PECAN: LLM-Guided Dynamic Progress Control with Attention-Guided Hierarchical Weighted Graph for Long-Document QA

Anonymous ACL submission

Abstract

Long-document QA presents challenges with large-scale text and long-distance dependencies. Recent advances in Large Language Models (LLMs) enable entire documents to be processed in a single pass. However, their computational cost is significantly high. Retrieval-Augmented Generation (RAG) methods split text into smaller chunks, but they often yield inferior results and may lose global context. Recent approaches that integrate LLMs into RAG via iterative summarization either underutilize LLM capabilities or still incur high computational costs. In this paper, we combine the high accuracy of LLMs with the efficiency of RAG and propose LLM-Guided Dynamic Progress Control with Attention-Based Hierarchical Weighted Graph (PECAN). Our method introduces two key improvements: (1) LLM-Guided Dynamic Progress Control: We leverage LLMs to dynamically control the retrieval process, adjusting the amount of retrieved information based on different queries to achieve a better balance of effectiveness and efficiency. (2) Attention-Guided Retrieval: We propose a novel retrieval method that constructs a hierarchical graph where edges are derived by LLM attention weights. Experimental results demonstrate that PECAN achieves LLM-level performance while maintaining computational complexity comparable to that of RAG methods on two single-document and two multi-document QA datasets. [1]

1 Introduction

Long-document Question Answering (QA) poses several challenges, including handling large-scale texts, uneven information distribution, and long-distance dependencies. While Large Language Models (LLMs) (Dubey et al., 2024; Yang et al., 2024; Team et al., 2024; OpenAI et al., 2024) can process entire documents after undergoing a context extension training stage, this approach is computationally intensive and inefficient, as queries often target only small segments of the text. In contrast, Retrieval-Augmented Generation (RAG) (Tay et al., 2022; Santhanam et al., 2022; Lin et al., 2023) mitigate this by segmenting long documents into chunks and retrieving the most relevant segments. However, studies like LongBench (Bai et al., 2024b) have demonstrated that the performance of RAG methods is often inferior to directly feeding the full document into LLMs (Zhang et al., 2023; Nair et al., 2023; Newman et al., 2023). RAG methods may suffer from incomplete retrieval, lack of global context, and struggle to capture long-distance dependencies. Recent methods, such as MeMWalker (Chen et al., 2023a) and RAPTOR (Sarthi et al., 2024) employ LLMs to iteratively summarize text into a tree. These methods use concise summaries generated by LLM to capture the global context while leveraging RAG to retrieve detailed information. However, MeMWalker still incurs a high computational cost since it repeatedly relies on the LLM to determine retrieval strategies, while RAPTOR only uses the LLM for summarization and heavily depends on traditional RAG.

In this paper, we propose a new method, LLM-Guided Dynamic Progress Control with Attention-Based Hierarchical Weighted Graph (**PECAN**), combining LLM accuracy and RAG efficiency. Unlike existing approaches, our method dynamically structures and searches information using an **Attention-Guided Hierarchical Graph**. Our approach comprises two stages: *Attention Graph Construction* and *Dynamic Graph Search*. In the *Attention Graph Construction* stage, we prompt an LLM to generate multiple **Information Points (IPs)**, each typically focusing on a single or a few events. These IPs form a Hierarchical Weighted Directed Acyclic Graph (HWDAG), where LLM attention weights between the generated nodes and the input nodes define relationships between lower- and higher-level nodes. This structure efficiently consolidate and summarize information about the same event from various sources, and enabling efficient event

[1] The source code will be released upon paper acceptance.

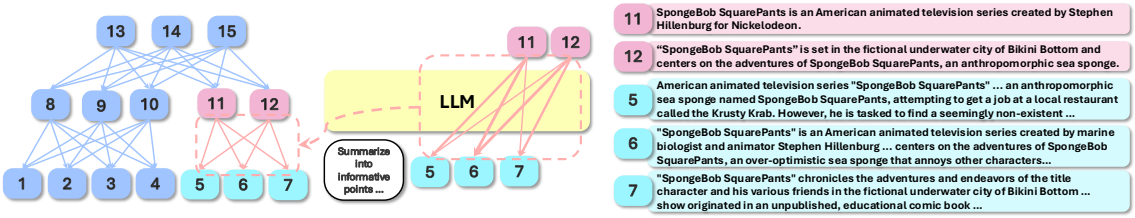

Figure 1: Overview of the attention-guided many-to-many summarization process for constructing the Hierarchical Weighted Directed Acyclic Graph (HWDAG). Each node represents an **Information Point (IP)**, typically summarizing one or a few events, and can have multiple predecessors and successors. The edge weights, derived from the LLM's attention during summarization, indicate the strength of relationships and the flow of information from lower-level nodes to higher-level nodes. The thickness of the red line on the right indicates the magnitude of the attention weights. For example, node #11 is more closely related to nodes #5 and #6, while node #12 is more associated with nodes #6 and #7. In this instance, the LLM extracts two events from three text chunks, with node #6 referencing both events. (For brevity, long text are truncated.)

identification, as illustrated in Figure 1.

Building on this graph, during the *Dynamic Graph Search* stage, instead of retrieving fixed-size text chunks, we use LLM attention to identify relevant high-level summaries and trace back to detailed information. This method effectively utilizes the LLM's knowledge, capturing both the generated summarization and the dynamic flow of information between high-level summaries and their corresponding detailed content within the graph. In addition, previous RAG methods retrieve a static number of top-ranked text chunks, which fails to account for the fact that different queries may require varying amounts of information. This fixed setting might lead to computational waste for simple queries and insufficient retrieval for more complex ones. We introduce a novel method called **Dynamic Progress Control**, which allows the LLM to adaptively determine the amount of information needed per query. This approach essentially reallocates computational resources based on the LLM's knowledge to achieve a better balance of effectiveness and efficiency. Meanwhile, previous inputs are KV-cached so that new documents can be appended without reprocessing the entire input. As a result, our method can effectively handle queries that require varying amounts of information, particularly when the necessary details are distributed across different parts of the graph. This avoids additional computational overhead and resolves the challenge of determining the optimal number of chunks to retrieve.

In experiments, PECAN achieves a good balance between effectiveness and efficiency, attaining LLM-level performance while keeping computational costs on par with RAG methods. In summary, our contributions are:

- We leverage LLMs to dynamically control the retrieval process, adjusting the amount of retrieved information per query without incurring additional computational overhead, leading to a more balanced effectiveness and efficiency.
- We propose a novel attention-guided retrieval paradigm that constructs a many-to-many graph using LLM attention weights. In this graph, edges are derived from attention weights, and retrieval is guided accordingly. Each node represents an Information Point (IP) focusing on one or a few events, allowing for structured event tracking rather than simple text chunk retrieval.
- Our empirical results show that PECAN achieves LLM-level accuracy while maintaining computational efficiency of traditional RAG methods.

2 Related Work

Long-Context Language Models Recent long-context language models have focused on overcoming fixed context window limitations, primarily through positional interpolation and training on full-length texts. Chen et al. (2023b); Peng et al. (2024); Fu et al. (2024) fine-tuned models on longer inputs and extended Rotary Position Embedding (RoPE; Su et al., 2023) for extended contexts. LongRoPE (Ding et al., 2024) performs direct extrapolation by rescaling RoPE with varied interpolation. LongAlign (Bai et al., 2024a) constructs a long-context dataset, adopting packing and sorted batching strategies. PoSE (Zhu et al., 2024) manipulates position indices by skipping bias terms. SkipAlign (Wu et al., 2024) synthesizes long-range dependencies from the aspect of position indices. Infini-Transformer (Munkhdalai et al., 2024) handles infinitely long inputs using compressive memory, masked local attention, and long-term attention mechanisms. Our primary focus is on effectively leveraging LLM capabilities, and the LLMs em-

ployed in these techniques can serve as the base models in our framework.

RAG Traditional retrieval techniques, such as TF-IDF (Jones, 1972) and BM25 (Robertson et al., 1995; Robertson and Zaragoza, 2009), rely on word-term matching. Subsequently, deep learning–based retrieval methods quickly gained popularity (Guu et al., 2020; Min et al., 2021; Izacard et al., 2022; Izacard and Grave, 2021; Wang et al., 2023b). Among these, DPR (Karpukhin et al., 2020) encodes queries and documents as dense embeddings. ColBERT (Khattab and Zaharia, 2020; Santhanam et al., 2022) produces multi-vector representations. DHR (Liu et al., 2021) leverages both document-level and passage-level semantics. CPT-text (Neelakantan et al., 2022) utilizes contrastive pre-training on unsupervised data. NCI (Wang et al., 2022) directly generates relevant document identifiers. RETRO (Borgeaud et al., 2022; Wang et al., 2023a) conditions on document chunks based on local similarity. HHR (Arivazhagan et al., 2023) combines sparse and dense retrieval methods. Dragon (Lin et al., 2023) uses contrastive learning and data augmentation to train a model, achieving state-of-the-art retrieval performance.

LLM and RAG Additionally, with the rise of LLMs, several studies have explored combining LLMs with RAG. GENREAD (Yu et al., 2023) prompts LLMs to generate contextual documents. RECITE (Sun et al., 2023) retrieves from the LLM's internal memory. KGP (Wang et al., 2023c) builds a knowledge graph with the LLM navigating. Recently, MeMWalker (Chen et al., 2023a) constructs tree-based summaries and uses LLMs to navigate. RAPTOR (Sarthi et al., 2024) creates tree-based summaries and leverages embedding similarities to select the most relevant nodes at each level for retrieval. In contrast, our approach makes greater use of LLM knowledge by employing attention mechanisms to construct a graph and perform the search, allowing navigation along multiple paths and termination at any depth.

3 Methodology

Our method consists of two main steps: *Attention Graph Construction* and *Dynamic Graph Search*. In the *Attention Graph Construction* stage, illustrated in Figure 1, we utilize the LLM's attention weights to build a Hierarchical Weighted Directed Acyclic Graph (HWDAG) from documents. This is a one-time preprocessing step for each document, after which it can be reused for any query to that document. In the *Dynamic Graph Search* stage, as illustrated in Figure 2, we dynamically control the volume of retrieved nodes and perform a search guided by the LLM.

Problem Setup The input consists of two parts: a document and multiple queries. The document is processed only by the *Attention Graph Construction* stage, while the queries are handled by the *Dynamic Graph Search* stage. Initially, a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is constructed from a document, where $\mathcal{V}$ denotes the collection of nodes and $\mathcal{E}$ denotes the collection of edges. Each node $v_i^l \in \mathcal{V}$ contains the text of an Information Point (IP) and belongs to level l. Each edge $e_{i,j} \in \mathcal{E}$ indicates the relatedness between node v_i and node v_j, derived from LLM attention weights. Each v_i contains tokens $\{x_n^i\}_{n=1}^{N_i}$, where N_i denotes the number of tokens in node v_i. The *Dynamic Graph Search* stage then uses the graph $\mathcal{G}$ to generate a response for each query q.

3.1 Attention Graph Construction

A document is initially split into chunks of 300 tokens, which serve as the first-level nodes $\mathcal{V}_1$. For each level, we iteratively summarize nodes from $\mathcal{V}_l$ by batching them and feeding them into the LLM to obtain the higher-level nodes $\mathcal{V}_{l+1}$, as illustrated in Figure 1. Specifically, we select nodes for each batch by sequentially adding them until the total text content length exceeds a threshold s. These nodes are then input into the LLM. We prompt the LLM to generate IPs in the form of bullet points.[2] We found that this bullet-point prompt ensures that each point contains only a single or a few events and remaining easy for the LLM to process.

The attention from a higher-level node v_i^{l+1} to a lower-level node v_j^l is averaged and then normalized across attention weights between nodes, yielding the edge weight $e_{i,j}$ from node v_i to node v_j. The value of $e_{i,j}$ quantifies the extent to which v_i^{l+1} relies on or extracts information from v_j^l. We focus only on attention weights between high-level and low-level nodes, omitting those within the same node. This approach emphasizes long-distance semantic relationships.

The computational process is elaborated below: **(a) Extracting attention weights**. During summarization, LLM attention weights are captured and averaged across attention heads and layers. In practice, these weights are iteratively accumulated during each layer's inference, thereby limiting memory

[2]The LLM prompt used is shown in Appendix A.1, and an example of the generated IPs is shown in Appendix D.

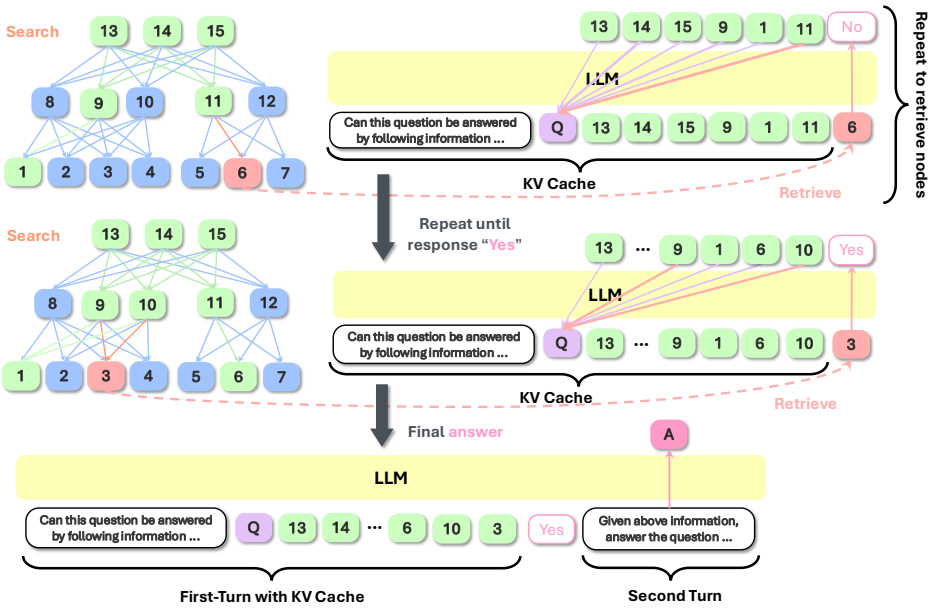

Figure 2: Overview of *Dynamic Graph Search*. At each iteration, a node is retrieved based on attention weights when searching the graph. The visited nodes are then fed into the LLM, prompting it to determine whether sufficient nodes have been gathered to answer the query. Due to KV caching, this process adds no additional computational cost. The search continues until the LLM indicates that enough relevant nodes have been retrieved, at which point the final answer is generated. This procedure dynamically adapts to the query, retrieving nodes flexibly across multiple graph paths and depths.

usage. This results in token-level attention e_{x^i,x^j} between tokens $x^i \in v_i^{l+1}$ and $x^j \in v_j^l$.

(b) Aggregating token-level attention to node-level. For each node v_j^l, the attention from all its tokens to token $\{e_{x^i,x_n^j}\}_{n=1}^{N_j}$ is averaged to to obtain the token-to-node attention e_{x^i,v_j^l}. Similarly, for node v_i^{l+1}, token-to-node attention $\{e_{x_n^i,v_j^l}\}_{n=1}^{N_i}$ is averaged to obtain the final node-level edge weight $e_{v_i^{l+1},v_j^l}$, abbreviated as $e_{i,j}$.

(c) Normalization. Finally, the edges are normalized as $e_{i,j} = \frac{e_{i,j}}{\sum_j e_{i,j}}$, ensuring that $\sum_j e_{i,j} = 1$. If no direct attention exists between two nodes, the corresponding edge weight is set to zero.

3.2 Dynamic Graph Search

The dynamic graph search process comprises two components: *Dynamic Progress Control*, which provides dynamic LLM-guided control, and *Graph Search*, which executes the search process.

3.2.1 Dynamic Progress Control

The process begins by initializing a visited set, denoted as $\mathcal{S} \subset \mathcal{V}$, with the top-level nodes of $\mathcal{V}$, which is fed into an LLM. The LLM is prompted to determine whether the current set of visited nodes $\mathcal{S}$ is sufficient to answer a given query. The prompt asks the LLM, "*Can this question be answered by the following information?*", followed by the query and visited nodes $\mathcal{S}$.[3] If the response is "No", the search continues, and the next node is retrieved. The newly retrieved node is added to the visited set $\mathcal{S}$, and this process repeats until the LLM responds with "Yes." Throughout the search, all previous inputs, including the prompt, query, and visited nodes $\mathcal{S}$, are cached using KV caching, as illustrated in Figure 2.[4] This ensures that no additional computational resources are required. Once the LLM responds with "Yes", the second turn of the prompt will ask the LLM to answer the query and the final answer is obtained. This dynamic approach, termed **Dynamic Progress Control**, better allocates limited resources among different queries to achieve a better balance between effectiveness and efficiency (see Section 4.3 for analysis). In contrast, previous methods typically relied on a fixed amount of retrieved content, which led to either computational waste or insufficient information retrieval.

3.2.2 Graph Search

After identifying a set of IP nodes required to answer a query, the set of visited nodes $\mathcal{S} \subset \mathcal{V}$ is provided to the LLM using the prompt shown in Table A2. The attention weight between a visited node v_i and the query q, denoted as r_i, is generally extracted following the method described in

[3]The complete prompt is shown in Appendix A.2

[4]See Appendix B for details on the use of KV caching.

Section 3.1.[5] This weight indicates the degree of attention that node v_i gives to the query q.

Specifically, for a node v_i, we define its predecessors as $\mathcal{P}_i$. The intersection of $\mathcal{P}_i$ and the set of visited nodes $\mathcal{S}$ is then $\mathcal{Q}_i = \mathcal{P}_i \cap \mathcal{S}$. For each node $v_j \in \mathcal{Q}_i$, the retrieval score z_i for a node $v_i \notin \mathcal{S}$ is computed as:

$$z_i = \sum_{j \in \mathcal{Q}_i} r_j e_{j,i} \tag{1}$$

where $e_{j,i}$ represents the edge weight from node v_j to node v_i. This operation is performed for all successors of nodes in $\mathcal{S}$.

This process can also be carried out via matrix multiplication. We define the adjacency matrix $\boldsymbol{E} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$, where $\boldsymbol{E}_{i,j} = e_{i,j}$ and $|\mathcal{V}|$ denotes the number of nodes in $\mathcal{V}$. Next, we construct the score vector $\boldsymbol{r} \in \mathbb{R}^{|\mathcal{V}|}$ such that $\boldsymbol{r}_i = r_i$ if $v_i \in \mathcal{S}$, and $\boldsymbol{r}_i = 0$ otherwise. The final score vector $\boldsymbol{z} \in \mathbb{R}^{|\mathcal{V}|}$ is computed as $\boldsymbol{z} = \boldsymbol{E}^T \boldsymbol{r}$, where each entry of $\boldsymbol{z}$ represents the corresponding node's score. Finally, the embedding similarity is added to $\boldsymbol{z}$ as the final score, and the node with the highest final score is retrieved. Notably, PECAN without embedding similarity only shows a slight performance degradation, whereas using only embedding similarity results in a larger performance decline (see the ablation study in Section 4.2).

The key idea is that if a node (i.e., an Information Point) containing an event is strongly correlated with the query, then the details about that event are likely to be more useful in answering the query. We use r to represent the relevance of a high-level node to the query and e to represent the relevance of a lower-level node to its associated high-level node. A high e indicates that the lower-level node contains more detailed information about the same event, as illustrated in Figures 1 and A1. If a retrieved node provides details that are not relevant to the query, its r score will be low, preventing the search from continuing along that node. If multiple nodes are highly relevant to both the query and the same successor, that successor node will accumulate scores from these multiple predecessors, resulting in a higher overall score.

4 Experiments

Dataset We use two single-document QA and two multi-document QA datasets from LongBench (Bai et al., 2024b): **NarrativeQA** (Kočiský et al., 2018) is a single-doc QA dataset containing 1,567 stories, including full texts of books and movie transcripts. **Qasper** (Dasigi et al., 2021) is a single-doc QA dataset with 1,585 papers, designed to seek information present in the papers. **HotpotQA** (Yang et al., 2018) is a multi-doc QA dataset that contains 112,779 examples, focusing on multi-hop QA. **MuSiQue** (Trivedi et al., 2022) is a multi-doc QA dataset with 24,814 examples featuring 2-4 hop questions and six reasoning types. [6]

Metrics We use F1 and ROUGE-L (Lin, 2004) as evaluation metrics. The final scores are computed using the evaluation source code from LongBench (Bai et al., 2024b) and Hugging Face Evaluate[7]. We also measure the average query-dependent TFLOPs (Tera Floating Point Operations) consumed during search and inference for each query.

Baseline We employ the following baselines: **BM25** (Robertson et al., 1995; Robertson and Zaragoza, 2009) is a bag-of-words based retrieval method. **SBERT** (Reimers and Gurevych, 2019) is a dense retrieval method that employs dense embeddings. **Dragon** (Lin et al., 2023) uses contrastive learning and data augmentation to train a model, achieving state-of-the-art retrieval performance among eight RAG baselines. **LongLLMLingua** (Jiang et al., 2024) is a prompt compression method that introduces question-aware compression based on LLMLingua (Jiang et al., 2023). **MeMWalker** (Chen et al., 2023a) summarizes context into a tree and navigates it to search for relevant information guided by the LLM within a limited input window. **RAPTOR** (Sarthi et al., 2024) constructs a tree by recursively embedding, clustering, and summarizing chunks of text. It has two variants: "tree traversal" (**RAPTOR-TT**) retrieves nodes along a single path from the top of the tree to the bottom, and "collapsed tree" (**RAPTOR-CT**) flattens all nodes for standard RAG-based retrieval. **Llama-3.1-8B** (Dubey et al., 2024) expands the context window, enabling documents to be fed into the model, except for a few from NarrativeQA.

We use Llama-3.1-8B (Dubey et al., 2024) as the LLM for PECAN and all baselines by running their source code. For MeMWalker, since the source code was not released, we implemented it based on the description in the paper using Llama-3.1. For LongLLMLingua, which employs a smaller model to compress prompts for GPT-3.5, we used the Phi-2-2.7B model provided by LongLLMLingua for compression. For all methods, we adopt a zero-shot approach without Chain-of-Thought (CoT). We use SBERT (Reimers and Gurevych, 2019) as the retrieval model in PECAN. We set the length threshold

[5] See Appendix C for more details.

[6] See Appendix E for more statistics.

[7] https://github.com/huggingface/evaluate

Method	NarrativeQA				Qasper			
	F1	ROUGE-L	TFLOPs	Ratio	F1	ROUGE-L	TFLOPs	Ratio
BM25 Top-5	52.7	51.8	26.7	0.86x	41.0	39.6	26.3	0.39x
SBERT Top-5	36.5	35.8	26.8	0.86x	44.4	42.4	26.0	0.39x
Dragon Top-5	53.8	52.9	26.9	0.87x	43.0	41.4	24.5	0.36x
MeMWalker	11.2	9.8	353.8	11.41x	39.0	36.8	123.9	1.85x
RAPTOR-TT	40.6	39.8	20.3	0.65x	42.1	40.1	17.7	0.26x
RAPTOR-CT Top-5	48.6	47.8	17.9	0.58x	44.6	42.7	16.6	0.25x
LongLLMLingua	50.5	49.5	1789.4	57.72x	43.2	43.0	159.7	2.39x
BM25 Top-X	53.7	52.9	37.5	1.21x	47.0	45.1	69.3	1.04x
SBERT Top-X	39.5	38.8	37.5	1.21x	46.6	44.5	68.9	1.03x
Dragon Top-X	55.1	54.2	37.5	1.21x	46.9	44.8	67.0	1.00x
RAPTOR-CT Top-X	52.0	51.2	35.1	1.13x	46.9	44.7	67.3	1.01x
Llama-3.1-8B	53.7	52.6	3361.9	108.45x	49.4	47.6	92.5	1.38x
PECAN	**61.1**	**60.2**	31.0	1.00x	**49.7**	**47.9**	66.9	1.00x
Method	**HotpotQA**				**MuSiQue**			
	F1	ROUGE-L	TFLOPs	Ratio	F1	ROUGE-L	TFLOPs	Ratio
BM25 Top-5	40.8	40.9	22.9	1.43x	28.7	28.7	26.3	0.85x
SBERT Top-5	40.9	40.8	22.6	1.41x	30.7	30.8	26.1	0.84x
Dragon Top-5	39.7	39.6	23.3	1.46x	28.5	28.4	28.1	0.91x
MeMWalker	39.7	38.9	93.4	5.84x	24.0	23.5	175.7	5.69x
RAPTOR-TT	38.6	38.5	8.4	0.53x	29.3	29.3	12.6	0.41x
RAPTOR-CT Top-5	40.9	40.4	15.3	0.96x	31.5	31.5	16.1	0.52x
LongLLMLingua	43.4	**43.5**	43.6	2.73x	34.5	34.4	78.9	2.55x
BM25 Top-X	40.7	40.8	20.0	1.25x	31.8	31.7	35.6	1.15x
SBERT Top-X	40.8	40.7	19.6	1.23x	32.5	32.5	35.6	1.15x
Dragon Top-X	39.2	39.1	20.6	1.29x	30.2	30.1	38.0	1.23x
RAPTOR-CT Top-X	40.7	40.7	17.9	1.12x	35.4	35.2	32.2	1.04x
Llama-3.1-8B	41.3	41.2	23.7	1.48x	35.8	35.7	40.6	1.31x
PECAN	**43.5**	**43.5**	16.0	1.00x	**36.9**	**36.8**	30.9	1.00x

Table 1: F1 (%), ROUGE-L (%), and TFLOPs of baselines and PECAN on NarrativeQA, Qasper, HotpotQA, and MuSiQue. TFLOPs are calculated during query-dependent inference. "Ratio" represents the ratio of the baselines' TFLOPs to PECAN's TFLOPs. In addition to Top-5, we also include a Top-X setting to match the TFLOPs of PECAN for a fair comparison. For BM25, SBERT, and Dragon we used Top-7 (NarrativeQA), Top-14 (Qasper), Top-4 (HotpotQA), and Top-7 (MuSiQue); for RAPTOR-CT, we used Top-20, Top-42, Top-12, and Top-22, respectively.

s to 8K. Across all datasets and steps of PECAN, including graph construction and search, the input window is capped by s and can be processed using an NVIDIA A100 80G GPU. A summary graph example is illustrated in Appendix D.

4.1 Main Results

The main results are shown in Table 1. In addition to Top-5 retrieval, we also include a Top-X setting to match the TFLOPs of PECAN for a fair comparison. For MeMWalker, RAPTOR, and PECAN, summarization TFLOPs are not included as summarization is query-independent. It is worth noting that our *Dynamic Progress Control* can determine the appropriate number of nodes to retrieve in a single pass, whereas these methods require extensive hyperparameter searches to find the optimal number. For Llama-3.1 on NarrativeQA, we used 8 A100 80G GPUs, with CPU offloading, to handle the long documents. However, we could only process an input window of 100K tokens under these resource constraints, resulting in 22.3% of documents being truncated.

The performance of BM25, SBERT, and Dragon was similar, with Dragon showing an advantage on NarrativeQA. When comparing the Top-5 and Top-X results, retrieving more fragments generally leads to better performance. LongLLMLingua achieves better results than Llama-3.1 on HotpotQA, possibly because it reorders documents to place the most relevant content upfront, mitigating the lost-in-the-middle effect (Liu et al., 2024). However, for other datasets, the deletion of sentences and tokens in LongLLMLingua negatively impacts its performance. The small size difference between Phi-2 and Llama-3.1-8B may not reflect the intended use cases of LongLLMLingua, making efficiency comparisons challenging.

MeMWalker requires the LLM to generate correct responses and formats at every node, which increases computational complexity and raises the risk of navigation failures when the tree becomes large. This contributed to its poor performance on NarrativeQA. RAPTOR-TT shows relatively poor

Method	NarrativeQA		Qasper		HotpotQA		MuSiQue	
	F1	ROUGE-L	F1	ROUGE-L	F1	ROUGE-L	F1	ROUGE-L
PECAN	61.1	60.2	49.7	47.9	43.5	43.4	36.9	36.9
w/o Attention-Guided Retrieval	53.0	52.4	46.9	45.5	41.9	41.7	33.1	32.9
w/o Dynamic Progress Control	53.5	52.9	37.7	36.4	39.4	39.3	27.0	27.1
w/o IP-based Graph	51.4	50.8	47.3	45.5	40.9	40.7	32.3	32.3
w/o Embedding Similarity	59.5	58.7	48.0	46.2	42.9	42.8	35.9	35.8

Table 2: Ablation study of the four components of PECAN with F1 (%) and ROUGE-L (%) on NarrativeQA, Qasper, HotpotQA, and MuSiQue.

performance as it is constrained by a fixed retrieval path from the top to the bottom. RAPTOR-CT Top-X achieves high performance but still underperforms compared to PECAN at comparable TFLOPs, suggesting that its tree-based summarization is less efficient. Llama-3.1-8B excelled on most datasets, demonstrating its strong capability. However, for particularly long inputs in NarrativeQA, Llama-3.1 incurred 108.45x TFLOPs, which is significantly more computationally expensive than using PECAN.

PECAN achieves a good balance between effectiveness and efficiency, and with fewer TFLOPs, outperforming all baselines across all four datasets in terms of F1 and ROUGE-L. Even for Top-X setting, with fewer TFLOPs, PECAN results are still better than those of RAPTOR-CT. Compared to LongLLMLingua, our method achieved better performance with lower TFLOPs, although the differing application scenarios limit direct comparison. PECAN slightly outperforms Llama-3.1 on Qasper, HotpotQA, and MuSiQue with fewer TFLOPs, and significantly surpasses Llama-3.1 on NarrativeQA while incurring much lower computational cost on a single GPU, demonstrating its capability to handle extremely long documents. Since PECAN begins with query-related information, the most relevant content is presented first, mitigating the "lost-in-the-middle" effect. With IPs organized by events, PECAN can better track these events, especially as long-distance relationships tend to weaken.

4.2 Ablation Study

In this section, we study how each component contributes to performance, as shown in Table 2, and describe them below.

w/o Attention-Guided Retrieval We remove the use of attention and rely solely on embedding similarity for node search, similar to RAPTOR. Performance dropped across all datasets, with the most significant drop occurring in NarrativeQA, indicating that attention scores are particularly effective in retrieving hierarchical information from long texts.

w/o Dynamic Progress Control We conduct retrieval using a fixed number of nodes, corresponding to the average used by PECAN across the four datasets. Instead of employing *Dynamic Progress Control*, we retrieve this fixed number of nodes. For documents with many top-level nodes, we limit the initial number of visited nodes $\mathcal{S}$ to the preset value divided by the number of levels, with top-level nodes selected using embedding similarity as in RAPTOR to ensure sufficient exploration. We retain the same search mechanism so that unselected top-level nodes can still be retrieved via embedding similarity. Without *Dynamic Progress Control*, performance degraded across all datasets, even though the average number of nodes retrieved remained unchanged. See Section 4.3 for a deeper analysis.

w/o IP-based Graph We do not use the IP-based many-to-many graph. Instead, following RAPTOR, we adopt a many-to-one summarization that ultimately forms a tree. Other search approaches remain unchanged. All datasets experience a performance decline, with NarrativeQA being particularly affected. This suggests that for complex and lengthy inputs, IPs with many-to-many graph-based relationships are more effective at organizing information and tracking events.

w/o Embedding Similarity We remove embedding similarity from the final score z, relying entirely on attention weights to compute retrieval scores. Across all four datasets, performance shows a slight decrease, though this drop is much smaller than that observed when the attention-guided retrieval is removed, indicating that attention-guided search plays a more critical role.

4.3 Dynamic Retrieval Analysis

In this section, we analyze the number of nodes retrieved by *Dynamic Progress Control*. Since each node primarily contains an IP with only a small amount of text, PECAN can retrieve more nodes using the same TFLOPs. NarrativeQA consists of stories, where each IP can be described succinctly, whereas Qasper comprises scientific papers, with IPs that are more complex and require longer descriptions. Figure 3 presents the frequency distribution of nodes retrieved per query for NarrativeQA,

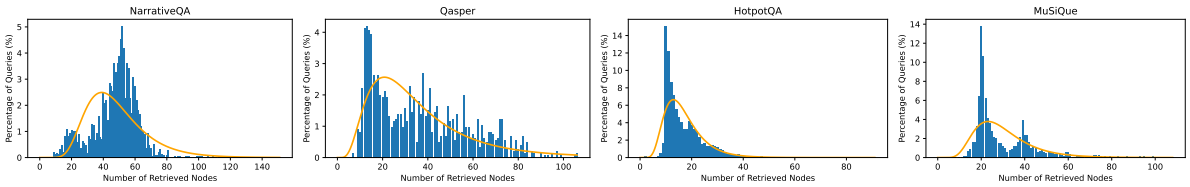

Figure 3: Frequency distribution of the number of nodes retrieved per query for NarrativeQA, Qasper, HotpotQA, and MuSiQue. The x-axis represents the number of nodes retrieved per query, while the y-axis indicates the percentage of queries retrieving that number of nodes. A log-normal distribution is fitted, as shown by yellow line.

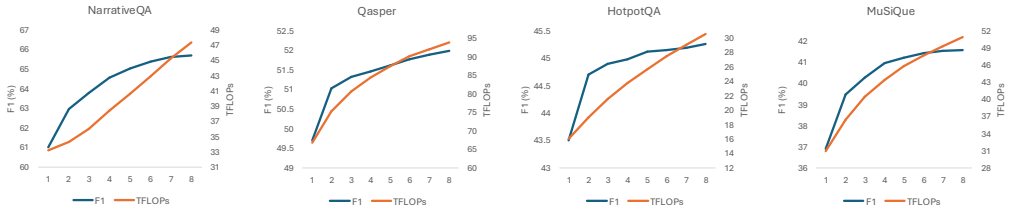

Figure 4: The F1 (%) and TFLOPs of PECAN on NarrativeQA, Qasper, HotpotQA, and MuSiQue with different stop patience p. The x-axis represents stop patience. The left y-axis shows the F1 (%) (blue line), and the right y-axis shows TFLOPs (orange line). As p increases from 1 to 8, both F1 and TFLOPs increase, but the increase in F1 slows when $p > 5$, while TFLOPs continue to rise.

Qasper, HotpotQA, and MuSiQue. As indicated by the log-normal yellow line, the distribution is generally right-skewed: most queries retrieve a moderate number of nodes, while a few require significantly more, forming a long tail. PECAN dynamically tailors the amount of retrieved information to accommodate this long tail—a scenario that previous methods cannot handle, as they typically retrieve a fixed number of node chunks. HotpotQA, in particular, exhibits a more standard right-skewed distribution. Qasper, which comprises 1,451 instances compared to HotpotQA's 7,405, shows a wider range of retrieved nodes (mostly within 80, versus mostly within 30 for HotpotQA), resulting in greater noise in Qasper. NarrativeQA displays a small peak at a low number of retrieved nodes, potentially corresponding to easier queries. This observation aligns with Kočiský et al. (2018)'s note that "a small number of questions and answers are shallow paraphrases of sentences in the full document." MuSiQue explicitly categorizes questions by the number of hops, with the majority being 2-hop questions and some requiring 3 or 4 hops, with fewer hops generally indicating a lower information requirement and vice versa. Figure 3 also shows two distinct peaks, an observation that aligns with MuSiQue's difficulty distribution and further demonstrates that PECAN can dynamically adapt to queries of varying complexity.

4.4 Effectiveness-Efficient Trade-off Study

In this section, we examine the trade-off between effectiveness and efficiency. Similar to the concept of early stopping patience, we introduce a stop patience hyper-parameter p, which means that the search terminates after the LLM responds with "Yes" p times. As shown in Figure 4, increasing p can further enhance performance beyond the results reported in Table 1, albeit at the cost of increased computational resources. When $p < 5$, the F1 score rises rapidly, and for $p > 5$, the improvement in F1 slows while the TFLOPs increase disproportionately. All four datasets exhibit a similar trend with respect to p, demonstrating that *Dynamic Graph Search* provides a good trade-off between effectiveness and efficiency. PECAN can adjust a single value of p across all datasets to achieve a similar effectiveness-efficiency trade-off, whereas previous methods require hyper-parameter searches on each dataset separately.

5 Conclusion

In this paper, we introduced PECAN, a novel retrieval method that incorporates two key innovations. The LLM-Guided Dynamic Progress Control adjusts the amount of information retrieved based on the query, achieving a better balance between effectiveness and efficiency. The Attention-Guided Retrieval constructs a many-to-many Hierarchical Weighted Directed Acyclic Graph using LLM attention weights to guide the search. Each node represents an Information Point that focuses on one or a few events, enabling the model to effectively track them. Empirical results demonstrate that PECAN achieves LLM-level performance while maintaining RAG-level computational complexity.

6 Limitations

PECAN primarily computes graph edges by averaging attention across all tokens, treating them as equally important. However, tokens may not actually be equally significant. Semantically irrelevant tokens (e.g., function words like "a," "an," "the," etc.) might introduce noise. Nonetheless, since we only retain the attention between nodes and omit the attention within nodes, this can partially mitigate the issue. Moreover, tokens in different texts may appear in similar proportions, so the added noise is less noticeable. Nevertheless, our experiments show that this averaging approach still yields good results. We leave a deeper exploration of this issue, the development of a more fine-grained attention averaging strategy, for future work. Similarly, we simply add the attention-based score and the embedding similarity score together, resulting in an equal weighting of both components. While this averaging approach also performs well in our experiments, future work could explore when and to what extent each score should be weighted differently to achieve a more refined strategy. In this paper, we focus on presenting the main framework of Dynamic Progress Control and Attention-Guided Retrieval, leaving further refinements for future work.

References

Manoj Ghuhan Arivazhagan, Lan Liu, Peng Qi, Xinchi Chen, William Yang Wang, and Zhiheng Huang. 2023. Hybrid hierarchical retrieval for open-domain question answering. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 10680–10689, Toronto, Canada. Association for Computational Linguistics.

Yushi Bai, Xin Lv, Jiajie Zhang, Yuze He, Ji Qi, Lei Hou, Jie Tang, Yuxiao Dong, and Juanzi Li. 2024a. Longalign: A recipe for long context alignment of large language models. *arXiv preprint arXiv:2401.18058*.

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. 2024b. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3119–3137, Bangkok, Thailand. Association for Computational Linguistics.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George van den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego de Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, Tom Hennigan, Saffron Huang, Loren Maggiore, Chris Jones, Albin Cassirer, Andy Brock, Michela Paganini, Geoffrey Irving, Oriol Vinyals, Simon Osindero, Karen Simonyan, Jack W. Rae, Erich Elsen, and Laurent Sifre. 2022. Improving language models by retrieving from trillions of tokens. *Preprint*, arXiv:2112.04426.

Howard Chen, Ramakanth Pasunuru, Jason Weston, and Asli Celikyilmaz. 2023a. Walking down the memory maze: Beyond context limit through interactive reading. *Preprint*, arXiv:2310.05029.

Shouyuan Chen, Sherman Wong, Liangjian Chen, and Yuandong Tian. 2023b. Extending context window of large language models via positional interpolation. *Preprint*, arXiv:2306.15595.

Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A. Smith, and Matt Gardner. 2021. A dataset of information-seeking questions and answers anchored in research papers. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4599–4610, Online. Association for Computational Linguistics.

Yiran Ding, Li Lyna Zhang, Chengruidong Zhang, Yuanyuan Xu, Ning Shang, Jiahang Xu, Fan Yang, and Mao Yang. 2024. Longrope: Extending llm context window beyond 2 million tokens. *arXiv preprint arXiv:2402.13753*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, et al. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Yao Fu, Rameswar Panda, Xinyao Niu, Xiang Yue, Hannaneh Hajishirzi, Yoon Kim, and Hao Peng. 2024. Data engineering for scaling language models to 128k context. *Preprint*, arXiv:2402.10171.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. 2020. Realm: retrieval-augmented language model pre-training. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org.

Gautier Izacard and Edouard Grave. 2021. Distilling knowledge from reader to retriever for question answering. In *International Conference on Learning Representations*.

Gautier Izacard, Patrick Lewis, Maria Lomeli, Lucas Hosseini, Fabio Petroni, Timo Schick, Jane Dwivedi-Yu, Armand Joulin, Sebastian Riedel, and Edouard Grave. 2022. Atlas: Few-shot learning with retrieval augmented language models. *Preprint*, arXiv:2208.03299.

Huiqiang Jiang, Qianhui Wu, , Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2024. LongLLMLingua: Accelerating and enhancing LLMs in long context scenarios via prompt compression. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1658–1677, Bangkok, Thailand. Association for Computational Linguistics.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2023. LLMLingua: Compressing prompts for accelerated inference of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 13358–13376, Singapore. Association for Computational Linguistics.

Karen Spärck Jones. 1972. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, pages 6769–6781, Online. Association for Computational Linguistics.

Omar Khattab and Matei Zaharia. 2020. Colbert: Efficient and effective passage search via contextualized late interaction over bert. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '20, page 39–48, New York, NY, USA. Association for Computing Machinery.

Tomáš Kočiský, 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.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

Sheng-Chieh Lin, Akari Asai, Minghan Li, Barlas Oguz, Jimmy Lin, Yashar Mehdad, Wen-tau Yih, and Xilun Chen. 2023. How to train your dragon: Diverse augmentation towards generalizable dense retrieval. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 6385–6400, Singapore. Association for Computational Linguistics.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173.

Ye Liu, Kazuma Hashimoto, Yingbo Zhou, Semih Yavuz, Caiming Xiong, and Philip Yu. 2021. Dense hierarchical retrieval for open-domain question answering. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 188–200, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Sewon Min, Kenton Lee, Ming-Wei Chang, Kristina Toutanova, and Hannaneh Hajishirzi. 2021. Joint passage ranking for diverse multi-answer retrieval. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6997–7008, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Tsendsuren Munkhdalai, Manaal Faruqui, and Siddharth Gopal. 2024. Leave no context behind: Efficient infinite context transformers with infini-attention. *Preprint*, arXiv:2404.07143.

Inderjeet Nair, Aparna Garimella, Balaji Vasan Srinivasan, Natwar Modani, Niyati Chhaya, Srikrishna Karanam, and Sumit Shekhar. 2023. A neural CRF-based hierarchical approach for linear text segmentation. In *Findings of the Association for Computational Linguistics: EACL 2023*, pages 883–893, Dubrovnik, Croatia. Association for Computational Linguistics.

Arvind Neelakantan, Tao Xu, Raul Puri, Alec Radford, Jesse Michael Han, Jerry Tworek, Qiming Yuan, Nikolas Tezak, Jong Wook Kim, Chris Hallacy, Johannes Heidecke, Pranav Shyam, Boris Power, Tyna Eloundou Nekoul, Girish Sastry, Gretchen Krueger, David Schnurr, Felipe Petroski Such, Kenny Hsu, Madeleine Thompson, Tabarak Khan, Toki Sherbakov, Joanne Jang, Peter Welinder, and Lilian Weng. 2022. Text and code embeddings by contrastive pre-training. *Preprint*, arXiv:2201.10005.

Benjamin Newman, Luca Soldaini, Raymond Fok, Arman Cohan, and Kyle Lo. 2023. A question answering framework for decontextualizing user-facing snippets from scientific documents. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3194–3212, Singapore. Association for Computational Linguistics.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke

Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, et al. 2024. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. 2024. YaRN: Efficient context window extension of large language models. In *The Twelfth International Conference on Learning Representations*.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, pages 3982–3992, Hong Kong, China. Association for Computational Linguistics.

Stephen Robertson, S. Walker, S. Jones, M. M. Hancock-Beaulieu, and M. Gatford. 1995. Okapi at trec-3. In *Overview of the Third Text REtrieval Conference (TREC-3)*, pages 109–126. Gaithersburg, MD: NIST.

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

Keshav Santhanam, Omar Khattab, Jon Saad-Falcon, Christopher Potts, and Matei Zaharia. 2022. ColBERTv2: Effective and efficient retrieval via lightweight late interaction. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3715–3734, Seattle, United States. Association for Computational Linguistics.

Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D Manning. 2024. RAPTOR: Recursive abstractive processing for tree-organized retrieval. In *The Twelfth International Conference on Learning Representations*.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. 2023. Roformer: Enhanced transformer with rotary position embedding. *Preprint*, arXiv:2104.09864.

Zhiqing Sun, Xuezhi Wang, Yi Tay, Yiming Yang, and Denny Zhou. 2023. Recitation-augmented language models. In *The Eleventh International Conference on Learning Representations*.

Yi Tay, Vinh Q. Tran, Mostafa Dehghani, Jianmo Ni, Dara Bahri, Harsh Mehta, Zhen Qin, Kai Hui, Zhe Zhao, Jai Gupta, Tal Schuster, William W. Cohen, and Donald Metzler. 2022. Transformer memory as a differentiable search index. In *Advances in Neural Information Processing Systems*.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M. Dai, Anja Hauth, Katie Millican, David Silver, Melvin Johnson, Ioannis Antonoglou, Julian Schrittwieser, Amelia Glaese, Jilin Chen, Emily Pitler, Timothy Lillicrap, Angeliki Lazaridou, Orhan Firat, James Molloy, Michael Isard, Paul R. Barham, Tom Hennigan, Benjamin Lee, Fabio Viola, Malcolm Reynolds, Yuanzhong Xu, Ryan Doherty, Eli Collins, Clemens Meyer, Eliza Rutherford, Erica Moreira, Kareem Ayoub, Megha Goel, Jack Krawczyk, Cosmo Du, Ed Chi, Heng-Tze Cheng, Eric Ni, Purvi Shah, Patrick Kane, Betty Chan, Manaal Faruqui, Aliaksei Severyn, Hanzhao Lin, YaGuang Li, Yong Cheng, Abe Ittycheriah, Mahdis Mahdieh, et al. 2024. Gemini: A family of highly capable multimodal models. *Preprint*, arXiv:2312.11805.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. MuSiQue: Multi-hop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Boxin Wang, Wei Ping, Peng Xu, Lawrence McAfee, Zihan Liu, Mohammad Shoeybi, Yi Dong, Oleksii Kuchaiev, Bo Li, Chaowei Xiao, Anima Anandkumar, and Bryan Catanzaro. 2023a. Shall we pretrain autoregressive language models with retrieval? a comprehensive study. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7763–7786, Singapore. Association for Computational Linguistics.

Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. 2023b. SimLM: Pre-training with representation bottleneck for dense passage retrieval. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2244–2258, Toronto, Canada. Association for Computational Linguistics.

Yu Wang, Nedim Lipka, Ryan A. Rossi, Alexa Siu, Ruiyi Zhang, and Tyler Derr. 2023c. Knowledge graph prompting for multi-document question answering. *Preprint*, arXiv:2308.11730.

Yujing Wang, Yingyan Hou, Haonan Wang, Ziming Miao, Shibin Wu, Hao Sun, Qi Chen, Yuqing Xia, Chengmin Chi, Guoshuai Zhao, Zheng Liu, Xing Xie, Hao Sun, Weiwei Deng, Qi Zhang, and Mao Yang. 2022. A neural corpus indexer for document retrieval. In *Advances in Neural Information Processing Systems*.

Wenhao Wu, Yizhong Wang, Yao Fu, Xiang Yue, Dawei Zhu, and Sujian Li. 2024. Long context alignment with short instructions and synthesized positions. *Preprint*, arXiv:2405.03939.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jianxin Yang, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, et al. 2024. Qwen2 technical report. *Preprint*, arXiv:2407.10671.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Conference on Empirical Methods in Natural Language Processing*.

Wenhao Yu, Dan Iter, Shuohang Wang, Yichong Xu, Mingxuan Ju, Soumya Sanyal, Chenguang Zhu, Michael Zeng, and Meng Jiang. 2023. Generate rather than retrieve: Large language models are strong context generators. In *The Eleventh International Conference on Learning Representations*.

Shiyue Zhang, David Wan, and Mohit Bansal. 2023. Extractive is not faithful: An investigation of broad unfaithfulness problems in extractive summarization. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2153–2174, Toronto, Canada. Association for Computational Linguistics.

Dawei Zhu, Nan Yang, Liang Wang, Yifan Song, Wenhao Wu, Furu Wei, and Sujian Li. 2024. PoSE: Efficient context window extension of LLMs via positional skip-wise training. In *The Twelfth International Conference on Learning Representations*.

A Prompt

A.1 Attention Graph Construction Prompt

In *Attention Graph Construction*, we employ an LLM to generate Information Points (IPs) in a bullet-point format, with each bullet point representing an IP. Specifically, the LLM prompt is shown in Table A1, along with an example from NarrativeQA (Kočiský et al., 2018). Each IP typically consists of a single sentence that describes one or a few events. T his approach enables *Graph Search* to more effectively retrieve information by tracking events rather than continuous text. We found that instructing the LLM to produce bullet-point lists aligns well with the LLM's inherent knowledge.

A.2 Graph Search Prompt

Graph Search uses a two-turn prompt, as shown in Table A2, with an example from NarrativeQA (Kočiský et al., 2018). In the first turn, the LLM is prompted to determine whether the current set of visited nodes $\mathcal{S}$ is sufficient to answer the query. The prompt asks the LLM, "*Can this question be answered by the following information?*", followed by the query and the visited nodes $\mathcal{S}$. If the response is "`No`", the search continues and the next node is retrieved, with all previous context KV cached to avoid additional computation. This process repeats until the LLM responds with "`Yes`". Once the LLM responds with "`Yes`", the second turn of the prompt asks the LLM to answer the query.

B KV Caching in Dynamic Progress Control

This section explains how the KV cache described in Section 3.2.1 is utilized in our approach. In conventional scenarios, KV caching is performed at the token level, where the LLM caches the key-value states of previous tokens when generating the next token. In contrast, our method employs a node-level KV cache.

Given the last retrieved node v_i in the visited set $\mathcal{S}$, let $\{x_n^i\}_{n=1}^{N_i}$ denote the tokens of node v_i, where N_i represents the number of tokens in v_i. The query and visited nodes are represented as $[q, v_1, \ldots, v_i, \ldots, v_{|S|}]$, with their corresponding tokens organized as $[x_1^q, \ldots, x_{N_q}^q, x_1^1, \ldots, x_{N_1}^1, \ldots, x_1^i, \ldots, x_{N_i}^i]$.

When a new node v_j is retrieved, the query, key, and value states for its tokens $\{x_n^j\}_{n=1}^{N_j} \in v_j$ are computed using the LLM's self-attention mechanism. Since v_j also attends to the previously visited nodes, the stored KV cache containing the tokens $[x_1^q, \ldots, x_{N_q}^q, x_1^1, \ldots, x_{N_1}^1, \ldots, x_1^i, \ldots, x_{N_i}^i]$ is provided as input to the LLM. At this point, the query states from v_j and the key-value states from q, $\mathcal{S}$, and v_j are available for self-attention.

After processing, v_j is added to the visited set $\mathcal{S}$, and the key-value states of its tokens $\{x_n^j\}_{n=1}^{N_j}$ are stored. These states are concatenated with the previous KV cache to form $[x_1^q, \ldots, x_{N_q}^q, \ldots, x_1^i, \ldots, x_{N_i}^i, x_1^j, \ldots, x_{N_j}^j]$, which is then used for the subsequent node retrieval.

Once the retrieval process is complete, the key-value states of all visited nodes in $\mathcal{S}$ are cached, and the LLM is prompted to answer the question using these cached states. Some LLMs may insert special tokens between the prompt and response, but these tokens are minimal, and the additional computation is negligible. The LLM follows standard decoding to generate tokens sequentially, leveraging the KV cache from previous tokens. Importantly, the query, key, and value states for all nodes, the query, and the answers are computed only once throughout the retrieval process, thereby avoiding additional computational overhead.

C Query-Node Attention Computation

This section describes our method for computing the query-node relevance r, which is derived from the attention weights introduced in Section 3.2.2.

We treat the query as a node and employ the averaging method described in Section 3.1 to compute the relevance score r_i between the query and each node v_i. Similarly, we extract only the attention between the node and the query, omitting other attention components such as intra-node attention. We do not apply the normalization procedure from Section 3.1, which constrains the relevance scores to sum to 1, because a query may be related to multiple nodes, and different queries can be associated with varying numbers of nodes. As a result, each node may potentially exhibit a strong relation to the query.

Instead, we adopt a heuristic normalization approach. Since the query is positioned before the visited nodes in the prompt, nodes appearing later may allocate some of their attention to earlier nodes, often resulting in lower attention scores for later nodes. Although placing the query at the end would avoid this issue, it would prevent the query from being cached in the key-value memory, thereby

creating a trade-off between performance and efficiency. To address this, we apply an empirical normalization to r_i by multiplying it by the position index of v_i, with the query q occupying the first position. For instance, in Figure 2, the extracted attention r_{15} is multiplied by 4 because it corresponds to the fourth position. Note that this is merely a heuristic approach rather than an ultimate solution. In this paper, we focus on the primary concept of leveraging attention weights and leave a more in-depth exploration of this detail to future work.

D Summary Graph Example

In this section, we present an example of generated Information Points (IPs) using parts of the first- and second-level nodes from HotpotQA (Yang et al., 2018) (see Figure A1). The nodes on the right represent second-level IPs, which are generated by aggregating information from the first-level chunk nodes on the left. The connections between these nodes denote attention weights, with higher weights visualized as redder and thicker lines.

It can be observed that the IPs are derived from multiple chunks. For instance, second-level nodes 10, 13, 14, 15, 16, 17, 18, and 19 are connected to several first-level nodes, while second-level nodes 11 and 20 primarily rely on a single first-level node with minimal connections elsewhere. From the perspective of the first-level nodes, nodes 2, 5, 6, 7, 8, and 9 are connected to multiple second-level nodes, whereas first-level node 3 is attended to only by second-level node 17.

The attention weights illustrate the dynamic connectivity between nodes. Some nodes connect to multiple others, while others connect to only one. Overall, our method leverages IPs and attention to explicitly capture the relationships between first-level and second-level nodes, enabling the model to understand how each piece of information is interconnected within the text.

E Datasets Statistics

Table A3 shows the token length statistics for NarrativeQA (Kočiský et al., 2018), Qasper (Dasigi et al., 2021), HotpotQA (Yang et al., 2018), and MuSiQue (Trivedi et al., 2022). In particular, NarrativeQA is much longer than the other datasets, followed by Qasper, while HotpotQA and MuSiQue contain relatively shorter texts.

F Efficiency Analysis

Table A4 presents the TFLOPs for graph construction per document (PECAN *Attention Graph Construction*) and for graph search per query (PECAN *Graph Search*). The combined TFLOPs of PECAN (*Attention Graph Construction* + *Graph Search*) represent the average TFLOPs per query for documents containing 1, 2, 4, or 8 queries.

As the number of queries per document increases, the TFLOPs for graph construction are amortized over more queries, reducing the average TFLOPs per query. In fact, when a document contains more than 8 queries, our method achieves lower average TFLOPs per query, even after accounting for summary generation.

For documents with only one query, our method's TFLOPs exceed those of Llama-3.1 on Qasper, HotpotQA, and MuSiQue. During graph construction, PECAN processes the entire document along with additional summary generation. However, on long documents such as those in NarrativeQA, PECAN uses fewer TFLOPs than Llama-3.1, since the Transformer (Vaswani et al., 2017) incurs quadratic complexity with very long inputs. Moreover, while Llama-3.1 processes the entire input at once, our method processes the document in chunks. As a result, PECAN operates with an 8K input window, enabling it to run on a single GPU. In contrast, running Llama-3.1 on NarrativeQA with a 100K-token input window required 8 A100 80G GPUs (even with CPU offloading). This limitation hinders the practical application of Llama-3.1 in real-world scenarios, whereas our method can easily run on a single GPU.

G Scientific Artifact Use

The scientific artifacts used in this paper are available for research use. Our utilization of these artifacts is for research purposes. The public datasets employed in this paper, including NarrativeQA, Qasper, HotpotQA, and MuSiQue, are not intended to contain offensive content.

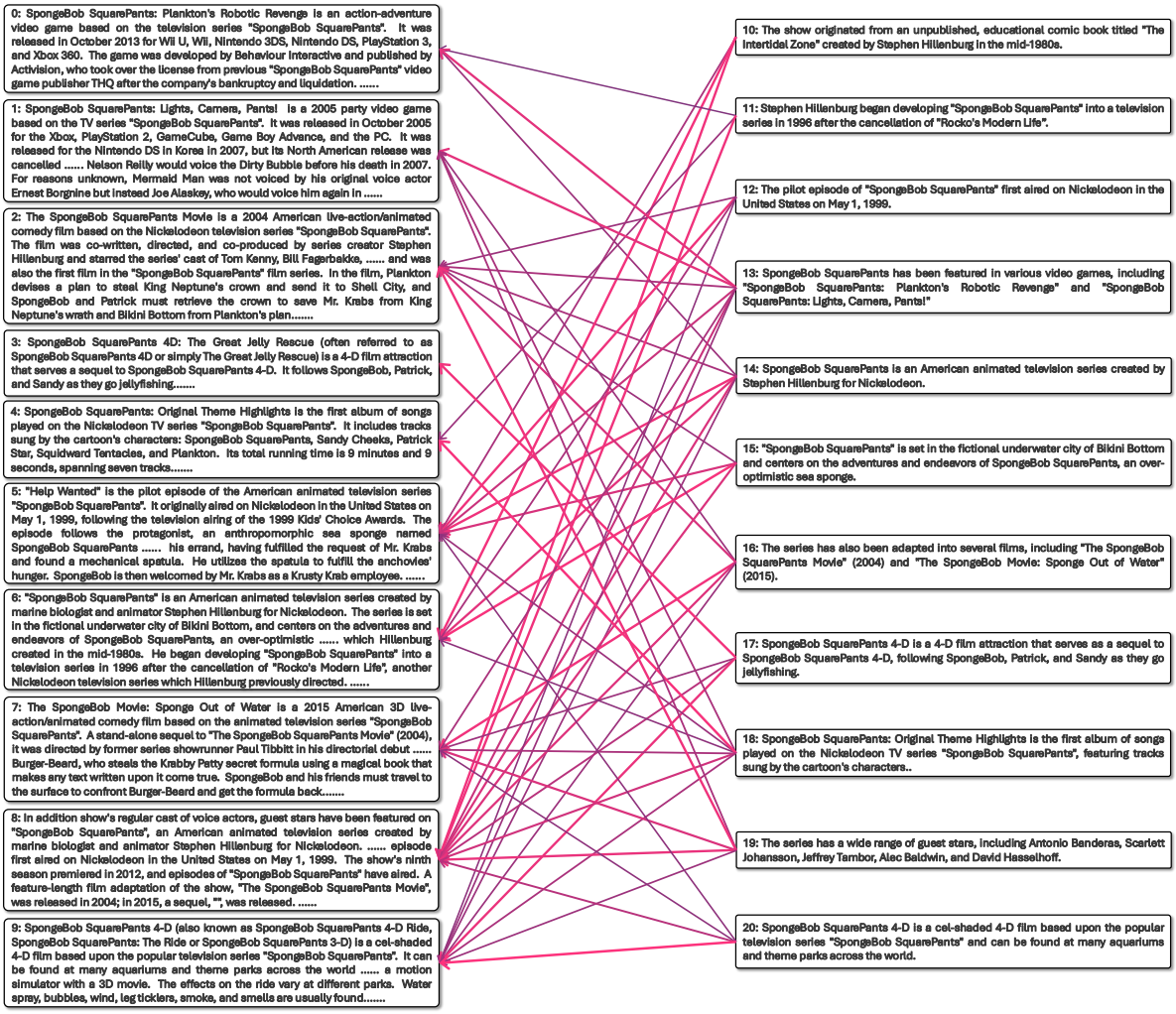

Figure A1: An example of generated IPs with portions of the first- and second-level nodes from HotpotQA. The nodes on the right are second-level nodes generated from the first-level nodes on the left. The connections in the middle represent attention weights, with higher weights shown as redder and thicker lines. For brevity, lines with attention weights less than 0.05 are omitted, and some lengthy text is truncated.

Attention Graph Construction
Prompt: Summary the following information. Each segment is separated by a new line symbol. ..... * *Mrs. Tabitha Twitchit expects "fine company" for tea and fetches the children before her friends arrive.* * *Tabitha dresses Moppet and Mittens in clean pinafores and tuckers, and Tom in "all sorts of elegant uncomfortable clothes" taken from a chest of drawers.* * *Tom is fat and bursts several buttons, but his mother sews them back on again.* * *Tabitha turns her kittens into the garden to keep them out of the way while she makes hot buttered toast for the party.* ..... Split your summary into different summary points according to the semantic information in these information points. It is not necessary to generate each summary point for each information point. Gather and organize information into summary points. In each summary point, try to avoid using pronouns like he/she/they and instead use full names. Generate in the format of: * summary point * summary point * summary point ..... Do not provide any explanation and start the summary directly.
Response: * *Mrs. Tabitha Twitchit expects fine company for tea and dresses Mittens, Tom Kitten, and Moppet in clean clothes.* * *Mrs. Tabitha Twitchit sends Mittens, Tom Kitten, and Moppet to the garden to keep them out of the way.* .....

Table A1: The prompt used for the LLM during *Attention Graph Construction*, along with an example from NarrativeQA. The *italicized* text represents the example context, while the remaining text represents the prompt instructions. The prompt instructs the LLM to generate Information Points (IPs) in a bullet-point format, with each bullet point representing an IP, which aligns well with the LLM's inherent knowledge.

Dynamic Graph Search
First-Turn Prompt: Can this question be answered by the following information? Response "Yes" or "No" in one word. Do not provide any explanation. Question: *Where does the mother send her kittens to keep them out of the way while getting ready for the party?* Information: ..... * *Mrs. Tabitha Twitchit sends Mittens, Tom Kitten, and Moppet to the garden to keep them out of the way.* * *Tabitha turns her kittens into the garden to keep them out of the way while she makes hot buttered toast for the party.* .....
First-Turn Response: Yes
Second-Turn Prompt: Given the above information and question, answer the question as concisely as you can.
Second-Turn Response: *The garden.*

Table A2: Prompt used for *Graph Search*, with an example from NarrativeQA. The *italicized* text represents the example context, while the remaining text represents the prompt instructions. In the first turn, the LLM is asked whether the current context is sufficient to answer the query. If the response is "`No`," additional context is appended to the prompt, and the query is repeated. Once the LLM responds with "`Yes`," the second turn prompts the LLM to provide the final answer directly.

Dataset	Average	Min	Max
NarrativeQA (Kočiský et al., 2018)	79,457	5,077	467,867
Qasper (Dasigi et al., 2021)	4,866	918	29,408
HotpotQA (Yang et al., 2018)	1,318	70	3,575
MuSiQue (Trivedi et al., 2022)	2,267	909	4,432

Table A3: Token length statistics for the NarrativeQA, Qasper, HotpotQA, and MuSiQue datasets, including the average, minimum, and maximum token lengths per document.

Method	NarrativeQA TFLOPs	Qasper TFLOPs	HotpotQA TFLOPs	MuSiQue TFLOPs
Llama-3.1-8B	3361.9	92.5	23.6	40.6
PECAN Attention Graph Construction	2042.8	136.6	40.2	66.7
PECAN Graph Search	31.0	66.9	16.0	30.9
PECAN Attention Graph Construction + Graph Search				
1 queries per document	2073.8	203.5	56.2	97.6
2 queries per document	1052.4	135.2	36.1	64.3
4 queries per document	541.7	101.1	26.1	47.6
8 queries per document	286.4	84.0	21.0	39.2

Table A4: TFLOPs for graph construction and graph search. `PECAN` *graph construction* + *graph search* shows the average TFLOPs per query when each document contains 1, 2, 4, or 8 queries.