Processing Queries with Complex Information Needs: University of Glasgow Terrier Team at TREC RAG 2025

Fangzheng Tian
f.tian.1@research.gla.ac.uk
University of Glasgow
Glasgow, United Kingdom

Debasis Ganguly
Debasis.Ganguly@glasgow.ac.uk
University of Glasgow
Glasgow, United Kingdom

Craig Macdonald
Craig.Macdonald@glasgow.ac.uk
University of Glasgow
Glasgow, United Kingdom

ABSTRACT

In our participation in the TREC 2025 Retrieval-Augmented Generation (RAG) Track, we investigate how existing RAG frameworks can be adapted to answer descriptive queries with complex information needs. Unlike short topical queries, TREC RAG 2025 queries are expressed as paragraph-length descriptions that often involve multiple aspects of a topic. To address this setting, we adapt two existing RAG paradigms: (1) a single-hop RAG pipeline that decomposes the original query into sub-queries and answers them independently, and (2) an iterative agentic RAG pipeline that decomposes and answers sub-queries in a cascading manner. We submitted 2 runs that instantiated these two paradigms. Our results show that directly adapting existing RAG systems designed for short queries provides a practical baseline for this task, but remains limited by the quality of query decomposition and long-horizon generation. According to the evaluation results, the generated answers often address only part of the original information need and fail to cover some vital nuggets. These findings highlight the limitations of current RAG frameworks when applied to complex, multi-aspect queries and suggest directions for future research on query decomposition and generation strategies in this emerging setting.

1 INTRODUCTION

In our participation in the TREC RAG 2025 Track, we aimed to develop Retrieval-Augmented Generation (RAG) pipelines tailored to the query format used in this year's task. Unlike the short topical queries commonly studied in information retrieval, TREC RAG 2025 queries are presented as descriptive paragraphs. For example,

"I'm trying to understand the various forms of discrimination and oppression people experience in the US, such as racial, gender, age, and housing. Can you explain their prevalence, how they affect individuals and society, and what laws or actions are in place to address them?"

Descriptive queries like this express complex information needs and typically ask about multiple aspects of a topic within a single description. On average, these descriptions contain 303.32 characters (45.22 words), making them approximately eight times longer than queries in benchmarks such as TREC Deep Learning and Natural Questions [4].

As the underlying information needs become more complex, we argue that query decomposition becomes an important component of an effective RAG system. In our submitted runs, we investigate two adaptations of existing RAG frameworks: (1) decomposing the original query into sub-queries at the start and performing RAG over them in parallel; and (2) decomposing the query iteratively and performing RAG over the resulting sub-queries in a cascading manner. The latter can be viewed as an agentic workflow, because the RAG model decides which sub-query to issue based on the information retrieved from external corpora.

The remainder of this paper is organised as follows: Section 2 describes the retrieval component used in our RAG pipelines. Section 3 presents two generation workflows designed for complex information needs and introduces our two submitted runs based on them. Section 4 summarises the performance of our submitted runs. Finally, Section 5 concludes the report and points out the direction for future study.

2 INDEXING AND RETRIEVAL

The TREC RAG track is built on the MS MARCO v2.1 passage collection [1], which contains over 113 million passages. Given the scale of this collection, an effective retrieval pipeline is essential for identifying relevant evidence that supports answer generation in a RAG system [8]. All the retrieval components in our experiments were implemented with PyTerrier [6].

During indexing, we build an E5 [9] FlexIndex[1] over the passage collection. At retrieval time, we adopt a two-stage retrieval architecture. In the first stage, we use the E5 bi-encoder dense retriever to retrieve candidate passages from the index. In the second stage, the top-20 retrieved candidates are re-ranked using the MonoT5 reranker [7]. Finally, the top 3 passages after re-ranking are selected as the context for answer generation, which we describe next in Section 3.

3 RUN DESCRIPTIONS

The main challenge of this year's TREC RAG task is how to organise generation for queries with complex, multi-aspect information needs. Unlike topical short queries, these descriptive long queries may not be effectively answered with a single retrieval-generation step. Instead, the model must identify and address multiple underlying sub-queries, either explicitly or implicitly, before producing the final answer. To deal with this challenge, we investigate two generation workflows built on top of the same retrieval pipeline. The first workflow decomposes the original query into subqueries in advance and answers them independently. The final answer is merged from the answers to the subqueries. The second workflow constructs the query plan incrementally, allowing the model to alternate between reasoning and retrieval iteratively until the final answer is reached.

[1]FlexIndex is a dense index format that allows for a variety of retrieval implementations and algorithms, see Pyterrier_dr at https://github.com/terrierteam/pyterrier_dr.

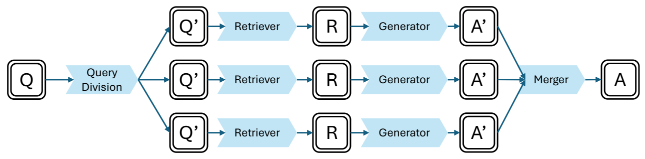


Figure 1: Generation workflow underlying our Run 1. It first divides the original query Q into subqueries Q', then merges the RAG answers A' for every subquery into the final answer A.

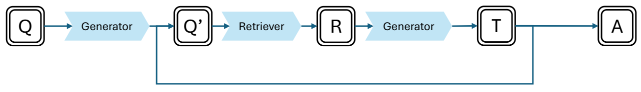


Figure 2: Generation workflow underlying our Run 2. It iteratively raises subqueries Q' and interacts with the retrieval model via LLM's reasonings T until the LLM decides it has enough information to yield the final answer A for the original query Q.

Our two submitted runs instantiate these two workflows, respectively. Both runs share the same retrieval pipeline described in Section 2.

Run 1: *genSubQ_merge*. Figure 1 illustrates the generation workflow used in Run 1. In this approach, we first construct a query plan by decomposing the original descriptive query into a set of shorter, more atomic sub-queries. For each sub-query, we apply a standard single-hop RAG pipeline to generate an answer. Because the target query typically contains multiple information needs, the answers to these sub-queries can be viewed as partial answers to the original query. We then use an LLM to combine these partial answers into a final response; in practice, this step can be regarded as a query-focused summarisation of the partial answers with respect to the original query. This framework is naturally multi-agent, as different LLMs may be used for query decomposition, sub-query answering, and final answer merging.

In our implementation, we use `gpt-4o-mini` to decompose each input query into sub-queries, with the prompt shown on the left side of Figure 3. We employ this model at the planning stage to leverage its stronger reasoning ability and richer parametric knowledge, aiming to produce higher-quality query decompositions before answer generation begins. To answer each sub-query, we adopt the single-hop RAG framework used in [8], using the top 3 documents returned by our retrieval pipeline as context along with a quantised Llama 3 8B model[2]. After answers generated for all sub-queries, we use the same Llama model to merge them into a final response with the prompt shown on the right side of Figure 3.

Run 2: *e5_monot5_searchR1*. Figure 2 illustrates the second generation workflow used in Run 2. Given the original query, an agentic generation model determines what additional information is needed and formulates a corresponding sub-query. The top-3 retrieved documents are then incorporated as context, enabling the model to decide whether further retrieval is necessary or whether sufficient information has been gathered to produce the final answer. In this way, the model alternates between reasoning and retrieval until it reaches the final answer. Compared with the first workflow, this approach relies more heavily on a single trained agentic model, which jointly performs sub-query generation, retrieval interaction, and answer generation.

In our implementation, we use Search-R1 [3], an agentic RAG model built on Qwen 2.5 7B [10], which is trained to interact with retrievers by issuing search queries during the reasoning process until it produces a final answer. The original Search-R1 is designed to generate short answers for datasets such as Natural Questions (NQ) [4] and HotpotQA [11], where queries typically express relatively simple information needs that can be satisfied with concise responses (typically a phrase) grounded in a small number of retrieved documents. In contrast, the TREC RAG 2025 queries in our setting are paragraph-length descriptions that often involve multiple sub-aspects and therefore require longer, more comprehensive answers. To adapt Search-R1 to this setting, we modify the prompting strategy to encourage paragraph-length answer generation. We use the Search-R1 implementation in PyTerrier-RAG [5] and provide the top 3 retrieved documents as context at each retrieval iteration. Facilitated by PyTerrier-RAG, agentic RAG pipelines can be implemented with relatively little engineering effort. An example is shown below.

```python
import pyterrier as pt
from pyterrier_rag import search_r1
from pyterrier_dr import E5
import pyterrier_dr
from pyterrier_t5 import MonoT5ReRanker

# loading the sparse index
sparse_index = pt.terrier.TerrierIndex(sparse_index_path)

# loading the dense index
e5_index = pyterrier_dr.FlexIndex(e5_index_path)

# loading the re-ranker
monoT5 = pyterrier_t5.MonoT5()

# construct the retrieval pipeline
retrieval_pipeline = E5() >> (e5_index.torch_retriever(fp16=True,
      num_results=20) >> sparse_index.text_loader(["text",
      "title"])
retrieval_pipeline = (retrieval_pipeline % 20) >> monoT5

# construct the retrieval pipeline and use it
r1_pipeline = pyterrier_rag.SearchR1(retrieval_pipeline % 3)
r1_pipeline.search(query_text)
```

4 RESULTS AND ANALYSIS

The organisers of TREC RAG released assessment results for 22 of the 105 target queries, using sub-narrative coverage[4] and strict

[2]We use an 8-bit quantised instruction-tuned variant of Llama 3 8B[3] [2].

[4]Sub-narrative coverage measures how well a response covers the decomposed subparts of a narrative query.

Prompt for Breaking a Query into Sub-Queries

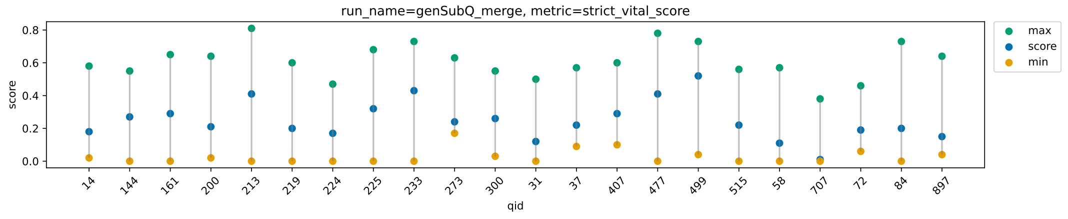
You are an expert at breaking down broad research questions into smaller, specific sub-queries. Given the following information need, produce a JSON array of sub-queries. Each sub-query must have these fields:
- id: sequential number
- query: the sub-query as a string
- topic: a short category describing the sub-query

Information need: {*the description*}

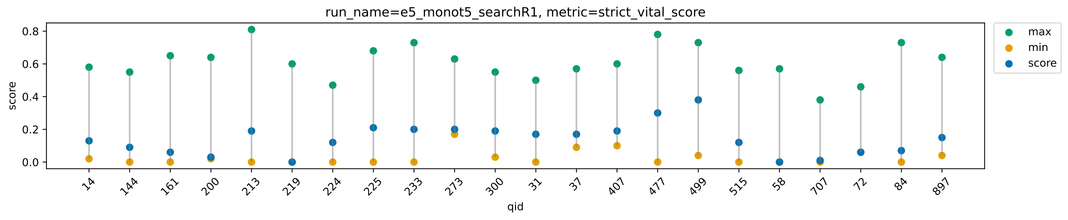
Prompt for Merging Answers

You are a helpful assistant. You will be given a description about the topic I want to search for, along with the answers for the sub-questions related to the topic.
A good answer should synthesise and refine the answers for the sub-questions and eventually answer the topic in the description.
Please write a paragraph to answer the question within <answer> and </answer>.
Description: {*the description*}.

Figure 3: The prompts used in Run-1. The left one is for dividing a description-style query into multiple sub-queries. The right one is for merging the partial answers into the final answer.



(a) The strict vital score of Run 1: genSubQ_merge



(b) The strict vital score of Run 2: e5_monot5_searchR1.

Figure 4: Comparison of strict_vital_score results for the two runs. The blue points mark the scores achieved in the corresponding run. The green and yellow points mark the maximum and minimum scores in all the submitted runs.

	Automatic		Post-Edited	
	SVS	SC	SVS	SC
genSubQ_merge	0.3071	0.5590	0.2465	0.5178
e5_monot5_searchR1	0.1510	0.3562	0.1383	0.3596
Track Median	0.3629	0.6805	0.3535	0.6770

Table 1: Average performance of Run 1 and Run 2, along with the median over all submitted runs. We report Strict Vital Score (SVS) and Sub-narrative Coverage (SC) under two nugget settings: *Automatic*, based on nuggets generated by the automatic nuggetizer, and *Post-Edited*, based on automatically generated nuggets refined by human assessors.

vital score[5] computed against automatically generated and post-edited nuggets for each query. Table 1 shows a performance summary of our submitted runs and the track median. Overall, both of our submitted runs perform below the track median. However, comparing our two submissions in Figure 4, Run 1 consistently outperforms Run 2.

To better understand this difference, we compare several characteristics of the two systems. Run 1, which follows the decomposing-and-generation strategy, issues 9.71 searches per query on average, whereas Run 2, based on iterative generation with Search-R1, issues only 3.82 searches per query on average. This suggests that Run 2 searches fewer sub-questions and, therefore, is more likely to miss important aspects of the target query. This observation is consistent with the complex, multi-aspect nature of TREC RAG

[5]Strict vital score measures the proportion of vital nuggets that are fully supported by a system answer, without awarding partial credit.

2025 queries, where broad coverage is often necessary to achieve high nugget recall.

Run 1 also tends to generate longer answers than Run 2, which is consistent with the design of the two systems. In Run 1, the original query is explicitly decomposed into multiple sub-queries, and the resulting partial answers are merged into a final response. This makes the final answer more likely to cover several aspects of the original information need. In contrast, Run 2 generates its response within a single iterative reasoning process and, because it issues fewer searches on average, often covers fewer aspects of the target query. Overall, this comparison suggests that explicit decomposition improves coverage but may produce less tightly integrated answers, whereas iterative generation can miss important aspects due to its reliance on the agentic model's ability to decide what to retrieve. These results indicate that adapting existing RAG frameworks provides a useful baseline for complex descriptive queries, but that answer quality remains strongly dependent on effective query decomposition and planning.

5 CONCLUSIONS

Our participation in the TREC RAG 2025 Track shows that existing RAG frameworks can provide a useful starting point for handling queries with complex information needs, a setting that is becoming increasingly important with the emergence of LLM-based applications. However, our results also highlight clear limitations in directly adapting these frameworks without modification. For agentic RAG models, a key challenge is over-reliance on the model's own ability to decide which sub-queries to issue, leading to limited control over query planning and potentially incomplete coverage of the original information need. In contrast, the decomposition-and-generation workflow achieves higher coverage, but may introduce less relevant sub-queries and produce answers that are less tightly integrated.

These findings suggest that more effective control over query planning can be a potential direction for future work. In particular, adding a dedicated query planning module may help existing RAG frameworks better balance coverage, relevance, and coherence when applied to complex descriptive queries across different application scenarios.

REFERENCES

[1] Nick Craswell, Bhaskar Mitra, Emine Yilmaz, Daniel Campos, and Jimmy Lin. 2025. Overview of the TREC 2021 deep learning track. arXiv:2507.08191 [cs.IR] https://arxiv.org/abs/2507.08191

[2] Abhimanyu Dubey et al. 2024. The Llama 3 Herd of Models. arXiv:2407.21783 [cs.AI] https://arxiv.org/abs/2407.21783

[3] Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. 2025. Search-R1: Training LLMs to Reason and Leverage Search Engines with Reinforcement Learning. arXiv:2503.09516 [cs.CL] https://arxiv.org/abs/2503.09516

[4] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. 2019. Natural Questions: A Benchmark for Question Answering Research. *Transactions of the Association for Computational Linguistics* 7 (2019), 452–466. https://aclanthology.org/Q19-1026/

[5] Craig Macdonald, Jinyuan Fang, Andrew Parry, and Zaiqiao Meng. 2025. Constructing and Evaluating Declarative RAG Pipelines in PyTerrier. In *Proceedings of SIGIR 2025*.

[6] Craig Macdonald, Nicola Tonellotto, Sean MacAvaney, and Iadh Ounis. 2021. PyTerrier: Declarative Experimentation in Python from BM25 to Dense Retrieval. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management* (Virtual Event, Queensland, Australia) *(CIKM '21)*. Association for Computing Machinery, New York, NY, USA, 4526–4533. https://doi.org/10.1145/3459637.3482013

[7] Rodrigo Frassetto Nogueira, Zhiying Jiang, Ronak Pradeep, and Jimmy Lin. 2020. Document Ranking with a Pretrained Sequence-to-Sequence Model. In *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020 (Findings of ACL, Vol. EMNLP 2020)*. Association for Computational Linguistics, 708–718. https://doi.org/10.18653/V1/2020.FINDINGS-EMNLP.63

[8] Fangzheng Tian, Debasis Ganguly, and Craig Macdonald. 2025. Is Relevance Propagated from Retriever to Generator in RAG?. In *Advances in Information Retrieval: 47th European Conference on Information Retrieval, ECIR 2025, Lucca, Italy* (Lucca, Italy). Springer-Verlag, Berlin, Heidelberg, 18 pages.

[9] Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. 2022. Text embeddings by weakly-supervised contrastive pre-training. *arXiv preprint arXiv:2212.03533* (2022).

[10] An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. 2025. Qwen2.5 Technical Report. arXiv:2412.15115 [cs.CL] https://arxiv.org/abs/2412.15115

[11] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Brussels, Belgium, 2369–2380. https://aclanthology.org/D18-1259/