SoftBank-Meisei at TREC RAG 2024

Aiswariya Manoj Kumar[1], Hiroki Takushima[1], Yuma Suzuki[1], Hayato Tanoue[1], Hiroki Nishihara[1], Yuki Shibata[1], Haruki Sato[2], Takumi Takada[3], Takayuki Hori[1], and Kazuya Ueki[4]

[1] AI Strategy Office, AI&Data Technology Planning Division, Technology planning department, SoftBank Corporation 1-7-1 Kaigan, Minato-ku, Tokyo 105–7529, Japan

[2] Agoop Corporation, 3-35-8 Jingumae, Shibuya, Tokyo 150-0001, Japan

[3] SB Intuitions Corporation, 1-7-1 Kaigan, Minato-ku, Tokyo 105–7529, Japan

[4] Department of Information Science, Meisei University, Room 27-1809, Hodokubo 2-1-1, Hino, Tokyo 191–8506, Japan

`aiswariya.manojkumar@g.softbank.co.jp`

Abstract. The SoftBank-Meisei team participated in the Retrieval (R), Augmented Generation (AG), and Retrieval Augmented Generation (RAG) tasks at TREC RAG 2024. In the retrieval task, we employed the hierarchical retrieval process of combining the sparse and dense retrieval methods. We submitted two runs for the task; one with the baseline implementation with additional preprocessing on the topic list and the other with the hierarchical retrieval results. In the Augmented Generation task, we used the GPT-4o API, as well as the LLama3-70b model along with our custom prompt for the generation. As for the Retrieval Augmented Generation task, we submitted two runs same as the R-task. The prompt used for the AG-task was used for the generation stage of the RAG-task too.

1 Retrieval Task

1.1 System Overview

Our pipeline consists of three stages including topics list pre-processing stage, retrieval stage and finally re-ranking stage. We implemented two pipelines where the differences between the two lies in the retrieval stage. For the first run, we used the baseline method BM25[1] indexing for retrieval, and a hybrid approach of combining sparse as well as dense retrieval for the second run.

Topics pre-processing : The pre-processing stage consists of correcting the grammatical/spelling errors in topics by prompting GPT-4o. It was noticed that the automated pre-processing by GPT-4o not only corrected the grammar/spelling errors but also did text completions for some incomplete topics. Once the automated pre-processing was done, we did a manual check to correct any improper additional texts. This mostly involved deleting the prefix to some of the pre-processsing results which contained the original topic. This check could also have been automated with a second stage processing, but we skipped it this time since the topics list was small enough to do a quick manual check.

Retrieval :
Sparse Retrieval: We employed the baseline method for sparse retrieval, which is using the BM25 algorithm for ranking and Lucene[2] inverted data structure to store the index. Top-100 results are returned as output.
Dense Retrieval: We first created flat dense embeddings of the msmarco-v2.1-doc-segmented corpus[3]. Search index was created using the Faiss library from Pyserini[5] for each document in the corpus. Hence 59 vector search indexes were created in total corresponding to the doc id.

For the first run, we employed sparse retrieval method using the msmarco-v2.1-doc-segmented prebuilt BM25 index from the Pyserini. First step retrieval results was obtained using the LueceneSearchlibrary from the pyserini toolkit.

For the second run, we first used sparse retrieval retrieval results to list up the doc ids that contains relevant information. Then the dense retrieval indexes of those filtered doc ids were used for the second stage retrieval. The resulting top-100 candidates from the dense retrieval results of each doc id candidates of the sparse retrieval results were combined together to have (sparse_retreival_candidates_count x 100) number of candidates. The combined list were then sorted in descending order of the retrieval score to get the top-100 retrieval results. This run will be referred to as hierarchical retrieval from here on since the retrieval was done from document to segment level.

Reranking : We used RankZephyr[4] from RankLLM for reranking the top-100 results from the retrieval stage.

2 Augmented Generation Task

The retrieval candidates list given as input was used along with the pre-processed topics list to generate answers using custom prompt.

2.1 System Overview

Azure OpenAI GPT-4o instance setup with content filtering was used for the generation process. Meta's LLama3-70b[5] model was used for the generation of those topics which got caught in content filtering.

3 Retrieval Augmented Generation Task

3.1 System Overview

We submitted two runs for the task; one with the sparse retrieval/generation results as described in R-task/Sparse retrieval and AG-task sections. And the second run with the hierarchical retrieval/generation results as described in R-task/Dense retrieval and AG-task sections.

[5] `https://github.com/castorini/pyserini`

4 Post-processing

As for the final processing step, we used the validator script provided by the organizers to drop sentences at the end of the generated response, so as to limit the total number of tokens as specified by the competition rules. The validator script also pointed out warnings of citations out of index 20.

5 Evaluation

The TREC RAG[6] evaluation process combines automatic and manual assessments, focusing on constructing information nuggets, which serve as "ground truth" for evaluating system answers.

For relevance assessment, "Auto Qrels" are initially constructed using relevance estimators with UMBRELA[6]. These auto-assessed Qrels are then post-edited by human annotators to create "Post-Edited Qrels," which serve as ground truth for the Retrieval task evaluation[7].

Evaluation criteria for the AG and RAG generation tasks include three metrics: Support, Fluency, and Nugget Assignment[8]. Support assesses how well each sentence in the answer is supported by its relevant cited segments. Fluency is assessed using LLMs to score the overall fluency and coherence of the generated answers. Finally, nugget assignment evaluates how well the generated answer captures the required and necessary information pertaining to the question.

Before submitting results, we attempted an automated LLM-based evaluation, comparing our output with the shared AG/RAG task baseline to determine which better answered the question. We used GPT-3.5-turbo[9] with a custom evaluation prompt and tabulated results for a few queries. However, because this evaluation method does not necessarily align with the actual evaluation guidelines, we did not pursue it extensively.

We manually reviewed several generated responses to verify they addressed the corresponding queries. Due to time constraints, we could not review the entire topic list or the correctness of citations. We plan to address citation correctness in future work.

6 Submissions and Results

We submitted a total of 5 runs for the three tasks:

R-task: Sparse retrieval run, Hierarchical retrieval run
AG-task: GPT-4o+LLama3-70b generation run
RAG task: Sparse retrieval run, Hierarchical retrieval run

According to the initial Nugget assessment results shared in[8], our submission secured the second-highest score for the primary metric among the AG task runs. This assessment is based on the fully automated evaluation of 301 topics. A separate tabulated result of 21 manually evaluated topics from NIST was also provided. These results also show that our AG task run secured second rank in terms of the primary metric. However, our retrieval-based runs did not achieve high evaluation scores, indicating the

[6] `https://trec-rag.github.io/`

need for further improvement.

In the future, we plan to conduct further analyses to understand the conditions under which the accuracy improves and refine our approach accordingly.

References

1. Robertson, S., Zaragoza, H., & Taylor, M. (2004, November). Simple BM25 extension to multiple weighted fields. In Proceedings of the thirteenth ACM international conference on Information and knowledge management (pp. 42-49).
2. Xian, J., Teofili, T., Pradeep, R., & Lin, J. (2024, March). Vector search with OpenAI embeddings: Lucene is all you need. In Proceedings of the 17th ACM International Conference on Web Search and Data Mining (pp. 1090-1093).
3. Ji, Z., Lee, N., Frieske, R., Yu, T., Su, D., Xu, Y., Ishii, E., Bang, Y.J., Madotto, A. and Fung, P., 2023. Survey of hallucination in natural language generation. ACM Computing Surveys, 55(12), pp.1-38.
4. Pradeep, R., Sharifymoghaddam, S., & Lin, J. (2023). RankZephyr: Effective and Robust Zero-Shot Listwise Reranking is a Breeze!. arXiv preprint arXiv:2312.02724.
5. Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F. and Rodriguez, A., 2023. Llama: Open and efficient foundation language models. arXiv preprint arXiv:2302.13971.
6. Upadhyay, S., Pradeep, R., Thakur, N., Craswell, N., & Lin, J. (2024). UMBRELA: UMbrela is the (Open-Source Reproduction of the) Bing RELevance Assessor. arXiv preprint arXiv:2406.06519.
7. Upadhyay, S., Pradeep, R., Thakur, N., Campos, D., Craswell, N., Soboroff, I., Dang, H.T & Lin, J. (2024). A Large-Scale Study of Relevance Assessments with Large Language Models: An Initial Look. arXiv preprint arXiv:2411.08275.
8. Pradeep, R., Thakur, N., Upadhyay, S., Campos, D., Craswell, N., & Lin, J. (2024). Initial nugget evaluation results for the trec 2024 rag track with the autonuggetizer framework. arXiv preprint arXiv:2411.09607.
9. OpenAI. (2024). OpenAI GPT-3 API [gpt-3.5-turbo]. Available at: https://platform.openai.com/docs/models#gpt-3-5-turbo