FragRel: Exploiting Fragment-level Relations in LLM External Memory

Anonymous ACL submission

Abstract

To process contexts with unlimited length using Large Language Models (LLMs), recent studies explore hierarchically managing the long text. Only several text fragments are taken from the external memory and passed into the temporary working memory, i.e., LLM's context window. However, existing approaches isolatedly handle the text fragments without considering their structural connections, thereby suffering limited capability on texts with intensive inter-relations, e.g., coherent stories and code repositories. This work attempts to resolve this by exploiting the fragment-level relations in external memory. First, we formulate the fragment-level relations and present several instantiations for different text types. Next, we introduce a relation-aware fragment assessment criteria upon previous independent fragment assessment. Finally, we present the fragment-connected Hierarchical Memory based LLM. We validate the benefits of involving these relations on long story understanding, repository-level code generation, and long-term chatting.

1 Introduction

The limited context window length constrains applications of Large Language Models (LLMs) in some practical scenarios, such as answering questions based on complete books or movie scripts (Kočiskỳ et al., 2018), writing codes within complete Github repositories (Zhang et al., 2023a), *etc.* To resolve this problem, some works (Ding et al., 2023; Han et al., 2023; Xiao et al., 2023) attempt to expand the context length of classical LLM inference framework via continual training or sparse attention mechanism. However, existing approaches are either limited to a finite expansion length (Packer et al., 2023; Schuurmans, 2023), or prone to performance degradation, especially when dealing with very long contexts (Liu et al., 2023).

Recent studies (Packer et al., 2023; Wang et al., 2023b; Ram et al., 2023) explore to hierarchically process the long text. Each time only partial fragments of long text are retrieved from external memory and fed into LLM's context window, a.k.a, temporary working memory, thereby eliminating the context length constraint and alleviating the inferior influence of substantial irrelevant content. However, current external memory managers simply split the complete long context into independent fragments, assessing their isolated importance during retrieval. This constrains the inference capability of Hierarchical Memory based LLMs, particularly in scenarios (*e.g.* understanding coherent story or code repository) where there are intensive associations across fragments of long text.

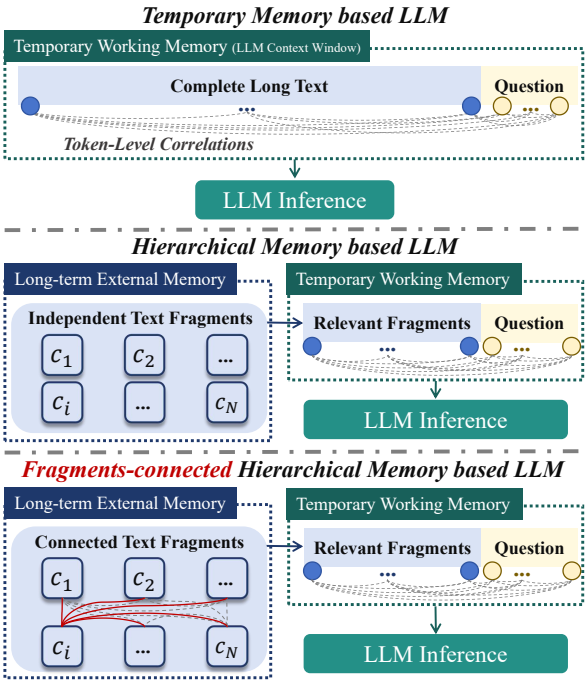


Figure 1: Schematic diagram of Fragment-connected Hierarchical Memory based LLM.

To address this issue, we propose to integrate these fragment-level relations into the external memory management by introducing a relation-aware fragments assessment score during retrieval. First, we formulate the relations between two frag-

ments as a real number, with higher values corresponding to stronger relation strength. The calculation of relation quantity could have different instantiations for different context types (*e.g.* narrative stories, code repositories, or historical dialog) and different relation types (*e.g.* semantic relations or structural relations). Next, based on the isolated relevance scores used in past external memory retrievers, we further calculate every fragment's environmental relevance score, which is defined as a normalized relation-weighted summation of other fragments' independent scores. During retrieval, the combination of independent score and environmental score is employed for assessing every fragment's importance. An adjustable coefficient is introduced to control the influence of environmental score. Finally, in the same as previous works (Packer et al., 2023; Ram et al., 2023), we concatenate the retrieved content and the requested instruction as LLM inference input.

Extensive experiments validate the benefits of incorporating these fragment-level relations during retrieval. The experiments encompass a variety of base LLMs (Llama2 (Touvron et al., 2023), ChatGPT, ChatGLM (Du et al., 2022), *etc.*), different temporary context lengths (1K, 4K, 8K, 20K, *etc.*), and multiple long context scenarios (Long Story Understanding (Kočiskỳ et al., 2018), Repository-level Code Completion (Zhang et al., 2023a), and Long-term Chatting with Human (Lu et al., 2023)).

2 Related Work

Temporarily Long Context Processing. One line of works explores how to process long context under the typical LLM inference framework, where all context is directly stored in the LLM context window (Temporary Memory). First, a series of works (Dai et al., 2019; Ding et al., 2023; Han et al., 2023; Xiao et al., 2023; Xiong et al., 2023) explore extending context length of Transformer-based models via efficient attention mechanism and recurrent inference strategic. In addition, some works (Li, 2023; Jiang et al., 2023a) investigate how to compress the length of long text content to mitigate the impact of excessive irrelevant text. Although the long-text processing capability of the LLM can be enhanced by expanding the context window or compressing the context content, the context length that can be handled remains limited.

External Memory augmented LLM. Another line of work introduces additional external memory, forming a hierarchical memory based inference framework, thereby processing context of any length. Relevant content is retrieved from external memory for knowledge updating (Wu et al., 2022; Wang et al., 2023b) or answering knowledge-intensive questions (Lewis et al., 2020; Guu et al., 2020; Borgeaud et al., 2022; Lan et al., 2023; Wang et al., 2023a). The most popular retrieval method is calculating the text embedding similarity for isolated fragments of external context using pre-trained embedding models (Su et al., 2022; Zhang et al., 2023b), and retrieving the text fragments with higher similarity to the requested question or current temporary context.

Benefiting from the zero-shot generalization capability of LLMs, the retrieved fragments can be directly concatenated with instructions as model input (Ram et al., 2023), eliminating the need for additional training. This further facilitates more flexible external memory augmented LLM inference frameworks. Firstly, some studies explore the collaborative use of retrieval and generation (Gao et al., 2022; Yan et al., 2024), as well as further multi-round retrieval-generation interleaving framework (Trivedi et al., 2022; Jiang et al., 2023b; Asai et al., 2023; Shao et al., 2023; Feng et al., 2023). Saad-Falcon et al. (2023) utilizes explicit prompts about the structure of external context for enhanced retrieval. Additionally, MemGPT (Packer et al., 2023) automatically reads and writes the external memory, enabling more flexible external memory reasoning.

3 Methodology

3.1 Preliminary

3.1.1 Temporary Memory based LLM

Formulation. Traditional language models receive input x and generate the output y by:

$$y = \text{LLM}(x). \tag{1}$$

The input x is straightly passed into the context window of LLM. Notably, the instructed LLMs (Ouyang et al., 2022) could take various user instructions x and produce the corresponding responses y. Inspired by human cognition, the LLM context window could be viewed as the *working memory* which temporarily stores information (Packer et al., 2023; Li et al., 2022). For clarity, we refer to this traditional LLM reasoning framework shown in Equation. 1 as *Temporary Memory based LLM* (TempMem-LLM).

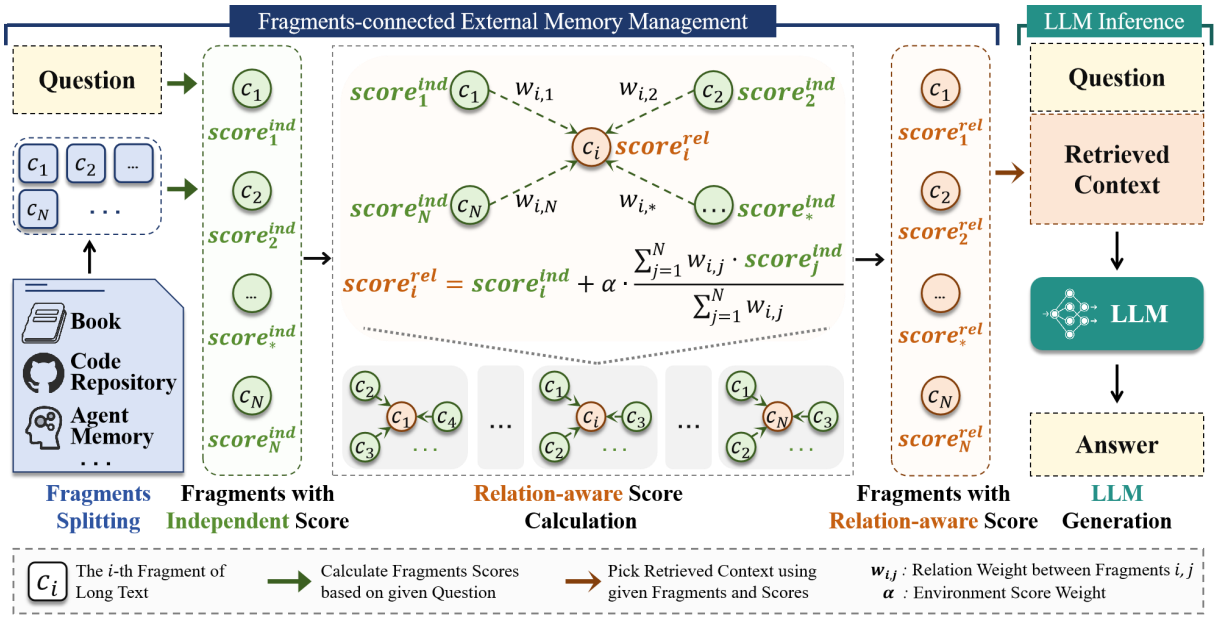


Figure 2: Framework of *Fragment-connected* Hierarchical Memory based LLM. The long context (complete book, code repository, or agent memory) is first split into a lot of fragments c_*. Then the independent relevance scores $Score_*^{ind}$ to the user question for every fragment is calculated. Next the relation-aware score $Score_*^{rel}$ is calculated as the combination of its independent score and the normalized relation-weighted summation of other fragments' independent score. Finally, the fragments within Top-K relation-aware scores are retrieved for LLM inference.

Long Text Processing. In many practical tasks, the input content includes not only user instructions x, but also an additional long context $\mathcal{T}$, such as answering questions based on complete storybooks or movie scripts (Kočiskỳ et al., 2018), writing codes in long Github repository (Zhang et al., 2023a), and constructing agents capable of engaging in long-term conversations (Lu et al., 2023; Zhong et al., 2023). In these scenarios, TempMem-LLM simply concatenates and processes the long context $\mathcal{T}$ and instruction x in its working memory:

$$y = \text{LLM}(x \oplus \mathcal{T}), \tag{2}$$

where $\oplus$ represents the concatenation operation.

When the text length of $\mathcal{T}$ exceeds the context window limitations of LLM, we could cut its additional content of $\mathcal{T}$ for executing inference. In next section, we present another framework for effectively processing the lengthy context $\mathcal{T}$ via storing it in external memory and retrieving relevant fragments for inference every time.

3.1.2 Hierarchical Memory based LLM

Unlike TempMem-LLM which handles all context in its temporary working memory, the Hierarchical Memory based LLM (HieraMem-LLM) integrates an additional non-parametric external memory for managing the long context $\mathcal{T}$.

Formulation. Instead of directly being concatenated with the user instruction x, the context $\mathcal{T}$ is independently processed in HieraMem-LLM. It consists of two decoupled modules, i.e., the external memory management module processing the long context $\mathcal{T}$, as well as the LLM forward inference module containing the temporary working memory. Formally, we have:

$$\begin{aligned} \mathcal{T}^{ret} &= \text{Mem-MGR}(x, \mathcal{T}), \\ y &= \text{LLM}(x \oplus \mathcal{T}^{ret}). \end{aligned} \tag{3}$$

Mem-MGR is the external memory manager, which takes the requested instruction x as input and returns relevant fragments from $\mathcal{T}$.

External Memory Manager. The typical external memory manager consists of three steps, i.e., fragment splitting, independent score calculation, and fragment selection.

1. *Fragment Splitting*. It splits the long text $\mathcal{T}$ into N short fragments c_*.
2. *Independent Score Calculation*. It calculates the independent score s_i^{ind} for every fragment c_i based on user input x, i.e.,

$$s_i^{ind} = \text{Similarity}(x, c_i). \tag{4}$$

The similarity function is often instantiated as the cosine similarity between embedding vectors of x and c_i, which could be calculated using pre-trained text embedding models (Izacard et al., 2021).

3. *Retrieved Context Picking*. With fragment scores s_i^{ind}, we select top related fragments and feed them into the temporary memory of LLM.

Although HieraMem-LLM effectively manages the long context $\mathcal{T}$ by utilizing external memory, the context is decomposed into unrelated segments, disrupting the structural integrity of the context.

TempMem *vs.* HieraMem. TempMem-LLM straightly handles the complete long text $\mathcal{T}$ in its working memory, integrating token-level correlations during inference. However, the simple concatenation approach in Equation 2 suffers the following issues: (1) When $\mathcal{T}$ exceeds the model's context length constraint, the LLM is unable to do prediction. (2) The information irrelevant to instruction x can interfere with the model's processing, leading to performance decline (Liu et al., 2023). (3) Reprocessing the lengthy context $\mathcal{T}$ each time consumes excessive computational resources. To address the issues of TempMem-LLM, HieraMem-LLM incorporates the external memory to manage the prolix context $\mathcal{T}$. Only a few related fragments are extracted for LLM inference.

However, existing external memory managers select fragments based on only isolated fragment content, overlooking intensive relations between fragments. While in TempMem-LLM, the long-term correlations could be employed via the attention mechanism over the complete text, enabling comprehensive context modeling.

3.2 Fragment Relations

3.2.1 Definition

We formulate the relations between every pair of fragments (c_i, c_j) as follows:

$$w_{i,j} = \mathcal{F}^{rel}(c_i, c_j),\ 1 \le i, j \le N, \qquad (5)$$

where $w_{i,j}$ is a real numbers measuring the relation strength between fragment c_i and fragment c_j. The larger value of $w_{i,j}$ indicates the stronger correlation between c_i and c_j. Next, we present several specific implementations for $\mathcal{F}^{rel}$.

3.2.2 Fragment Relation Instantiations

This section introduces several instantiations of fragment-level relations and discusses the importance of considering these relations.

Semantic Relation. Semantic association is the most common type of connection between text segments. The semantic correlation between text fragments can be measured by the cosine similarity between the latent embeddings of fragments,

$$\mathcal{F}^{rel}(c_i, c_j) = 1 - \frac{e_i \cdot e_j}{\|e_i\|\|e_j\|}, \qquad (6)$$

where e_i and e_j represent the latent embeddings of fragments c_i and c_j respectively.

Context Structure Relation. In consecutive books or long-term dialog, there are significant content correlations between the contiguous preceding and following fragments. The fragments with closer positions in the context have stronger relevance. This contextual relationship strength can be defined as:

$$\mathcal{F}^{rel}(c_i, c_j) = w_{rel}^{|loc_i - loc_j|},\ w_{rel} \in [0, 1]. \qquad (7)$$

where loc_i refers the absolute position of fragment c_i in the external context $\mathcal{T}$. w_{rel} can be adjusted to control the relation strength between fragments. When $w_{rel} = 0$, it represents there is no relation between fragments, and our method degrades to previous fragment-independent external memory.

Code Structure Relation. Compared to natural language, code repository fragments have more complex interrelations. We construct the structure graph $\mathcal{G}$ for a complete code repository. The code graph $\mathcal{G}$ consists of all code parsing nodes (including function definition, function body, assignment expression, *etc.*), and edges based on the parsing tree, function calling relation, and the files directory structure. The edge weights take values in $[0, 1]$ and are set based on the edge types. Section. A.1 presents more details about the construction of $\mathcal{G}$. The i-th code fragment c_i contains $N_{c_i}^G$ non-overlapping graph nodes, represented as $\{g_k^{c_i}\}_{k=1}^{N_{c_i}^G}$. Based on the code graph $\mathcal{G}$, we formulate the code structure relation as follows:

$$\mathcal{F}^{rel}(c_i, c_j) = \frac{\sum_{k=1}^{N_{c_i}^G}\sum_{l=1}^{N_{c_j}^G} K_{k,l}^{c_i,c_j} \cdot Dis(g_k^{c_i}, g_l^{c_j})}{\sum_{k=1}^{N_{c_i}^G}\sum_{l=1}^{N_{c_j}^G} K_{k,l}^{c_i,c_j}},$$
$$K_{k,l}^{c_i,c_j} = len_k^{c_i} \cdot len_l^{c_j} \qquad (8)$$

where $len_k^{c_i}$ represents the text length of the k-th paring node in fragment c_i. $Dis(g_k^{c_i}, g_k^{c_j})$ is the shortest path distance between node $g_k^{c_i}$ and $g_k^{c_j}$ on the code graph $\mathcal{G}$. Section. A.1 shows more detail.

More Relations. More relations could be designed for specific text properties and practical needs. For example, we can gauge the correlation strength between academic papers based on citation relationships and author associations.

Importance of Fragments Relations. The fragments-level relations are significant for:
1. Ubiquitous existence. These fragments-level relations ubiquitously appear in a variety of long texts. For example, in narrative books or movie scripts, the storyline progresses coherently from beginning to end, with each fragment c_* intricately connected to its preceding and following fragments c_*. In code repository, structural correlations exist among different code lines, and function calling or variable passing relationships exist between different code files and functions. Therefore, the content of different code fragments c_* are densely related.
2. Assisting long text understanding. Unlike TempMem-LLM latently utilizes long-term relations, we posit that involving explicit inter-relations plays its role in external memory management by forming a more comprehensive assessment criteria for fragment selection. Specifically, when neglecting the fragments relations, the external memory retriever greedily supposes only the fragments with high *direct similarity* to the input x is helpful (Previous). The consideration of fragment relations allows a more comprehensive fragments assessment criteria: *i.e.* the fragments with both higher *direct similarity* and *contextual similarity* to x is important (Ours). The contextual similarity of fragment c_i refers to the similarity between the c_i's related fragments and the input x. The new assessment criteria retrieve not only directly similar fragments but also take account of: (a) fragments within a relevant environment, (b) contextual fragments of relevant fragments, which could help understand the directly relevant fragment, (c) indirectly relevant fragments.

3.3 Fragment-connected Memory Retrieval

This section introduces the integration of fragment relations by calculating the relation-aware assessment scores, and presents the overall framework of Fragment-connected HieraMem-LLM.

3.3.1 Relation-aware Fragment Assessment

Different from vanilla retriever which considers the independent importance of every fragment using independent score s_i^{ind}, we instead propose to calculate a *Relation-aware Score* for more comprehensively considering the importance of every fragment.

Definition. For the i-th fragment, the relation-aware score is composed of two parts: its independent score s_i^{ind} and its *environment score* s_i^{env}. The independent score measures its direct relevance degree with question x, defined in Equation 4. The environment score s_i^{env} assesses its related fragments' relevance with question x. We formulate s_i^{env} as the normalized weighted summation over independent scores s_j^{env} of related fragments:

$$s_i^{env} = \frac{\sum_j w_{i,j} \cdot s_j^{ind}}{\sum_j w_{i,j}}, \tag{9}$$

where $w_{i,j}$ is the fragment relation defined in Equation 5. The normalization operation is introduced to ensure the consistent numerical scale of s_i^{env} with s_i^{ind}.

Combining s_i^{ind} and s_i^{env}, we define *Relation-aware Score* of the i-th fragment as follows:

$$s_i^{rel} = s_i^{ind} + \alpha \cdot s_i^{env}, \tag{10}$$

where α is an adjustable coefficient, employed to control the influence of environment score.

Relation Distance and Complexity. The utilization of explicit fragment-level relations shown in Equation. 9 is irrelevant with fragments distance, while TempMem-LLM extracts relations within ranges limited by the context window length. Additionally, some complex relations (*e.g.* code structure relations) are challenged to automatically learn while they could be employed explicitly.

3.3.2 Fragment-connected HieraMem-LLM

Based on the proposed relation-aware score, we introduce the overall framework of *Fragment-connected HieraMem-LLM*, shown in Figure. 2.

We first split the long text $\mathcal{T}$ into fragments and acquire their independent scores. Next, considering the extracted relations, we calculate the relation-aware score s_i^{rel} for every fragment using Equation 10. Based on these fragments along with relation-aware scores, we select relevant fragments as retrieved context:

$$\begin{aligned} \mathcal{T}_{ret}^{rel} &= c_{r_1} \oplus c_{r_2} \oplus ... \oplus c_{r_K}, \\ r_1, r_2, ..., r_K &= arg\,TopK_i \ \ s_i^{rel}, \end{aligned} \tag{11}$$

where $\oplus$ represents operation of concatenating two text fragments, r_* is the indexes of retrieved fragments. The final response is generated y with LLM as follows:

$$\hat{y} = LLM(x, \mathcal{T}_{ret}^{rel}). \tag{12}$$

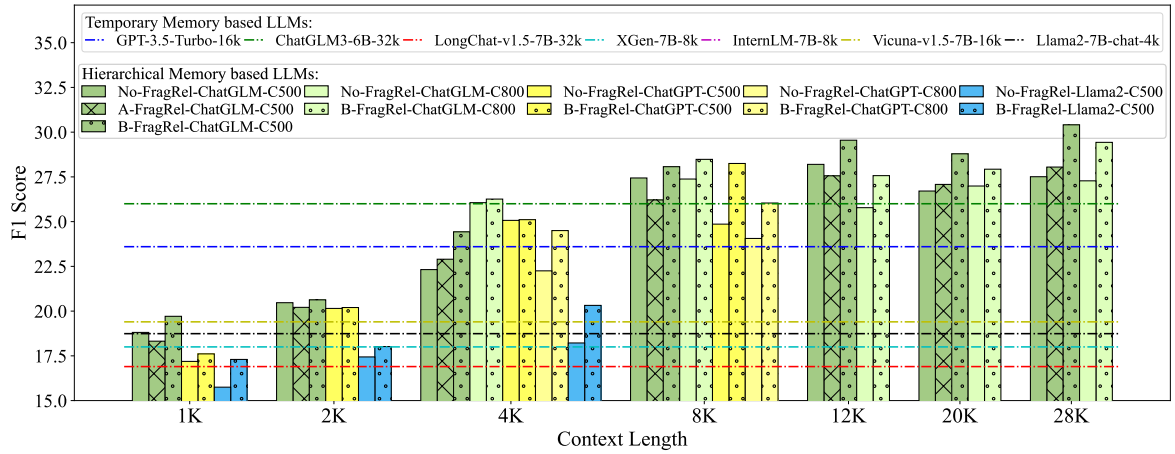


Figure 3: Performance comparison on NarrativeQA (Kočiskỳ et al., 2018). The horizontal lines and columnas represent Temporary Memory based LLMs and Hierarchical Memory based LLMs respectively. The Hierarchical Memory based LLMs contain 3 categories: no relation incorporation, semantic relation incorporation and context structure relation incorporation, denoted as "No-FragRel-*", "A-FragRel-*", and "B-FragRel-*" respectively.

4 Experiment

We evaluate the proposed *Fragment-connected HieraMem-LLM* on three long text understanding tasks: long story understanding (Section. 4.1), repository-level code generation (Section. 4.2), and memory-enhanced chatting agent (Section. 4.3).

4.1 Long Story Understanding

4.1.1 Setup

Dataset. NarrativeQA (Kočiskỳ et al., 2018) is a challenging story comprehension benchmark, consisting of human-written question-answer paris based on long (average 18K words) story books or movie scrips. LLMs are required to understand the long-term relations in the lengthy stories to answer these questions. We employ the 200 extracted testing samples in LongBench (Bai et al., 2023).

Metrics. Following LongBench (Bai et al., 2023), we assess the generated response with the F1 Score, a widely used metric in question-answering tasks.

Baselines. We classify baselines into 2 categories: Temporary Memory based LLMs (TempMem LLMs) and Hierarchical Memory based LLMs (HieraMem LLMs). **(a)** TempMem-LLMs: We directly compare the results shown in LongBench (Bai et al., 2023), covering LLMs with extensive parameter amount and context length (including the context length expanded LLMs). When the input text exceeds LLM's context length, the content in the middle position of the text is truncated. **(b)** HieraMem-LLMs: We experiment with different context lengths (1K, 2K, ..., 20K, 28K words), and different fragments lengths (500 and 800 words per fragment). The embeddings are calculated using the pre-trained Contriever (Izacard et al., 2021). The base LLMs includes Llama2-7B-4K (Touvron et al., 2023), ChatGLM3-6B-32K (Du et al., 2022) and GPT-3.5-Turbo-16K.

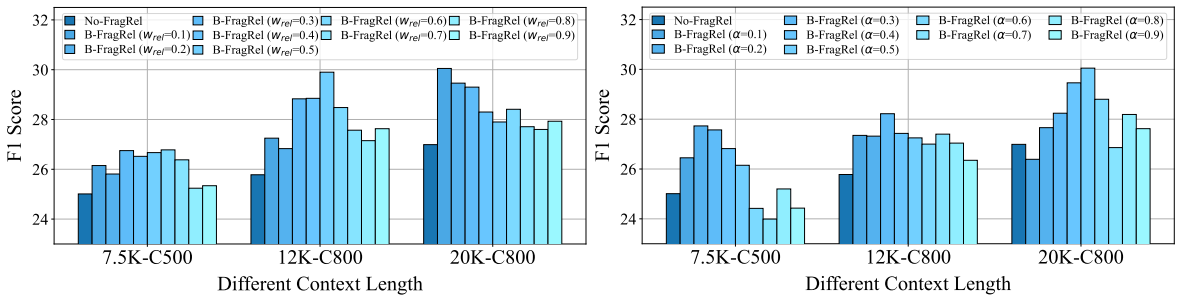
Relation Integration Details. We calculate the fragment relations using semantic relation (Equation. 6) and context structure relevance (Equation. 7), denoted as "A-FragRel" and "B-FragRel" respectively. Except for specific statements, we set $w_{rel} = 0.3$ and $\alpha = 0.5$.

4.1.2 Result

TempMem *vs.* HieraMem. According to Figure. 3, the Hierarchical Memory based LLMs (the columns) could outperform Temporary Memory based LLMs (the horizontal lines), especially on large context windows (more than 8K tokens). This is consistent with the conclusion that retrieval augmentation could help improve long context LLM in previous works (Xu et al., 2023).

Benefits of Fragment-level Relations. As shown in Figure. 3, across all base LLMs and context length, the structural relation augmented HieraMem-LLMs (noted as B-FragRel-*) consistently outperforms the counterpart HieraMem-LLMs (noted as No-FragRel-*). The performance enhancement is especially noticeable under enough long context length. This indicates that the introduction of fragment-level correlations effectively alleviates the deficiencies of HieraMem-LLM in terms of external memory management.

Semantic Relation *vs.* Structure Relation. Figure. 3 compares semantic relation ("A-FragRel-*") and context structure relation ("B-FragRel-*") on



(a) Varied w^{rel} (b) Varied α

Figure 4: Performance improvement using different edge weight w^{rel} and relation weight α.

fragment length 500. The semantic relation offers a slight enhancement under enough long context, while the context structure relation provides consistent and significant improvement across various context lengths. We posit this is due to that the embedding based retrieval has implicitly considered the semantic association between segments.

Different Relation Parameters. Figure. 4 presents the performance with varied relation parameters w_{rel} and α on different context lengths and fragment lengths. Although the optimal values of relations parameters (w_{rel} and α) for different setups (including fragment length, context types, and context length constraint *etc.*) are difficult to ascertain, most empirical values ($\alpha \in [0.2, 0.5], w_{rel} \in [0.1, 0.7]$) can lead to performance enhancement.

4.2 Repository-level Code Generation

4.2.1 Setup

Dataset. We conduct the code generation experiment on RepoEval (Zhang et al., 2023a), a benchmark constructed using the latest repositories source from GitHub. Specifically, two code completion tasks are considered: **(a)** *Line Completion*: completing random code lines, **(b)** *Api Invocation Completion*: completing random code lines that invoke in-repository apis. Both tasks contain 1600 test samples across 8 repositories.

Metrics. Following previous works (Zhang et al., 2023a; Lu et al., 2021, 2022), we evaluate the code generation performance using two metrics: *Exact Match* (EM Score) and *Edit Similarity* (ES Score). EM score evaluates how many completions are exactly the same to real code. ES score represents the similarity between the generated and real code.

Baselines. We employ Codellama-34b (Roziere et al., 2023) with 4K context length as the base LLMs, and the same prompt formats as RepoCoder (Zhang et al., 2023a). To extensively evaluate the integrated relations, we consider not only the single step Vanilla Retriever but also the iterative retrieval method RepoCoder (Zhang et al., 2023a). Noted that we report the result of oracle iterative retrieval, *i.e.* the upper bound of performance during the iterative retrieval procedure. Section. B.1 presents more implementation detail.

Method	EM Score	ES Score
Single Step Retrieval		
Vanilla Retriever	46.31	66.26
Vanilla Retriever + FragRel	**48.25**	**67.05**
Iterative Retrieval		
RepoCoder (Zhang et al., 2023a)	49.13	68.39
RepoCoder + FragRel	**50.44**	**68.50**

Table 1: Performance evaluation on line completion task of RepoEval (Zhang et al., 2023a) using Codellama-34b (Roziere et al., 2023).

Method	EM Score	ES Score
Single Step Retrieval		
Vanilla Retriever	40.00	66.32
Vanilla Retriever + FragRel	**40.94**	**66.39**
Iterative Retrieval		
RepoCoder (Zhang et al., 2023a)	41.81	68.31
RepoCoder + FragRel	**43.00**	**69.07**

Table 2: Performance evaluation on api invocation completion task of RepoEval (Zhang et al., 2023a) using Codellama-34b (Roziere et al., 2023).

Relation Integration Details. In this experiment, we use the Code Structure Relation shown in Equation. 8 , and we set relation weight $\alpha = 0.5$.

4.2.2 Result

Table 1, 2 present the results of the line completion task and api invocation completion task, respectively. On the two tasks, the integrated relation consistently improves the performance of both the single step retrieval inference framework and the iterative retrieval inference framework. This empirically demonstrates that fragment-level relations are greatly helpful in long code scenarios.

Method	Auto-rated Score by GPT4-4K (1-100)			
	Retrospection	Continuation	Conjunction	Average
ChatGPT-2K	52.11	55.33	48.22	51.89
MPC-ChatGPT (Lee et al., 2023)	53.00	61.22	49.33	54.52
MemoryBank-ChatGPT (Zhong et al., 2023)	23.39	55.28	48.67	42.44
MemoChat-ChatGPT (Lu et al., 2023)	66.28	73.50	72.50	70.76
MemRetrieval-ChatGPT	81.17	74.56	69.39	75.04
MemRetrieval-ChatGPT + FragRel	**81.56**	**77.89**	**82.17**	**80.54**

Table 3: Performance comparison on MTBench+ (Lu et al., 2023).

4.3 Memory-enhanced Chatbot

4.3.1 Setup

Dataset. We perform the long-term chatting experiment on MTBench+ (Lu et al., 2023). Every chatting stream consists of $12 \sim 15$ turns dialogs, covering topics such as "STEM exams", and "literary writing". At the end of every dialog, a challenging question is added by the experts. The questions could be classified into 3 categories: **(a)** "Retrospection" requires the model to respond content mentioned previously, **(b)** "Continuation" requires the model to finish a further task about talked topics, **(c)** "Conjunction" requires answering questions involving multiple topics existing in the dialog. There are 54 test samples, 18 for every type.

Metrics. Following the origin benchmark (Lu et al., 2023), we assess the generated response using LLM-as-a-judge method (Zheng et al., 2023), where GPT4 is instructed to check the faithfulness of the model response and produces a $1 \sim 100$ integer score. We utilize exactly the same testing setup (including prompt format, GPT4 version, hyperparameters, *etc.*) as (Lu et al., 2023).

Baselines. We consider the ChatGPT-based baselines reported in MemoChat (Lu et al., 2023), including various external memory enhanced frameworks (Zhong et al., 2023; Lee et al., 2023). In addition, we introduce a dense retrieval augmentation baseline, named MemRetrieval-ChatGPT. Following previous work (Lu et al., 2023), we constraint the temporary context length as 2K tokens. Section. B.2 presents more implementation detail.

Relation Integration Details. Based on MemRetrieval-ChatGPT, we integrate the context structure relations defined in Equation. 7. We set $w_{rel} = 0.8$ and $\alpha = 0.5$ in our experiments.

4.3.2 Result

Inference Expense Comparison. Approximately, MemoChat (Lu et al., 2023) consumes about 1M input tokens and 170K output tokens. MemRetrieval (+FragRel) costs about 680K input tokens and 65K output tokens. The introduced dense retrieval framework relatively reduces about 32% input tokens and 62% output tokens cost.

Performance Comparison. Table 3 presents the experimental results. Utilizing the same 2K temporary context length, the introduced MemRetrieval achieves comparable performance to MemoChat (Lu et al., 2023). Our fragment relations augmented methods consistently outperform all baseline, validating the effectiveness of incorporating fragments relations on the long-term chatting task.

5 Conclusion

This work focuses on the isolated fragment assessment issue in current external memory retrieval methods and proposes a fresh relation-embedded fragment assessment criteria. Experiments across multiple long text processing tasks substantiate the advantage of the proposed method. We hope our findings provide inspiration for future explorations about External Memory enhanced LLMs.

6 Limitation

Despite the fragment-level relations significantly improve the external memory management, current framework still suffers following limitations: (1) The best value for relation parameters (including w_{rel} and α) varies for different text types, fragment lengths, and relation categories. We can only empirically select a relatively good, but not the best parameter value. (2) The relation definitions shown in Equation. 6, 7, 8 are empirically defined, thus they have limited generalizability. (3) The relation incorporation method shown in Section. 3.3 is only applicable upon dense similarity based retrievers while overlooking other retrieval methods.

References

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2023. Self-rag: Learning to retrieve, generate, and critique through self-reflection. *arXiv preprint arXiv:2310.11511*.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2023. Longbench: A bilingual, multitask benchmark for long context understanding. *arXiv preprint arXiv:2308.14508*.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, et al. 2022. Improving language models by retrieving from trillions of tokens. In *International conference on machine learning*, pages 2206–2240. PMLR.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. 2019. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*.

Jiayu Ding, Shuming Ma, Li Dong, Xingxing Zhang, Shaohan Huang, Wenhui Wang, Nanning Zheng, and Furu Wei. 2023. Longnet: Scaling transformers to 1,000,000,000 tokens. *arXiv preprint arXiv:2307.02486*.

Zhengxiao Du, Yujie Qian, Xiao Liu, Ming Ding, Jiezhong Qiu, Zhilin Yang, and Jie Tang. 2022. Glm: General language model pretraining with autoregressive blank infilling. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 320–335.

Zhangyin Feng, Xiaocheng Feng, Dezhi Zhao, Maojin Yang, and Bing Qin. 2023. Retrieval-generation synergy augmented large language models. *arXiv preprint arXiv:2310.05149*.

Luyu Gao, Xueguang Ma, Jimmy Lin, and Jamie Callan. 2022. Precise zero-shot dense retrieval without relevance labels. *arXiv preprint arXiv:2212.10496*.

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

Chi Han, Qifan Wang, Wenhan Xiong, Yu Chen, Heng Ji, and Sinong Wang. 2023. Lm-infinite: Simple on-the-fly length generalization for large language models. *arXiv preprint arXiv:2308.16137*.

Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. 2021. Unsupervised dense information retrieval with contrastive learning.

Huiqiang Jiang, Qianhui Wu, Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2023a. Longllmlingua: Accelerating and enhancing llms in long context scenarios via prompt compression. *arXiv preprint arXiv:2310.06839*.

Zhengbao Jiang, Frank F Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023b. Active retrieval augmented generation. *arXiv preprint arXiv:2305.06983*.

Tomáš Kočiskỳ, Jonathan Schwarz, Phil Blunsom, Chris Dyer, Karl Moritz Hermann, Gábor Melis, and Edward Grefenstette. 2018. The narrativeqa reading comprehension challenge. *Transactions of the Association for Computational Linguistics*, 6:317–328.

Tian Lan, Deng Cai, Yan Wang, Heyan Huang, and Xian-Ling Mao. 2023. Copy is all you need. In *The Eleventh International Conference on Learning Representations*.

Gibbeum Lee, Volker Hartmann, Jongho Park, Dimitris Papailiopoulos, and Kangwook Lee. 2023. Prompted llms as chatbot modules for long open-domain conversation.

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

Daliang Li, Ankit Singh Rawat, Manzil Zaheer, Xin Wang, Michal Lukasik, Andreas Veit, Felix Yu, and Sanjiv Kumar. 2022. Large language models with controllable working memory. *arXiv preprint arXiv:2211.05110*.

Yucheng Li. 2023. Unlocking context constraints of llms: Enhancing context efficiency of llms with self-information-based content filtering. *arXiv preprint arXiv:2304.12102*.

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

Junru Lu, Siyu An, Mingbao Lin, Gabriele Pergola, Yulan He, Di Yin, Xing Sun, and Yunsheng Wu. 2023. Memochat: Tuning llms to use memos for consistent long-range open-domain conversation.

Shuai Lu, Nan Duan, Hojae Han, Daya Guo, Seungwon Hwang, and Alexey Svyatkovskiy. 2022. Reacc: A retrieval-augmented code completion framework. *arXiv preprint arXiv:2203.07722*.

Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin B. Clement,

Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, Ming Gong, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie Liu. 2021. Codexglue: A machine learning benchmark dataset for code understanding and generation. *CoRR*, abs/2102.04664.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744.

Charles Packer, Vivian Fang, Shishir G Patil, Kevin Lin, Sarah Wooders, and Joseph E Gonzalez. 2023. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*.

Ori Ram, Yoav Levine, Itay Dalmedigos, Dor Muhlgay, Amnon Shashua, Kevin Leyton-Brown, and Yoav Shoham. 2023. In-context retrieval-augmented language models. *arXiv preprint arXiv:2302.00083*.

Stephen Robertson and Hugo Zaragoza. 2009. The probabilistic relevance framework: Bm25 and beyond. *Found. Trends Inf. Retr.*, 3(4):333–389.

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

Jon Saad-Falcon, Joe Barrow, Alexa Siu, Ani Nenkova, Ryan A Rossi, and Franck Dernoncourt. 2023. Pdftriage: Question answering over long, structured documents. *arXiv preprint arXiv:2309.08872*.

Dale Schuurmans. 2023. Memory augmented large language models are computationally universal. *arXiv preprint arXiv:2301.04589*.

Zhihong Shao, Yeyun Gong, Yelong Shen, Minlie Huang, Nan Duan, and Weizhu Chen. 2023. Enhancing retrieval-augmented large language models with iterative retrieval-generation synergy. *arXiv preprint arXiv:2305.15294*.

Hongjin Su, Weijia Shi, Jungo Kasai, Yizhong Wang, Yushi Hu, Mari Ostendorf, Wen-tau Yih, Noah A Smith, Luke Zettlemoyer, and Tao Yu. 2022. One embedder, any task: Instruction-finetuned text embeddings. *arXiv preprint arXiv:2212.09741*.

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

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. *arXiv preprint arXiv:2212.10509*.

Boxin Wang, Wei Ping, Lawrence McAfee, Peng Xu, Bo Li, Mohammad Shoeybi, and Bryan Catanzaro. 2023a. Instructretro: Instruction tuning post retrieval-augmented pretraining. *arXiv preprint arXiv:2310.07713*.

Weizhi Wang, Li Dong, Hao Cheng, Xiaodong Liu, Xifeng Yan, Jianfeng Gao, and Furu Wei. 2023b. Augmenting language models with long-term memory. *arXiv preprint arXiv:2306.07174*.

Yuhuai Wu, Markus N Rabe, DeLesley Hutchins, and Christian Szegedy. 2022. Memorizing transformers. *arXiv preprint arXiv:2203.08913*.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2023. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*.

Wenhan Xiong, Jingyu Liu, Igor Molybog, Hejia Zhang, Prajjwal Bhargava, Rui Hou, Louis Martin, Rashi Rungta, Karthik Abinav Sankararaman, Barlas Oguz, et al. 2023. Effective long-context scaling of foundation models. *arXiv preprint arXiv:2309.16039*.

Peng Xu, Wei Ping, Xianchao Wu, Lawrence McAfee, Chen Zhu, Zihan Liu, Sandeep Subramanian, Evelina Bakhturina, Mohammad Shoeybi, and Bryan Catanzaro. 2023. Retrieval meets long context large language models. *arXiv preprint arXiv:2310.03025*.

Shi-Qi Yan, Jia-Chen Gu, Yun Zhu, and Zhen-Hua Ling. 2024. Corrective retrieval augmented generation. *arXiv preprint arXiv:2401.15884*.

Fengji Zhang, Bei Chen, Yue Zhang, Jin Liu, Daoguang Zan, Yi Mao, Jian-Guang Lou, and Weizhu Chen. 2023a. Repocoder: Repository-level code completion through iterative retrieval and generation. *arXiv preprint arXiv:2303.12570*.

Peitian Zhang, Shitao Xiao, Zheng Liu, Zhicheng Dou, and Jian-Yun Nie. 2023b. Retrieve anything to augment large language models. *arXiv preprint arXiv:2310.07554*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *arXiv preprint arXiv:2306.05685*.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, and Yanlin Wang. 2023. Memorybank: Enhancing large language models with long-term memory. *arXiv preprint arXiv:2305.10250*.

A Relation Calculation Details

A.1 Code Relation Calculation

Code Repository Graph $\mathcal{G}$ Construction. The code repository graph is created with the following steps: (1) For a specific code repository, we first construct the code syntax tree for every code source file using tree-sitter [1] (edge weight is set as 0.5). (2) In addition, we construct the file directory tree where every node corresponds to a file or directory in the repository (edge weight is set as 0.3). (3) Next, we connect the root node of every code syntax tree with its counterpart file node on the file directory tree (edge weight is set as 1.0), obtaining the code repository parsing tree. (4) Finally, we create node connections (edge weight is set as 0.8) between every function invoking node and its corresponding function definition node (including both in-file invoking and cross-file invoking).

Distance calculation. On code repository graph $\mathcal{G}$, the edge weight is set according to the edge types (a connection between files, a connection between in-file code syntax node, *etc.*), and constrained in $[0, 1]$.

The length of a path through multiple nodes is defined as the product of the weights of all edges on the path, thus a longer path will have a smaller weight. The relation strength $Dis(g_k^{c_i}, g_l^{c_i})$ of the relationship between two nodes is defined as the weight of maximum weight path between the two nodes $g_k^{c_i}$ and $g_l^{c_i}$). We calculate $Dis(\cdot, \cdot)$ via the Dijkstra algorithm in our experiment.

B Framework Implementation Details

B.1 Code Completion Framework Details

Retrieval. Every fragment consists of exactly S_w lines of code and adjacent have S_s overlapped lines of code. We set $S_w = 20$ and $S_s = 10$ same as Zhang et al. (2023a). Following RepoCoder (Zhang et al., 2023a), the BM25 (Robertson and Zaragoza, 2009) is used for retrieval. The fragments within Top 10 similarity scores to the completed code context are taken as retrieved results every time.

LLM Inference. We employ Codellama-34b (Roziere et al., 2023) with 4K context length as the base LLMs in our experiment, and exactly the same prompt formats as used in RepoCoder (Zhang et al., 2023a).

[1] https://tree-sitter.github.io

B.2 Memory-enhanced Chatbot Details

Retrieval In MemRetrieval-ChatGPT (+FragRel), every fragment consists of exactly one turn of the dialog. The text embedding is calculated using the text-embedding-ada-002 model. Every time we load related historical fragments within Top 8 embedding similarity to the recent dialog (last turn of dialog and latest user prompt). The retrieved fragments are reordered according to their chatting time.

LLM Inference. Same as MemoChat (Lu et al., 2023), we use the GPT-3.5-Turbo as the base LLM for inference, and the context length is constrained to 2K tokens. In the initial rounds of conversation, all historical records are directly inputted into the model. Once the length of the historical record exceeds 1K tokens or the conversation rounds surpass 10, the historical chat content will be stored in external memory. Subsequently, each conversation will load relevant fragments to the recent chatting context from the external memory.