Generative Multi-hop Retrieval

Hyunji Lee **Sohee Yang** **Hanseok Oh** **Minjoon Seo**
KAIST AI
{hyunji.amy.lee, sohee.yang, hanseok, minjoon}@kaist.ac.kr

Abstract

A common practice for text retrieval is to use an encoder to map the documents and the query to a common vector space and perform a nearest neighbor search (NNS); multi-hop retrieval also often adopts the same paradigm, usually with a modification of iteratively reformulating the query vector so that it can retrieve different documents at each hop. However, such a bi-encoder approach has limitations in multi-hop settings; (1) the reformulated query gets longer as the number of hops increases, which further tightens the embedding bottleneck of the query vector, and (2) it is prone to error propagation. In this paper, we focus on alleviating these limitations in multi-hop settings by formulating the problem in a fully generative way. We propose an encoder-decoder model that performs multi-hop retrieval by simply *generating* the entire text sequences of the retrieval targets, which means the query and the documents interact in the language model's parametric space rather than L2 or inner product space as in the bi-encoder approach. Our approach, Generative Multi-hop Retrieval (GMR), consistently achieves comparable or higher performance than bi-encoder models in five datasets while demonstrating superior GPU memory and storage footprint.[1]

1 Introduction

Finding the relevant knowledge from a massive collection of information is often formulated as a text retrieval problem. A large portion of the text retrieval literature focuses on finding the single most relevant paragraph or document (i.e., no hop) to the given query (Karpukhin et al., 2020; Chen et al., 2017). When we cannot answer a query with a single document, the task is often formulated as a multi-hop retrieval problem, where one needs to retrieve multiple documents that together provide sufficient evidence to answer the query (Yang et al., 2018; Joshi et al., 2017; Dalvi et al., 2021). For example, to answer the question "Where did the form of music played by Die Rhöner Säuwäntzt originate?" (Figure 1), we first need to retrieve *the form of music* played by Die Rhöner Säuwäntzt and then *where the form originated from*.

No-hop and multi-hop retrieval tasks are often approached by encoding both the query and retrieval sequences to a common vector space and then finding the sequence whose embedding is closest to the query. This bi-encoder approach for retrieval is often considered as a *de facto* standard; heavy computations such as extracting the dense embeddings of the items in the corpus can be done offline, and one can search over a large number of items with low latency through the nearest neighbor search (NNS) or maximum inner product search (MIPS) (Lewis et al., 2020; Chen et al., 2020; Wu et al., 2020; Roller et al., 2021). While such a bi-encoder approach performs well on many retrieval tasks, it has also shown to suffer from information loss when encoding a long query or document into a fixed-size embedding (Luan et al., 2021; Izacard et al., 2020). The problem becomes even more critical in multi-hop retrieval as previously retrieved items are appended to the query while iterating through multiple hops. The augmented query gets longer as the number of hops increases; therefore, the query embedding gradually becomes incapable of containing the entire information.

In this paper, we argue that a fully generative approach to multi-hop retrieval may be the solution; it overcomes the bottleneck problem by interacting in the whole parametric space of the model trained on the target corpus during the retrieval process, rather than operating on L2 or inner product space as in bi-encoder approach (Figure 1). We propose Generative Multi-hop Retrieval (GMR), an encoder-decoder model that attempts to memorize the entire target corpus in a generative man-

[1]https://github.com/amy-hyunji/Generative-Multihop-Retrieval

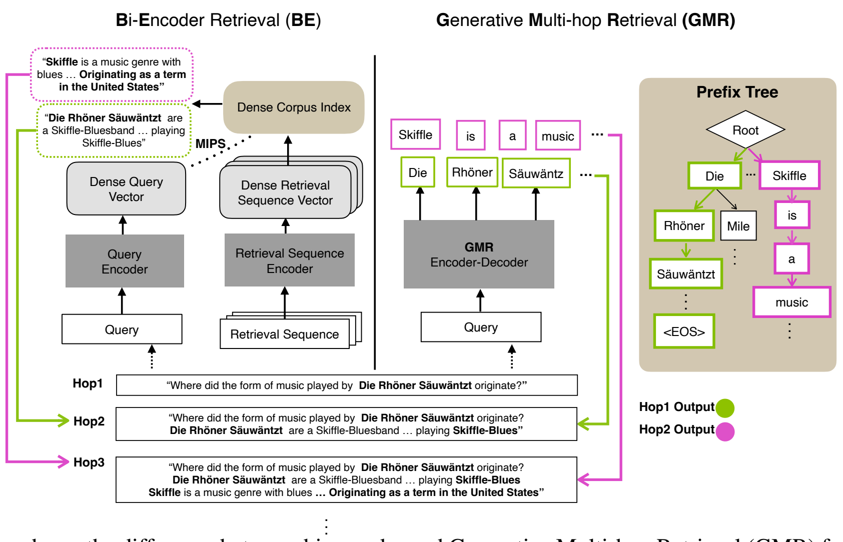

Bi-Encoder Retrieval (BE)
Generative Multi-hop Retrieval (GMR)
"Skiffle is a music genre with blues … Originating as a term in the United States"
"Die Rhöner Säuwäntzt are a Skiffle-Bluesband … playing Skiffle-Blues"
Dense Corpus Index
MIPS
Dense Query Vector
Dense Retrieval Sequence Vector
Query Encoder
Retrieval Sequence Encoder
Query
Retrieval Sequence
Skiffle
is
a
music
Die
Rhöner
Säuwäntz
GMR Encoder-Decoder
Query
Prefix Tree
Root
Die
Skiffle
Rhöner
Mile
is
a
Säuwäntzt
music
<EOS>
Hop1
"Where did the form of music played by Die Rhöner Säuwäntzt originate?"
Hop2
"Where did the form of music played by Die Rhöner Säuwäntzt originate? Die Rhöner Säuwäntzt are a Skiffle-Bluesband … playing Skiffle-Blues"
Hop3
"Where did the form of music played by Die Rhöner Säuwäntzt originate? Die Rhöner Säuwäntzt are a Skiffle-Bluesband … playing Skiffle-Blues Skiffle is a music genre with blues … Originating as a term in the United States"
Hop1 Output
Hop2 Output

Figure 1: The figure shows the difference between bi-encoder and Generative Multi-hop Retrieval (GMR) for multi-hop retrieval. In the first hop retrieval of GMR, to generate the second token, *Rhöner*, it finds the potential next tokens ([Rhöner, Mille]) by searching through the prefix tree with the previously generated tokens. We mask out tokens that are not in the potential next tokens and find the token with the maximum score from unmasked tokens, which in this example is *Säuwäntzt*. Finally, when it retrieves the <*EOS*> token, the generation ends, and the generated output is the retrieval sequence of the query. In the second hop retrieval, the concatenation of the query and the previously retrieved sequence is the input query for both approaches.

ner and retrieves the most relevant sequence from the corpus by generating the *entire* sequence with the aid of constrained decoding. We also propose memorization methods (LM and multi-hop memorization) to encourage GMR to memorize the target corpus. Earlier work in generative retrieval (Tay et al., 2022; Cao et al., 2021; Bevilacqua et al., 2022) performs retrieval by generating the entity or the document id that represents the target paragraph or document; GMR instead generates the entire text of the target paragraph, which we believe is more suitable for multi-hop retrieval that requires modeling the interaction between longer queries and more fine-grained text segments.

The main contributions of our paper are that:

- We show the limitations of bi-encoder retrieval in multi-hop retrieval tasks: its performance decreases as the number of hops increases and is vulnerable to error propagation.
- We show that Generative Multi-hop Retrieval (GMR) is robust in solving multi-hop retrieval tasks, performing higher or comparable on five datasets. It is especially strong in multi-hop retrieval settings close to real-world scenarios and datasets with a low unseen rate.
- We introduce *multi-hop memorization* which effectively memorizes the target corpus and improves the performance of GMR.

Given that generative retrieval shows high performance with high storage efficiency in multi-hop retrieval task compared to the traditional bi-encoder approach, we suggest that generative retrieval has the potential to be a practical alternative for not only the no-hop text retrieval tasks, as shown in Tay et al. (2022); Bevilacqua et al. (2022), but also for multi-hop retrieval tasks as explored in this work.

2 Related Work

Multi-hop Retrieval Multi-hop retrieval, which answers a query by integrating multiple documents, is often necessary to solve complex queries; it is an active area of research due to its importance. There has been a line of previous works in multi-hop retrieval with non-textual metadata such as knowledge bases, Wikipedia hyperlinks, or entity linking which leverage such metadata to solve the tasks (Asai et al., 2020; Nie et al., 2019; Zhao, 2020; Dhingra et al., 2020). However, they are not expandable to cases where such metadata does not exist. Another line of research focuses on expanding the bi-encoder architecture which has shown high performance on no-hop retrieval to multi-hop retrieval (Xiong et al., 2021; Zhao et al., 2021). While such methods have shown good performance, previous studies (Luan et al., 2021; Izacard et al., 2020) show that bi-encoder approach suffers from information loss when condensing text into a fixed-size vector. Since the input text gets longer as the number of hops increases in multi-hop retrieval, it is highly likely for the bi-encoder to fall into the bottleneck problem when the number of hops is large. Therefore, to overcome such limitations, it is worth exploring the changes in the fundamental approach; our work suggests that a *generative* method can be an effective alternative to the bi-encoder approach for multi-hop retrieval tasks.

Generative Retrieval Cao et al. (2021) first propose a generative retrieval model, which achieves comparable or higher performances on entity retrieval tasks compared to bi-encoder models. Moreover, concurrent works DSI (Tay et al., 2022) and SEAL (Bevilacqua et al., 2022) show generative retrieval methods in no-hop retrieval settings. DSI (Differentiable Search Engine) generates structured identifiers of each corpus and shows higher performance than the bi-encoder approach in the NQ dataset. SEAL (Search Engines with Autoregressive LMs) retrieves an item (paragraph or document) by finding an item containing generated ngram using FM index. It tests on NQ and KILT benchmarks and shows that the generative retrieval model can even outperform well-designed bi-encoder models such as DPR (Karpukhin et al., 2020) and GAR (Mao et al., 2021)[2]. To see the effectiveness of explicitly generating the entire retrieval sequence, we compare GMR with our re-implementation of DSI[3] which we expand to a multi-hop retrieval setting for fair comparison.

3 Generative Multi-hop Retrieval

Multi-hop retrieval is a task of retrieving a *set* of sequences (e.g., sentences or paragraphs) from a target corpus D given a query x. It is often approached by iterating through multiple hops where the previously retrieved sequences are appended to the query and form an *augmented query* to model the relationship between the target sequences (Asai et al., 2020; Xiong et al., 2021; Khattab et al., 2021; Qi et al., 2021). In this paper, we focus on multi-hop retrieval tasks that resemble a real-world scenario: the oracle number of hops and the correct order of retrieval sequences are not given for each query at the inference time, and the number of oracle hops varies in a wide range.

Canonical text retrieval can be formulated as retrieving a sequence $d_{\hat{y}} = \arg\max_{d \in D} P(d|x)$, where x is the query, d is a retrieval sequence in the target corpus D, and $\hat{y}$ is the index of retrieved sequence in D. The retrieval is considered successful if $d_{\hat{y}} = d_y$ where d_y is the ground truth target. On the other hand, multi-hop retrieval aims on finding a *set* of sequences retrieved through k hops, $\mathscr{D}_{\hat{y}} = \{d_{\hat{y}_1}, \cdots, d_{\hat{y}_k}\}$, given the query. Here, $d_{\hat{y}_i}$ is the sequence retrieved at the i-th hop, $d_{\hat{y}_i} = \arg\max_{d \in D} P(d|x, d_{\hat{y}_{<i}})$, where x is the query and $d_{\hat{y}_{<i}}$ is sequences retrieved at previous hops. As the canonical text retrieval of a bi-encoder approach is modeled as $\arg\max_{d \in D} P(d|x) \propto \arg\max_{d \in D} F(d) \cdot G(x)$, which is the inner product between the query vector from an encoder G and the retrieval sequence vector from an encoder F, its extension to multi-hop is defined as $P(d|x, d_{\hat{y}_{<i}}) \propto F(d) \cdot G(x, d_{\hat{y}_{<i}})$. As the number of hops increases, the augmented query $(x, d_{\hat{y}_{<i}})$ gets longer, and it increases the burden of query encoder G to encode the long augmented query into a fixed-size vector. We investigate that such a burden on the query encoder worsens the *bottleneck problem* of the bi-encoder model and that such models are vulnerable to *error propagation* (Section 5.2).

To alleviate such limitations of the bi-encoder approach in a multi-hop setting, we formulate the problem in a fully *generative* way; the generative approach can interact in the whole parametric space of the model trained on the target corpus during the retrieval process rather than operating only on L2 or inner product space as in bi-encoder approach. We propose Generative Multi-hop Retrieval (GMR), an encoder-decoder model that retrieves the most relevant sequence at each hop from the target corpus by *generating* the sequence using constrained decoding as in the right side of Figure 1. The generation goes over multiple hops to retrieve a set of sequences. In training time, the objective is to maximize:

$$P((d_{y_1}, \cdots, d_{y_n})|x) \propto \prod_{i=1}^{n} P(d_{y_i}|x, d_{y_{<i}}) \quad (1)$$

$$= \prod_{i=1}^{n} \prod_{j=1}^{|d_{y_i}|} P(d_{y_i}^{(j)}|x, d_{y_{<i}}, d_{y_i}^{(<j)}) \quad (2)$$

to generate the tokens $d_{y_i}^{(j)}$ of the ground truth text to retrieve d_{y_i} at retrieval hops $i = 1, \cdots, n$,[4] when the query x and the ground truth target sequences of the previous hops $d_{y_{<i}}$ are given as an input to the encoder and all tokens up to the previous step at the current hop ($d_{y_i}^{(<j)}$) are given as the input to the decoder. In inference time, GMR

[2]Note that SEAL and GENRE perform multi-hop QA datasets in no-hop retrieval setting.

[3]We re-implement DSI as the source code is not released (Appendix C.1.4).

[4]While multi-hop retrieval targets to retrieve a *set* of sequences, it is oftentimes difficult to train over all possible permutations of the set. Therefore, most previous works assume the order heuristically and train the model.

decides the sequence to retrieve by $P(d|x, d_{\hat{y}_{<i}}) \propto \prod_{j=1}^{|d|} P(d^{(j)}|x, d_{\hat{y}_{<i}}, d^{(<j)})$, i.e. the probability of generating the token $d^{(j)}$ conditioned on the query x, text sequences $d_{\hat{y}_{<i}}$ retrieved until the i-th hop, and the tokens $d^{(<j)}$ previously generated at the current hop. To ensure that the generated sequence is in the corpus, we build a prefix tree and perform constrained decoding with the tree (Cao et al., 2021)[5].

Since GMR generates a sequence in a uni-directional way (left to right) during the retrieval process, it cannot know the information at the end of a sequence in advance if the model has not been previously trained to generate the sequence. However, the training set for the target multi-hop retrieval task may not cover all the sequences in the corpus. This may negatively affect the performance, especially when the length of the sequence is long and the training set does not contain enough sequences in the target corpus. To alleviate the issue, we propose *LM memorization* and *multi-hop memorization*, two corpus memorization methods that aim to store the target corpus in the parameters. By training GMR with the methods, it is able to leverage the memorized information in the parameters on multi-hop retrieval tasks (Appendix A).

LM memorization LM (Language Modeling) memorization is an intermediate task applied before training on a multi-hop retrieval task. During LM memorization, GMR is trained on the texts in the corpus using the standard LM objective function: when a corpus D with texts d ($d \in D$) is given, the model learns to maximize the LM probability $P(d) = \prod_{j=1}^{|d|} P(d^{(j)}|d^{(<j)})$ for all d in D. By training the retrieval task on top of the parameters trained on LM memorization, the model is able to be aware of the contents at the end of the sequence it generates beforehand since it has seen the sequence during the LM memorization. To make the input of LM memorization similar to that of multi-hop retrieval task, we make the first m (randomly chosen) tokens of the text to generate to serve as the encoder input when maximizing $P(d)$ so that the model is trained to maximize $P(d^{(\geq m)}|d^{(<m)}) = \prod_{j=m}^{|d|} P(d^{(j)}|d^{(<m)}, d^{(\geq m,<j)})$ where $d^{(<m)}$ is the input to the encoder.

[5]The prefix tree is built by aggregating the tokenization result of texts in the corpus. Tokens that create strings that are not a sub-string of any text in the corpus are masked out, and only the next top-k tokens from the unmasked and thus valid set of tokens are passed to the model as the potential next tokens list.

Multi-hop Memorization While LM memorization has the benefit that it can be easily applied to GMR on all datasets, one limitation is that it is an *unconditional* generation task (i.e., not depending on a query) while multi-hop retrieval is a conditional generation task where a query is always given. Therefore, we propose an advanced *conditional* variant: multi-hop memorization. Multi-hop memorization is a task of maximizing $P(D'_y|x')$ where x' is a pseudo-multi-hop query generated from a query generation model Q[6] and $D'_y = (d_{y'_1}, \cdots, d_{y'_n})$ is a list of pseudo target sequences.

We perform data augmentation to construct the training data for multi-hop memorization, $\{(x', D'_y\}$. First, using the original retrieval dataset $\{(x, D_y)\}$, we train a query generator Q to generate query x given the concatenation of ground truth target sequences for the query. After training Q, we sample pseudo-target sequences $D_{y'} = (d_{y'_1}, \cdots, d_{y'_k})$ from the target corpus D and generate the corresponding pseudo query x' by feeding $D_{y'}$ to Q. This sampling-generation step is repeated to create enough set of pairs $\{(x', D'_y)\}$.

For the method to be beneficial for multi-hop retrieval, we simulate the target distribution of the original dataset D_y when sampling $D_{y'}$: each adjoining $d_{y'_i}$ and $d_{y'_{i+1}}$ in $D_{y'}$ are relevant. Therefore, we first find important words or phrases (e.g., entity, subject) I_d for each sequence $d \in D$. Then, for all $d \in D$, we construct $D_{y'} = (d_{y'_1}, \cdots, d_{y'_k})$ by first setting $d_{y'_1} = d$. This is an iterative process that $d_{y'_{i+1}}$ is sampled from the set of text sequences $\{d' : |I_{d_{y'_i}} \bigcap I_{d'}| > 1\}$, stopped when $|D_{y'}| = k$ where k is randomly chosen.

Following the described approach, the constructed dataset $\{(x', D'_y)\}$ becomes similar to the original data. We apply filtering to ensure the quality (Appendix A). Since the objective function of multi-hop memorization has the same form with that of multi-hop retrieval, we perform the training in a multi-task manner, using $\{(x', D'_y)\}$ and $\{(x, D_y)\}$ together as the training data at once, rather than the two-phase training as in LM memorization.

[6]We use pre-trained T5-large to initialize Q.

4 Experimental Setup

4.1 Fixed and Dynamic Multi-hop Retrieval

We formulate two settings of multi-hop retrieval tasks: fixed and dynamic multi-hop retrieval settings. Our ultimate goal of multi-hop retrieval tasks in the inference step is to retrieve a set of relevant items when given an input query x. However, since k, the oracle number of items in a set, varies depending on x and the task, it is difficult to know k beforehand in a real-world scenario. Therefore, in most cases, k is fixed to a certain number.

Fixed and Dynamic multi-hop retrieval settings differ by whether the retrieval process continues until the maximum retrieval hops k or stops in the middle (Appendix B.1). Fixed setting is commonly used in previous multi-hop retrieval tasks, which a model retrieves till the maximum retrieval hop. Whereas dynamic setting is more applicable to solving multi-hop retrieval tasks close to a real-world scenario; rather than iterating until the given maximum number of hops, the model itself predicts when to stop the process by generating the special token (*DONE*) and stops in the middle.

4.2 Datasets

We use five datasets with various characteristics (Appendix B.2).

HotpotQA (Yang et al., 2018) is an open domain multi-hop question answering dataset, which requires two Wikipedia pages to answer the query.

Entailment TreeBank (EntailBank) (Dalvi et al., 2021) is a reasoning tree construction task where it forms a tree with a hypothesis as the root node and evidence sentences as leaf nodes. We experiment on Task3: retrieve leaf nodes from the corpus when given a question and an answer as an input.

StrategyQA (Geva et al., 2021) is an open-domain multi-hop question answering dataset where the reasoning steps are implicit in the question. It requires strategies to answer the question.

Explagraphs-Open (EG-Open) (Saha et al., 2021) is a generative and structured commonsense-reasoning task. We reformulate it to open-domain retrieval task (Explagraphs-Open), which considers a single path (subject-relation-object) as a retrieval sequence.

RuleTaker-Open (RT-Open) (Clark et al., 2021) is a synthetic rule-based dataset to measure the model's reasoning ability over rules. We reformulate it to open-domain retrieval task (RuleTaker-Open) which considers nodes of the graph (sentences) as a retrieval sequence (Appendix B.3).

4.3 Bi-Encoder Retrieval Models

For each dataset, we compare the results with a bi-encoder retrieval model (BE) as a baseline. For the HotpotQA dataset, we use MDR (Xiong et al., 2021), a widely used bi-encoder model for the corresponding dataset. For the rest of the datasets, we compare with ST5 (Ni et al., 2021).

We train and inference BE similar to GMR for both fixed and dynamic settings. In a fixed setting, BE maximize $P(d_{y_i}|x, d_{y_{<i}})$ by concatenating the query x and the retrieval sequences of the previous steps $d_{y_{<i}}$ as an input to the query encoder. In a dynamic setting, we add a special token *DONE* to the corpus, and BE is trained to retrieve the special token after the last hop (Appendix B.4).

MDR is an iterative bi-encoder retrieval model which extends DPR (Karpukhin et al., 2020) to a multi-hop retrieval.

ST5 is an encoder-decoder model[7] that serves as our baseline bi-encoder to compare the performance with GMR using the same number of parameters and architecture, T5 (Raffel et al., 2020).

4.4 Evaluation Metric

In the fixed multi-hop retrieval, we evaluate HotpotQA following the MDR evaluation metric[8]. For the rest, we first calculate the recall rate (R@k) of each query and average over the number of queries (Dalvi et al., 2021; Saha et al., 2021). In the dynamic multi-hop retrieval, since the number of predicted retrieval sequences varies, we measure the F1 score (F1@k) by retrieving a maximum of k sequences. For RT-Open, we newly define an evaluation metric (Appendix B.3) that measures the graph construction success rate.

5 Experimental Results

In Section 5.1, we compare the results of GMR and bi-encoder models in fixed and dynamic settings with five different datasets. In Section 5.2, we show the limitations of bi-encoder retrieval models, discuss the effect of unseen rate in GMR, and show GMR's efficiency on storage and inference time.

[7] We use ST5-EncDec which extracts sequence embedding by the first output of decoder

[8] https://github.com/facebookresearch/MDR

Table 1: Recall rate (R@5) of fixed setting and F1 score (F1@5, F1@10, F1@20 where each number indicates the maximum retrieval step) of dynamic setting on the test set. We compare results between GMR and ST5 (bi-encoder retrieval) where GMR outperforms ST5 for all four datasets. GMR_L is GMR with LM memorization, and GMR_M is GMR with Multi-hop memorization. The bold text shows the best score of each dataset. Results with * are evaluated by success rate (Appendix B.3).

	EntailTree			StrategyQA				EG-Open				RT-Open*		
	ST5	GMR	GMR_L	ST5	GMR	GMR_L	GMR_M	ST5	GMR	GMR_L	GMR_M	ST5	GMR	GMR_L
Fixed R@5	31.5	53.6	**54.3**	37.4	44.9	45.5	**45.6**	27.0	32.9	32.4	**34.6**	-	-	-
Dynamic F1@5	24.9	**48.2**	47.4	38.1	41.9	42.6	**43.1**	25.0	35.5	35.7	**36.2**	-	-	-
Dynamic F1@10	19.4	**52.1**	51.7	36.9	44.3	45.0	**45.2**	24.6	40.0	40.8	**42.1**	-	-	-
Dynamic F1@20	16.9	**52.5**	52.2	36.5	46.6	47.1	**47.9**	25.4	41.5	41.3	**42.6**	17.0	51.0	**65.5**

Table 2: Recall rate of HotpotQA official full-wiki dev set. Scores of DPR, MDR- and MDR are from Table 3 of Xiong et al. (2021). MDR- indicates a variant of MDR without linked negatives, memory bank, and shared encoder.

Method	DPR	MDR-	MDR	fix-GMR	fix-GMR_L
Top-2	25.2	59.9	65.9	57.7	55.0
Top-10	45.4	70.6	77.5	68.8	65.3
Top-20	52.1	73.1	80.2	73.9	71.4

Table 3: Recall rate (R@5) of fixed setting on the test set. We compare results between GMR and DSI* to show the effectiveness of explicitly generating the entire sequence in a multi-hop retrieval task. GMR outperforms both DSI* models on all three datasets.

Model	EntailTree	StrategyQA	EG-Open
atomic-DSI*	28.0	-	23.4
naive-DSI*	7.7	-	8.6
fix-GMR	53.6	44.9	32.9

5.1 Results

Bi-Encoder (BE) vs. GMR Table 1 shows the overall performance of the bi-encoder baseline (BE) and GMR variants on four datasets (EntailTree, EG-Open, StrategyQA, RT-Open) in fixed and dynamic multi-hop retrieval settings. We further compare results between our base model (GMR) and GMR with memorization methods: multi-hop memorization (GMR_M) and LM memorization (GMR_L). Across all datasets, GMR consistently shows a higher recall rate of top-5 in the fixed setting and a higher F1 score in the dynamic setting than bi-encoder models. Also, in most cases, both LM memorization and multi-hop memorization methods help improve the performance of GMR (Section 5.2). Moreover, the dynamic setting consistently outperforms the fixed setting for both BE and GMR (Appendix C.1.1), which suggests that the dynamic setting is more adaptable to multi-hop retrieval with a larger number of hops.

Table 2 compares the result between GMR and MDR (Xiong et al., 2021) on HotpotQA. While the score of GMR is lower than that of MDR, it is comparable to MDR- (a variant of MDR without linked negative, memory bank, and shared encoder). One reason why the performance of GMR is similar to MDR-, not MDR, would be that the techniques such as hard negative training or memory bank are crucial for higher performance yet are not applicable to GMR; this suggests an important future direction to close the gap. Also, since HotpotQA is a fixed to *two*-hop setting, bi-encoder models would suffer less from bottleneck and error propagation problems (Appendix 5.2) compared to the other datasets that require larger numbers of hops. Results in Table 1 on RT-Open dataset, a task to construct a reasoning graph for the given query in the dynamic setting, suggest that GMR is strong at retrieving sequences interdependent to one another. GMR and *GMR_M* outperform BE on success rate[9] by 300% and 385%, respectively, and construct more complex and diverse reasoning graphs through the retrieval process (Appendix C.1.2).

While BE needs to create and store a large index of embeddings and often loads it on GPUs for low latency, GMR only needs to create a prefix tree on CPUs, which leads to higher efficiency on offline computation, storage (Appendix C.1.3), and GPU memory; GMR shows 69.7% and 79.5% decrease of storage and GPU memory, respectively, compared to BE with the same number of parameters (ST5). During inference time, GMR can be time-inefficient if it has to generate every word in the retrieval target text. In practice, however, one can stop generation as soon as the partially generated text can uniquely identify the target text. By leveraging the optimization, GMR with greedy search is able to achieve a 40% inference time reduction with respect to ST5 in HotpotQA. Note that without the optimization, GMR is 24.6 times slower than ST5, signifying the importance of early stopping.

[9]F1 cannot be calculated on RT-Open because the ground truth retrieval sequence is not known at each step.

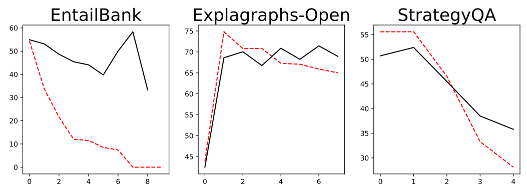


Figure 2: Hop-R@5_{oracle} (y-axis) over number of hops (x-axis). The red dotted line and the solid black line show performance of the ST5 (bi-encoder) and GMR, respectively. For all three datasets, ST5 tends to degrade as the number of hops increases, whereas GMR shows relatively consistent performance.

Importance of Explicit Generation in Multi-hop Retrieval Task GMR performs retrieval by explicitly generating the *entire* retrieval sequence using constrained decoding, unlike the previous generative retrieval methods, in order for the retrieval model to better grasp and understand the relationship between the input query and retrieval sequences. We compare GMR with our implementation of DSI (Tay et al., 2022), a concurrent work that assigns an id for each document in the corpus and retrieves relevant documents by generating ids. We expand DSI (which experiments only on no-hop settings) to multi-hop settings to retrieve the id of a relevant document and construct an augmented query by adding the text of the retrieved id at the end of the input query as in GMR. Table 3[10] shows that GMR outperforms DSI on all datasets, implying the benefit of generating the entire sequence in a multi-hop retrieval task. GENRE (Cao et al., 2021), which performs document retrieval by generating the title of the target Wikipedia page with constrained decoding, is not directly applicable to our multi-hop settings since most retrieval sequences in the datasets do not have such titles.

5.2 Analysis

Limitation of Bi-Encoder Retrieval Models We investigate limitations of bi-encoder in multi-hop retrieval and show (1) *bottleneck problem*: performance of bi-encoder consistently decreases as the number of hops increases, and (2) *error propagation*: the bi-encoder approach is more vulnerable to error propagation than generative approach.

For ease of analysis, we compare the performance of the bi-encoder retriever (ST5) and generative retriever (GMR) on three datasets (StrategyQA, EG-Open, EntailBank) under the setting where we assume that a ground truth order of the sequences to retrieve exists. The goal is to retrieve the one gold target sequence d_{y_i} of each i-th hop. The performance is measured as hop-R@$5_{\text{oracle}} = \mathbb{1}\{d_{y_i} \in \text{top-5}_{d \in D} P(d|x, d_{y_{<i}})\}$, where top-5 is a function that returns a set of the five sequences with the largest probabilities.

(1) Bottleneck Problem As shown in previous works (Luan et al., 2021; Izacard et al., 2020), bi-encoder approaches have an inherent limitation that their performance degrades proportionally to the size of the embedding. Luan et al. (2021) especially shows that the performance decreases more severely as the length of the encoded sequence gets longer. We hypothesize that such a limitation would be even more problematic in multi-hop retrieval with a large number of hops; as the previously retrieved sequences are added at the end of the input query, it results in a longer input sequence compared to the canonical retrieval tasks.

To test the hypothesis, we experiment over ST5 (bi-encoder) and GMR, where the ground truth retrieval targets up to the previous hops are given to seclude the effect of error propagation from the analysis. The result in Figure 2 shows that hop-R@5_{oracle} (y-axis) at each hop (x-axis) of the bi-encoder retriever deteriorates more severely than GMR as the number of hops increases. We could observe that such limitation also occurs in HotpotQA (Appendix C.2.1), where the highest error case of the bi-encoder approach is when it fails to retrieve the second hop correctly although the first hop is correctly retrieved.

It seems like the bi-encoder retriever finds it more difficult to encode the lengthened query into a fixed-size embedding. In contrast, the generative retriever is more robust in modeling the information, possibly because it can mimic the behavior of powerful one-tower cross-encoders (where all tokens in the query and retrieval sequence perform attention to each other), unlike the shallow bottlenecked interaction of bi-encoder. We also check that the finding of the previous works still holds in our multi-hop setup: the performance of the bi-encoder retriever monotonically decreases as the embedding size decreases using an additional linear layer at the top of the model (Appendix C.2.2).

(2) Error Propagation We perform experiments to analyze the robustness of the retrievers on error propagation. We simulate the case

[10]Since DSI is not open-sourced, we reproduced the model ourselves (DSI*). We show NQ results of DSI* in Appendix C.1.4. We skip the result of DSI-semantic as we could not reproduce the result. '-' in the table indicates that the model failed on all test cases. We hypothesize it is due to its difficulty in generalization to datasets with a large size corpus. We plan to update the table when the official code of DSI is released.

Table 4: Error propagation rate of a bi-encoder model (BE) and GMR on three datasets: StrategyQA (Str), EG-Open (Exp), and EntailBank (Ent). Details of Minor and Major tasks are in Section 5.2.

	Minor			Major		
	Str	**Exp**	**Ent**	**Str**	**Exp**	**Ent**
BE	23.6%	46.9%	14.0%	71.2%	91.1%	55.1%
GMR	1.7%	49.3%	11.1%	20.7%	75.8%	39.6%

where it has retrieved an irrelevant sequence at the previous hop. At i-th hop, the retriever is given a query x, ground truth retrieval target until the $i-1$-th hop ($d_{y_{<i-1}}$), and an irrelevant sequence $d_{\text{error}_{i-1}}$ at the $i-1$-th hop, and we test whether the retriever can still correctly retrieve the ground truth target at the i-th hop. We evaluate the robustness of the retrieval model by the error propagation rate (error propagation rate (%) $= \left(1 - \frac{\text{hop-R@5}_{error}}{\text{hop-R@5}_{oracle}}\right) \times 100$), which is the relative drop rate of hop-R@5$_{error}$ (hop-R@5$_{\text{error}}$ $= \mathbb{1}\{d_{y_i} \in \text{top-5}_{d \in D} P(d|x, d_{y_{<i-1}}, d_{\text{error}_{i-1}})\}$) from the oracle setup hop-R@5$_{\text{oracle}}$. We experiment over two tasks (major and minor) which differ by how relevant $d_{\text{error}_{i-1}}$ is to the ground truth target at the $i-1$-th hop $d_{y_{i-1}}$: (1) *minor* task is when we find the most relevant sequence excluding the ground truth from the corpus set using BM25 and (2) *major* task is when we randomly sample any sequence from the corpus set.

Results in Table 4 indicate that the bi-encoder approach is highly vulnerable to error propagation, where the average error propagation rate is 28.2% and 72.5% for minor and major tasks, respectively. On the other hand, somewhat surprisingly, GMR is much more robust to error propagation: 20.7% and 45%, respectively. We hypothesize that such robustness is due to its cross encoding capability over all input tokens (query and retrieval sequences at the previous hops) and its ability to leverage the distribution of sequential text tokens, learned during pretraining, on the retrieval task.

Effect of Unseen Rate of GMR The unseen rate indicates the rate of queries in the test set that needs to retrieve sequences never seen during training as the ground truth target. Therefore, datasets with high unseen rates can be considered similar to a zero-shot retrieval setting. Comparing the relative F1@20 (Table 1) of GMR to corresponding bi-encoder models in dynamic settings, the improvement is 305.4% on datasets with low unseen rates (EntailTree, RT-Open), whereas 145.6% on datasets with high unseen rates (StrategyQA, EG-Open), implying the importance of reducing the unseen rate for GMR.

We thus train GMR with LM memorization (GMR$_L$) where the model is first trained on target corpus using standard language modeling objective and then finetuned on retrieval task (Section 3). Table 1 shows that LM memorization is consistently helpful in StrategyQA and RT-Open, while the gain is inconsistent in other datasets. We hypothesize that such inconsistency is because the training objective function of LM memorization is not aligned well with that of multi-hop setting retrieval task; the memorization objective function $P(d) = \prod_{j=1}^{|d|} P(d^{(j)}|d^{(<j)})$ resembles the no-hop retrieval training objective function of maximizing $P(d_y|x)$ rather than that of the multi-hop retrieval, $P((d_{y_1}, \cdots, d_{y_n})|x)$, which goes through multiple retrieval hops. Its strength is that it can be easily applied to any dataset, but it does not show consistent improvement on different multi-hop datasets.

We also train GMR with multi-hop memorization (GMR$_M$), where the objective function is similar to that of multi-hop retrieval; we generate pseudo-multi-hop data $\{(x', D'_y)\}$ and use it as additional training data for the retrieval task (Section 3). The unseen rates of StrategyQA and EG-Open are greatly reduced by applying the method, and GMR$_M$ consistently outperforms both GMR and GMR$_L$: the unseen rates of StrategyQA and EG-Open are reduced by 40.4% and 60%, respectively. While the method is difficult to apply as-is to datasets with low unseen rates due to the filtering process (Appendix A), it is also not necessary as most retrieval sequences in the target corpus are covered by the training set.

6 Conclusion

In this paper, we show that the bi-encoder approach has limitations in multi-hop retrieval; the bottleneck problem becomes a more severe problem as the number of hops increases, and is more susceptible to error propagation. We present Generative Multi-hop Retrieval (GMR), an encoder-decoder model that performs retrieval by *generating* the entire target sequences with the aid of constrained decoding. We show that GMR is more robust on multi-hop retrieval tasks where it achieves higher or comparable performance in five datasets. We also introduce two corpus memorization methods, LM memorization and multi-hop memorization, to further improve GMR's performance. Our experimental results demonstrate that in multi-hop retrieval, a generative approach is highly competitive with bi-encoder methods and deserves further explorations in the community.

Limitations

As shown in Table 2, GMR is still not as good as a well-designed bi-encoder retrieval (MDR) for HotpotQA. We suspect that there are largely two reasons: first, HotpotQA has exactly two hops, whereas GMR seems to be more advantageous when the number of hops is large and dynamic; second, bi-encoder retrieval is a relatively mature research area, whereas generative retrieval is quite new and the community is yet to discover advanced techniques that fully leverage it. Early stopping of GMR (Section 5.1) helps inference speed but degrades the performance as it is difficult to calculate the total beam score with early stopping; it does not generate till the last token of the target sequence which it also cannot calculate the beam score over all tokens. More research will be needed to achieve both.

Acknowledgements

We would like to thank Eunbi Choi, Joel Jang, Miyoung Ko, and Yongrae Jo for helpful discussion. This work was partly supported by Samsung Research grant (2021, Multi-grained Passage Embedding via Cross-to-Bi Encoder Distillation, 40%) and Institute of Information communications Technology Planning Evaluation (IITP) grants funded by the Korea government (MSIT) (No.2022-0-00264, Comprehensive Video Understanding and Generation with Knowledge-based Deep Logic Neural Network, 55%; No.2019-0-00075, Artificial Intelligence Graduate School Program (KAIST), 5%).

References

Akari Asai, Kazuma Hashimoto, Hannaneh Hajishirzi, Richard Socher, and Caiming Xiong. 2020. Learning to retrieve reasoning paths over wikipedia graph for question answering. In *ICLR*.

Michele Bevilacqua, Giuseppe Ottaviano, Patrick Lewis, Wen-Tau Yih, Sebastian Riedel, and Fabio Petroni. 2022. Autoregressive search engines: Generating substrings as document identifiers. *ArXiv*.

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.".

Nicola De Cao, Gautier Izacard, Sebastian Riedel, and Fabio Petroni. 2021. Autoregressive entity retrieval. In *ICLR*.

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. 2017. Reading wikipedia to answer open-domain questions. In *ACL*.

Qianglong Chen, Feng Ji, Haiqing Chen, and Yin Zhang. 2020. Improving commonsense question answering by graph-based iterative retrieval over multiple knowledge sources. In *COLING*.

Peter Clark, Oyvind Tafjord, and Kyle Richardson. 2021. Transformers as soft reasoners over language. In *IJCAI*.

Bhavana Dalvi, Peter Alexander Jansen, Oyvind Tafjord, Zhengnan Xie, Hannah Smith, Leighanna Pipatanangkura, and Peter Clark. 2021. Explaining answers with entailment trees. In *EMNLP*.

Bhuwan Dhingra, Manzil Zaheer, Vidhisha Balachandran, Graham Neubig, Ruslan Salakhutdinov, and William W. Cohen. 2020. Differentiable reasoning over a virtual knowledge base. In *ICLR*.

Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. 2021. Did aristotle use a laptop? a question answering benchmark with implicit reasoning strategies. *TACL*.

Gautier Izacard, Fabio Petroni, Lucas Hosseini, Nicola De Cao, Sebastian Riedel, and Edouard Grave. 2020. A memory efficient baseline for open domain question answering. *ArXiv*.

Joel Jang, Seonghyeon Ye, Sohee Yang, Joongbo Shin, Janghoon Han, Gyeonghun KIM, Stanley Jungkyu Choi, and Minjoon Seo. 2022. Towards continual knowledge learning of language models. In *ICLR*.

Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. 2017. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. In *ACL*.

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 *EMNLP*.

O. Khattab, Christopher Potts, and Matei A. Zaharia. 2021. Baleen: Robust multi-hop reasoning at scale via condensed retrieval. In *NeurIPS*.

James Kirkpatrick, Razvan Pascanu, Neil C. Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *NeurIPS*.

Yi Luan, Jacob Eisenstein, Kristina Toutanova, and Michael Collins. 2021. Sparse, dense, and attentional representations for text retrieval. *TACL*.

Yuning Mao, Pengcheng He, Xiaodong Liu, Yelong Shen, Jianfeng Gao, Jiawei Han, and Weizhu Chen. 2021. Generation-augmented retrieval for open-domain question answering. In *ACL-IJCNLP*.

Jianmo Ni, Gustavo Hernández Ábrego, Noah Constant, Ji Ma, Keith B Hall, Daniel Cer, and Yinfei Yang. 2021. Sentence-t5: Scalable sentence encoders from pre-trained text-to-text models. *CoRR*.

Yixin Nie, Songhe Wang, and Mohit Bansal. 2019. Revealing the importance of semantic retrieval for machine reading at scale. In *EMNLP*.

Peng Qi, Haejun Lee, OghenetegiriTGSido, and Christopher D. Manning. 2021. Answering open-domain questions of varying reasoning steps from text. In *EMNLP*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *JMLR*.

Stephen Roller, Emily Dinan, Naman Goyal, Da Ju, Mary Williamson, Yinhan Liu, Jing Xu, Myle Ott, Kurt Shuster, Eric Michael Smith, Y.-Lan Boureau, and Jason Weston. 2021. Recipes for building an open-domain chatbot. In *EACL*.

Swarnadeep Saha, Sayan Ghosh, Shashank Srivastava, and Mohit Bansal. 2020. Prover: Proof generation for interpretable reasoning over rules. In *EMNLP*.

Swarnadeep Saha, Prateek Yadav, Lisa Bauer, and Mohit Bansal. 2021. Explagraphs: An explanation graph generation task for structured commonsense reasoning. In *EMNLP*.

Oyvind Tafjord, Bhavana Dalvi, and Peter Clark. 2021. Proofwriter: Generating implications, proofs, and abductive statements over natural language. In *Findings of the ACL-IJCNLP*.

Yi Tay, Vinh Quang 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. *ArXiv*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and Jamie Brew. 2020. Transformers: State-of-the-art natural language processing. In *EMNLP*.

Ledell Yu Wu, Fabio Petroni, Martin Josifoski, Sebastian Riedel, and Luke Zettlemoyer. 2020. Scalable zero-shot entity linking with dense entity retrieval. In *EMNLP*.

Zhengnan Xie, Sebastian Thiem, Jaycie Martin, Elizabeth Wainwright, Steven Marmorstein, and Peter Jansen. 2020. Worldtree v2: A corpus of science-domain structured explanations and inference patterns supporting multi-hop inference. In *LREC*.

Wenhan Xiong, Xiang Li, Srini Iyer, Jingfei Du, Patrick Lewis, William Yang Wang, Yashar Mehdad, Scott Yih, Sebastian Riedel, Douwe Kiela, and Barlas Oguz. 2021. Answering complex open-domain questions with multi-hop dense retrieval. In *ICLR*.

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 *EMNLP*.

Chen Zhao. 2020. Complex factoid question answering with a free-text knowledge graph. *Proceedings of The Web Conference 2020*.

Chen Zhao, Chenyan Xiong, Jordan Boyd-Graber, and Hal Daumé, III. 2021. Multi-Step reasoning over unstructured text with beam dense retrieval. *ArXiv*.

A Generative Multi-hop Retrieval

LM memorization For the path retrieval task (RT-Open, EG-Open), the subject and the relation are given, and the model generates the object of the sentence. For paragraph retrieval tasks (HotpotQA, EntailBank, StrategyQA), the first 70% of the sentence is given as input, and the model generates the rest.

Multi-Hop Memorization For a *conditional* memorization method, we experiment GMR with multi-hop memorization in which we generate pseudo-multi-hop queries x' and train a retriever with not only the original training dataset $\{(x, D_y)\}$, where x is a query and D_y is a list of target sequences, but also with generated pseudo-datasets $\{(x', D_{y'})\}$, where $D_{y'}$ is a list of pseudo target sequences, during the retrieval step. To keep the distribution of $\{(x', D_{y'})\}$ similar to $\{(x, D_y)\}$, we ensure that elements in the sequence $D_{y'}$ are interconnected to one another by constructing the set to have more than one other elements with the same important words or phrases (e.g., entity, subject).

For StrategyQA, we consider the entities as important words or phrases and extract the entities by NER (Named Entity Recognition) [11]. We remove

[11]We use NER model provided from huggingface (https://huggingface.co/pipelines)

sentences that do not contain any entity or contains more than four entities from the target corpus. We found that such removal of sentences with many entities is critical in the performance as sentences with lots of entities tend to be irrelevant to one another although they do have one common entity, which deviates from our original purpose of sampling: to ensure that elements in a sequence $D_{y'}$ are interconnected to one another. For EG-Open, as the items in the target corpus are a path, we consider the subject and the object as important words.

We add a filtering process for both when sampling a list of pseudo target sequences $D_{y'}$ and when generating multiple pseudo-multi-hop queries x'. When sampling $D_{y'}$, we remove sequences that contains all element in a sequence D_a where $D_a \in \{D_y\}$. After generating multiple pseudo-multi-hop queries x' with sampled $D_{y'}$, we remove queries that do not generate till the end token; the sentence stops the generation in the middle due to its higher beam score compared to when generating till the end. We keep the number of items in $D_{y'}$ to be within the same range as in D_y.

In Table 5, we show examples of pseudo-multi-hop data ($\{(x', D_{y'})\}$) of StrategyQA and EG-Open. By adding pseudo-multi-hop data to the original training dataset, we could reduce the unseen rate of StrategyQA and EG-Open by 40.4% and 60%, respectively, and increase the performance on both datasets. While the method is difficult to apply to datasets with low unseen rates due to the filtering process of removal when a set of target sequence contains a set in the original retrieval dataset as a subset, it is also not necessary as most retrieval sequences in the target corpus are covered by the training set. We leave the method of generating effective pseudo-multi hop data for datasets with low unseen rate as future work.

B Experimental Setup

B.1 Fixed and Dynamic Multi-hop Retrieval

We formulate two settings of multi-hop retrieval: fixed and dynamic multi-hop retrieval settings. In the inference step of the multi-hop retrieval, we eventually aim on retrieving a set of items by retrieving for k hops when given an initial input query x. However, since k varies depending on x and task, it is difficult to know the exact number of k beforehand in a real-world scenario. Therefore due to this limitation of real-world setting, we fix the number of retrieval step k for all datasets and queries,

Algorithm 1 Inference step of Fixed Conditional Retrieval

```
Require: trained retriever $R$, fixed number of iteration step $k$,
  input query $q$, and target corpus $\{d_i\}$

  $Y'$ := An empty set to store all retrieved sequences
  $s$ := a number of iteration step with an initial value of 1

  while $s \leq k$ do
      $y' = R(x, Y')$
      $Y'$.add($y'$)
  end while

  return $Y'$
```

Algorithm 2 Inference step of Dynamic Conditional Retrieval

```
Require: trained retriever $R$, fixed number of iteration step $k$,
  input query $q$, and target corpus $\{d_i\} \cup \{\emptyset\}$

  $Y'$ := An empty set to store all retrieved sequences
  $s$ := number of iteration step which initial value is 1

  while $s \leq k$ do
      $y' = R(x, Y')$

      // If retriever retrieves null element, stop the iteration
  and fill the set with the null element
      if $y' == \emptyset$ then
          Fill $Y'$ with $\emptyset$ so that $len(Y') == k$ break
      else
          $Y'$.add($y'$)
      end if

  end while

  return $Y'$
```

which is $k = 5$ in this paper.[12] Fixed conditional retrieval task is an expansion of canonical text retrieval task and dynamic conditional retrieval task is more likely on solving multi-hop retrieval task by stopping the retrieval process before filling all k. In fixed multi-step, the number of items to retrieve is given as an oracle, whereas in dynamic multi-step, the model also needs to determine when to stop retrieving the next item.

Fixed Multi-hop Retrieval Setting *Fixed Multi-hop Retrieval Setting* is a basic setup for multi-hop retrieval tasks, where we assume the given k as the oracle number of hops and iterate till the maximum retrieval hop k. In training process, when given a query x with 3 elements in an oracle set $\mathscr{D}_y = \{d_{y_1}, d_{y_2}, d_{y_3}\}$, the training examples for the query is $\{(x, d_{y_1}), (x; d_{y_1}, d_{y_2}), (x; d_{y_1}; d_{y_2}, d_{y_3})\}$ where ; is a string concatenation operator and each element is (input, ground truth output). In the inference process, when given a trained retriever and a target

[12]We fix k to 5 since it is near the average for all datasets.

Table 5: Examples of pseudo-multi-hop data. The top two examples are pseudo-multi-hop data from StrategyQA and the bottom two examples are from EG-Open dataset.

Input	**Output**
Did Mary, Queen of Scots know Jesus?	Output 1
	Mary, Queen of Scots was Queen of Scotland in the 1500s
	Output 2
	According to the Gospel of Matthew, Joseph and Mary resettled in Nazareth after returning from the flight from Bethlehem to Egypt
	Output 3
	She accompanied Joseph to Bethlehem, where Jesus was born
Is the iPhone still the most popular computer brand in the world?	Output1
	All generations of the iPhone use Apple's iOS mobile operating system software
	Output2
	"upset" about the price drop, Apple gave store credit to early adopters
	Output3
	Apple stores stock only Mac brand computers
belief: Everyone is abusive. / argument: Some people are just harmful to others.	Output1
	everyone; synonym of; people
	Output2
	people; not has property; abusive
	Output3
	abusive; synonym of; harmful
belief: Social media is negative. / argument: Social media keeps us connected.	Output1
	social media; causes; connected
	Output2
	connected; capable of; causing content
	Output3
	causing content; not capable of; negative

corpus, it iterates k times to find a set of retrieval sequences related to a given query as in Algorithm 1.

Dynamic Multi-hop Retrieval Setting *Dynamic Multi-hop Retrieval Setting* is a setting where the retriever has to predict the correct number of oracle hops k (when to stop retrieving). It differs from the fixed multi-hop retrieval setting, which assumes that the oracle number of hops for each input query is given and fixed. It can be considered efficient compared to the commonly used fixed multi-hop setting because (1) in real-world scenarios, we may not know the exact number of texts to retrieve in advance and the number differs by the query; (2) the model has to retrieve till the maximum hop even if it has already retrieved all the relevant sequences in fixed multi-hop setting, which would not only cause unnecessary extra time and computation but also harm the downstream tasks performance (Qi et al., 2021) by providing hard negative (false positive) sequences which are difficult to distinguish to the downstream module. The additional ability to detect the stopping point of the retrieval hop (dynamic multi-hop retrieval) can resolve the fixed multi-hop retrieval issues above.

For the setting, we additionally add a null element ($\emptyset$)[13] to the given target corpus in fixed multi-hop setting D; in this setup, a retrieved sequence is an item in $D \cup \{\emptyset\}$. In the training process, for a query, we add one extra step at the end of each end step from the fixed multi-hop retrieval training set: given all oracle sequences as conditions, retrieve null element $\emptyset$. For example, when given a query x with 3 elements in an oracle

[13]We add special token DONE to the model and the corpus which is the null element during training and inference.

Table 6: Overview of the five datasets. **Seq Len** column shows the average number of retrieval sequence tokens for each retrieval sequence in given target corpus. **Unseen Rate** column shows the rate of test queries consisting of only the retrieval sequences unseen during the training process.

Dataset	Corpus (MB)	Seq Len	Unseen Rate
HotpotQA	1,595	78.6	18.9%
EntailBank	0.7	12.5	2.7%
StratgyQA	7.0	13.1	98.2%
EG-Open	0.5	9.6	95.5%
RT-Open	0.7	13.1	0.0%

set $\mathcal{D}_y = \{d_{y_1}, d_{y_2}, d_{y_3}\}$, the training examples for the query is $\{(x, d_{y_1}), (x; d_{y_1}, d_{y_2}), (x; d_{y_1}; d_{y_2}, d_{y_3}), (x; d_{y_1}; d_{y_2}; d_{y_3}, \emptyset)\}$ where ; is a string concatenation operator and each element is (input, ground truth output). In the inference process, when given a trained retriever and a target corpus, as in fixed multi-hop retrieval, it iterates k times to find a set of retrieval sequences related to a given query. However, when the retriever retrieves the null element, it ends the iteration as in Algorithm 2.

B.2 Datasets

B.2.1 Datasets Details

Table 6[14] shows the overview of the five datasets. HotpotQA can be download in https://hotpotqa.github.io/, and the rest of the dataset can be download in https://allenai.org/data. Note that all datasets are in English.

HotpotQA Yang et al. (2018) propose an open domain multi-hop question answering dataset, which requires aggregating multiple Wikipedia passages through logical reasoning or sequential processing. The number of retrieval sequences is fixed to two. HotpotQA consists of two types of questions: comparison and bridge. Comparison questions, a rationale/evidence type of multi-hop dataset, do not necessitate iterative retrieval since the two entities can be retrieved by the query itself. However, bridge questions consist of evidence in the reasoning chain from where it has to retrieve the second step based on the first one. We use the official Wikipedia dump provided by Yang et al. (2018), use 2% of the official train dataset as a dev set, and report the scores on the official dev set.

[14]For RT-Open unseen rate, we calculate it with prediction result since there are no gold retrieval sequences.

Entailment TreeBank (EntailBank) Dalvi et al. (2021) propose a reasoning tree construction task where it forms a tree with a hypothesis as the root node and evidence sentences are leaf nodes. The dataset has three settings, and among them, we experiment on Task3, an open setting. Task3 consists of two steps; the first is to select a leaf node from the corpus set when given a question and an answer, and the second is to construct a reasoning tree through the selected leaf node. We perform the first step, the leaf node retrieval. Since the leaf node and the root node are not directly connected, there is a less tight connection between the input query and gold outputs than in other datasets. We experiment on the first step of Task3 (leaf node retrieval). As in the paper, we use both EntailBank and WorldTreeV2 (Xie et al., 2020) datasets when training a retrieval model. We compare the results with ST5 since there is no released bi-encoder model, and as in the paper, we use both EntailBank and WorldTreeV2 (Xie et al., 2020) datasets when training a retrieval model.

StrategyQA Geva et al. (2021) propose a multi-hop open-domain question answering dataset where the reasoning steps are implicit in the question and need some strategy to answer the question. When given a question, the model retrieves the evidence sentences from the corpus. Since only the train dataset contains evidence annotation, we split it into 75/5/20 (%) and used it as a train/val/test set, respectively. Also, based on the given corpus, we split the given paragraph-level corpus to sentence level using NLTK (Bird et al., 2009) to match the granularity of the evidence and add the annotated evidence sentences to the corpus.

RuleTaker-Open (RT-Open) Clark et al. (2021) propose a synthetic rule-based dataset to measure the model's reasoning ability over the rules expressed in natural language. Based on the released dataset, we create a new task, RuleTaker-Open, to make the task close to a real-world setting. Given a query, the model retrieves nodes of the graph, which is a sentence from the corpus, and the nodes are connected in order to construct a graph. Details of the construction method are described in Appendix B.3.

Explagraphs-Open (EG-Open) Saha et al. (2021) propose a generative and structured commonsense-reasoning task. When given a belief and an argument, a model predicts whether the

Algorithm 3 Finding the missing edge

```
Require: Input Corpus $P$
  $T$ := An empty list to append or remove facts from $P$

  for all sentence $s \in P$ do
      if $s$ is a rule then
          divide $s$ to assumptions $A$ and result $r$
          for all assumption $a \in A$ do
              if $a$ in $T$ then
                  $T$.remove($a$)
              else
                  return False                          ▷ Missing edge
              end if
          end for
          $T$.append($r$)
      else
          $T$.append($s$)
      end if
  end for

  if $T$ is empty then
      return True                                       ▷ No missing edge
  else
      return False                                      ▷ Missing edge
  end if
```

argument supports or counters the belief and generates (retrieves) a reasoning graph to explain the prediction. While the original dataset needs generation on constructing the reasoning graph, which is limited to generative models only, we expand the task to an open-domain retrieval setting to compare with the bi-encoder models by constructing the corpus and name it Explagraphs-Open. We consider a single path (*subject-relation-object*) as a retrieval unit and construct the corpus by dumping all the possible paths provided from the dataset.

B.2.2 Datasets Examples

Examples of each dataset (input and output forms) are in Table 7.

B.3 Details of RuleTaker-Open (RT-Open)

RuleTaker dataset is a synthetic rule-based dataset used to measure the model's ability on reasoning over rules (Clark et al., 2021; Tafjord et al., 2021; Saha et al., 2020). Given a small corpus of textual facts and rules, the model has to answer the question, retrieve, and construct the graph-structured proofs. As in Tafjord et al. (2021), we use the maximum depth dataset *D5* for training. To evaluate the model performance in the open-setting (i.e., Task3 in Dalvi et al. (2021)), we newly construct a large corpus and divide the train/dev/test dataset by the unique query set from the original D5 dataset.

Dataset Construction We dump all the facts and rules from the original D5 train/dev/test datasets to construct the corpus and collect 1621 unique queries, which we split into 1300/121/200. We remove cases with NAF and FAIL cases for rule-based evaluation, remove graphs with less than two nodes to ensure that the fact from the corpus itself could not be the proof, and remove graphs with more than ten nodes to fit in the maximum length of T5 model. Also, we added *DONE* at the end of graph construction for dynamic stopping.

Evaluation Metric In RT-Open, there are various possible answer graphs for a query, unlike the previous RuleTaker dataset. Therefore, to check whether the prediction graph is correct, a new evaluation metric is necessary. Since each textual sentence can be divided into a simple format, *subject-relation-object*, when considering the constructed method (Clark et al., 2021), we evaluate the result by a new rule-based method.
We check whether the constructed graph is well-constructed by four steps.

- *Node Num Error*: The number of evidence should be larger than 2.
- *Start Node Error*: First word (*subject*) should be the same.
- *End Node Error*: Last word (*object*) should be the same.
- *Missing Edge Error*: There should be no missing edge.

Table 8 shows the rate on each constraint for both the bi-encoder model and GMR. Each error in the table corresponds to the item on top with the same name.
Missing Edge Error is evaluated by Algorithm 3; when given a prediction graph (*P*), we divide the sentences into rules and facts and check for the missing edge in the prediction order. When the algorithm returns *True*, the graph is considered to have no missing edge.

B.4 Bi-Encoder Retrieval Models

We use ST5 model (Ni et al., 2021) as the architecture of the bi-encoder baseline to compare the performance with GMR using the same number of parameters. The input text is fed into T5-encoder, and the first decoder output of the T5-decoder is taken as the sentence embedding. We follow the implementation details in Ni et al. (2021) except for

Table 7: Dataset examples

Task	Input	Output
Paragraph Retrieval (HotpotQA)	Step 1 Input (a query)	Step 1 output (evidence passage)
	<QUESTION> The Oberoi family is part of a hotel company that has a head office in what city? </QUESTION>	<TITLE> Oberoi family </TITLE> The Oberoi family is an Indian family that is famous for its involvement in hotels, namely through The Oberoi Group.
	Step 2 Input (a query with previous output)	Step 2 Output (evidence passage)
	<QUESTION> The Oberoi family is part of a hotel company that has a head office in what city? </QUESTION> <EVIDENCE> <TITLE> Oberoi family </TITLE> The Oberoi family is an Indian family that is famous for its involvement in hotels, namely through The Oberoi Group. </EVIDENCE>	<TITLE> The Oberoi Group </TITLE> The Oberoi Group is a hotel company with its head office in Delhi. Founded in 1934, the company owns and/or operates 30+ luxury hotels and two river cruise ships in six countries, primarily under its Oberoi Hotels & Resorts and Trident Hotels brands.
Sentence Retrieval (EntailmentBank, StrategyQA)	Step 1 Input (a query)	Step 1 output (evidence sentence)
	<QUESTION> Does a dentist treat Bluetooth problems? </QUESTION>	A dentist is a surgeon who specializes in dentistry, the diagnosis, prevention, and treatment of diseases and conditions of the oral cavity
	Step 2 Input (a query + Step 1 Output)	Step 2 Output (evidence sentence)
	<QUESTION> Does a dentist treat Bluetooth problems? </QUESTION> <EVIDENCE> A dentist is a surgeon who specializes in dentistry, the diagnosis, prevention, and treatment of diseases and conditions of the oral cavity </EVIDENCE>	Technological problems are typically handled by IT professionals
	Step 3 Input (a query + Step 1 & Step 2 Output)	Step 3 Output (evidence sentence)
	<QUESTION> Does a dentist treat Bluetooth problems? </QUESTION> <EVIDENCE> A dentist is a surgeon who specializes in dentistry, the diagnosis, prevention, and treatment of diseases and conditions of the oral cavity </EVIDENCE> <EVIDENCE> Technological problems are typically handled by IT professionals </EVIDENCE>	Bluetooth is not a physical entity
Reasoning Path Retrieval (RuleTakers, Explagraphs)	Step 1 Input (a query)	Step 1 output (evidence sentence)
	<QUESTION> belif: marriage is the best for a family unit. argument: Marriage is a predictor of health and happiness. </QUESTION>	marriage; created by; love
	Step 2 Input (a query + Step 1 Output)	Step 2 Output (evidence sentence)
	<QUESTION> belif: marriage is the best for a family unit. argument: Marriage is a predictor of health and happiness. </QUESTION> <EVIDENCE> marriage; created by; love </EVIDENCE>	love; causes; health and happiness
	Step 3 Input (a query + Step 1 & Step 2 Output)	Step 3 Output (evidence sentence)
	<QUESTION> belif: marriage is the best for a family unit. argument: Marriage is a predictor of health and happiness. </QUESTION> <EVIDENCE> marriage; created by; love </EVIDENCE> <EVIDENCE> love; causes; health and happiness </EVIDENCE>	health and happiness; used for; family unit

two settings: (1) as in Karpukhin et al. (2020), we use the inner product instead of cosine similarity when calculating the similarity since inner produce shows a higher recall rate than cosine similarity for overall dataset (2) we change the hyperparameters for a fair comparison with GMR.

B.5 Details

We train both ST5 and GMR using pre-trained T5-large checkpoint (770 million parameters) from Wolf et al. (2020) as the initial checkpoint. We use the same hyperparameter setting when training GMR and ST5 model for a fair comparison. We observe that hyperparameter change does not

Table 8: Error rate for each error type in RT-Open. Results are from 200 test sets.

Error Rate (%)	GMR	ST5
Node Num Error	0.5	5
Start Node Error	9.5	0
End Node Error	20	28
Missing Edge Error	19	50
Success	51	17

change the tendency of results after experimenting over a combination of settings used in previous models (Karpukhin et al., 2020; Ni et al., 2021; Raffel et al., 2020). Also, we use different hyperparameters for different tasks: retrieval corpus memorization and retrieval. For all experiments, we use 8 32GB V100 GPUs. In retrieval task, training a epoch of HotpotQA take 1.5 hours and for the rest it take less than 0.5 hours.

LM Memorization The LM memorization step aims to show GMR a corpus it will retrieve and sa it implicitly before the retrieval step. We keep the learning rate to 1e-5, which is relatively low than the retrieval step, to maintain the linguistic ability the model learned during pretraining (Jang et al., 2022). We train the model from T5 pre-trained checkpoint for every dataset using Adafactor with a constant learning rate of 1e-5 with batch size 240 till the maximum of 3 epochs.

Increasing the LM memorization epoch does not always lead to higher performance. This is because as the model is trained on a new dataset, catastrophic forgetting of previously learned parts occurs (Kirkpatrick et al., 2017), and in this case, the linguistic ability of the model learned during the pretraining step. To prevent the following process from occurring, we follow Jang et al. (2022) and reduce the learning rate to 1e-5 and use checkpoint of epoch 3 as the initial checkpoint for all retrieval tasks.

Multi-Hop Memorization We train a model that generates a pseudo-multi-hop query for multi-hop memorization when given a set of retrieval sequences. We dump all retrieval datasets to train such a model and concatenate all retrieval sequences as a long sequence as an input and the corresponding query as an output. Generated pseudo-multi-hop queries after the filtering process are 11k and 1.9K for StrategyQA and EG-Open, respectively. We set the configuration the same as in *Retrieval Step*.

Table 9: Retrieval sequence F1 score of model trained on fixed multi-hop retrieval setting (*-fix) and dynamic multi-hop retrieval setting (*-dyn) on test set. F1@k is a retrieval sequence F1 score with maximum retrieval step of k. Models trained on dynamic setting show consistently higher F1 score compared to those trained in fixed setting.

Dataset	Model	F1@5	F1@10	F1@20
EntailTree	BE-fix	20.1	15.0	9.7
	BE-dyn	24.9	19.4	16.9
	GMR-fix	33.6	23.5	13.8
	GMR-dyn	48.2	52.1	52.5
StrategyQA	BE-fix	22.8	15.6	9.3
	BE-dyn	38.1	36.9	36.5
	GMR-fix	30.1	19.3	11.2
	GMR-dyn	41.9	44.3	46.6
EG-Open	BE-fix	24.6	20.1	13.1
	BE-dyn	27.0	24.6	25.4
	GMR-fix	26.9	22.1	15.1
	GMR-dyn	35.5	40.0	41.5

Retrieval Step The retrieval step aims to retrieve the gold item from a large-scale corpus. For GMR with LM memorization (GMR_L), we use the checkpoint from LM-memorization as the initial checkpoint, and for the rest of the models (ST5, GMR, GMR with multi-hop memorization (GMR_M)), we use the T5 pre-trained checkpoint as the initial checkpoint. For GMR_M, we train the model using both the training dataset and generated dataset from the T5 pre-trained checkpoint. For both ST5 and GMR (including GMR_L, GMR_M), we train using Adafactor with a learning rate 1e-4 with a linear warm-up for the first 10% of training and then linear decay with batch size 120 till a maximum of 30 epochs.

C Experimental Results

C.1 Results

C.1.1 Fixed vs. Dynamic Multi-Hop Settings

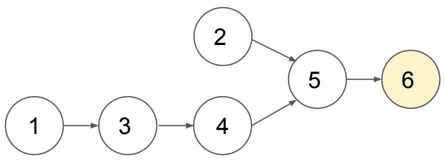
Table 9 shows the retrieval sequence F1 score (F1@k) of Fixed and Dynamic Multi-Hop Settings, where k is the number of maximum retrieval steps, and we evaluate k=5, 10, and 20. For all three evaluations and both the bi-encoder approach and GMR, models trained in the dynamic setting show higher scores than those trained in the fixed setting, emphasizing the importance of using dynamic multi-hop retrieval settings to solve multi-hop retrieval tasks near a real-world scenario.

C.1.2 RT-Open Results

The prediction result from the model, predicted corpus (*P*), is in the gray box, and the final node is colored in yellow. The Missing nodes are colored in red, and the leftover nodes are colored in blue. If there is a red or blue node, it means that it failed to construct the reasoning graph. We show two examples for each retrieval method and success and failure cases (missing edge error case) in Figure 3, Figure 4, Figure 5, and Figure 6.

Predicted Corpus (P):

1. The cat is kind
2. The cat is kind
3. If something is kind then it chases the cat
4. If something chases the cat then it is young.
5. If something is kind and young then it is cold.
6. If something is cold then it visits the dog.



Predicted Corpus (P):

1. The lion is young.
2. If something is young then it eats the lion.
3. If something eats the lion then it is kind.
4. The lion is young.
5. If something is young then it eats the lion.
6. If something eats the lion then it likes the lion.
7. If something is kind and it likes the lion then the lion eats the cow.
8. If something eats the cow then it eats the rabbit.

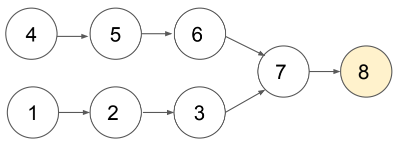
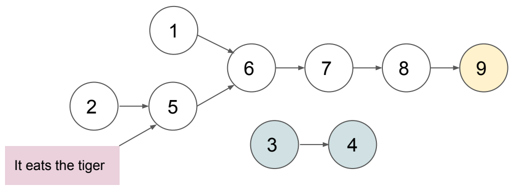


Figure 3: Success Examples of GMR

Predicted Corpus (P):

1. The mouse is cold.
2. The mouse is cold.
3. The mouse is cold.
4. If something is cold then it eats the dog.
5. If something is cold and it eats the tiger then the tiger is kind.
6. If something is cold and kind then it is red.
7. If something is red then it sees the mouse.
8. If something sees the mouse then the mouse is green.
9. If something is green then it sees the squirrel.



Predicted Corpus (P):

1. The bear is kind.
2. The bear is kind.
3. If something is kind then it chases the bear.
4. If something chases the bear then it is big.
5. If something is kind and big then it is rough.
6. If something is rough then it likes the bald eagle.
7. If something likes the bald eagle then it likes the tiger.
8. If something likes the tiger then it likes the lion.
9. If something likes the bear and it likes the bald eagle then it is round.
10. If something is round then it likes the bear.

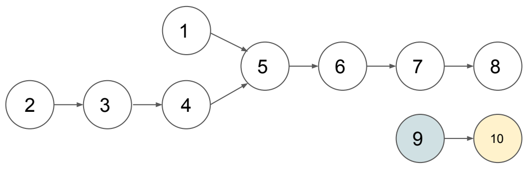
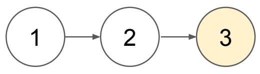


Figure 4: Failure Examples of GMR where blue nodes indicate the the leftover nodes and red are the missing nodes

Predicted Corpus (P):

1. The rabbit is blue.
2. If something is blue then it sees the rabbit.
3. If something sees the rabbit then it is big.



Predicted Corpus (P):

1. The rabbit is big.
2. If someone is big then they need the rabbit.
3. If someone needs the rabbit then they are big.
4. If someone is big then they like the rabbit.
5. If they like the rabbit then the rabbit is kind.
6. If someone is kind then they visit the rabbit.

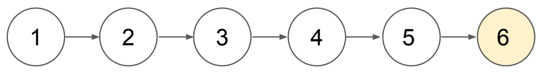
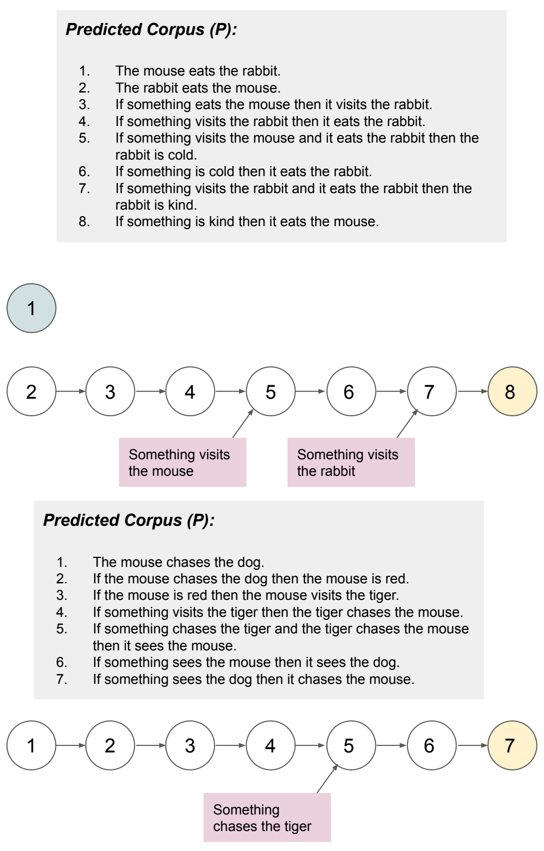


Figure 5: Success Examples of Bi-encoder (ST5) Retrieval

C.1.3 Storage Footprint

Table 10 shows the overall storage footprint of three models: MDR, GMR, and GMR with early stopping. Where GMR with early stopping does not generate every word in the retrieval target text but stops generation as soon as the partially generated text can uniquely identify the target text and saves only till the point. Also, GMR shows higher memory efficiency with a higher decrease rate of the index size (with respect to that of the bi-encoder counterpart) when the granularity of the corpus is small and when items in the given target corpus are similar to one another with the same prefix. HotpotQA, which uses paragraphs as the retrieval unit, has a lower reduction rate of 86.68%. RT-Open shows the highest decrease rate of 99.9% since its corpus consists of short texts, and the items in the corpus are highly similar to one another due to its synthetic rule-based data construction process.



Predicted Corpus (P):

1. The mouse eats the rabbit.
2. The rabbit eats the mouse.
3. If something eats the mouse then it visits the rabbit.
4. If something visits the rabbit then it eats the rabbit.
5. If something visits the mouse and it eats the rabbit then the rabbit is cold.
6. If something is cold then it eats the rabbit.
7. If something visits the rabbit and it eats the rabbit then the rabbit is kind.
8. If something is kind then it eats the mouse.

Predicted Corpus (P):

1. The mouse chases the dog.
2. If the mouse chases the dog then the mouse is red.
3. If the mouse is red then the mouse visits the tiger.
4. If something visits the tiger then the tiger chases the mouse.
5. If something chases the tiger and the tiger chases the mouse then it sees the mouse.
6. If something sees the mouse then it sees the dog.
7. If something sees the dog then it chases the mouse.

Figure 6: Failure Examples of Bi-encoder (ST5) Retrieval where blue nodes indicate the the leftover nodes and red are the missing nodes

Table 10: Storage footprint for GMR and bi-encoder in GB. GMR * is a model of GMR with early stopping. GMR shows average of 73.49% reduction on total memory usage compare to the bi-encoder model.

Dataset	Model	Retriever	Index	Total
HotpotQA	MDR	0.48	15.33	15.81
	GMR	2.75	2.04	4.79
	GMR *	2.75	0.20	2.95

Shorter retrieval sequences result in a higher index decrease rate since bi-encoder retrievers use a fixed-size dense embedding regardless of the sequence length, whereas GMR stores fewer tokens for shorter sequences. Moreover, when the number of retrieval sequences in the corpus increases, the storage footprint of the bi-encoder model increases linearly, whereas that of GMR increases more slowly as it needs to store only the additional token ids (integers) that are not in the prefix tree.

C.1.4 DSI

As DSI (Tay et al., 2022) is not open-sourced, we reproduce both the model and the dataset (NQ-10k) ourselves. In Table 11, we show results of DSI* and DSI which DSI* is our reproduced model and DSI is the model from the original paper. For the Atomic Docid method, DSI* shows near twice the performance of DSI in both Hits@1 and Hits@10. For the Naive String Docid method, DSI* shows near 80% of DSI performance in Hits@1 and Hits@10.

Table 11: Result of NQ-10k dataset of our reproduced DSI model (DSI*). DSI are results from Table 3 of Tay et al. (2022). Although we tried to replicate the same setting as in DSI since the dataset NQ-10k is unreleased, the DSI and DSI* datasets may differ. We use T5-base with initial checkpoint from Wolf et al. (2020). We did not report the score of the Semantic String Docid method since both Hits@1 and Hits@10 of DSI* are very low.

Method	Model	Hits@1	Hits@10
Atomic Docid	DSI	13.0	38.4
	DSI*	38.2	60.1
Naive String Docid	DSI	28.1	48.0
	DSI*	22.6	37.0

In Table 3, we could see that DSI especially shows a low recall score in StrategyQA dataset, which has a corpus set four times larger than the other two datasets (EG-Open and EntailBank). Such tendency of the performance degradation as the size of the target corpus increases can also be seen in the DSI paper when comparing the result between NQ-10k and NQ-320k. These results suggest the possible difficulty of expanding to a larger corpus set in DSI unlike GMR.

C.2 Analysis

C.2.1 Manual Analysis on HotpotQA

We conduct manual analysis on HotpotQA by comparing the top-2 prediction result of the GMR and MDR, a bi-encoder retrieval model. From the two question categories in HotpotQA (bridge and comparison questions), we manually inspect 30 sampled examples where one model fits and the other is wrong. MDR mostly got wrong by missing the second hop item though it got the first hop correct and GMR was wrong for cases where the first-hop item is not written explicitly in the query but by sharing a specific part of a sentence. When the item is written explicitly in the query, GMR tend to get it correct, which shares with the result that GMR shows a higher score on comparison questions than MDR. We suggest this result is because GMR can directly cross-encode between the input and the output without any information loss.

To be specific, we divide the error case into four:

(1) When the first-hop retrieval item is not written explicitly in the query but by sharing a specific part of a sentence.
(2) Though it is written explicitly in the query, it retrieves the wrong document by giving attention to an irrelevant part of the query.
(3) Detail of the title is wrong (i.e., when the gold document has the title *Do you Love Me (Not That I Can Dance)*, the model retrieves a document with the title *Do you Love Me (2NE1 song)* instead; when *do you love me* is in a query, the model misses to understand the details correctly.)
(4) The retriever got the first hop correct but failed to retrieve the second hop item correctly.
When comparing the number of models matched in the bridge question with each error case, among the four cases, MDR is often wrong in the second (1.3 times) and fourth cases (2.2 times), and the GMR is most often wrong in the first case (6 times) along with the third case (2.8 times)[15].

C.2.2 BottleNeck Problem in Bi-Encoder Models

Previous work has shown the inherent limitations of bi-encoder approaches; by encoding all information in given text into fixed-size embedding, it has shown a bottleneck problem (Luan et al., 2021). By adding a linear layer at the top of the model and decreasing the dimension, we could see that such a bottleneck problem still holds in our bi-encoder models in multi-hop retrieval tasks. As in Figure 7, as the embedding size decreases from 1028 to 128, hop-R@5 of the bi-encoder retriever monotonically decreases. The x-axis is the number of hops in multi-hop retrieval tasks and the y-axis is the score of hop-R@5. For all three datasets (EntailBank, EG-Open, and StrategyQA), we can see that performance of the bi-encoder tends to degrade as (1) the number of hops increases after a certain threshold value and (2) as the size of embedding decreases.

[15] the value in parentheses shows the ratio of the error rate compared to the other model

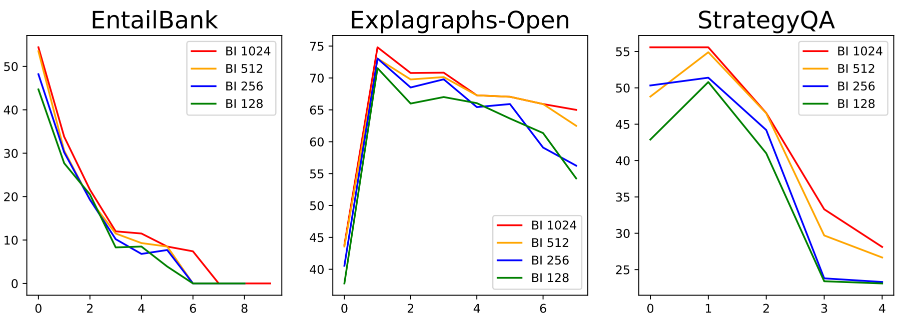


Figure 7: We plot hop-R@5 (y-axis) over number of hops in multi-hop retrieval task (x-axis) in the figure. As we experiment on ST5 using T5-large, the initial embedding size is 1024. We experiment by reducing or retaining the 1024 embedding to 1024 (red), 512 (orange), 256 (blue), and 128 (green) dimensions by adding a linear layer at the end of the model. For all three datasets, we can see that performance of the bi-encoder tends to degrade as (1) the number of hops increases after a certain threshold value and (2) as the size of embedding decreases.