GARLIC: LLM-GUIDED DYNAMIC PROGRESS CONTROL WITH HIERARCHICAL WEIGHTED GRAPH FOR LONG DOCUMENT QA

Anonymous authors
Paper under double-blind review

ABSTRACT

In the past, Retrieval-Augmented Generation (RAG) methods split text into chunks to enable language models to handle long documents. Recent tree-based RAG methods are able to retrieve detailed information while preserving global context. However, with the advent of more powerful LLMs, such as Llama 3.1, which offer better comprehension and support for longer inputs, we found that even recent tree-based RAG methods perform worse than directly feeding the entire document into Llama 3.1, although RAG methods still hold an advantage in reducing computational costs. In this paper, we propose a new retrieval method, called LLM-Guided Dynamic Progress Control with Hierarchical Weighted Graph (GARLIC), which outperforms previous state-of-the-art baselines, including Llama 3.1, while retaining the computational efficiency of RAG methods. Our method introduces several improvements: (1) Rather than using a tree structure, we construct a Hierarchical Weighted Directed Acyclic Graph with many-to-many summarization, where the graph edges are derived from attention mechanisms, and each node focuses on a single event or very few events. (2) We introduce a novel retrieval method that leverages the attention weights of LLMs rather than dense embedding similarity. Our method allows for searching the graph along multiple paths and can terminate at any depth. (3) We use the LLM to control the retrieval process, enabling it to dynamically adjust the amount and depth of information retrieved for different queries. Experimental results show that our method outperforms previous state-of-the-art baselines, including Llama 3.1, on two single-document and two multi-document QA datasets, while maintaining similar computational complexity to traditional RAG methods. [1]

1 INTRODUCTION

Retrieval-Augmented Generation (RAG) methods (Robertson et al., 1995; Robertson & Zaragoza, 2009; Reimers & Gurevych, 2019; Karpukhin et al., 2020; Khattab & Zaharia, 2020; Tay et al., 2022; Santhanam et al., 2022; Lin et al., 2023) have been a popular approach for handling QA tasks. Longer documents were segmented into chunks, and the most relevant chunks were retrieved and fed into a language model to generate answers. With the advent of Large Language Models (LLMs) (Touvron et al., 2023a;b), tree-based approaches such as RAPTOR (Sarthi et al., 2024) and MeMWalker (Chen et al., 2023a) have emerged. These models utilize LLMs to iteratively summarize the text, constructing tree-based summaries. By integrating information from different parts of the text, these methods facilitate the retrieval of both granular and high-level information, thereby improving performance through a balance of detailed understanding and global context, while managing longer documents effectively. However, as LLMs evolved, their capacity expanded, and models such as Llama 3.1 (Dubey et al., 2024) started supporting inputs of up to 128K tokens with enhanced comprehension capabilities. Studies like LongBench (Bai et al., 2024b) have demonstrated that the performance of RAG methods is often inferior to feeding the full document into LLMs directly (Zhang et al., 2023; Nair et al., 2023; Newman et al., 2023). Our experimental results also find that both RAPTOR and MeMWalker perform less well when compared to directly inputting the text into Llama 3.1. Nevertheless, retrieval methods remain beneficial in reducing input lengths and managing computational costs.

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

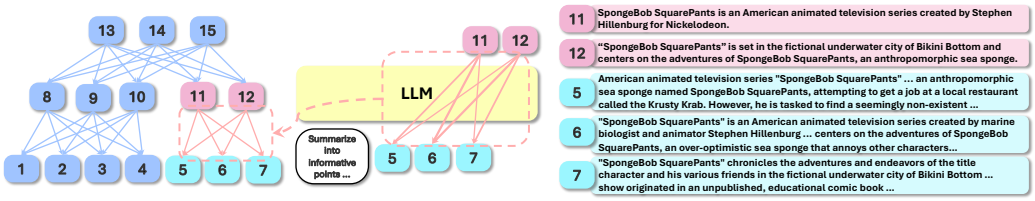


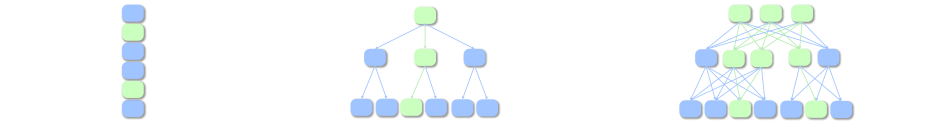
Figure 1: Overview of the Hierarchical Weighted Directed Acyclic Graph for summarization. Each node contains an Information Point (IP) and has multiple parent and child nodes, i.e., multiple successors and predecessors. Each time, the LLM is fed multiple nodes and prompted to generate multiple IPs. The weights of edges between nodes are computed based on the attention weights from LLM summarization. Some example IPs and chunks are shown on the right. For brevity, some long text is omitted.

In this paper, we introduce a new method, **LLM-Guided Dynamic Progress Control with Hierarchical Weighted Graph (`GARLIC`)**, which outperforms RAG baselines and even Llama 3.1 while preserving the benefits of retrieval with lower computational costs. Our method consists of two stages: *Summary Graph Construction* and *Dynamic Graph Search*. In the *Summary Graph Construction* stage, we prompt an LLM to generate multiple bullet-point sentences, termed **Information Points** (**IPs**). Each IP typically focuses on a single or very few events. IPs are initially generated from text segments and then fed back into an LLM to recursively generate higher-level IPs. This process forms a many-to-many graph in which each node can have multiple lower-level child nodes and higher-level parent nodes, i.e., multiple successors and predecessors. During summarization, the LLM attention weights between the generated IPs and the input IPs are extracted to capture their relationships, as shown in Figure 1. The LLM consolidates and summarizes the same event from different sources, enabling the model to efficiently identify specific events rather than scanning through entire summary texts. The attention mechanism records which parts of the text contribute to an IP. The resulting structure is a *Hierarchical Weighted Directed Acyclic Graph* (HWDAG), where the extracted attention serves as the edge weights of the graph.

In the *Dynamic Graph Search* stage, we retrieve nodes from the HWDAG and feed them into the LLM to predict answers. We introduce a method called *LLM-guided Dynamic Progress Control*, which dynamically determines when to stop the search. Starting with top-level nodes, one node is selected per iteration. The LLM evaluates if the retrieved nodes contain sufficient information to answer a given query. The search continues until the LLM signals that enough information has been gathered. The prompt asks: "*Can this question be answered by the following documents?*" followed by the query and the text of the retrieved node. Each time a new node is retrieved, the previous inputs are KV cached, allowing new documents to be appended without reprocessing the entire input. Therefore, although we use LLM to decide when to stop the retrieval process, the method does not introduce additional computational overhead. This approach effectively resolves the prior challenge of determining how many chunks to search.

During the search, the attention between the retrieved nodes and the query is extracted, enabling it to assess relevance based on the LLM's knowledge. We combine this attention with the attention from the *Dynamic Graph Search* stage to guide the search and retrieve the next node. We employ Greedy Best-First Search (GBFS), a variant of Best-First Search (BFS), to traverse the graph. In contrast, RAPTOR (Sarthi et al., 2024) and MeMWalker (Chen et al., 2023a) utilize search methods that are more similar to Depth-First Search (DFS). We treat the graph as an adjacency matrix, considering all nodes connected to the retrieved node during the search, enabling the exploration of multiple paths. On the contrary, previous methods, as shown in Figure 2b, perform a search by following a single path from the top node to the bottom node since the number of nodes to retrieve was uncertain. Our method, however, is more flexible, as shown in Figure 2c, allowing for multiple paths and enabling the search to terminate at any level. Therefore, our method can handle queries that require varying amounts of information and information spread across different parts of the graph more effectively.

The combination of HWDAG and attention-based search enables us to introduce an alternative retrieval approach that solely relies on attention weights. GBFS integrates well with the HWDAG's structure, which consists of numerous smaller nodes, allowing the flexible retrieval of multiple IPs in any quantity and order. GBFS equipped with Dynamic Progress Control empowers the LLM to decide when to stop, making the retrieval process adaptable to different queries and allowing for



(a) Chunk-based retrieval. (b) Tree-based retrieval. (c) Our HWDAG-based retrieval.

Figure 2: Comparison of three retrieval methods. Nodes shaded in green are retrieved nodes. **(a) Chunk-based retrieval**. **(b) Tree-based retrieval**, starting from the top node and selecting a child node at each level until reaching the bottom node. **(c) Retrieval based on a HWDAG**. The node search is flexible, allowing multiple paths from the top level, and the search can stop at any level.

termination at various points along the search paths and at different levels of detail. In experiments, GARLIC outperforms other baselines, including Llama 3.1, without incurring additional inference computational costs. In summary, the contributions of this paper are as follows:

- We propose Dynamic Progress Control using the LLM to control the retrieval process while employing KV caching to avoid additional time complexity.
- We introduce a novel attention-based retrieval method that relies on LLM attention weights using a Hierarchical Weighted Directed Acyclic Graph with many-to-many summarization, where each node represents an Information Point (IP) focused on a single or very few events, and the graph edges are derived from LLM attention during summarization.
- Our method surpasses previous state-of-the-art baselines, including Llama 3.1, while maintaining the computational efficiency of retrieval methods.

2 Related Work

Retrieval Traditional retrieval techniques, such as TF-IDF (Jones, 1972) and BM25 (Robertson et al., 1995; Robertson & Zaragoza, 2009), retrieve information based on word terms. Subsequently, deep learning–based retrieval methods quickly became popular. REALM (Guu et al., 2020) augments the language model pre-training with a latent knowledge retriever using masked language modeling. DPR (Dense Passage Retrieval) (Karpukhin et al., 2020) encodes queries and documents as dense embeddings, with similarity computed between them. ColBERT (Khattab & Zaharia, 2020; Santhanam et al., 2022) produces multi-vector representations at the token level. JPR (Joint Passage Retrieval) (Min et al., 2021) is a joint passage retrieval model with an autoregressive reranker that selects a sequence of passages. DHR (Dense Hierarchical Retrieval) (Liu et al., 2021) leverages both macroscopic document-level semantics and microscopic passage-level semantics. Fusion-in-Decoder (Izacard & Grave, 2021) employs both DPR and BM25 in a knowledge distillation manner, which does not require annotated query-document pairs. CPT-text (Neelakantan et al., 2022) utilizes contrastive pre-training on unsupervised data. NCI (Wang et al., 2022) directly generates relevant document identifiers for a given query. Atlas (Izacard et al., 2022) fine-tunes an encoder-decoder model with a retriever to address knowledge-intensive tasks with minimal training examples. RETRO (Borgeaud et al., 2022; Wang et al., 2023a) conditions on document chunks based on local similarity with preceding tokens. HHR (Hybrid Hierarchical Retrieval) (Arivazhagan et al., 2023) combines sparse and dense retrieval methods across both document and passage retrieval stages. SimLM (Wang et al., 2023b) proposes a new loss function to reduce the mismatch between pre-training and fine-tuning input distributions. Dragon (Lin et al., 2023) uses contrastive learning and data augmentation to train a model, achieving state-of-the-art retrieval performance among eight baselines. Additionally, with the rise of LLMs, some research has explored the use of LLMs as retrievers. GENREAD (Yu et al., 2023) prompts LLMs to generate contextual documents based on a given query. RECITE (Sun et al., 2023) retrieves relevant passages from the LLM's internal memory via sampling. KGP (Knowledge Graph Prompting) (Wang et al., 2023c) builds a knowledge graph from multiple documents, with the LLM navigating. Recently, MeMWalker (Chen et al., 2023a) constructs tree-based summaries and uses LLMs to navigate through the tree. RAPTOR (Sarthi et al., 2024) also creates tree-based summaries with clustering and uses embedding similarities to select the most relevant nodes at each level for retrieval. However, our approach differs from these methods. We construct a summary graph with IPs and employ attention mechanisms and GBFS for retrieval along any path, whereas MeMWalker and RAPTOR follow a single path from the top level to the bottom. Additionally, our method uses the LLM to dynamically determine when to stop the search, whereas MeMWalker also uses LLM to navigate but incurs significantly higher computational costs.

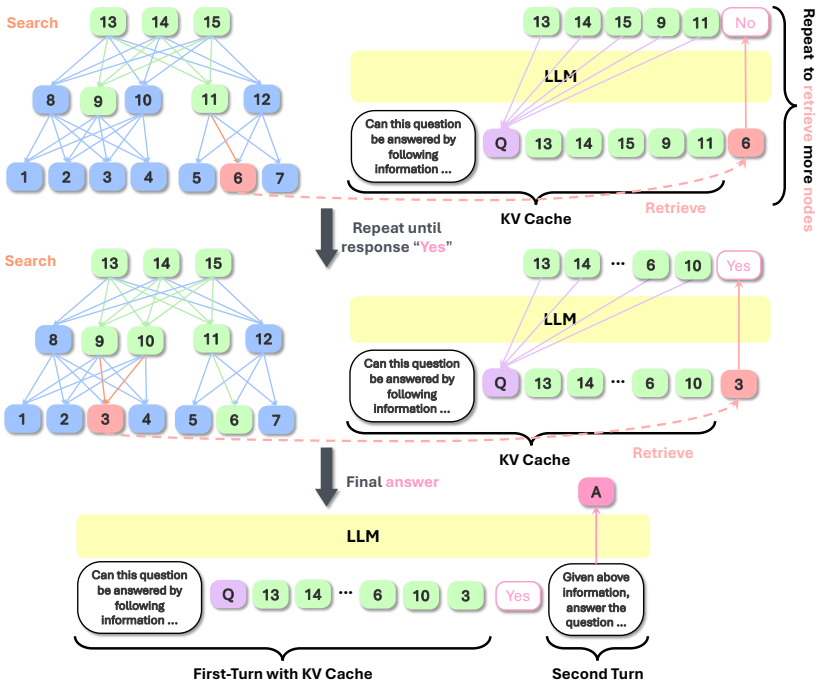


Figure 3: Overview of **Dynamic Graph Search**. Each time, a node is retrieved by Greedy Best-First Search using attention weights. The visited nodes are fed into the LLM, prompting the LLM to determine if sufficient nodes have been gathered to answer the query. This process incurs no additional computational cost due to KV caching. The search continues until the LLM signals that enough relevant nodes are retrieved, at which point the final answer is generated. The process adjusts dynamically based on the query, retrieving nodes flexibly across multiple graph paths and depths.

Long-Context Language Models Recent long-context language models have focused on overcoming the limitations of context window size, primarily through positional interpolation to extend long-context capabilities by training on full-length texts. Chen et al. (2023b) used positional interpolation on RoPE (Rotary Position Embedding) (Su et al., 2023) to extend context length. Ding et al. (2024) proposed LongRoPE which performs direct extrapolation by rescaling RoPE with varied interpolation across RoPE dimensions at different token positions. Peng et al. (2024) and Fu et al. (2024) fine-tuned models on longer inputs and extended RoPE for longer contexts. LongLoRA (Chen et al., 2024) shifts sparse attention on LoRA (Hu et al., 2022) to extend model capacity for longer inputs. 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 in each chunk. SkipAlign (Wu et al., 2024) synthesizes long-range dependencies from the aspect of position indices. Liu et al. (2024) showed that performance can degrade significantly when the position of relevant information is altered. Infini-Transformer (Munkhdalai et al., 2024) handles infinitely long inputs using compressive memory, masked local attention, and long-term attention mechanisms. Our method is complementary to these approaches. The LLMs used in these methods could serve as the base model in our approach to further reduce computational demands. Our focus is on utilizing LLMs effectively rather than improving the LLMs, and our method is compatible with these long-context LLMs.

3 METHODOLOGY

Our method consists of two main steps: *Summary Graph Construction* and *Dynamic Graph Search*. During *Summary Graph Construction*, as depicted in Figure 1, we iteratively construct an HWDAG from the documents, where the nodes represent IPs. In *Dynamic Graph Search*, as illustrated in Figure 3, given a query, we dynamically retrieve the IPs from the constructed HWDAG by performing a search guided by an LLM. Once enough nodes are retrieved, the LLM generates the final answer based on them. *Summary Graph Construction* is independent of the query, while *Dynamic Graph Search* is query-dependent.

3.1 SUMMARY GRAPH CONSTRUCTION

This step generates a HWDAG, defined as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ represents the collection of nodes, and $\mathcal{E}$ represents the collection of edges. Each node $v_i^l \in \mathcal{V}$ contains the text and is in level l. The subset of nodes at level l is denoted by $\mathcal{V}_l \subset \mathcal{V}$. Each edge $e_{i,j} \in \mathcal{E}$ represents a scalar value indicating the relatedness between node v_i and node v_j. Since $\mathcal{G}$ is a directed graph, $e_{i,j} \neq e_{j,i}$. By default, if there is no relationship between nodes v_i and v_j, then $e_{i,j}, e_{j,i} = 0$.

The text content 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 choosing them until the total text content length exceeds a threshold n_s. This threshold n_s includes both the prompt and the generated summary. These nodes are then input into the LLM. The prompt instructs the LLM to generate summaries in bullet-point format, with each point referred to as an **Information Point (IP)**. The LLM prompt used for summarization is shown in Table 4 in Appendix A.1, and a graph example is shown in Appendix D.

In previous retrieval methods, whether using chunks or summaries, multiple events are often contained in a single text chunk, requiring the retrieval of the entire chunk even if only one event is relevant (Karpukhin et al., 2020; Lin et al., 2023; Chen et al., 2023a; Sarthi et al., 2024). In contrast, each IP corresponding to each node in our method describes a single event and reduces the smallest retrievable unit from a chunk to an IP.

The attention from a higher-level node v_i^{l+1} to a lower-level node v_j^l is averaged across all tokens and layers to produce a scalar value $e_{i,j}$, which serves as the edge value from node v_i to node v_j, where $\sum_j e_{i,j} = 1$. The value of $e_{i,j}$ represents how much information the higher-level node v_i^{l+1} extracts from the lower-level node v_j^l. For example, in Figure 3, node v_9 directs to nodes $\{v_j\}_{j=1}^4$, with edge values $\{e_{9,j}\}_{j=1}^4$, and $\sum_{j=1}^4 e_{9,j} = 1$. Detailed computation can be found in Appendix B.

3.2 DYNAMIC GRAPH SEARCH

This section explains the process of retrieving nodes from the graph $\mathcal{G}$. An overview of dynamic graph search is illustrated in Figure 3. The process consists of two steps: *Dynamic Progress Control*, discussed in Section 3.2.1, and *Graph Search* in Section 3.2.2.

3.2.1 DYNAMIC PROGRESS CONTROL

This subsection describes the dynamic control of the search process. Initially, a visited set, denoted as $\mathcal{S} \subset \mathcal{V}$, which is initialized with the top-level nodes of $\mathcal{V}$, is fed into the LLM. We use a two-turn prompt system. 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 visited nodes $\mathcal{S}$. The complete prompt is shown in Table 5 in Appendix A.2, including an example from NarrativeQA (Kočiský et al., 2018). If the response is "`No`", the search continues, and the next node is retrieved. The details of the search process will be introduced in Section 3.2.2. The newly retrieved node is then appended to the end of the visited set $\mathcal{S}$, and the LLM is queried again. 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 3. Additional details can be found in Appendix C. This ensures that no additional computational resources are required. Some LLMs may insert special tokens between the prompt and response, but these tokens are minimal, and the additional computation is negligible. Once the LLM responds with "`Yes`", the second turn of the prompt will ask the LLM to answer the query. At this point, the final answer is obtained.

This approach allows the LLM to dynamically determine the number of nodes needed for retrieval based on the query. Different queries may require varying amounts and types of content. For example, a query that requires only high-level information can be answered with just a few high-level nodes, whereas a query spanning multiple detailed aspects of the document may require the retrieval of both high- and low-level nodes. Previous methods typically had a fixed length of retrieved content, which could lead to either too much or too little information being retrieved for certain queries.

Similar to the concept of early stopping patience, we introduce a stop patience p. With this approach, the search stops after the LLM responds "`Yes`" p times. We observed that increasing p can even

further improve performance, though this creates a trade-off between performance gains and computational cost. Performance improvements tend to plateau when $p > 5$, while the computational cost continues to increase. See Section 4.3 for more details.

3.2.2 GRAPH SEARCH

This section describes the process of searching the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Beyond the embedding similarity used in dense retrieval, as described in (Sarthi et al., 2024), we introduce another approach by leveraging the attention mechanisms of the LLM.

First, we define the adjacency matrix as $\boldsymbol{E} \in \mathbb{R}^{n_v \times n_v}$, where $\boldsymbol{E}_{i,j} = e_{i,j}$ and n_v represents the number of nodes in $\mathcal{V}$. Each time, the visited nodes $\mathcal{S} \subset \mathcal{V}$ are fed into the LLM using the prompt shown in Table 5. The attention weight between the visited node v_i and the query q is extracted and averaged across tokens and layers, denoted as a_i, which represents the attention that node v_i gives to the query q. Given that the query precedes the visited nodes in the prompt, nodes that appear later in the sequence may distribute some of their attention to earlier nodes. We observe that nodes positioned later in the sequence tend to have lower attention scores. To address this issue, we apply an empirical normalization to a_i by multiplying of the corresponding position v_i, with q occupying the first position. For example, in Figure 3, the extracted attention a_{15} is multiplied by 3, as a_{15} is in the third position. We found this adjustment yields good experimental results. After obtaining all attention values on all nodes, we construct a vector $\boldsymbol{a} \in \mathbb{R}^{n_v}$, where $\boldsymbol{a}_i = a_i$, and the remaining elements are set to 0, i.e., $\boldsymbol{a}_j = 0$ for $v_j \notin \mathcal{S}$.

Once the adjacency matrix $\boldsymbol{E} \in \mathbb{R}^{n_v \times n_v}$ and the query attention vector $\boldsymbol{a} \in \mathbb{R}^{n_v}$ are computed, the score vector $\boldsymbol{z} \in \mathbb{R}^{n_v}$ is calculated as follows: $\boldsymbol{z} = \boldsymbol{E}^T \boldsymbol{a}$, where $\boldsymbol{z}$ represents the score for each node, indicating how likely it is to be retrieved. The intuition behind this is that, if a node, i.e., an IP, is strongly correlated with the query, the details about this node will be more helpful for answering the query. $\boldsymbol{E}$ represents the relationships between nodes, while $\boldsymbol{a}$ highlights which of the currently visited nodes is more relevant. Through matrix multiplication, we can identify nodes that are related to the current query but are not yet part of the visited set $\mathcal{S}$. Nodes that are more closely related to the query q, will have their related successor nodes assigned higher scores. If a retrieved node is not relevant to the query, it will receive a low score in $\boldsymbol{z}$, thus preventing the search from continuing through that node. When sufficient relevant details are retrieved, Dynamic Progress Control will stop the search to avoid retrieving unnecessary details. We use $\boldsymbol{E}^T_{j,i}$ for the calculation of $\boldsymbol{z}$, which represents the attention received by v_j from v_i, because the goal is to calculate candidate scores for each node, focusing on the nodes that are the focus of attention. If multiple nodes are highly related to q and also connected to a successor, that successor node will receive a higher score, as it accumulates scores from multiple predecessors. For example, in Figure 3, v_3 will receive scores from both v_9 and v_{10}. We normalize $\boldsymbol{z}$ and the query-node embedding similarity so that the sum of their elements equals 1, and then we add the embedding similarity to $\boldsymbol{z}$ as the final score. The node $v \notin \mathcal{S}$ that is not yet in the visited set and has the highest score is selected as the next node to retrieve.

4 EXPERIMENTS

4.1 SETUP

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. See Appendix E for more statistics.

Metrics We use F1, ROUGE-L (Lin, 2004), and BLEU-4 (Papineni et al., 2002) as evaluation metrics. The final scores are computed using the evaluation source code from LongBench (Bai et al., 2024b) and Hugging Face Evaluate[2]. Additionally, we measure the average TFLOPs (Tera Floating Point Operations) during search and inference for each query.

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

Baseline We employ both traditional and recent baselines: **BM25** (Robertson et al., 1995; Robertson & Zaragoza, 2009) is a bag-of-words based retrieval method that ranks documents based on the query terms appearing in them. **SBERT** (Reimers & Gurevych, 2019) is a dense retrieval method that employs dense embeddings obtained through the encoder model. **Dragon** (Lin et al., 2023) is a dense retrieval method that uses contrastive learning and data augmentation to train a model, achieving state-of-the-art retrieval performance among eight baselines. **LongLLMLingua** (Jiang et al., 2024) introduces question-aware compression based on LLMLingua (Jiang et al., 2023), a prompt compression method. **MeMWalker** (Chen et al., 2023a) processes context into a tree of summary nodes and navigates the tree to search for relevant information guided by the LLM, to handle long-text QA tasks within a limited input window. **RAPTOR** (Sarthi et al., 2024) constructs a tree by recursively embedding, clustering, and summarizing chunks of text for retrieval. RAPTOR has two variants: "tree traversal" (**RAPTOR-TT**) and "collapsed tree" (**RAPTOR-CT**). **Llama3.1-8B** (Dubey et al., 2024) is the 8B version of Llama3.1, which expands the context window to 128K tokens, allowing documents from the four datasets, except for some from NarrativeQA, to be directly fed into the model.

For all baselines and `GARLIC`, we use Llama3.1-8B (Dubey et al., 2024) as the LLM for both summarization and inference. For MeMWalker, since the source code was not released, we implemented it according to the paper using Llama3.1. For LongLLMLingua, which was originally proposed to use a smaller model to compress prompts for GPT-3.5, we used the published model Phi-2-2.7B, as provided by LongLLMLingua, to compress the text before inputting it into Llama3.1-8B. For other models, we ran their source code using Llama3.1-8B. For all inferences, we did not use Chain-of-Thought (CoT). We use SBERT (Reimers & Gurevych, 2019) as the retrieval model in `GARLIC`. In the experiments, we set the stop patience $p = 1$ by default and the length threshold n_s to 8K. Across all datasets and steps of our method, including graph construction and search, the input window is capped by n_s and can be processed using an NVIDIA A100 80G GPU. A summary graph example is illustrated in Appendix D.

4.2 Main Results

The main results are shown in Table 1. Here, TFLOPs refers to the query-dependent inference. For MeMWalker, RAPTOR, and `GARLIC`, summarization TFLOPs are not included as summarization is query-independent. For BM25, SBERT, and Dragon, in addition to top-5, we also add a top-X set to match the TFLOPs of `GARLIC` for a fair comparison. Specifically, we used Top-7, Top-14, Top-4, and Top-7 for NarrativeQA, Qasper, HotpotQA, and MuSiQue, respectively. A similar *top-X* applies to RAPTOR-CT, with Top-20, Top-42, Top-12, and Top-22, respectively. It is worth noting that our Dynamic Progress Control can determine the appropriate number of chunks or nodes to retrieve in a single pass, whereas these methods require extensive hyperparameter searches to find the optimal number. For Llama3.1-8B on NarrativeQA, we used 8 A100 80G GPUs, with CPU offloading, to handle the dataset. However, we could only process an input window of 100K tokens with these resources, resulting in 22.3% of documents being truncated. For LongLLMLingua, since the compression is query-dependent, we included the compression TFLOPs.

The performance of BM25, SBERT, and Dragon was relatively similar, with Dragon showing an advantage on NarrativeQA. Comparing the top-5 and top-X results, we found that for Llama3.1-8B, retrieving more chunks generally leads to better results. LongLLMLingua achieves better results than Llama3.1-8B 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. Given the small size difference between Phi-2 and Llama3.1-8B, the compression TFLOPs take up a significant portion of the overall computation. This may not reflect the intended use cases of LongLLMLingua, making efficiency comparisons challenging. MeMWalker did not perform well and lagged behind traditional retrieval methods. MeMWalker was initially designed to overcome the input length limitations for LLMs, but it struggles to navigate large trees effectively. It requires the LLM to generate correct responses and formats at every node, and when the tree becomes too large, navigation is prone to failure. This likely contributed to its poor performance on NarrativeQA. Additionally, its high computational complexity arises from the need to invoke the LLM at every node. RAPTOR outperformed other retrieval methods at lower TFLOPs. Its summarization is able to extract key information from the document, which improves performance and reduces TFLOPs. RAPTOR-TT is constrained by its fixed retrieval path, which sequentially retrieves nodes from the top level to the bottom. While RAPTOR-CT top-X achieves higher per-

Table 1: F1 (%), ROUGE-L (%), BLEU-4 (%), and TFLOPs of baselines and GARLIC on NarrativeQA, Qasper, HotpotQA, and MuSiQue. TFLOPs are calculated during query-dependent inference. "Ratio" represents the ratio of the baselines' TFLOPs to GARLIC's TFLOPs. *Top-X* of BM25, SBERT, and Dragon denotes Top-7, Top-14, Top-4, and Top-7 for NarrativeQA, Qasper, HotpotQA, and MuSiQue, respectively, where the numbers are selected to ensure the baselines have similar TFLOPs to GARLIC for a performance comparison at the same TFLOPs. A similar *top-X* applies to RAPTOR-CT, with Top-20, Top-42, Top-12, and Top-22, respectively.

Method	NarrativeQA					Qasper				
	F1	ROUGE-L	BLEU-4	TFLOPs	Ratio	F1	ROUGE-L	BLEU-4	TFLOPs	Ratio
BM25 top-5	52.7	51.8	13.4	26.7	0.86x	41.0	39.6	21.0	26.3	0.39x
SBERT top-5	36.5	35.8	6.6	26.8	0.86x	44.4	42.4	23.9	26.0	0.39x
Dragon top-5	53.8	52.9	13.6	26.9	0.87x	43.0	41.4	22.8	24.5	0.36x
RAPTOR-TT	40.6	39.8	7.8	20.3	0.65x	42.1	40.1	17.2	17.7	0.26x
RAPTOR-CT	48.6	47.8	11.8	17.9	0.58x	44.6	42.7	19.5	16.6	0.25x
LongLLMLingua	50.5	49.5	10.1	1789.4	57.72x	43.2	43.0	21.0	159.7	2.39x
MeMWalker	11.2	9.8	2.6	353.8	11.41x	39.0	36.8	17.4	123.9	1.85x
BM25 top-X	53.7	52.9	14.0	37.5	1.21x	47.0	45.1	22.8	69.3	1.04x
SBERT top-X	39.5	38.8	7.3	37.5	1.21x	46.6	44.5	23.3	68.9	1.03x
Dragon top-X	55.1	54.2	13.6	37.5	1.21x	46.9	44.8	22.1	67.0	1.00x
RAPTOR-CT top-X	52.0	51.2	11.8	35.1	1.13x	46.9	44.7	20.8	67.3	1.01x
Llama3.1-8B	53.7	52.6	10.4	3361.9	108.45x	49.4	47.6	26.9	92.5	1.38x
GARLIC	61.1	60.2	18.6	31.0	1.00x	49.7	47.9	27.0	66.9	1.00x
Method	**HotpotQA**					**MuSiQue**				
	F1	ROUGE-L	BLEU-4	TFLOPs	Ratio	F1	ROUGE-L	BLEU-4	TFLOPs	Ratio
BM25 top-5	40.8	40.9	7.7	22.9	1.43x	28.7	28.7	5.1	26.3	0.85x
SBERT top-5	40.9	40.8	8.0	22.6	1.41x	30.7	30.8	6.3	26.1	0.84x
Dragon top-5	39.7	39.6	6.9	23.3	1.46x	28.5	28.4	5.4	28.1	0.91x
RAPTOR-TT	38.6	38.5	6.7	8.4	0.53x	29.3	29.3	4.7	12.6	0.41x
RAPTOR-CT	40.9	40.4	7.2	15.3	0.96x	31.5	31.5	5.5	16.1	0.52x
LongLLMLingua	43.4	43.5	8.1	43.6	2.73x	34.5	34.4	5.6	78.9	2.55x
MeMWalker	39.7	38.9	13.9	93.4	5.84x	24.0	23.5	9.9	175.7	5.69x
BM25 top-X	40.7	40.8	7.7	20.0	1.25x	31.8	31.7	5.6	35.6	1.15x
SBERT top-X	40.8	40.7	7.5	19.6	1.23x	32.5	32.5	6.4	35.6	1.15x
Dragon top-X	39.2	39.1	6.7	20.6	1.29x	30.2	30.1	6.0	38.0	1.23x
RAPTOR-CT top-X	40.7	40.7	7.2	17.9	1.12x	35.4	35.2	7.2	32.2	1.04x
Llama3.1-8B	41.3	41.2	6.3	23.7	1.48x	35.8	35.7	5.6	40.6	1.31x
GARLIC	43.5	43.5	7.2	16.0	1.00x	36.9	36.8	5.7	30.9	1.00x

formance by using more nodes, it still underperforms compared to our method at similar TFLOPs, demonstrating that its tree-based summarization is not as efficient as ours. Llama3.1-8B's excelled on most datasets except NarrativeQA, where input truncation likely affected its performance. For particularly long inputs in NarrativeQA, Llama3.1-8B reached 3361.9 TFLOPs due to an average token length of 794,457, suggesting that directly feeding very long texts into an LLM may not be the optimal choice.

Our method outperformed all baselines and Llama3.1-8B across all four datasets. Even with the same TFLOPs, our results were better than those of BM25, SBERT, and Dragon. Compared to LongLLMLingua, our method achieved better performance with lower TFLOPs, although the differing application scenarios limit direct comparison. Compared to MeMWalker and Llama3.1-8B, our method achieved higher performance at a lower computational cost. Similarly, our method outperformed RAPTOR while omitting the clustering step, as our retrieval process stops only after gathering sufficient nodes. Overall, our method demonstrates both performance advantages and low computational complexity. Its superiority over Llama3.1-8B can be attributed to the effective summarization of document key points through IPs, and the utilization of attention to enhance search effectiveness, combined with Dynamic Progress Control to ensure adequate information collection.

4.3 Dynamic Search Stop Study

Following the concept of early stop patience, the stop patience p in this paper refers to the number of times the LLM responds "Yes" before the search stops, as introduced in Section 3.2.1. In Table 1, we set the stop patience $p = 1$. In this section, we investigate how the stop patience p influences both performance and efficiency.

As shown in Figure 4, increasing p can further improve performance beyond the results in Table 1, but at the cost of increased computational resources. It can be observed that with our method and

Llama3.1, retrieving more nodes allows the model to provide more accurate answers. When $p < 5$, the F1 score increases rapidly, indicating that, on average, there are still up to 5 nodes containing useful information that have not yet been retrieved. However, when $p > 5$, the improvement in the F1 score slows, and the TFLOPs curve begins to exceed the F1 increase, suggesting that the additional retrieved nodes do not contribute significantly to the model's performance, thus reducing the cost-effectiveness of further computation. The slowdown in performance improvement as p increases also validates the LLM's ability to stop the search, as few additional nodes are beneficial once the search stops. By adjusting p, GARLIC can balance between effectiveness and efficiency. Due to the presence of Dynamic Progress Control, there is a lower bound for this adjustment range, specifically when $p = 1$, and across all four datasets, performance increases more rapidly when $p < 5$. In contrast, previous methods required adjusting the number of chunks to retrieve based on different data distributions or even individual queries, resulting in higher hyper-parameter search costs.

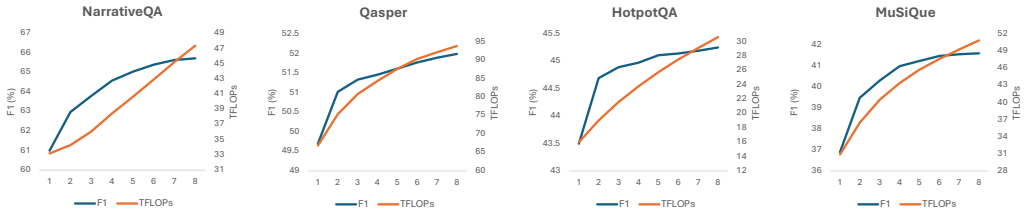


Figure 4: The F1 (%) and TFLOPs of GARLIC on NarrativeQA, Qasper, HotpotQA, and MuSiQue with different stop patience values p. The horizontal axis represents stop patience p. The left vertical axis shows the F1 (%) corresponding to the blue line, and the right vertical axis shows TFLOPs corresponding to the red 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.

4.4 Ablation Study

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

Table 2: Ablation study of the four components of GARLIC with F1 (%), ROUGE-L (%), and BLEU-4 (%) on NarrativeQA, Qasper, HotpotQA, and MuSiQue.

Method	NarrativeQA			Qasper			HotpotQA			MuSiQue		
	F1	ROUGE-L	BLEU-4	F1	ROUGE-L	BLEU-4	F1	ROUGE-L	BLEU-4	F1	ROUGE-L	BLEU-4
GARLIC	61.1	60.2	18.6	49.7	47.9	27.0	43.5	43.4	7.2	36.9	36.9	5.7
w/o Graph-based Summary	51.4	50.8	9.2	47.3	45.5	23.8	40.9	40.7	6.4	32.3	32.3	4.1
w/o Dynamic Progress Control	53.5	52.9	16.0	37.7	36.4	20.8	39.4	39.3	5.9	27.0	27.1	3.4
w/o Attention Search	53.0	52.4	18.3	46.9	45.5	25.3	41.9	41.7	6.8	33.1	32.9	5.1
w/o Embedding Similarity Search	59.5	58.7	17.6	48.0	46.2	23.5	42.9	42.8	6.6	35.9	35.8	5.0

w/o Graph-based Summary: We drop the graph-based summary and instead follow a tree-based manner, as illustrated in Figure 2b, by iteratively summarizing nodes and conducting the search with it. All datasets show a decline in performance, especially for NarrativeQA. For more complex and longer inputs, IPs help organize information more effectively.

w/o Dynamic Progress Control: We measured the average number of nodes used across the four datasets with GARLIC, which were 48, 36, 17, and 30 for NarrativeQA, Qasper, HotpotQA, and MuSiQue, respectively. Instead of using the LLM to dynamically decide when to stop the search, we employ a fixed number of nodes based on these averages. The search stops after retrieving this predefined number of nodes. For documents with many top-level nodes, the initial number of visited nodes $\mathcal{S}$ is limited to the preset value divided by the number of levels, ensuring sufficient room to search. The most relevant top-level nodes are selected using embedding similarity, similar to RAPTOR (Sarthi et al., 2024). We retain the same search mechanism using attention and embedding similarity, so unselected top-level nodes can still be retrieved via embedding similarity. Without Dynamic Progress Control, performance dropped across all datasets, even though the average number of nodes retrieved remained unchanged. Some queries retrieve unnecessary information, while others still lack the required information, leading to decreased performance. GARLIC dynamically adjusts the amount of information retrieved for each query, ensuring a proper amount is retrieved.

w/o Attention Search: We remove the use of attention and rely solely on embedding similarity for node search, similar to RAPTOR (Sarthi et al., 2024). 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 Embedding Similarity Search: We exclude embedding similarity from the final score $\boldsymbol{z}$, relying entirely on attention weights to compute retrieval scores. While performance decreased slightly, it was not a major drop. The score for NarrativeQA was not significantly affected, indicating that while embedding similarity is beneficial, attention-based search plays a more critical role.

4.5 Efficiency Analysis

Table 1 lists the TFLOPs for search and inference for each query. In this section, we provide a closer analysis of efficiency. Table 3 shows the graph construction TFLOPs per document as GARLIC *graph construction* and the graph search TFLOPs per query as GARLIC *graph search*. GARLIC *graph construction* + *graph search* shows the average TFLOPs per query when each document contains 2, 4, or 8 queries.

Table 3: TFLOPs for graph construction and graph search. GARLIC *graph construction* + *graph search* shows the average TFLOPs per query when each document contains 2, 4, or 8 queries.

Method	**NarrativeQA TFLOPs**	**Qasper TFLOPs**	**HotpotQA TFLOPs**	**MuSiQue TFLOPs**
Llama3.1-8B	3361.9	92.5	23.6	40.6
GARLIC graph construction	2042.8	136.6	40.2	66.7
GARLIC graph search	31.0	66.9	16.0	30.9
GARLIC graph construction + graph search				
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

If each document has only one query, the TFLOPs of our method exceed those of Llama3.1 on Qasper, HotpotQA, and MuSiQue. During graph construction, the entire document is processed by the GARLIC, along with the additional summary generation. However, Llama3.1 uses more TFLOPs on NarrativeQA than GARLIC as the complexity of the Transformer (Vaswani et al., 2017) increases quadratically with input length for very long inputs. Even though the total amount of text processed by our method is longer, Llama3.1 processes the entire input at once, whereas our method processes the document in chunks, resulting in lower TFLOPs. As the number of queries per document increases, the TFLOPs for graph construction are amortized, reducing the average TFLOPs per query. When each document has more than 8 queries, our method achieves lower average TFLOPs per query, even when accounting for the summary.

Additionally, GARLIC is able to use an input window of 8K, allowing it to run on an NVIDIA A100 80G GPU. In contrast, we had to use 8 A100 80G GPUs, even with CPU offloading, to run Llama3.1 on NarrativeQA with an input window of 100K tokens. This limits the practical application of Llama3.1 in real-world scenarios, whereas our method can easily run on a single GPU.

5 Conclusion and Future Work

In this paper, we propose a new retrieval method, called GARLIC, which constructs a Hierarchical Weighted Directed Acyclic Graph with many-to-many summarization. Each node represents an Information Point focusing on a single or few events, with edges derived from summarization attention. We also introduce a novel retrieval method using LLM attention weights and LLM-controlled retrieval, with efficiency maintained through KV caching. Our Dynamic Progress Control can be easily adapted to other retrieval methods. Experiments show our method outperforms baselines while maintaining retrieval computational efficiency, and increasing stop patience can further improve performance. There are some areas that could be improved. As we placed the query at the beginning, we had to apply an empirical normalization, which might not be optimal. Additionally, the equal weighting of attention-based and embedding similarity scores could be refined. Further work could explore incorporating Chain-of-Thought (CoT) reasoning to further enhance performance.

REFERENCES

Manoj Ghuhan Arivazhagan, Lan Liu, Peng Qi, Xinchi Chen, William Yang Wang, and Zhiheng Huang. Hybrid hierarchical retrieval for open-domain question answering. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 10680–10689, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.679. URL https://aclanthology.org/2023.findings-acl.679.

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

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. LongBench: A bilingual, multitask benchmark for long context understanding. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3119–3137, Bangkok, Thailand, August 2024b. Association for Computational Linguistics. URL https://aclanthology.org/2024.acl-long.172.

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. Improving language models by retrieving from trillions of tokens, 2022. URL https://arxiv.org/abs/2112.04426.

Howard Chen, Ramakanth Pasunuru, Jason Weston, and Asli Celikyilmaz. Walking down the memory maze: Beyond context limit through interactive reading, 2023a. URL https://arxiv.org/abs/2310.05029.

Shouyuan Chen, Sherman Wong, Liangjian Chen, and Yuandong Tian. Extending context window of large language models via positional interpolation, 2023b. URL https://arxiv.org/abs/2306.15595.

Yukang Chen, Shengju Qian, Haotian Tang, Xin Lai, Zhijian Liu, Song Han, and Jiaya Jia. Longlora: Efficient fine-tuning of long-context large language models, 2024. URL https://arxiv.org/abs/2309.12307.

Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A. Smith, and Matt Gardner. A dataset of information-seeking questions and answers anchored in research papers. In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (eds.), *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 4599–4610, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.365. URL https://aclanthology.org/2021.naacl-main.365.

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

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, Emily Dinan, Eric Michael

Smith, Filip Radenovic, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaoqing Ellen Tan, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aaron Grattafiori, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alex Vaughan, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Franco, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, Danny Wyatt, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Firat Ozgenel, Francesco Caggioni, Francisco Guzmán, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Govind Thattai, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Karthik Prasad, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kun Huang, Kunal Chawla, Kushal Lakhotia, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Maria Tsimpoukelli, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navy-

ata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikolay Pavlovich Laptev, Ning Dong, Ning Zhang, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Rohan Maheswari, Russ Howes, Ruty Rinott, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Kohler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vítor Albiero, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaofang Wang, Xiaojian Wu, Xiaolan Wang, Xide Xia, Xilun Wu, Xinbo Gao, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yuchen Hao, Yundi Qian, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, and Zhiwei Zhao. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Yao Fu, Rameswar Panda, Xinyao Niu, Xiang Yue, Hannaneh Hajishirzi, Yoon Kim, and Hao Peng. Data engineering for scaling language models to 128k context, 2024. URL https://arxiv.org/abs/2402.10171.

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

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Gautier Izacard and Edouard Grave. Distilling knowledge from reader to retriever for question answering. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=NTEz-6wysdb.

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

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. LLMLingua: Compressing prompts for accelerated inference of large language models. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 13358–13376, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.825. URL https://aclanthology.org/2023.emnlp-main.825.

Huiqiang Jiang, Qianhui Wu, , Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. LongLLMLingua: Accelerating and enhancing LLMs in long context scenarios via prompt compression. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1658–1677, Bangkok, Thailand, August 2024. Association for Computational Linguistics. URL https://aclanthology.org/2024.acl-long.91.

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

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, pp. 6769–6781, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.550. URL https://aclanthology.org/2020.emnlp-main.550.

Omar Khattab and Matei Zaharia. 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, pp. 39–48, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450380164. doi: 10.1145/3397271.3401075. URL https://doi.org/10.1145/3397271.3401075.

Tomáš Kočiský, Jonathan Schwarz, Phil Blunsom, Chris Dyer, Karl Moritz Hermann, Gábor Melis, and Edward Grefenstette. The NarrativeQA reading comprehension challenge. *Transactions of the Association for Computational Linguistics*, 6:317–328, 2018. doi: 10.1162/tacl_a_00023. URL https://aclanthology.org/Q18-1023.

Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pp. 74–81, Barcelona, Spain, July 2004. Association for Computational Linguistics. URL https://aclanthology.org/W04-1013.

Sheng-Chieh Lin, Akari Asai, Minghan Li, Barlas Oguz, Jimmy Lin, Yashar Mehdad, Wen-tau Yih, and Xilun Chen. How to train your dragon: Diverse augmentation towards generalizable dense retrieval. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 6385–6400, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-emnlp.423. URL https://aclanthology.org/2023.findings-emnlp.423.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024. doi: 10.1162/tacl_a_00638. URL https://aclanthology.org/2024.tacl-1.9.

Ye Liu, Kazuma Hashimoto, Yingbo Zhou, Semih Yavuz, Caiming Xiong, and Philip Yu. Dense hierarchical retrieval for open-domain question answering. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2021*, pp. 188–200, Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.findings-emnlp.19. URL https://aclanthology.org/2021.findings-emnlp.19.

Sewon Min, Kenton Lee, Ming-Wei Chang, Kristina Toutanova, and Hannaneh Hajishirzi. Joint passage ranking for diverse multi-answer retrieval. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 6997–7008, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.560. URL https://aclanthology.org/2021.emnlp-main.560.

Tsendsuren Munkhdalai, Manaal Faruqui, and Siddharth Gopal. Leave no context behind: Efficient infinite context transformers with infini-attention, 2024. URL https://arxiv.org/abs/2404.07143.

Inderjeet Nair, Aparna Garimella, Balaji Vasan Srinivasan, Natwar Modani, Niyati Chhaya, Srikrishna Karanam, and Sumit Shekhar. A neural CRF-based hierarchical approach for linear text segmentation. In Andreas Vlachos and Isabelle Augenstein (eds.), *Findings of the Association for Computational Linguistics: EACL 2023*, pp. 883–893, Dubrovnik, Croatia, May 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-eacl.65. URL https://aclanthology.org/2023.findings-eacl.65.

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. Text and code embeddings by contrastive pre-training, 2022. URL https://arxiv.org/abs/2201.10005.

Benjamin Newman, Luca Soldaini, Raymond Fok, Arman Cohan, and Kyle Lo. A question answering framework for decontextualizing user-facing snippets from scientific documents. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 3194–3212, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.193. URL https://aclanthology.org/2023.emnlp-main.193.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In Pierre Isabelle, Eugene Charniak, and Dekang Lin (eds.), *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pp. 311–318, Philadelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics. doi: 10.3115/1073083.1073135. URL https://aclanthology.org/P02-1040.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. YaRN: Efficient context window extension of large language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=wHBfxhZu1u.

Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (eds.), *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, pp. 3982–3992, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1410. URL https://aclanthology.org/D19-1410.

Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Found. Trends Inf. Retr.*, 3(4):333–389, April 2009. ISSN 1554-0669. doi: 10.1561/1500000019. URL https://doi.org/10.1561/1500000019.

Stephen Robertson, S. Walker, S. Jones, M. M. Hancock-Beaulieu, and M. Gatford. Okapi at trec-3. In *Overview of the Third Text REtrieval Conference (TREC-3)*, pp. 109–126. Gaithersburg, MD: NIST, January 1995. URL https://www.microsoft.com/en-us/research/publication/okapi-at-trec-3/.

Keshav Santhanam, Omar Khattab, Jon Saad-Falcon, Christopher Potts, and Matei Zaharia. ColBERTv2: Effective and efficient retrieval via lightweight late interaction. In Marine Carpuat, Marie-Catherine de Marneffe, and Ivan Vladimir Meza Ruiz (eds.), *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 3715–3734, Seattle, United States, July 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.naacl-main.272. URL https://aclanthology.org/2022.naacl-main.272.

Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D Manning. RAPTOR: Recursive abstractive processing for tree-organized retrieval. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=GN921JHCRw.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding, 2023. URL https://arxiv.org/abs/2104.09864.

Zhiqing Sun, Xuezhi Wang, Yi Tay, Yiming Yang, and Denny Zhou. Recitation-augmented language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=-cqvvvb-NkI.

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. Transformer memory as a differentiable search index. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=Vu-B0clPfq.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023a. URL https://arxiv.org/abs/2302.13971.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023b. URL https://arxiv.org/abs/2307.09288.

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

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

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. Shall we pretrain autoregressive language models with retrieval? a comprehensive study. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 7763–7786, Singapore, December 2023a. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.482. URL https://aclanthology.org/2023.emnlp-main.482.

Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. SimLM: Pre-training with representation bottleneck for dense passage retrieval. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2244–2258, Toronto, Canada, July 2023b. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.125. URL https://aclanthology.org/2023.acl-long.125.

Yu Wang, Nedim Lipka, Ryan A. Rossi, Alexa Siu, Ruiyi Zhang, and Tyler Derr. Knowledge graph prompting for multi-document question answering, 2023c. URL https://arxiv.org/abs/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. A neural corpus indexer for document retrieval. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=fSfcEYQP_qc.

Wenhao Wu, Yizhong Wang, Yao Fu, Xiang Yue, Dawei Zhu, and Sujian Li. Long context alignment with short instructions and synthesized positions, 2024. URL https://arxiv.org/abs/2405.03939.

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

Wenhao Yu, Dan Iter, Shuohang Wang, Yichong Xu, Mingxuan Ju, Soumya Sanyal, Chenguang Zhu, Michael Zeng, and Meng Jiang. Generate rather than retrieve: Large language models are strong context generators. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=fB0hRu9GZUS.

Shiyue Zhang, David Wan, and Mohit Bansal. Extractive is not faithful: An investigation of broad unfaithfulness problems in extractive summarization. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2153–2174, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.120. URL https://aclanthology.org/2023.acl-long.120.

Dawei Zhu, Nan Yang, Liang Wang, Yifan Song, Wenhao Wu, Furu Wei, and Sujian Li. PoSE: Efficient context window extension of LLMs via positional skip-wise training. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=3Z1gxuAQrA.

A PROMPT

A.1 SUMMARY GRAPH CONSTRUCTION PROMPT

In *Summary Graph Construction*, the prompt instructs the LLM to generate summaries in bullet-point format, with each point referred to as an Information Point (IP). The LLM prompt used is shown in Table 4, with an example from NarrativeQA (Kočiský et al., 2018). In the prompt, we use the term "summary point" to refer to the IPs in the response, distinguishing them from the IPs in the input for LLM. Each IP in our method describes a single or very few events, and during retrieval, each IP, corresponding to each node, reduces the smallest retrievable unit from a chunk to an IP. An IP usually consists of just one sentence, allowing our method to retrieve specific pieces of knowledge more efficiently.

Table 4: The prompt used for the LLM during *Summary Graph Construction*, with an example from NarrativeQA. The *italicized* text represents the example context, while the remaining text is the prompt instruction. To differentiate between the input and output IPs, the output IPs are named as "summary points" in the prompt to avoid confusion for LLM. Each summary point starts with an asterisk and typically consists of a single sentence.

Summary 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.* .....

A.2 DYNAMIC GRAPH SEARCH PROMPT

The two-turn prompt is shown in Table 5 in Appendix A.2, including 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 visited nodes $\mathcal{S}$. If the response is "`No`", the search continues, and the next node is retrieved. This process repeats until the LLM responds with "`Yes`". Once the LLM responds with "`Yes`", the second turn of the prompt will ask the LLM to answer the query.

Table 5: The two-turn prompt used for *Dynamic Graph Search*, with an example from NarrativeQA. The *italicized* text represents the example context, while the remaining text is the prompt instruction. The first-turn prompt asks the LLM to determine if the current context is sufficient to answer the query. If the response is "`No`," more context will be added to the end of the prompt, and the LLM will be asked again, with all previous context KV cached to avoid additional computation. Once the LLM responds with "`Yes`," the second turn will prompt the LLM to provide the final answer directly.

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.*

B Attention Averaging and Normalization

This section introduces the detailed computation of the final attention weight $e_{i,j} \in \mathcal{E}$ between a high-level node v_i^{l+1} and a low-level node v_j^l, as introduced in Section 3.1. We define the lower-level node v_j^l with tokens $\{x_k^j\}_{k=1}^{K_j}$ and the higher-level node v_i^{l+1} with tokens $\{x_k^i\}_{k=1}^{K_i}$, where K_j and K_i denote the number of tokens in nodes v_j^l and v_i^{l+1}, respectively.

First, we extract all attention weights from the LLM during summarization. These attention weights are averaged across attention heads and layers. In practice, we iteratively accumulate attention weights averaged over attention heads for each layer to reduce memory usage. This process yields the attention weight e_{x^i,x^j} between tokens $x^i \in v_i^{l+1}$ and $x^j \in v_j^l$.

Next, we average attention weights at the token level within each node. For the lower-level node v_j^l, we average $\{e_{x^i,x_k^j}\}_{k=1}^{K_j}$ into e_{x^i,v_j^l}, which represents the attention weight from token $x^i \in v_i^{l+1}$ to the node v_j^l. Similarly, for the higher-level node v_i^{l+1}, we average $\{e_{x_k^i,v_j^l}\}_{k=1}^{K_i}$ into $e_{v_i^{l+1},v_j^l}$. For simplicity, we refer to $e_{v_i^{l+1},v_j^l}$ as $e_{i,j}$ in Section 3.1.

This process is repeated for all nodes. If no connection exists between two nodes, the edge weight is set to zero. Finally, the attention weights are normalized as $e_{i,j} = \frac{e_{i,j}}{\sum_j e_{i,j}}$, ensuring that $\sum_j e_{i,j} = 1$. This provides the final attention weights for the edges $\mathcal{E}$ in the graph $\mathcal{G}$.

We classify attention into two types: syntactic attention, which reflects grammatical relationships, and semantic attention, which captures meaning connections, often between paragraphs. When averaging token-level attentions, we only extract attentions between high-level and low-level nodes, omitting attentions within the same node. This process emphasizes long-distance semantic attention, which effectively reflects the semantic relationships between nodes.

C KV CACHING IN DYNAMIC PROGRESS CONTROL

This section introduces the usage of the KV cache described in Section 3.2.1. Given the last retrieved node v_i in the visited set $\mathcal{S}$, let $\{x_k^i\}_{k=1}^{K_i}$ denote the tokens of node v_i, where K_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 $[x_1^q, \ldots, x_{K_q}^q, x_1^1, \ldots, x_{K_1}^1, \ldots, x_1^i, \ldots, x_{K_i}^i]$.

When a new node v_j is retrieved, the query, key, and value states of the tokens $\{x_k^j\}_{k=1}^{K_j}$ in v_j are computed using the self-attention mechanism of the LLM. Since v_j also attend to the previously visited nodes, the stored KV cache containing the tokens $[x_1^q, \ldots, x_{K_q}^q, x_1^1, \ldots, x_{K_1}^1, \ldots, x_1^i, \ldots, x_{K_i}^i]$ is input into 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. The LLM performs self-attention over these query, key, and value states.

After processing, v_j is added to the visited set $\mathcal{S}$, and the key-value states of its tokens $\{x_k^j\}_{k=1}^{K_j}$ are stored. These states are concatenated with the previous KV cache to form $[x_1^q, \ldots, x_{K_q}^q, x_1^1, \ldots, x_{K_1}^1, \ldots, x_1^i, \ldots, x_{K_i}^i, x_1^j, \ldots, x_{K_j}^j]$. This updated KV cache is then used for the next node retrieval. In this common scenario, KV caching is typically performed at the token level, where the LLM caches the key-value states of previous tokens when generating the next token. In contrast, our approach utilizes a node-level KV cache for 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. The LLM follows standard decoding to generate each token one at a time, leveraging the KV cache of the previous tokens. The query, key, and value states for all nodes, the query, and the answers are computed only once throughout the retrieval process to avoid additional computation.

D SUMMARY GRAPH EXAMPLE

In this section, we present an example of generated IPs with part of the first and second-level nodes in Figure 5 from HotpotQA (Yang et al., 2018). The nodes on the right are higher-level IPs generated from the low-level chunk nodes on the left. The connections in the middle represent attention weights. The higher the attention weight, the redder and thicker the line. Lines with attention weights less than 0.05 are omitted from the figure. It can be observed that IPs are generated from multiple chunks. For example, high-level nodes 10, 13, 14, 15, 16, 17, 18, and 19 are connected to multiple low-level nodes. In contrast, high-level nodes 11 and 20 mainly rely on single low-level nodes, with minimal connections to other low-level nodes. From the perspective of low-level nodes, nodes 2, 5, 6, 7, 8, and 9 are connected to many high-level nodes. Meanwhile, low-level node 3 is only attended to by node 17. Different nodes automatically adjust their relevance based on attention with other nodes. Some nodes connect to multiple nodes, while others connect to a single node. Our method uses IPs and attention to make the relationships between low-level nodes and higher-level nodes more explicit, allowing the model to understand how each piece of information is connected to different parts of the text through the graph.

E DATASETS STATISTICS

Table 6 shows the token length statistics of the datasets NarrativeQA (Kočiský et al., 2018), Qasper (Dasigi et al., 2021), HotpotQA (Yang et al., 2018), and MuSiQue (Trivedi et al., 2022). NarrativeQA is much longer than the other datasets, followed by Qasper. HotpotQA and MuSiQue are relatively shorter.

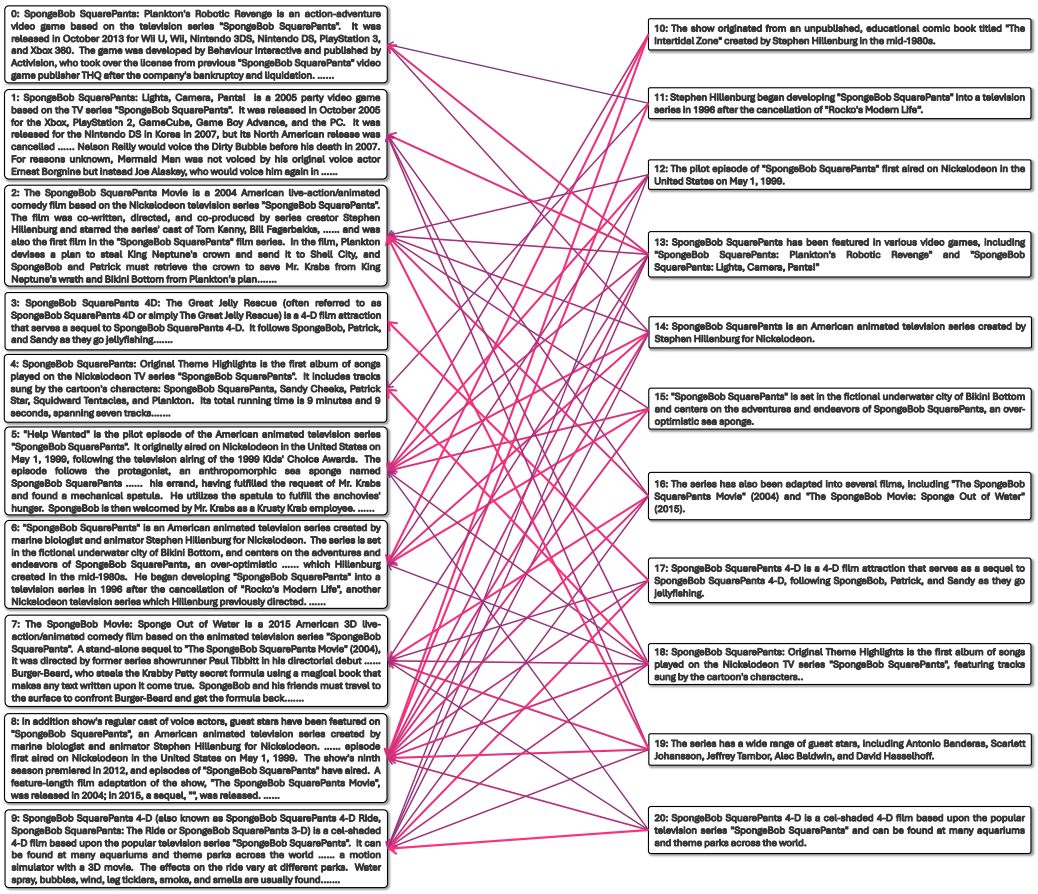


Figure 5: An example of generated IPs with part of the first and second-level nodes from HotpotQA. The nodes on the right are higher-level IPs generated from the low-level chunk nodes on the left. The connections in the middle represent attention weights. The higher the attention weight, the redder and thicker the line. Lines with attention weights less than 0.05 are omitted from the figure. For brevity, some long text in the nodes on the left is omitted.

Table 6: Token length statistics for the NarrativeQA, Qasper, HotpotQA, and MuSiQue datasets, including the average, minimum, and maximum token length per document.

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