get wrong answers when 11 ≤ K ≤ 13.
Get the wrong answer [Option_2] When K =14.

Fig. 8. A case of *noisy retrieval.*

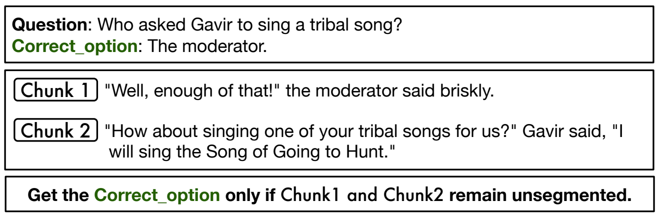


Question: What technology have the people of Venus not developed?
Option_1: Electricity
Option_2: Glassmaking
Option_3: Creating fire
Option_4: Metallurgy

Get the wrong answer [Option_4] when K ≤ 6.
Might get correct answers when 7 ≤ K ≤ 14.
Get the wrong answer [Option_1] When K = 15.

Scores of Chunks	
Chunk 1	5.080
Chunk2	3.854
Chunk 3	3.016
Chunk 4	1.734
Chunk 5	1.560
Chunk 6	1.146
	0.842
Chunk 7	0.823
Chunk 8	0.685
...	...

Fig. 9. A case of *missing retrieval.*



Question: Who asked Gavir to sing a tribal song?
Correct_option: The moderator.

Chunk 1 "Well, enough of that!" the moderator said briskly.

Chunk 2 "How about singing one of your tribal songs for us?" Gavir said, "I will sing the Song of Going to Hunt."

Get the Correct_option only if Chunk1 and Chunk2 remain unsegmented.

Fig. 10. A case of *ineffective corpus segmentation.*

2) Non-semantics-based corpus segmentation will impair the effectiveness of RAG systems (See **Exp-14**).
3) The proficiency level of LLMs plays a crucial role in the effectiveness of RAG systems (See **Exp-15**).
4) Interestingly, embedding models, though useful, are not as important as LLMs (See **Exp-15**).

▶ **Exp-11: Case Study of noisy retrieval.**

Figure 8 demonstrates a real case from `QuALITY` dataset, illustrating the issue of noisy retrieval. For the given question, the correct answer is `Option1`, with the `Target Chunk` being ranked second in terms of relevance. However, there are some noisy chunks containing some information supporting the `Option2` in the other chunks. When with fewer noisy chunks ($2 \leq K \leq 10$), a LLM could choose the correct option. However, when with more noisy chunks ($K \geq 11$), the LLM might be misled into choosing the wrong answer `Option2`.

▶ **Exp-12: Case Study of missing retrieval.**

Figure 9 demonstrates a real case from `QuALITY` dataset, showing the issue of missing retrieval. For the given question

"Which technology not be developed?", it needs lots of context to use the elimination method to choose a correct answer. Therefore, we can find that $K < 6$ will lead to a wrong answer, and $K = 15$ will lead to a correct answer. From the relevance scores of chunks, we can see that the sorted scores are smooth. With our gradient-based chunk selection algorithm, SAGE will choose more chunks and get the correct answer.

► **Exp-13: Case Study of incomplete chunks.**

Figure 10 illustrates a case showing the issue of ineffective corpus segmentation. In the original corpus, Chunk1 and Chunk2 are consecutive. A LLM could produce a correct answer if consecutive (Chunk1, Chunk2) are retrieved. However, if using fixed-length segmentation, they may be segmented. Furthermore, the relevance score of Chunk2 is higher than Chunk1, so they are impossible to retrieve together as in the original corpus. Finally, for the given question, it is infeasible to conclude that "The moderator asked Gavir to sing a tribal song.", resulting in a wrong answer.

► **Exp-14: Different LLMs and embedding models.**

To evaluate the effectiveness of different LLMs, Table XII shows the Accuracy of RAG methods using two proficiency levels of LLMs (GPT-3.5-turbo and GPT-4o-mini) on QuALITY dataset. The results show that RAG methods using GPT-4o-mini outperform RAG methods using GPT-3.5-turbo 17.38%, 20.86%, and 19.53% on average. These results indicate the importance of the proficiency level of LLMs in RAG. To evaluate the effectiveness of different embedding models, Tabel II shows the performance of RAG methods with different embedding models. We can find that the performance order is SBERT > OpenAI Embedding > DPR > BM25, but the variance is smaller than that with different LLMs. This result suggests that while embedding models contribute to RAG's performance, they are not as influential as the choice of LLMs.

IX. Related Work

A. Data Management

Our work is a general framework to enhance the retrieval accuracy of information retrieval applications. It can be applied to the recent data management works that need to retrieve data according to the embedding distances of retrieved items. For instance, in multimodal retrieval, such as the systems developed by [66] and [25], our framework could improve semantic alignment and cross-modal consistency, thereby enhancing overall retrieval precision. For some efficient machine learning systems-related work, such as [16], [23], [47], [49]–[51], [56]–[61], [64], [65], our framework can be used in conjunction with these works within large models of RAG systems, thus ensuring both inference efficiency and retrieval accuracy. For some systems in databases, such as [43], [53]–[55], [62], [63], our framework can be utilized to detach unstructured data from databases and use RAG technology for retrieval. For distributed systems like those discussed in [52], integrating our framework could optimize data retrieval processes, enhancing efficiency and reducing operational costs.

B. Retrieval-Augmented Generation.

The Retrieval-Augmented Generation (RAG) framework is a key advancement in natural language processing. It improves how generation systems understand and create text by using extra information from outside sources. RAG works by finding relevant information first and then using it to make better and more relevant text outputs. For example, one study [22] introduced a model that efficiently locates pertinent documents to aid in answering questions or verifying facts. This method was much better at these tasks because it used external knowledge. Another important development, Dense Passage Retrieval (DPR) [18], makes it easier to find the right pieces of information in large sets of data. This helps in creating accurate and to-the-point texts, which is the main goal of RAG. Recent works have further refined RAG's effectiveness through various strategies, including prompt engineering [24], [40], [68], rewriting questions [13], [29], adding knowledge basese [2], [67], and iterative answering [6], [42], [46]. However, these methods often overlook the need to segment the corpus semantically and reduce irrelevant context retrieved, that SAGE, aims to solve. In summary, RAG continues to evolve through continuous improvement in merging retrieval and generation. Our system SAGE focuses on the retrieval side.

X. Conclusion and Future Work

In this paper, we propose a framework of precise retrieval for RAG systems, named SAGE. We propose to train a segmentation model to segment a given corpus semantically and with low latency. Moreover, we propose a chunk selection algorithm to select the most relevant chunks rather than a larger fixed number of chunks. Lastly, we propose a self-feedback method to enable LLMs to adjust the number of retrieved chunks automatically. Experiments show that SAGE can achieve better QA performance with a lower cost of LLM inference than other baselines.

Looking ahead, we find three promising directions: (1) **Multi-hop retrieval.** We find the importance of scenarios where answers come from multiple documents like AiR Baleen [20] and leave a comprehensive design for such scenarios as future work. (2) **Integration of SAGE with fine-tuning of LLMs.** We have found that the proficiency level of LLMs is very important for the performance of a RAG system (See Section VIII). However, using the most powerful LLMs, e.g., GPT-4, is expensive. Fine-tuning is a simple way to enhance the QA ability of a LLM for a given corpus. For example, we can generate several batches of question-answer pairs to fine-tune GPT-3.5-turbo. Then, we might achieve the same QA performance based on the inexpensive LLM. (3) **A more flexible chunk selection strategy.** Currently, SAGE selects top chunks with higher relevance scores. Although SAGE selects a dynamic number of chunks, it is still possible there are useless chunks, e.g., the chunk with the highest relevance score is useless. Therefore, a more flexible chunk selection strategy might help. For example, we can train a LLM smart enough to select relevant chunks directly.

REFERENCES

[1] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[2] J. Baek, S. Jeong, M. Kang, J. C. Park, and S. J. Hwang. Knowledge-augmented language model verification. *arXiv preprint arXiv:2310.12836*, 2023.

[3] S. Banerjee and A. Lavie. Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. In *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, pages 65–72, 2005.

[4] I. Beltagy, M. E. Peters, and A. Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

[5] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[6] X. Cheng, D. Luo, X. Chen, L. Liu, D. Zhao, and R. Yan. Lift yourself up: Retrieval-augmented text generation with self-memory. *Advances in Neural Information Processing Systems*, 36, 2024.

[7] F. Cuconasu, G. Trappolini, F. Siciliano, S. Filice, C. Campagnano, Y. Maarek, N. Tonellotto, and F. Silvestri. The power of noise: Redefining retrieval for rag systems. In *SIGIR*, pages 719–729, 2024.

[8] P. Dasigi, K. Lo, I. Beltagy, A. Cohan, N. A. Smith, and M. Gardner. A dataset of information-seeking questions and answers anchored in research papers. *arXiv preprint arXiv:2105.03011*, 2021.

[9] M. Dong, B. Zou, Y. Li, and Y. Hong. Colisa: inner interaction via contrastive learning for multi-choice reading comprehension. In *European Conference on Information Retrieval*, pages 264–278. Springer, 2023.

[10] M. Douze, A. Guzhva, C. Deng, J. Johnson, G. Szilvasy, P.-E. Mazaré, M. Lomeli, L. Hosseini, and H. Jégou. The faiss library. *arXiv preprint arXiv:2401.08281*, 2024.

[11] D. Edge, H. Trinh, N. Cheng, J. Bradley, A. Chao, A. Mody, S. Truitt, and J. Larson. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*, 2024.

[12] W. Foundation. Wikimedia downloads.

[13] L. Gao, X. Ma, J. Lin, and J. Callan. Precise zero-shot dense retrieval without relevance labels. *arXiv preprint arXiv:2212.10496*, 2022.

[14] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, and H. Wang. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2023.

[15] P. He, J. Gao, and W. Chen. Debertav3: Improving deberta using electra-style pre-training with gradient-disentangled embedding sharing. *arXiv preprint arXiv:2111.09543*, 2021.

[16] Y. Hu, W. Huang, Z. Liang, C. Chen, J. Zhang, J. Zhu, and J. Chen. Identifying sensitive weights via post-quantization integral. *arXiv preprint arXiv:2503.01901*, 2025.

[17] M. Joshi, E. Choi, D. Weld, and L. Zettlemoyer. triviaqa: A Large Scale Distantly Supervised Challenge Dataset for Reading Comprehension. *arXiv e-prints*, page arXiv:1705.03551, 2017.

[18] V. Karpukhin, B. Oğuz, S. Min, P. Lewis, L. Wu, S. Edunov, D. Chen, and W.-t. Yih. Dense passage retrieval for open-domain qa. *arXiv preprint arXiv:2004.04906*, 2020.

[19] D. Khashabi, S. Min, T. Khot, A. Sabharwal, O. Tafjord, P. Clark, and H. Hajishirzi. Unifiedqa: Crossing format boundaries with a single qa system. *arXiv preprint arXiv:2005.00700*, 2020.

[20] O. Khattab, C. Potts, and M. Zaharia. Baleen: Robust multi-hop reasoning at scale via condensed retrieval. *Advances in Neural Information Processing Systems*, 34:27670–27682, 2021.

[21] T. Kočiskỳ, J. Schwarz, P. Blunsom, C. Dyer, K. M. Hermann, G. Melis, and E. Grefenstette. The narrativeqa reading comprehension challenge. *Transactions of the Association for Computational Linguistics*, 6:317–328, 2018.

[22] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

[23] B. Li, J. Chen, and J. Zhu. Memory efficient optimizers with 4-bit states. *Advances in Neural Information Processing Systems*, 36, 2024.

[24] G. Li, X. Zhou, and X. Zhao. LLM for data management. *Proc. VLDB Endow.*, 17(12):4213–4216, 2024.

[25] T. Li, X. Yang, Y. Ke, B. Wang, Y. Liu, and J. Xu. Alleviating the inconsistency of multimodal data in cross-modal retrieval. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*, pages 4643–4656. IEEE, 2024.

[26] X. Li and J. Li. Angle-optimized text embeddings. *arXiv preprint arXiv:2309.12871*, 2023.

[27] C.-Y. Lin. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81, 2004.

[28] LlamaIndex. Llamaindex, 2024. Available online: https://www.llamaindex.ai/open-source.

[29] X. Ma, Y. Gong, P. He, H. Zhao, and N. Duan. Query rewriting for retrieval-augmented large language models. *arXiv preprint arXiv:2305.14283*, 2023.

[30] X. Mou, M. Yu, B. Yao, C. Yang, X. Guo, S. Potdar, and H. Su. Frustratingly hard evidence retrieval for qa over books. *arXiv preprint arXiv:2007.09878*, 2020.

[31] OpenAI. Guide to embeddings, 2023. Available online: https://platform.openai.com/docs/guides/embeddings.

[32] OpenAI. gpt-4o-mini, 2024. Available online: https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/.

[33] OpenAI. text-embedding-3-small, 2024. Available online: https://openai.com/index/new-embedding-models-and-api-updates/.

[34] R. Y. Pang, A. Parrish, N. Joshi, N. Nangia, J. Phang, A. Chen, V. Padmakumar, J. Ma, J. Thompson, H. He, et al. Quality: Question answering with long input texts, yes! *arXiv preprint arXiv:2112.08608*, 2021.

[35] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318, 2002.

[36] QuALITY Team. Quality leadboard, 2024. Available online: https://nyu-mll.github.io/quality/.

[37] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

[38] N. Reimers. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*, 2019.

[39] S. E. Robertson, S. Walker, S. Jones, M. M. Hancock-Beaulieu, M. Gatford, et al. Okapi at trec-3. *Nist Special Publication Sp*, 109:109, 1995.

[40] P. Sahoo, A. K. Singh, S. Saha, V. Jain, S. Mondal, and A. Chadha. A systematic survey of prompt engineering in large language models: Techniques and applications. *arXiv preprint arXiv:2402.07927*, 2024.

[41] P. Sarthi, S. Abdullah, A. Tuli, S. Khanna, A. Goldie, and C. D. Manning. Raptor: Recursive abstractive processing for tree-organized retrieval. *arXiv preprint arXiv:2401.18059*, 2024.

[42] Z. Shao, Y. Gong, Y. Shen, M. Huang, N. Duan, and W. Chen. Enhancing retrieval-augmented large language models with iterative retrieval-generation synergy. *arXiv preprint arXiv:2305.15294*, 2023.

[43] J. Sun, J. Zhang, Z. Sun, G. Li, and N. Tang. Learned cardinality estimation: A design space exploration and a comparative evaluation. *Proceedings of the VLDB Endowment*, 15(1):85–97, 2021.

[44] R. Teja. Evaluating the ideal chunk size for a rag system using llamaindex. *LLAMAi,[Online]. Available: https://www. llamaindex. ai/blog/evaluating-the-ideal-chunk-size-for-a-ragsystem-using-llamaindex-6207e5d3fec5*, 30:31, 2023.

[45] vast.ai. vast.ai. Available online: https://cloud.vast.ai/.

[46] J. Wang and G. Li. Aop: Automated and interactive llm pipeline orchestration for answering complex queries. *CIDR*, 2025.

[47] Z. Wang, J. Chen, and J. Zhu. Efficient backpropagation with variance-controlled adaptive sampling. *arXiv preprint arXiv:2402.17227*, 2024.

[48] J. Wu, L. Ouyang, D. M. Ziegler, N. Stiennon, R. Lowe, J. Leike, and P. Christiano. Recursively summarizing books with human feedback. *arXiv preprint arXiv:2109.10862*, 2021.

[49] H. Xi, Y. Chen, K. Zhao, K. J. Teh, J. Chen, and J. Zhu. Jetfire: Efficient and accurate transformer pretraining with int8 data flow and per-block quantization. *arXiv preprint arXiv:2403.12422*, 2024.

[50] H. Xi, S. Yang, Y. Zhao, C. Xu, M. Li, X. Li, Y. Lin, H. Cai, J. Zhang, D. Li, et al. Sparse videogen: Accelerating video diffusion transformers with spatial-temporal sparsity. *arXiv preprint arXiv:2502.01776*, 2025.

[51] S. Yang, H. Xi, Y. Zhao, M. Li, J. Zhang, H. Cai, Y. Lin, X. Li, C. Xu, K. Peng, et al. Sparse videogen2: Accelerate video generation with sparse attention via semantic-aware permutation. *arXiv preprint arXiv:2505.18875*, 2025.

[52] F. Yao, Q. Tao, W. Yu, Y. Zhang, S. Gong, Q. Wang, G. Yu, and J. Zhou. Ragraph: A region-aware framework for geo-distributed graph processing. *Proceedings of the VLDB Endowment*, 17(3):264–277, 2023.

[53] C. Zhang, G. Li, and T. Lv. HyBench: A New Benchmark for HTAP Databases. *Proceedings of the VLDB Endowment*, 17(5):939–951, 2024.

[54] C. Zhang, G. Li, J. Zhang, X. Zhang, and J. Feng. Htap databases: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 2024.

[55] C. Zhang, J. Lu, P. Xu, and Y. Chen. UniBench: A Benchmark for Multi-model Database Management Systems. In *TPCTC*, pages 7–23. Springer, 2018.

[56] J. Zhang, H. Huang, P. Zhang, J. Wei, J. Zhu, and J. Chen. Sageattention2: Efficient attention with smoothing q and per-thread quantization. 2025.

[57] J. Zhang, H. Huang, P. Zhang, J. Wei, J. Zhu, and J. Chen. Sageattention2: Efficient attention with thorough outlier smoothing and per-thread int4 quantization. In *International Conference on Machine Learning (ICML)*, 2025.

[58] J. Zhang, J. Wei, P. Zhang, X. Xu, H. Huang, H. Wang, K. Jiang, J. Zhu, and J. Chen. Sageattention3: Microscaling fp4 attention for inference and an exploration of 8-bit training. *arXiv preprint arXiv:2505.11594*, 2025.

[59] J. Zhang, C. Xiang, H. Huang, J. Wei, H. Xi, J. Zhu, and J. Chen. Spargeattn: Accurate sparse attention accelerating any model inference. *arXiv preprint arXiv:2502.18137*, 2025.

[60] J. Zhang, C. Xiang, H. Huang, J. Wei, H. Xi, J. Zhu, and J. Chen. Spargeattn: Training-free sparse attention accelerating any model inference. 2025.

[61] J. Zhang, X. Xu, J. Wei, H. Huang, P. Zhang, C. Xiang, J. Zhu, and J. Chen. Sageattention2++: A more efficient implementation of sageattention2. 2025.

[62] J. Zhang, C. Zhang, G. Li, and C. Chai. Autoce: An accurate and efficient model advisor for learned cardinality estimation. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*, pages 2621–2633. IEEE, 2023.

[63] J. Zhang, C. Zhang, G. Li, and C. Chai. Pace: Poisoning attacks on learned cardinality estimation. *Proceedings of the ACM on Management of Data*, 2(1):1–27, 2024.

[64] J. Zhang, P. Zhang, J. Zhu, J. Chen, et al. Sageattention: Accurate 8-bit attention for plug-and-play inference acceleration. In *The Thirteenth International Conference on Learning Representations*.

[65] P. Zhang, J. Wei, J. Zhang, J. Zhu, and J. Chen. Accurate int8 training through dynamic block-level fallback. *arXiv preprint arXiv:2503.08040*, 2025.

[66] J. Zheng, M. Liang, Y. Yu, Y. Li, and Z. Xue. Knowledge graph enhanced multimodal transformer for image-text retrieval. In *ICDE*, pages 70–82. IEEE, 2024.

[67] X. Zhou, G. Li, Z. Sun, Z. Liu, W. Chen, J. Wu, J. Liu, R. Feng, and G. Zeng. D-bot: Database diagnosis system using large language models. *Proc. VLDB Endow.*, 17(10):2514–2527, 2024.

[68] X. Zhou, Z. Sun, and G. Li. DB-GPT: large language model meets database. *Data Sci. Eng.*, 9(1):102–111, 2024.