Sequence Structure Aware Retriever for Procedural Document Retrieval: A New Dataset and Baseline

Zhenqi Ye[1,2]*, Haopeng Ren[3]*, Yi Cai[2,1]†, Qingbao Huang[4], Jing Qin[5], Pinli Zhu[1,2], Songwen Gong[1,2]

[1]School of Software Engineering, South China University of Technology, Guangzhou, China,
[2]Key Laboratory of Big Data and Intelligent Robot(SCUT), MOE of China,
[3]School of Artificial Intelligence, Guangzhou University, Guangzhou, China,
[4]School of Electrical Engineering, Guangxi University, Nanning, China,
[5]School of Nursing, The Hong Kong Polytechnic University, Hong Kong, China
sezqye@mail.scut.edu.cn, renhp@gzhu.edu.cn, ycai@scut.edu.cn

Abstract

Execution failures are common in daily life when individuals perform procedural tasks, such as cooking or handicrafts making. Retrieving relevant procedural documents that align closely with both the content of steps and the overall execution sequence can help correct these failures with fewer modifications. However, existing retrieval methods, which primarily focus on declarative knowledge, often neglect the execution sequence structures inherent in procedural documents. To tackle this challenge, we introduce a new dataset *Procedural Questions*, and propose a retrieval model Graph-Fusion Procedural Document Retriever (GFPDR) which integrates procedural graphs with document representations. Extensive experiments demonstrate the effectiveness of GFPDR, highlighting its superior performance in procedural document retrieval compared to existing models.[1]

1 Introduction

In real life, humans often encounter execution failures when performing procedural tasks such as cooking or handicrafts making, as shown in Figure 1. The causes and solutions to failures can be found in procedural documents (e.g., cookbooks or operation tutorials) containing the used materials with their specific amounts, the sequence of operating steps, operating precautions, and so on (Zhou et al., 2022). However, it is often labor-intensive and time-consuming for humans to correctly retrieve the reference procedural documents that enable solving the failures effectively. In our paper, we refer to such retrieval scenario as *procedural document retrieval*.

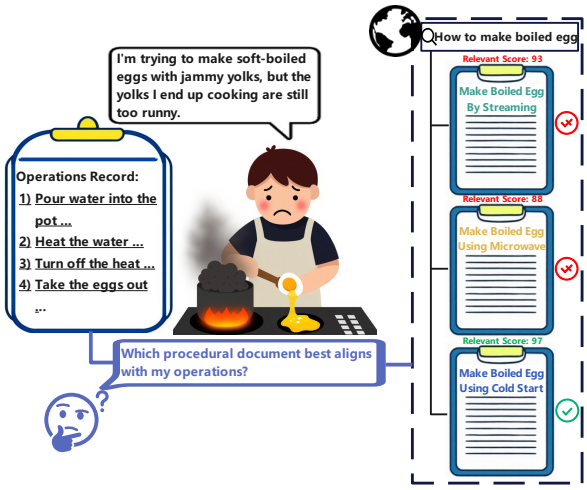

Figure 1: Procedural document retrieval scenario. There are various procedural documents for a procedural task.

Current retrieval scenarios are predominantly centered around declarative knowledge (Jacobs and Paris, 1987). Popular retrieval datasets like *MS-MARCO* (Bajaj et al., 2016) and *Natural Questions* (Kwiatkowski et al., 2019) primarily consist of declarative documents (e.g., *Wikipedia pages*) and queries are simple like *"What's the largest ice sheet?"*. Retrievers (Wang et al., 2023; Zhou et al., 2024) mainly focus on capturing the key elements of declarative documents, e.g., subjects, facts, and entities. However, they neglect the execution sequence of steps within procedural document, which provides crucial clues for retrieving relevant documents in procedural document retrieval. For instance, in Figure 2, although both documents contain the same topic and similar step contents (i.e., *Place eggs*, *Heat water*, and *Simmer*), the sequences of these steps are executed differently. *Document 2* places eggs into water first, then heats the water, whereas *Document 1* reverses the sequence. This difference in sequence affects the simmering time for different states of eggs. Since both the step contents and sequence structure in

* Equal Contributions
† Corresponding Author
[1]Data and code is available at https://github.com/YeZhenqi/GFPDR

Findings of the Association for Computational Linguistics: EMNLP 2025, pages 4180–4198
November 4-9, 2025

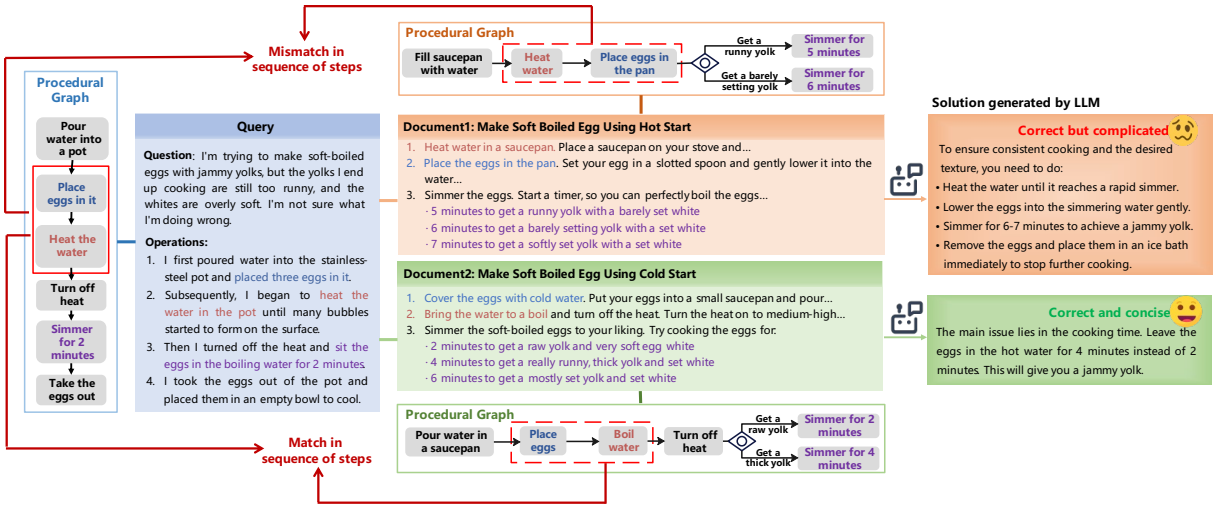

Figure 2: An example of procedural document retrieval. The *Query* describes a failure including the issue and the operations taken. *Document 1* and *Document 2* are two relevant documents sharing several steps, such as Place eggs in pan, Heat water, and Simmer eggs, but with different execution sequences. The execution sequence in *Document 2* aligns more closely with the *Query*, resulting in a more concise and effective solution. However, ColBERTv2 incorrectly ranks *Document 1* as more relevant than *Document 2*. A preliminary study on how retrieval quality affects solution generation is provided in Appendix A.

Document 2 align better with the query's procedure, it is more relevant to the query and leads to a more streamlined solution with fewer modifications.

Furthermore, we observe that the execution sequence structures in procedural documents often include multiple semantic dependencies beyond simple step-by-step sequences. These dependencies (Pal et al., 2021; Ren et al., 2023) can involve selection (e.g., *Step 3* of *Document 2* in Figure 2) and inclusion, where one action functions as a sub-action of another. Additionally, declarative sentences frequently appear to provide supplementary or constraints for specific actions. Existing retrieval methods lack explicit modeling of these diverse dependencies within procedural documents, limiting their ability to learn the sequence structure. Recent studies (Du et al., 2024; Shirai et al., 2023) structure these semantic dependencies through procedural graphs, which are graph-based representations for procedural knowledge. These structured representations facilitate capturing the sequence structures in procedural documents. According to our observation, as shown in Figure 2, procedural graphs can summarize key procedural steps and their dependencies, aiding in the identification of differences in execution sequence structure between queries and documents.

In our paper, we propose **G**raph-**F**usion **P**rocedural **D**ocument **R**etriever (GFPDR), a retriever that leverages procedural graphs to learn the sequence structure of procedural documents. We employ an Edge-Aware Graph Attention Module to capture sequence structure from the graph. Then, the Graph-Fusion Document Encoder encodes procedural document and fuses it with graph features. To maintain low online computational complexity, GFPDR adopts a late-interaction architecture (Khattab and Zaharia, 2020), which allows offline document encoding and performs fine-grained similarity computation during retrieval. Furthermore, due to the lack of retrieval datasets that use erroneous operations as queries, we construct a new dataset named *Procedural Questions*. It contains erroneous operations in procedures from various procedural tasks as queries, paired with the most relevant procedural document.

Our contributions are summarized as follows:

- We explore a problem of procedural document retrieval, emphasizing the need to consider both step contents and execution sequences between queries and documents. To the best of our knowledge, this is the first study focusing on procedural document retrieval with consideration of sequence structure.

- We design a retriever GFPDR that captures the sequence structure of procedural documents by integrating procedural graphs. Existing retrieval methods overlook the impact of sequence structure on retrieving relevant procedural documents.

- We create a procedural document retrieval dataset *Procedural Questions*. Extensive experiments on our dataset demonstrate the effectiveness of our proposed method in procedural document retrieval.

2 Related Work

2.1 Procedural Knowledge

Prominent knowledge bases like *WikiData* (Vrandečić and Krötzsch, 2014), *Wikipedia* (Lehmann et al., 2015), and *FreeBase* (Bollacker et al., 2007) primarily focus on representing *declarative knowledge*, which involves attributes or features of things. However, these resources do not adequately cover *procedural knowledge*—knowledge about the sequence of actions required to achieve specific goals. Previous research on procedural knowledge falls broadly into two categories. One is procedural knowledge acquisition, aiming to better represent procedural knowledge (Du et al., 2024; Shirai et al., 2023; Ren et al., 2024; Zhou et al., 2022). The other is procedural knowledge applications (Li et al., 2024; Luo et al., 2021; Tandon et al., 2015). Luo et al., 2021 formulates the problem of operations diagnosis within procedural graphs. However, their approach assumes that there is only a single, fixed way to accomplish a task, overlooking the fact that multiple methods may exist for the same procedural task.

2.2 Retriever and Reranker Architecture

With the advancement of language models (Yuan et al., 2024), fine-tuned pre-trained models have significantly improved retrieval and reranking performance. The bi-encoder architecture is the most commonly used (Karpukhin et al., 2020), where the query and document are passed through encoders to generate a single vector, and the relevance is computed by the similarity of embeddings. Recent works (Xiao et al., 2023; Wang et al., 2023) have enhanced bi-encoders through advanced distillation techniques. In addition to the standard bi-encoder, there are variants (Gao et al., 2020; Santhanam et al., 2022) that introduce dense interactions for improved effectiveness. ColBERT (Khattab and Zaharia, 2020) employs a late-interaction paradigm, computing token-wise dot products between the query and document vectors, followed by max-pooling and sum-pooling to obtain a relevance score. Cross-encoders (Kenton and Toutanova, 2019; Dai and Callan, 2019), which encode process query and document information during inference, have proven effective in traditional retrieval settings. Additionally, generative models such as T5 (Raffel et al., 2020) have shown to be effective for retrieval and reranking (Zhuang et al., 2023; Yoon et al., 2024). In this paper, we investigate the performance of these architectures for procedural document retrieval.

3 Methods

Task Formulation: The procedural document retrieval task aims to search the most relevant documents given a query. Formally, given a query q and a collection of procedural documents D, the task is to retrieve top-k relevant documents $D_q = d_1^*, ..., d_k^*$ from D.

Graph-Fusion Procedural Document Retriever (GFPDR) aims to capture sequence structure from procedural graph and fuse with document embeddings. Our method has three components: 1) Edge-Aware Graph Attention Module, which is designed to construct procedural graphs from procedural documents and extract the sequence structure features from the graphs. 2) Graph-Fusion Document Encoder, which encodes procedural documents and integrates with procedural graph. 3) Query Encoder, a language model for encoding the query. The overall architecture is shown in Figure 3.

3.1 Edge-Aware Graph Attention Module

3.1.1 Procedural Graph Construction

To construct procedural graphs, we first define the types of nodes and edges for general procedural graphs, as shown in Table 1. We follow the method proposed by Du et al., 2024, which leverages LLMs to extract procedural graphs from documents. This approach transforms a procedural document into a directed weighted graph $G = \{V, E\}$, where V represents nodes comprising actions, constraints, and gateways, while E denotes the dependencies between nodes. An example and additional details are provided in App. B.

3.1.2 Edge-Aware Graph Attention Layer

The nodes and edges extracted from the procedural graph above have text attributes. To obtain node and edge features, we encode their text attributes with a language model. Given a triple $t = \{u, v, e\}$ in the procedural graph, where u, v and e are source node, destination node, and edge

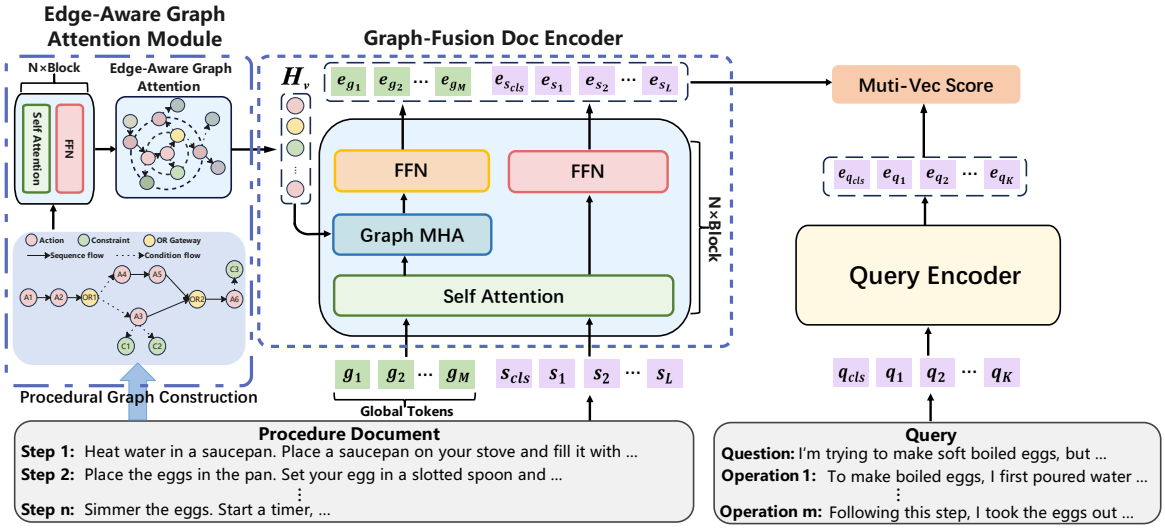

Figure 3: The overall architecture of our Graph-Fusion Procedural Document Retriever.

Node Type	**Meaning**
Action	a specific step in a procedure
Constraint	essential notices for the execution of the actions
AND	gateway that indicates the following actions are executed in parallel
OR	gateway that indicates one or more of the following actions are executed
LOOP	gateway that indicates the following actions need to be re-executed
Edge Type	**Meaning**
Sequence flow	flow that represents the execution of sequential actions
Condition flow	flow that indicates the following action or constraint is under the condition

Table 1: Types of node and edge in the procedural graph and their meanings.

respectively. Their features are processed as:

$$h_u = LM(t_u) \quad h_v = LM(t_v) \quad h_e = LM(t_e)$$

where t_u, t_v, t_e are the text of node u, node v and edge e respectively. LM is a pre-trained language model such as BERT(Kenton and Toutanova, 2019). $h_u, h_v, h_e \in \mathbb{R}^d$ are the $[CLS]$ embeddings which serve as the features of node u, node v and edge e respectively. After obtaining the node and edge features, we use GAT (Veličković et al., 2017) to update nodes. Note that procedural graph is a directed weighted graph. The direction and text attribute of the edges are important properties in the procedural graph. Thus, we introduce edge features when updating the graph by self-attention mechanism as follows:

$$e_{ij}^{(l)} = \phi\left(\mathbf{a}^{(l)\top}\left[\mathbf{W}_{\mathbf{src}}^{(\mathbf{l})} h_{v_i}^{(l)} \,\|\, \mathbf{W}_{\mathbf{dst}}^{(\mathbf{l})} h_{v_j}^{(l)} \,\|\, \mathbf{W}_{e}^{(l)} h_{e_{ij}}\right]\right)$$

$$\alpha_{ij}^{(l)} = \frac{\exp(e_{ij}^{(l)})}{\sum_{k \in \mathcal{N}(v_i^{(l)})} \exp(e_{ik}^{(l)})}$$

$$h_{v_i}^{(l+1)} = \sigma\left(\sum_{j \in \mathcal{N}(v_i)} \alpha_{ij}^{(l)} \mathbf{W}_{\mathbf{att}}^{(\mathbf{l})} h_{v_j}^{(l)}\right)$$

where ϕ denotes the LeakyReLU activation function and $\mathbf{a}$ is a learnable vector. $\mathbf{W_{src}}$ and $\mathbf{W_{dst}}$ are learnable weight matrices for source nodes and destination nodes. $\mathbf{W_e}$ and $\mathbf{W}_{att}$ are learnable weight matrices for edges and attention calculation. $\|$ denotes concatenation, and α_{ij} is the attention coefficient. l is the number of EAGAT layers. After k EAGAT layers, each node aggregates features from its k-hop neighbors. We do not update edge features as their text attributes are simple and clear. Preserving the original edge features across layers helps reduce learning complexity.

3.2 Graph-Fusion Document Encoder

Procedural graph focuses on providing a clear sequence structure of the procedure while procedural document comprises more details. However, node embeddings of the graph and token representations of the document are in different feature spaces. To eliminate the modality gap between two embeddings and effectively integrate two modalities,

inspired by Li et al., 2023, we propose the Graph-Fusion Document Encoder, which is a transformer encoder aims to align modalities by fusing token representations with node embeddings. Specifically, we apply a set of learnable tokens as input, termed *global tokens* $G = \{g_i\}_{i=1}^{M}$, where M is the number of global tokens. Given a sequence of text tokens $S = \{s_1, s_2, ..., s_L\}$ of the document, both global tokens and text tokens interact with each other through self-attention layers to capture overarching semantic information of the document. Then, the global tokens interact with node embeddings $H_v = \{h_1, h_2, ..., h_I\}$ of the graph to introduce sequence structure features through cross-modality multi-head attention layer(Graph-MHA) as follows:

$$G^q = E_G W^q, H^k = H_v W^k, G^v = H_v W^v$$

$$E'_G = \text{SoftMax}\left(G^q (H^k)^\top / \sqrt{d}\right) G^v W^{out}$$

where E_G is the embeddings of global tokens after self-attention layers. Finally, two feed-forward networks are used to encode the global tokens and text tokens respectively. The document representations $E_D = \{e_{g_1}, ..., e_{g_M}, e_{cls}, e_{s_1}, ..., e_{s_L}\}$, where $\{e_{g_i}\}_{i=1}^{M}$ and $\{e_{s_j}\}_{j=1}^{L}$ are the correspond token embeddings of g_i and s_j encoded by Graph-Fusion Document Encoder. We share the parameters of self-attention layers and feed-forward layers with the language model of Edge-Aware Graph Attention Module to better align the two modalities.

3.3 Query Encoder

Although the users' queries contains steps about how they perform the procedural task, we observe that these steps are always sequential and concise. Based on the observation, we only adopt a language model to encode query. Given a sequence of tokens $Q = \{q_1, q_2, ..., q_K\}$ of query, the query representation $E_Q = \{e_{cls}, e_{q_1}, ..., e_{q_K}\}$, where $\{e_{q_i}\}_{i=1}^{K}$ is the correspond token embedding of q_i.

3.4 Objective Function

Given document representation E_D and query representation E_Q, late interaction similarity (Santhanam et al., 2022), a multi-vector similarity computation, is utilized as follows:

$$Sim(Q, D) = \sum_{e_i \in E_Q} \max_{e_j \in E_D} (e_i \cdot e_j)$$

We apply InfoNCE Loss as the objective function. Specifically, for a given query Q, it computed negative log-likelihood of a positive document D^+ against a set of negatives $\{D_1^-, D_2^-, ..., D_l^-\}$ as follows:

$$\mathcal{L}_{NCE} = -log \frac{e^{Sim(Q,D^+)/\tau}}{e^{Sim(Q,D^+)/\tau} + \sum_l e^{Sim(Q,D_l^-)/\tau}}$$

To better align node embeddings with text embeddings, we introduce an auxiliary loss function termed the Graph-Document Match Loss. Specifically, given a graph-document pair (G, D), the embeddings of global tokens are passed through a binary classifier and averaged to compute a match score, defined as:

$$s(G, D) = \frac{1}{M} \sum_{i=1}^{M} f_{cls}(e_{g_i})$$

where e_{g_i} is the embedding of $i\text{-}th$ global token and f_{cls} is a binary classifier. The loss is as follows:

$$\mathcal{L}_{GDM} = BCE(s(G, D), y(G, D))$$

where $y(G, D) \in \{0, 1\}$ indicates whether graph and document match and BCE denotes to Binary Cross Entropy Loss. The overall objective function is as follows:

$$\mathcal{L} = \mathcal{L}_{NCE} + \mathcal{L}_{GDM}$$

3.5 Time Complexity

Despite incorporating a GAT module for document encoding, GFPDR adopts a late interaction design that allows document embeddings to be computed offline and efficiently indexed. At query time, only query encoding and similarity computation are performed. The resulting online time complexity is $O(q^2 + n \times q \times d')$, where q is the number of query tokens, and $d' = d + M$ denotes the number of document tokens plus global tokens. Here, $O(q^2)$ corresponds to query encoding, and $O(n \times q \times d')$ to similarity computation. It is comparable to ColBERTv2's $O(q^2 + n \times q \times d)$.

4 Experiments

4.1 Dataset & Annotation

Constructing a dataset through web scraping is challenging, as procedural questions on the internet are dispersed and most answers lack corresponding reference documents. Thus, we curate our dataset *Procedural Questions* manually. We collect the corpus from Wu et al., 2022 and Yuan et al., 2023, which

Dataset Statistic	Train	Dev	Test
Total Queries	4500	500	921
Avg. Tokens per Query	196	195	188
Avg. Sentences per Query	10	9.9	9.2
# Corpus		361500	
Avg. Sentences per Doc.		564	

Table 2: Dataset statistics for *Procedural Questions*.

provides a collection of human-created *how-to* articles on WikiHow[2] style. We develop an LLM-based framework to generate erroneous procedural operations as queries. Specifically, we define three common types that cause failures when users execute procedural tasks: 1) **Missing**. Failure occurs due to the missing of steps. 2) **Swap**. Failure occurs due to the incorrect order of several steps. 3) **Wrong details**. Failure occurs due to incorrect details, such as time and dosage.

Given a procedural document, our framework simulates failure scenarios based on predefined types to generate queries consisting of a failure description and an erroneous step sequence. A self-refine step ensures that the step sequence logically leads to the failure, and a rewriting phase enhances query diversity. This process guarantees that generated queries differ from the reference document while keeping the reference as the most relevant. By automating query generation, the framework significantly reduces annotation workload, requiring only minimal human review. We use DeepSeek-V2.5 (DeepSeek-AI, 2024) and LLaMA3 (AI@Meta, 2024) to improve query diversity. More details are provided in App. C.

Four well-educated annotators were employed to review the queries. Before starting the annotation, we provided detailed guidelines and examples to ensure consistency. With the framework's assistance, the annotators reviewed approximately 2,300 queries, filtering out 997. A second-round review by a single annotator further refined the dataset to 921 queries, which are used as the test set. The inner-annotator Cohen's Kappa of 0.68 and retention rate of 0.85 indicate substantial agreement. An additional 5,000 unreviewed queries are allocated to the training and development sets. The statistics of the dataset are summarized in Table 2, with review criteria provided in App. C. Moreover, our framework generates corrective solutions for each query, allowing the *Procedural Questions* dataset to also serve as a QA benchmark.

[2] https://www.wikihow.com/Main-Page

4.2 Experiment Settings

4.2.1 Baselines

We evaluate 14 models across various architectures. For sparse retrievers, BM25 (Robertson et al., 2009) a classic term-frequency-based method, and SPLADE++ (Formal et al., 2022), which leverages transformer-based expansion. For bi-encoders, SimLM (Wang et al., 2023) employs a bottleneck architecture with self-supervised pretraining, Longtriever (Yang et al., 2023) handles long documents with hierarchical encoding, BGE-base (Xiao et al., 2023) is trained on large-scale corpora, and T5-2K (Coelho et al., 2024) mitigates positional bias with RoPE. For generative retrievers, ListT5 (Yoon et al., 2024) introduces a novel efficient listwise reranking algorithm. RepLLaMA(Ma et al., 2024) is an LLM dense retriever fine-tuning by LLaMA2-7B; RankZephyr(Pradeep et al., 2023) and PRP-Sliding(Qin et al., 2024) are two zero-shot LLM rerankers using different listwise reranking strategies. For cross-encoders, MonoBERT (Nogueira and Cho, 2019) applies BERT to encode query–document pairs, while Longformer (Beltagy et al., 2020) extends input capacity to handle long contexts. Re2g (Glass et al., 2022) improves reranking through multi-stage training. Finally, ColBERTv2 (Santhanam et al., 2022) employs a late-interaction architecture.

4.2.2 Implementation Details

Considering that several rerankers are selected as baselines and reranking the whole corpus would be computationally expensive, we unify the experimental setting by having all models rerank the top 50 candidates retrieved by BM25. The top 50 candidates ensure coverage of relevant solutions for each procedural task.

Our GFPDR consists of two EAGAT layers. The Graph-Fusion Document Encoder and the query encoder are initialized separately with RoBERTa-based parameters. The number of global tokens is set to 32. During training, each query is paired with two hard negatives. The maximum input lengths for the documents and queries are 512 and 300, respectively. When training baselines, we align key hyperparameters such as input length. We employ a token reduction algorithm to shorten lengthy documents while preserving as much semantic information as possible. Additional implementation details are provided in App. D.1. Our dataset, code and model weights are publicly available.

Model	Hit@1	Hit@5	NDCG@3	MRR@5	MRR@10
Sparse retriever					
BM25†(Robertson et al., 2009)	22.91	54.72	36.65	35.29	36.34
SPLADE++†(Formal et al., 2022)	39.2	66.45	50.93	49.57	50.76
Bi-encoder					
Longtriever(Yang et al., 2023)	45.17	84.04	64.55	59.96	61.53
SimLM (Wang et al., 2023)	59.28	87.19	70.98	69.97	71.25
T5-2K (Coelho et al., 2024)	48.97	85.34	64.21	62.61	63.9
BGE-base (Xiao et al., 2023)	67.53	89.58	77.07	76.63	77.14
Cross-encoder					
MonoBERT (Nogueira and Cho, 2019)	65.26	85.34	74.48	73.25	73.99
Longformer (Beltagy et al., 2020)	68.51	89.58	77.07	76.95	77.81
Re2g(Glass et al., 2022)	59.28	74.59	66.78	65.51	65.88
Generative model					
ListT5(Yoon et al., 2024)	72.68	90.21	80.65	79.65	80.22
RepLLaMA-7B(Ma et al., 2024)	73.94	92.73	82.18	81.29	81.96
RankZephyr-7B†(Pradeep et al., 2023)	27.47	60.80	40.47	41.86	41.95
PRP-Sliding-20B†(Qin et al., 2024)	21.72	61.70	37.85	36.80	36.91
Late-interaction					
ColBERTv2(Santhanam et al., 2022)	74.59	94.03	83.82	82.40	82.99
GFPDR (Ours)	**85.45**	**96.74**	**91.03**	**90.17**	**90.42**

Table 3: The experiment results of retrieval methods on *Procedural Questions*. † denotes zero-shot methods.

No.	Method	Hit@1	MRR@10
I	GFPDR	85.45	90.42
II	I w/o $\mathcal{L}_{GDM}$	84.04	89.37
III	II w/o EAGAM	82.95	88.62
IV	III w/o init sepa.	81.54	87.58

Table 4: Ablation study for GFPDR.

4.3 Main Results

We evaluate the performance on the test set of *Procedural Questions* using Hit@n, NDCG@3, and MRR@n, as shown in Table 3. Our model achieves the best performance across all metrics, with the most significant improvement of +10.86% in Hit@1, highlighting its superior ability to distinguish between similar procedural documents by integrating sequence structure features of procedural graphs. We observe distinct performance gaps between different retriever architectures in this scenario. Sparse retrievers underperform due to their reliance on lexical matching, which fails to capture procedural sequence structure. Bi-encoder models perform poorly due to the limited capacity of single vector representation. For cross-encoder models, Longformer performs much better than MonoBERT and Re2g by handling longer context inputs. RepLLaMA shows competitive performance with better semantic representation, and ListT5 performs well as it uses a novel sort algorithm for listwise reranking. In contrast, zero-shot LLM-based rerankers perform significantly worse, suggesting these methods do not yet generalize well in procedural document retrieval. ColBERTv2 is the strongest among baselines, likely due to its fine-grained similarity computation, which can effectively leverage the detailed procedural queries in procedural document retrieval. The zero-shot results of baselines are provided in App. D.2.

4.4 Ablation Study

Table 4 shows the performance of GFPDR with progressively removed components. In Experiment II, the model is trained with InfoNCE loss only. Experiment III removes the Edge-Aware Graph Module, and Experiment IV shares language model parameters between the query and document encoders instead of using separate initializations. Additional results on NDCG@3, MRR@5, and Hit@5 are provided in App. Table 6.

Comparing Experiments I and III, the proposed Edge-Aware Graph Module proves to be effective, leading to improvements of +2.5% in Hit@1 and +1.8% in MRR@10. It means that GFPDR can leverage the sequence structure features captured

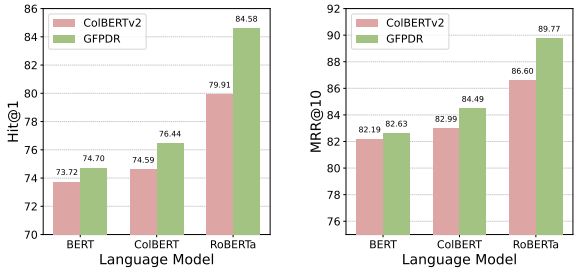

Figure 4: Impact study for language model.

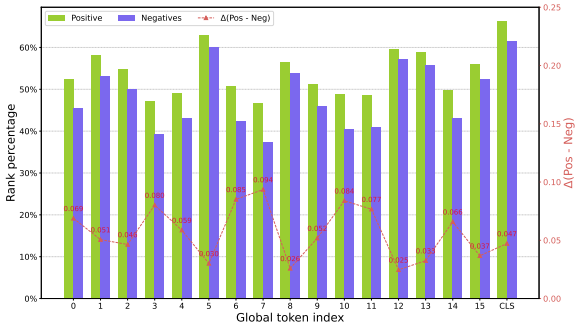

Figure 5: Rank percentage of global tokens during similarity computation. "Positive" refers to their rank in the positive document, and "Negatives" in the negatives.

by the graph module. Experiments I and II, we observe a 1.05% drop in MRR@10 after removing the GDM loss, indicating that the loss helps better align the representations of the graph and text. Experiments III and IV show that using separate language models for the query and document encoders performs better than sharing. We hypothesize it is because the steps in queries are always sequential, while procedural documents exhibit more complex sequence structures. Separate encoders allow better modeling of these distinct characteristics.

Impact Study of Language Model: We observe that the choice of language model backbone significantly affects retrieval performance. As shown in Figure 4, we compare GFPDR and ColBERTv2 initialized with three different backbones. GFPDR achieves the best performance when using RoBERTa-base. We hypothesize the strong semantic representation and generalization capacity of RoBERTa make it better to capture long-range dependencies in both procedural graphs and texts. Additionally, GFPDR consistently outperforms ColBERTv2 across all backbones, highlighting the effectiveness of our method. We discuss the impact of the number of global tokens in App. D.3.

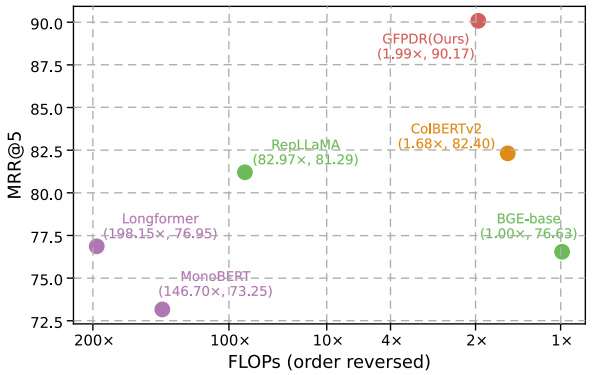

Figure 6: FLOPs of online computation at query time.

4.5 Further Analysis

To evaluate the capabilities of global tokens, we analyze their role during similarity computation. Specifically, all document tokens are computed with each query token via doc-product, and we measure the average rank percentage of the dot-product scores of global tokens relative to other document tokens for each query token. This calculation is performed separately for positive and negative documents.

We evaluate two GFPDR variants with 16 and 32 global tokens. As shown in Figure 5 and App. Figure 13, global tokens in positive documents consistently achieve higher rankings, suggesting they carry a denser semantic representation compared to most regular tokens. Moreover, the gap in average rank percentage between positive and negative documents highlights the sensitivity of global tokens to the differences between query and irrelevant documents. Most global tokens exhibit larger margins than the [CLS] token, indicating they offer stronger discriminative signals for procedural document retrieval. Additionally, we provide a case study in App. D.5 to illustrate the effectiveness of GFPDR.

Efficiency in FLOPs: We report per-query FLOPs in Figure 6, focusing on the online computations required at query time. BGE-base is the most efficient due to its bi-encoder architecture. GFPDR achieves comparable efficiency while outperforming others in MRR@5. Longformer and MonoBERT incur much higher costs, since their inference is fully performed online. Overall, GFPDR offers a better trade-off between efficiency and effectiveness. Measurement details are provided in App. D.4. Additional analyses, including domain-specific performance, false negative evaluation, and graph construction analysis, are presented in App.D.6.

5 Conclusion

In this paper, we address the problem of procedural document retrieval, where both step content and execution sequence structure should be considered between queries and documents. While existing retrieval methods primarily focus on learning topical content, they overlook the complex sequence structures inherent in procedural documents. To tackle this, we introduce a new dataset for procedural document retrieval and propose a retriever that integrates procedural graphs with document representations. Extensive experiments demonstrate the effectiveness of our approach.

Limitations

Our study has some limitations. First, the procedural graphs we construct may contain noise, such as nodes representing multiple actions. The analysis of graph construction in the Appendix shows that there is a correlation between graph imperfections and retrieval failures. We believe a more robust method for procedural graph construction can help model better capture the sequence structure.

Second, like many retrieval datasets, our dataset may suffer from false negatives, where positive documents are incorrectly labeled as negatives. For example, some procedural documents have largely similar step contents and execution sequences but differ in titles. Although we conduct an initial experiment about false negatives in the Appendix, a more comprehensive evaluation would be necessary to fully understand its impact.

Third, our corpus primarily consists of WikiHow-style procedural documents, which are relatively detailed. In contrast, documents from other sources such as Allrecipes[3] are more concise. Expanding the evaluation to include more diverse sources would better test the generalization capability of our method.

Finally, while our experiments show that LLM-based rerankers perform poorly on procedural document retrieval, we have not deeply explored this phenomenon. Given the increasing attention to LLM-based retrievers and rerankers, a direction for future work is to investigate how procedural graphs can be effectively integrated with LLMs, enabling rerankers to better capture sequence structures.

[3] https://www.allrecipes.com/

Acknowledgements

This research is supported by the National Natural Science Foundation of China (62476097), the Fundamental Research Funds for the Central Universities, South China University of Technology (x2rjD2250190), Guangdong Provincial Fund for Basic and Applied Basic Research—Regional Joint Fund Project (Key Project) (2023B1515120078), Guangdong Provincial Natural Science Foundation for Outstanding Youth Team Project (2024B1515040010), the National Natural Science Foundation of China (62276072), Guangxi Natural Science Foundation Key Project (2025GXNSFDA069017).

References

AI@Meta. 2024. Llama 3 model card.

Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, and 1 others. 2016. Ms marco: A human generated machine reading comprehension dataset. *arXiv preprint arXiv:1611.09268*.

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

Iz Beltagy, Matthew E. Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv:2004.05150*.

Kurt Bollacker, Robert Cook, and Patrick Tufts. 2007. Freebase: A shared database of structured general human knowledge. In *AAAI*, volume 7, pages 1962–1963.

João Coelho, Bruno Martins, Joao Magalhaes, Jamie Callan, and Chenyan Xiong. 2024. Dwell in the beginning: How language models embed long documents for dense retrieval. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 370–377, Bangkok, Thailand. Association for Computational Linguistics.

Zhuyun Dai and Jamie Callan. 2019. Deeper text understanding for ir with contextual neural language modeling. In *Proceedings of the 42nd international ACM SIGIR conference on research and development in information retrieval*, pages 985–988.

DeepSeek-AI. 2024. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *Preprint*, arXiv:2405.04434.

Weihong Du, Wenrui Liao, Hongru Liang, and Wenqiang Lei. 2024. Paged: A benchmark for procedural graphs extraction from documents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10829–10846.

Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. 2024. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*.

BV Elasticsearch. 2018. Elasticsearch. *software], version*, 6(1).

Thibault Formal, Carlos Lassance, Benjamin Piwowarski, and Stéphane Clinchant. 2022. From distillation to hard negative sampling: Making sparse neural ir models more effective. In *Proceedings of the 45th international ACM SIGIR conference on research and development in information retrieval*, pages 2353–2359.

Luyu Gao, Zhuyun Dai, and Jamie Callan. 2020. Modularized transfomer-based ranking framework. *arXiv preprint arXiv:2004.13313*.

Yunfan Gao, Yun Xiong, Meng Wang, and Haofen Wang. 2024. Modular rag: Transforming rag systems into lego-like reconfigurable frameworks. *arXiv preprint arXiv:2407.21059*.

Michael Glass, Gaetano Rossiello, Md Faisal Mahbub Chowdhury, Ankita Naik, Pengshan Cai, and Alfio Gliozzo. 2022. Re2g: Retrieve, rerank, generate. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2701–2715.

Janis E Jacobs and Scott G Paris. 1987. Children's metacognition about reading: Issues in definition, measurement, and instruction. *Educational psychologist*, 22(3-4):255–278.

Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. *arXiv preprint arXiv:2004.04906*.

Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of naacL-HLT*, volume 1. Minneapolis, Minnesota.

Omar Khattab and Matei Zaharia. 2020. Colbert: Efficient and effective passage search via contextualized late interaction over bert. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pages 39–48.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, and 1 others. 2019. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466.

Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, Sören Auer, and 1 others. 2015. Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic web*, 6(2):167–195.

Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. 2023. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In *International conference on machine learning*, pages 19730–19742. PMLR.

Zhuoqun Li, Hongyu Lin, Yaojie Lu, Hao Xiang, Xianpei Han, and Le Sun. 2024. Meta-cognitive analysis: Evaluating declarative and procedural knowledge in datasets and large language models. *arXiv preprint arXiv:2403.09750*.

Ruipu Luo, Qi Zhu, Qin Chen, Siyuan Wang, Zhongyu Wei, Weijian Sun, and Shuang Tang. 2021. Operation diagnosis on procedure graph: The task and dataset. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 3288–3292.

Xueguang Ma, Liang Wang, Nan Yang, Furu Wei, and Jimmy Lin. 2024. Fine-tuning llama for multi-stage text retrieval. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2421–2425.

Rodrigo Nogueira and Kyunghyun Cho. 2019. Passage re-ranking with bert. *arXiv preprint arXiv:1901.04085*.

Kuntal Kumar Pal, Kazuaki Kashihara, Pratyay Banerjee, Swaroop Mishra, Ruoyu Wang, and Chitta Baral. 2021. Constructing flow graphs from procedural cybersecurity texts. *arXiv preprint arXiv:2105.14357*.

Ronak Pradeep, Sahel Sharifymoghaddam, and Jimmy Lin. 2023. Rankzephyr: Effective and robust zero-shot listwise reranking is a breeze! *arXiv preprint arXiv:2312.02724*.

Zhen Qin, Rolf Jagerman, Kai Hui, Honglei Zhuang, Junru Wu, Le Yan, Jiaming Shen, Tianqi Liu, Jialu Liu, Donald Metzler, and 1 others. 2024. Large language models are effective text rankers with pairwise ranking prompting. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 1504–1518.

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. *Journal of machine learning research*, 21(140):1–67.

Haopeng Ren, Yushi Zeng, Yi Cai, Zhenqi Ye, Li Yuan, and Pinli Zhu. 2024. Grounded multimodal procedural entity recognition for procedural documents: A new dataset and baseline. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 7971–7981.

Haopeng Ren, Yushi Zeng, Yi Cai, Bihan Zhou, and Zetao Lian. 2023. Constructing procedural graphs with multiple dependency relations: A new dataset and baseline. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 8474–8486.

Stephen Robertson, Hugo Zaragoza, and 1 others. 2009. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389.

Keshav Santhanam, Omar Khattab, Jon Saad-Falcon, Christopher Potts, and Matei Zaharia. 2022. Colbertv2: Effective and efficient retrieval via lightweight late interaction. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3715–3734.

Keisuke Shirai, Hirotaka Kameko, and Shinsuke Mori. 2023. Towards flow graph prediction of open-domain procedural texts. *arXiv preprint arXiv:2305.19497*.

Weihang Su, Yichen Tang, Qingyao Ai, Zhijing Wu, and Yiqun Liu. 2024. Dragin: Dynamic retrieval augmented generation based on the information needs of large language models. *arXiv preprint arXiv:2403.10081*.

Niket Tandon, Gerard De Melo, Abir De, and Gerhard Weikum. 2015. Knowlywood: Mining activity knowledge from hollywood narratives. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 223–232.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903*.

Denny Vrandečić and Markus Krötzsch. 2014. Wikidata: a free collaborative knowledgebase. *Communications of the ACM*, 57(10):78–85.

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

Te-Lin Wu, Alex Spangher, Pegah Alipoormolabashi, Marjorie Freedman, Ralph Weischedel, and Nanyun Peng. 2022. Understanding multimodal procedural knowledge by sequencing multimodal instructional manuals. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4525–4542.

Shitao Xiao, Zheng Liu, Peitian Zhang, and Niklas Muennighoff. 2023. C-pack: Packaged resources to advance general chinese embedding. *Preprint*, arXiv:2309.07597.

Junhan Yang, Zheng Liu, Chaozhuo Li, Guangzhong Sun, and Xing Xie. 2023. Longtriever: a pre-trained long text encoder for dense document retrieval. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3655–3665.

Soyoung Yoon, Eunbi Choi, Jiyeon Kim, Hyeongu Yun, Yireun Kim, and Seung-won Hwang. 2024. Listt5: Listwise reranking with fusion-in-decoder improves zero-shot retrieval. *arXiv preprint arXiv:2402.15838*.

Li Yuan, Yi Cai, Haopeng Ren, and Jiexin Wang. 2024. A logical pattern memory pre-trained model for entailment tree generation. *Preprint*, arXiv:2403.06410.

Li Yuan, Jin Wang, Liang-Chih Yu, Yi Cai, and Xuejie Zhang. 2025. Pruning self-attention with local and syntactic dependencies for aspect sentiment triplet extraction. *IEEE Transactions on Audio, Speech and Language Processing*.

Siyu Yuan, Jiangjie Chen, Ziquan Fu, Xuyang Ge, Soham Shah, Charles Robert Jankowski, Yanghua Xiao, and Deqing Yang. 2023. Distilling script knowledge from large language models for constrained language planning. *arXiv preprint arXiv:2305.05252*.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. 2019. Bertscore: Evaluating text generation with bert. *arXiv preprint arXiv:1904.09675*.

Shuyan Zhou, Li Zhang, Yue Yang, Qing Lyu, Pengcheng Yin, Chris Callison-Burch, and Graham Neubig. 2022. Show me more details: Discovering hierarchies of procedures from semi-structured web data. *arXiv preprint arXiv:2203.07264*.

Yucheng Zhou, Tao Shen, Xiubo Geng, Chongyang Tao, Jianbing Shen, Guodong Long, Can Xu, and Daxin Jiang. 2024. Fine-grained distillation for long document retrieval. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19732–19740.

Honglei Zhuang, Zhen Qin, Rolf Jagerman, Kai Hui, Ji Ma, Jing Lu, Jianmo Ni, Xuanhui Wang, and Michael Bendersky. 2023. Rankt5: Fine-tuning t5 for text ranking with ranking losses. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2308–2313.

Algorithm 1 Document Token Length Reduction

Require: Document steps $\{S_1, S_2, \ldots, S_N\}$, Token limit L

Ensure: Reduced document D

```
Split each step S_i into sentences {s_i1, s_i2, ..., s_im}
Tokenize each sentence s_ij and compute total token length T_total
while T_total > L do
    Remove the last sentence from each step, if available
    Recompute T_total
end while
Reconstruct the document D by concatenating remaining sentences
return D
```

A Preliminary Study on Retrieval-Augmented Solution Generation

To investigate whether providing relevant procedural documents helps generate more concise solutions to user's erroneous operations, we conduct a preliminary experiment using a subset of our proposed dataset. Specifically, we vary the number of reference documents k ($k = 0, 1, 5, 10$) supplied to the LLM for answer generation. When $k > 0$, the reference set always includes the positive (i.e., most relevant) document.

The ground truth answers are concise solutions manually annotated by humans. We evaluate the responses generated by DeepSeek-V2.5 using automatic metrics, including METEOR (Banerjee and Lavie, 2005) and BERTScore (Zhang et al., 2019). The results are presented in Figure 7. We observe that the quality of answers is lowest when no reference document is provided. When the LLM is given only the positive document, the quality significantly improves, achieving a +4.5% increase in Meteor and a +3.8% increase in BERTScore compared to the zero-shot setting. However, as the number of reference documents increases, the quality of the generated answers tends to decline. We find that this decline is due to the LLM extracting generalized suggestions from multiple procedural documents rather than providing more targeted solutions to specific failures.

Although we adopt a native RAG paradigm in our experimental setup, the results demonstrate that retrieving relevant procedural documents significantly improves answer quality—particularly when the retrieved document closely aligns with the user's original operations. Current RAG systems have many variants with different retrieval strategies. However, no matter what retrieval strategy the RAG system uses, it still relies on a robust retriever to provide reliable references. For example, Iterative RAG (Gao et al., 2024) performs multiple rounds of retrieval, refining the query based on previous outputs. Despite its iterative nature, it still depends on an effective retriever to identify relevant documents. Similarly, Demand-based RAG (Su et al., 2024) dynamically determines whether to retrieve external documents based on the query. This approach is similar to our setting and likewise requires a retriever when internal knowledge is insufficient. Graph-based RAG (Edge et al., 2024) extends retrieval to structured knowledge sources, such as knowledge graphs. However, building a procedural graph system for RAG is particularly challenging, as it requires modeling both intra-document step relationships and inter-document dependencies, making it difficult to implement in practice.

B Procedural Graph Construction

An example of a generated procedural graph is shown in Figure 12. We adopt the method proposed by Du et al., 2024 to construct procedural graphs by extracting nodes and edges from procedural documents using LLMs with few-shot in-context learning and chain-of-thought reasoning. This approach involves summarizing key procedural steps and extracting node-edge-node triples in text format. Considering our task involves simpler node and edge types compared to Du et al., 2024, we adjust their prompts accordingly. To further enhance the model's understanding, we provide high-quality document-graph examples that cover all possible node and edge types. Due to the high computational cost of constructing procedural graphs for the entire corpus, we focus on generating graphs only for documents serving as negatives in queries. Notably, around 95% of the extracted procedural graphs conform to the required graph structure.

C Dataset

Figure 8 illustrates the overall framework, while Table 11 presents the designed prompts for each turn. Queries for the training and development sets are generated using DeepSeek. To enhance nar-

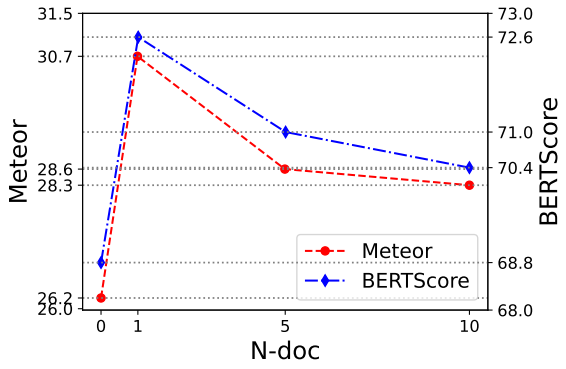

Figure 7: Performance of answer generation by LLM with different numbers of reference documents.

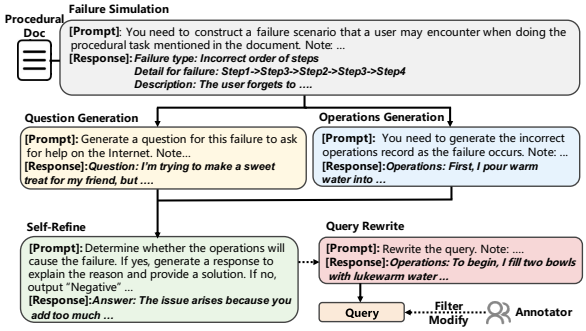

Figure 8: The illustration of the overall dataset construction framework.

rative diversity, test set queries are generated by both DeepSeek and LLaMA3, then mixed and reviewed by annotators. Figures 9 show the statistics of failure types and query domains for the test set.

Additionally, since the documents in Yuan et al., 2023 only contain key steps for completing procedural tasks, we leverage DeepSeek to reconstruct them into WikiHow-style procedural documents, incorporating more detailed steps and extra tips. Considering the potential real-world issue of procedural document redundancy due to multi-source origins, we synthetically generate additional documents by rewriting the top 20 negative test set documents using LLMs, which also increases procedural document diversity.

For human review, annotators adhere to the following guidelines:

1) Validate the positive document associated with the query, excluding recommendation-style documents (e.g., "How to be a good person").

2) Ensure that the LLM-generated procedure leads to task failure and that the failure is causally linked to the procedural deviation.

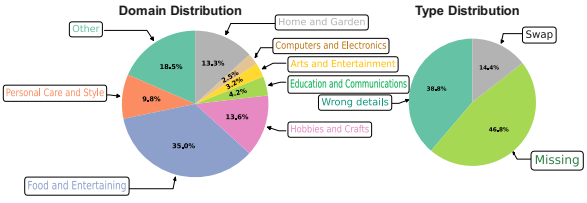

Figure 9: Domains and types statistics of the test set. The "Other" category includes queries for which the category labels of positive documents are missing, as well as those corresponding to domains with fewer than 20 queries.

3) Annotators may add, remove, or modify step details or order, provided that such adjustments do not affect the final execution outcome or the failure reason.

D Experiments

D.1 Experiments Settings

Existing studies (Yuan et al., 2025) have employed various methods to enhance the robustness of GNN across different domains. To improve robustness of GFPDR against potential noise in procedural graphs generated by LLMs, we introduce structural perturbations during training. Specifically, we randomly add edges and merge nodes in the graph to simulate noise, and treat these perturbed graphs as negative samples in the GDM loss. This encourages the Edge-Aware Graph Attention Module to recognize and filter out incorrect structural information. The perturbation rate is set to 0.3. During GFPDR training, hard negative samples are dynamically mined from the top-5 documents retrieved by BM25. We found that the in-batch negative strategy did not yield performance improvements in our setting, and thus it was not adopted. The model is trained on a single NVIDIA L20 GPU for approximately 10 hours, using automatic mixed precision and gradient accumulation for efficiency.

We provide the hyper-parameters for both baseline methods and our model in Table 5. ElasticSearch (Elasticsearch, 2018) is used to implement the BM25 algorithm with its default parameter settings. For SPLADE++, we use the *splade-cocondenser-ensembledistil* language model checkpoint [4]. Except for MonoBERT and Longformer, all neural network models are fine-tuned on their respective checkpoints, which were initially trained

[4] https://huggingface.co/naver/splade-cocondenser-ensembledistil

Model	lr	batch size	# negatives	Len(query)	Len(doc)	in-batch negatives
Longtriever	1e-5	8	2	300	780	✓
BGE-base	1e-5	16	5	300	512	✗
SimLM	2e-5	16	50	300	512	✓
T5-2K	5e-6	4	4	300	512	✓
ListT5	1e-4	16	4	-	-	✗
Longformer	1e-5	16	2	400	800	✗
MonoBERT	3e-6	16	2	-	-	✗
RepLLaMA-7B	1e-4	8	2	300	512	✗
ColBERTv2	1e-4	16	2	300	512	✓
Ours	1e-5	8	2	300	512	✗

Table 5: The hyper-parameters for the baseline methods and our approach are summarized. # Negatives refers to the number of negative documents per sample. Len(query) and Len(doc) indicate the maximum input lengths for queries and documents, respectively. MonoBERT concatenates the query and document, truncating the input to a maximum length of 512 tokens.

Method	NDCG@3	MRR@5	Hit@5
GFPDR	91.03	90.17	96.74
I w/o $\mathcal{L}_{GDM}$	89.93	89.04	96.09
II w/o EAGAM	89.57	88.33	95.66
III w/o init sepa.	88.30	87.23	95.01

Table 6: Ablation studies for GFPDR on NDCG@3, MRR@5 and Hit@5.

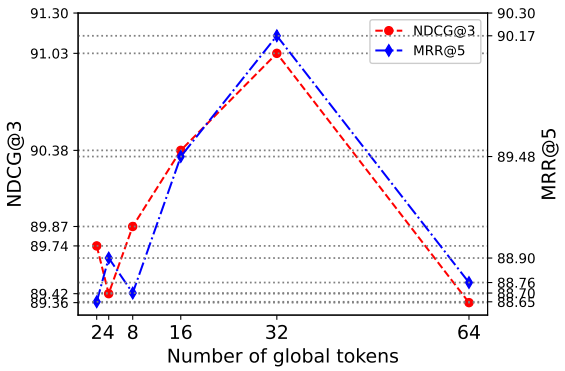

Figure 10: Impact study for the number of global tokens.

on retrieval datasets like *MSMARCO*, as training on large-scale retrieval datasets can serve as an effective pretraining. We present the prompt templates of the LLM-based rerankers RankZephyr and PRP-Sliding in Table 10. To ensure faithful reproduction, we directly adopt the original templates from their respective papers without modification.

For some procedural documents that exceed the 512-token limit (the maximum input length for several baseline methods using BERT as a language model), we employ a token reduction algorithm, as described in Algorithm 1, to reduce the total token count while retaining as much semantic information as possible. Specifically, we observe that the latter part of procedural documents typically contains declarative sentences that provide supplementary information. Removing these sentences has minimal impact on the overall sequence structure of the procedure. Based on this observation, when a document exceeds the token limit, we first split each step into individual sentences, then iteratively remove the last sentence from each step until the document's length satisfies the token constraint. This approach minimizes the loss of important content while reducing the document length.

D.2 Zero-shot Retrieval Performance

We evaluate the zero-shot retrieval performance of models with different architectures, as shown in Table 7. RepLLaMA-7B achieves the best overall performance, likely due to the strong representation capabilities of large language models. Among lightweight baselines, ColBERTv2 performs best, which we attribute to its fine-grained similarity computation, particularly effective for handling long procedural queries. ListT5 performs poorly in the zero-shot setting, but its strong performance after fine-tuning suggests that generative models hold promise for procedural document retrieval. Bi-encoders, particularly BGE-base, demonstrate strong performance, which can be attributed to their advanced distillation techniques. However, their reliance on a single-vector representation limits their capacity, leading to limited performance improvements after fine-tuning.

Model	Hit@1	Hit@5	NDCG@3	MRR@10
BM25	22.91	54.72	36.65	36.34
SPLADE++	39.2	66.45	50.93	50.76
Longtriever	19.22	47.67	29.76	31.38
BGE-base	30.18	65.47	42.97	45.19
SimLM	25.51	48.10	33.33	36.87
T5-2K	26.17	55.59	37.30	39.09
ListT5	15.89	41.24	25.71	27.40
RankZephyr	27.47	60.80	40.47	41.95
PRP-Sliding	21.72	61.70	37.85	36.91
ColBERTv2	36.48	69.82	50.32	50.53
RepLLaMA	**42.78**	**77.63**	**58.09**	**56.46**

Table 7: The zero-shot performance of baselines with different architectures.

Model	Complexity	TFLOPs ($\times 10^{12}$FLOPs)	Times
BGE-base	$O(q^2 + n)$	0.017	1×
ColBERTv2	$O(q^2 + n \times q \times d)$	0.028	1.68×
GFPDR(Ours)	$O(q^2 + n \times q \times d)$	0.033	1.99×
RepLLaMA	$O(q^2 + n)$	1.395	82.97×
MonoBERT	$O(n \times (q + d)^2)$	2.467	146.70×
Longformer	$O(n \times (q + d)^2)$	3.332	198.15×

Table 8: Per-query online inference cost (in TFLOPs) and relative computational complexity across different model architectures. q and d denotes to the length of query and document, while n denotes to the number of documents.

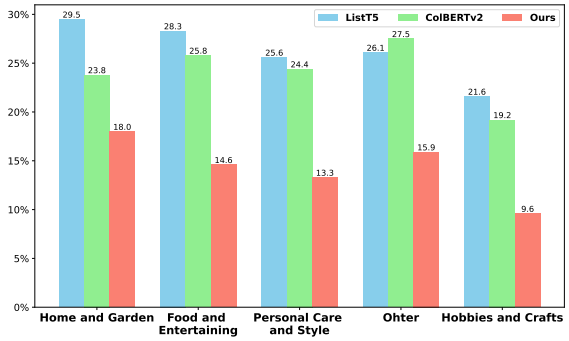

Figure 11: Error rate comparison across top-5 domains with highest retrieval failure rates.

D.3 Impact Study of The Number of Global Tokens

We investigate the impact of varying the number of global tokens (M = 2, 4, 8, 16, 32, 64) on the performance of GFPDR. The corresponding results are presented in Figure 10. The model achieves the best performance when $M = 32$, and performance drops when M increases to 64. We hypothesize that an excessive number of global tokens may hinder their ability to generate compact and information-rich representations. In contrast, too few global tokens may fail to capture the complete sequence structure features of the procedural graph.

D.4 Complexity Analysis

We measure the FLOPs of different models using the FlopCountAnalysis tool from the fvcore library[5]. Specifically, we compute the total FLOPs involved in query-time (online) computation and report the average per query. For late-interaction models such as GFPDR and ColBERTv2, online computation includes query encoding and multi-vector similarity calculation, while document encoding can be done offline, similar to bi-encoder models. In contrast, cross-encoders must concatenate the query with each candidate document and encode the entire pair online, resulting in significantly higher online computational cost. For LLM-based rerankers, the complexity depends on the specific reranking strategy used. Regardless of the approach, their query-time FLOPs are substantially higher than those of lightweight retrieval models. The measured FLOPs and computational complexity of all baselines are summarized in Table 8.

D.5 Case Study

To intuitively explain the effectiveness of our proposed model, we conduct a case study on procedu-

[5] https://github.com/facebookresearch/fvcore

ral document retrieval by comparing ColBERTv2 (Santhanam et al., 2022) and GFPDR, as shown in Figure 14. The query describes a failure in making a cocoa drink, where the candy does not dissolve properly. We analyze the top-ranked documents retrieved by two models. We can observe that both documents share the same topic of making cocoa drinks and contain key actions, such as *pour water*, *microwave the drink*, and *add cream* which align with the query. However, ColBERTv2 fails to capture the actions of adding mint candy. Two steps (i.e., *chill cocoa drink* and *add ice*) of the document retrieved by ColBERTv2 are mismatched with query's execution sequence structure. Additionally, the execution sequence of microwaving the drink differs between the query and this document. In the query, the user microwaves the water first and then pours it into a mug containing cocoa powder and a candy cane, whereas in the document, the cocoa powder is added to the mug first before microwaving. This difference in sequence may affect the heating time. In contrast, GFPDR correctly retrieves a document that closely aligns with the query's operations in both step content and sequence structure. The experimental results in this case study highlight GFPDR can better capture the execution sequence structure of procedural documents, along with the detailed steps within the structure, leading to more accurate retrieval compared to ColBERTv2.

D.6 Further Analysis

D.6.1 Domain Analysis

We investigate which domains of queries pose challenges for retrieval and where our model outperforms the baselines. Figure 11 presents the top five domains with the highest retrieval error rates. Our model exhibits superior performance in all domains compared to the other two baselines. We hypothesize that sequence structure plays a more critical role in these domains and our model leverages its strengths effectively. In contrast, two baselines underperform in *Home and Garden* and *Food and Entertaining*, indicating that sequence structure may be more important in these domains.

D.6.2 Experiments about False Negative

We conduct a simple experiment about the false negative. Considering the scale of the document corpus (360K documents), large-scale detection of false negatives (i.e., documents with similar content but different titles) presents a computational challenge. To estimate their influence, we use SentenceBERT to compute similarity scores for the top-50 negative documents in the test set. Documents with scores above a threshold m ($m = 0.9$ and $m = 0.95$) are treated as duplicates. Results are shown in Table 9. We observe that false negatives have a limited impact on evaluation. However, due to time and resource constraints, we are unable to analyze the entire corpus, which may influence the training process.

D.6.3 Analysis of Graph Construction Error

To better understand the impact of graph construction quality on retrieval performance, we manually inspected 75 randomly sampled queries. Among these, 21 cases resulted in retrieval failures, while the remaining 54 were successfully retrieved. Across all samples, we found that 26 positive documents contained imperfections in their procedural graphs rather than outright errors. Among them, 11 graphs exhibited minor issues such as missing partial content, 13 had moderate flaws related to few incorrect dependency extraction, and 4 showed more serious structural inconsistencies. Notably, the error rate in graphs corresponding to incorrectly retrieved queries was 57%, compared to only 25% in the correctly retrieved ones.

These findings suggest a clear correlation between graph construction errors and retrieval failures. Inaccurate or noisy graphs can misrepresent the sequence structure and dependencies of procedural steps, potentially leading the retriever to misjudge document relevance.

D.7 Other Metrics on Ablation Study

Table 6 reports the corresponding NDCG@3, MRR@5, and Hit@5 scores on ablation study for GFPDR.

Threshold	Model	Hit@1	Hit@5	NDCG@3	MRR@5
Original	GFPDR	85.45	96.74	91.03	90.17
	ColBERTv2	74.59	94.03	83.82	82.40
$m = 0.9$	GFPDR	86.64	96.74	91.55	90.82
	ColBERTv2	75.68	94.57	84.68	83.27
$m = 0.95$	GFPDR	85.67	96.74	91.11	90.28
	ColBERTv2	74.70	94.03	83.86	82.46

Table 9: Comparison of GFPDR and ColBERTv2 under different thresholds m

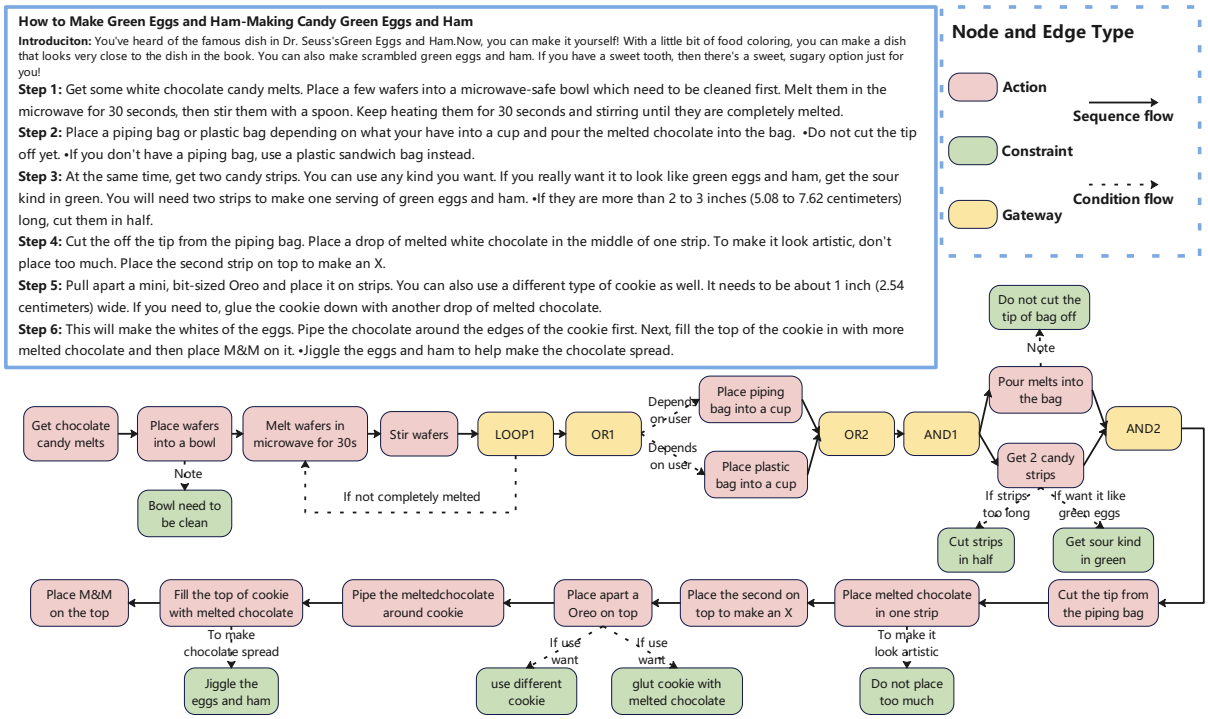

Figure 12: An example of a procedural graph constructed.

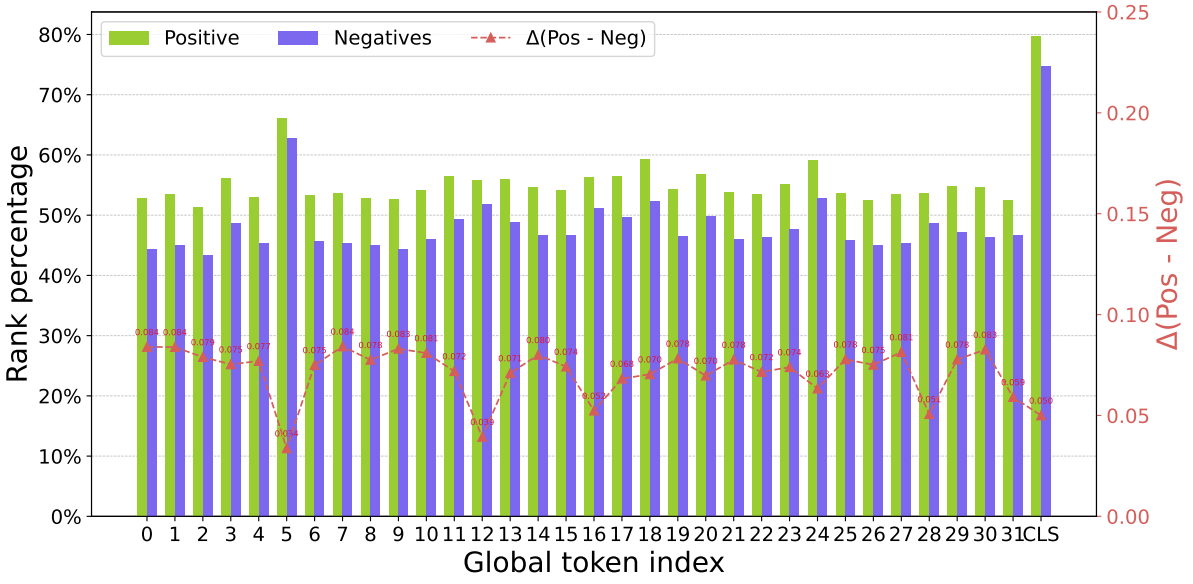

Figure 13: Rank percentage analysis of GFPDR with 32 global tokens during similarity computation.

Model	Prompt
RankZephyr	<|system|> You are RankLLM, an intelligent assistant that can rank passages based on their relevancy to the query. <|user|> I will provide you with {num} passages, each indicated by a numerical identifier []. Rank the passages based on their relevance to the search query: {query}. [1] {passage 1} [2] {passage 2} ... [{num}] {passage num} Search Query: {query}. Rank the {num} passages above based on their relevance to the search query. All the passages should be included and listed using identifiers, in descending order of relevance. The output format should be [] > [], e.g., [4] > [2]. Only respond with the ranking results, do not say any word or explain.
PRP-Sliding	Given a query query, which of the following two passages is more relevant to the query? Passage A: document1 Passage B: document2 Output Passage A or Passage B:

Table 10: Prompt templates of LLM-based rerankers.

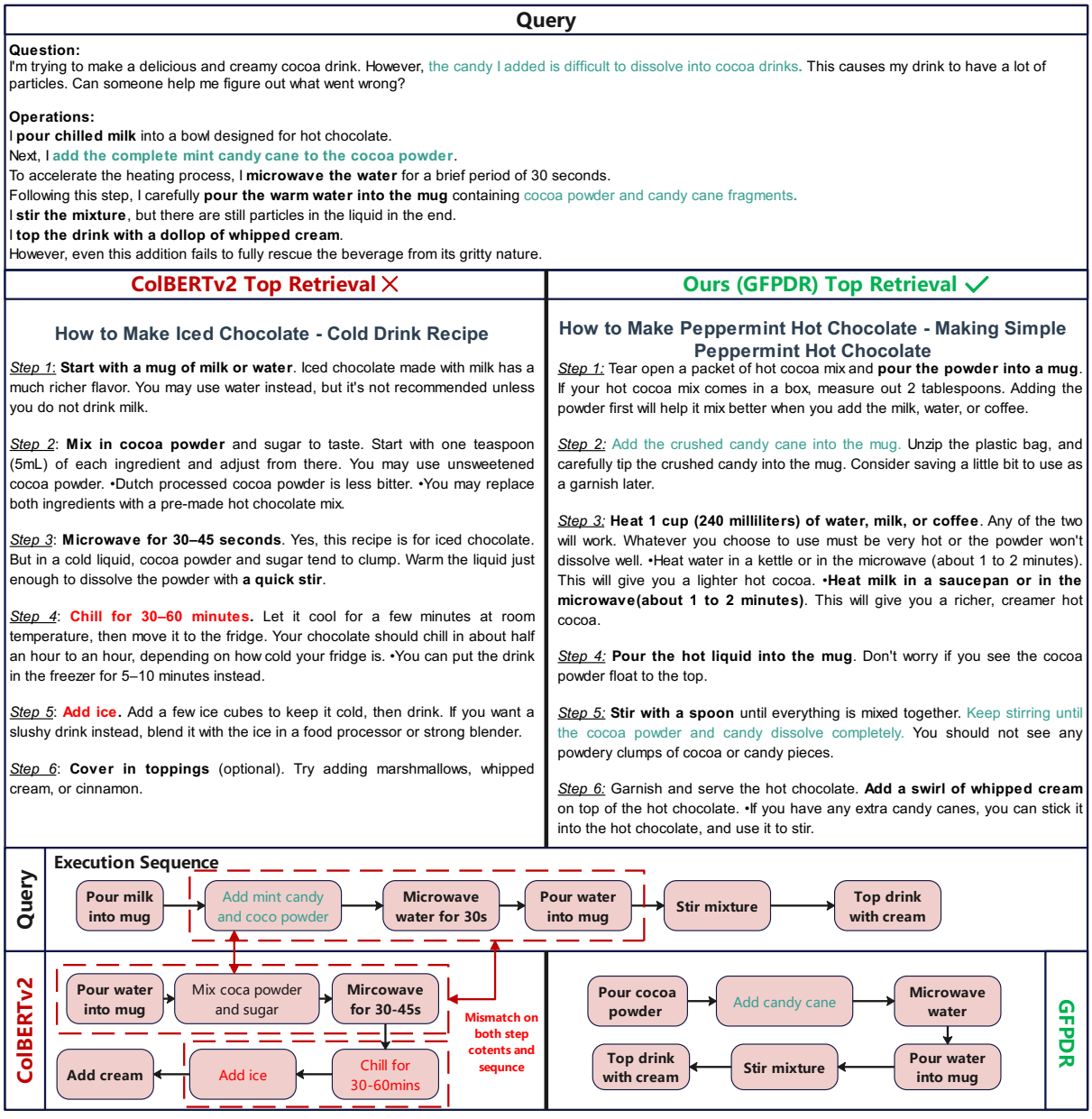

Figure 14: Retrieval comparison between *ColBERTv2* and our proposed model *GFPDR* using a query about a cocoa drink failure, where the candy does not dissolve properly. Both *ColBERTv2* and *GFPDR* retrieve documents have the same topic of making cocoa drinks. Key actions, such as **pour water**, **microwave the drink**, and **add cream**, are shared between the query and both documents (highlighted in **bold**). Actions in red represent mismatched operations with the query, while actions in green are exclusive to the query and the top retrieval document from *GFPDR*.

Phase	Prompt
Failure Simulation	{*Procedural Document*} Read the above procedural document, which is standard and correct. However, in reality, people often fail to execute tasks related to procedural documents due to various reasons, which may include the following category: 1. Incorrect execution order of steps: The order of steps has been swapped; 2. Missing steps: missing step(can be multiple steps); 3. Specific details of the step are incorrect: the time, amount, tools, or materials involved in a certain step are incorrect; You now need to construct a failure scenario that a user may encounter when executing the process task, and the reason for the failure is the type mentioned above. When constructing a failed situation, you need to pay attention to the following rules: A. If the reason for the failure is a missing step, ensure that the deleted step is not a necessary condition for the subsequent steps to be executed, that is, the subsequent steps can still be executed without the missing step, but it will cause the failure to occur. B. Ensure that the entire process still works causally and logically after modifying steps, that is, each step can still be executed, but it will cause the failure to occur. C. Try to ensure the uniqueness of the reason for the failure as much as possible, that is, the failure can only be caused by an error in this detail in the step, and it should be targeted. Now, according to the above requirements, if a failure situation can be generated, your output should include the type of failure reason, the specific reason for the failure, and a description of the failure situation. Here is an output format example: {*Examples*} If you are unable to generate a failure situation for this procedural document due to the above rules, simply output NO.
Question Generation	Suppose you don't know the cause of this failure, and you need to ask for help on the Internet. Now please generate a help question for this failure from the first person perspective. The questions you generate should include the following: 1. Briefly explain the procedural task you are currently working on. Note: You only need to explain what you are doing, not what you are using or how you are doing it. Try to use more diverse expressions, such as using different words or applying the programmatic task in different scenarios. Do not directly use the titles of procedural documents. You can use synonyms instead. 2. Explain your failure situation and provide a brief description of the failure. The questions you generate must not include the following: A. You must not mention information related to procedural documents, such as steps. You need to pretend that you have not read the procedural documents. B. You must not mention the reason for the failure you generated in the previous conversation, whether it is the type of reason or the specific reason. You need to pretend that you are unaware of the reason for the failure. Here are examples of output formats: {*Examples*} Now generate a question with less than 50 words.
Operations Generation	If the failure occurs during the execution of the procedural task due to the reason you mentioned. You need to generate this incorrect operations record to seek help on the Internet. NOTE: 1. The operations record you generate should be able to lead to the occurrence of failure situations. That is, if the reason for the failure is missing steps, you should exclude that step from your operations record and ensure that there are no logical or causal conflicts before and after the operation after excluding that step. If the reason for the failure is an incorrect execution order of steps, the corresponding execution order in your operations record should also be incorrect. Ensure that there are no logical or causal conflicts before and after the operation after swapping the execution order. If the reason for the failure is incorrect specific details of the step, your operations record should also include a description of that detail and ensure that there are no logical or causal conflicts before and after modifying the details. 2. You must provide more additional detailed information on the operation details, such as time, dosage, etc.. But ensure that these details do not affect the final result. 3. Do not directly mention procedural documents and the failure reason. Pretend not to have read the procedural documentation, but know the necessary steps. Here are two examples of output format: {*Examples*} Now generate operation records within 10 sentences. Remember the operations record you generate should be able to lead to the occurrence of failure situations due to the reason you mentioned.
Self-Refine	You now have the reason for the failure and the record of the failed operation. Generate a response for this failed operations record to explain the reason for the failure and provide a solution, with a length of no more than 100 words. If you find that the reason for the failure of the operation does not match the reason you mentioned in the previous conversation, simply output 'Negative'. NOTE: Do not mention the information about procedural document. That is, statements like 'according to step 3' should not appear. Here are two examples of output formats: {*Examples*} Now generate an answer.
Query Rewrite	{*Query*} The above content is an operations record. Now you need to rewrite this operations record. Use more diverse words(such as synonyms) while maintaining the same meaning. Especially in the expression of actions and nouns. In addition, you can also add extra details to some operations, such as enriching the actions before and after the operation, providing details on usage or time, etc. But it is important to ensure that the added details do not affect the final result. NOTE: The operations record is not completely correct, so please use an objective and neutral writing style when rewriting. Do not use positive words, such as "satisfying me" and "achieving perfection". Now rewrite the operation record within 10 sentences.

Table 11: Prompts for datasets generation.