SemEval 2026-Task 8: MTRAGEval - Evaluating Multi-Turn RAG Conversations

Mohit Gupta* **Rayaan Lodhi*** **Yifan Tian***
University of Colorado Boulder
{Mohit.Gupta-1, Rayaan.Lodhi, Yifan.Tian}@colorado.edu

Abstract

This paper presents a unified approach to SemEval-2026 Task 8 (MTRAGEval), which evaluates multi-turn Retrieval-Augmented Generation (RAG) systems through three subtasks: passage retrieval (A), generation with reference passages (B), and full retrieval-augmented generation (C). We implemented retrieval and generation pipelines using local large language models and vector databases, also evaluate our pipelines using official benchmarks provided by the benchmark paper. We included Results section where we analyze the performance of our approaches Our implementation and results provide practical insights for building efficient, evidence-grounded RAG systems under real-world hardware limitations.

1 Introduction

More recent publications on Retrieval-Augmented Generation systems have highlighted the need to have solid evaluation with respect to how models retrieve, select and ground external evidence-especially in multi-turn conversations whose context varies with time. The MTRAG benchmark (Katsis et al., 2025), implemented through the MTRAGEval framework, is divided into three subtasks: **Subtask A: Retrieval Only**, which evaluates a system's ability to retrieve and rank relevant passages; **Subtask B: Generation with Reference Passages**, which measures grounded answer generation using gold passages; and **Subtask C: RAG (Gao et al., 2024) (Retrieval + Generation)**, which requires systems to both retrieve evidence and generate faithful responses.

In this paper, the three subtasks are explored with reference to a shared retrieval and generation pipeline. In the case of retrievals, we benchmark dense model MiniLM (Wang et al., 2020) and Snowflake-Arctic-Embed and improved them with Maximum Marginal Relevance and re-ranker. We employed Qwen2-1.5B-Instruct equipped with answerability check and long-term memory as generation model, which will be elaborated in following sections.

2 Dataset

We build our pipelines on the MTRAG benchmark, a recently released multi-turn retrieval-augmented generation (RAG) dataset designed for end-to-end evaluation of conversational, document-grounded question answering.[1] MTRAG consists of 110 human-authored multi-turn conversations, covering four domains (Wikipedia, cloud documentation, finance, and government). Each conversation is linked to a domain-specific corpus and exhibits realistic RAG phenomena such as follow-up questions, clarification, partial evidence, and unanswerable queries. The raw conversations average 7.7 turns and are converted into 842 turn-level generation tasks.

The benchmark includes four corresponding corpora: ClapNQ (Wikipedia), Cloud (technical documentation), FiQA (finance), and Govt (government). Each corpus is pre-segmented into passages in a BEIR-style layout.

In this preliminary report, we focus on ClapNQ due to time and hardware constraints.

3 Method

The system we use has a multi-turn retrieval augmented generation (RAG) pipeline adhering to the MTRAG protocol. The pipeline is divided into two large parts, namely Retrieval and Generation. Subtask A is associated with the retrieval module; Subtask B will involve generation with the gold reference passages that are presented by the dataset

*These authors contributed equally to this work.

[1] Our code is published at github: https://github.com/IamMohitGupta/NLP-Project-SemEval-Task8.

Corpus	Domain	#Documents	#Passages
ClapNQ	Wikipedia	4,293	183,408
Cloud	Technical Docs	57,638	61,022
FiQA	Finance	7,661	49,607
Govt	Government	8,578	72,422

Table 1: Document corpora released with the MTRAG benchmark.

and Subtask C will involve generation with references made by our integrated retrieval module. To be effectively developed in parallel, the three subtasks were applied parallel to each other. Outputs are in the form of the SemEval Task 8 evaluation scripts, in JSON format.

4 Retrieval

The first important step in any RAG system is typically retrieval where the most relevant supportive evidence must be located in a large corpus prior to the actual answer being generated. Effective retrieval, given the context of multi-turn question answering, must not only be able to handle changing conversational context, long-range dependencies, and a variety of knowledge sources, but be computationally efficient.

The existing RAG pipelines apply to dense retrieval: during indexing, queries and documents are instantiated in a common semantic space and searched with ANN search in order to retrieve top-k candidates. The workflow may be backed by a variety of backends, such as Milvus which is high throughput vector search, pgvector which is database integrated retrieval, Snowflake Arctic with vector extensions or lightweight local such as FAISS (Johnson et al., 2017).

Some of the retrieval configurations we used in this work on SemEval Task 8 (2026) are: (i) a Milvus-based system that can be used to search the large-scale ANN, (ii) a pgvector+Snowflake system that can be used to filter the embeddings in a SQL-native fashion, and lastly, (iii) local retrieval variants that can be used to experiment and ablate. Next we concentrate on describing our key implementation in detail.

5 Subtask A: Retrieval

Subtask A checks the capability of a system to access the most relevant passages with regard to a particular conversational query. On the benchmark we operate two datasets; namely, the **ClapNQ** and **Cloud** datasets which have different document distributions and grounding properties. All data sets are loaded based on their JSONL source files with each record having a distinct identifier and raw text. We do not change the text at all, as the embedding models we have used (e.g., MiniLM, Snowflake-Arctic-Embed, Contriever) have their own built-in tokenization, and do not need any linguistic preprocessing.

In all retrieval methods, queries and passages are coded into dense semantic vectors with a set of pretrained encoders. MiniLM-L6-v2 (Wang et al., 2020) to ClapNQ because it is efficient and can be run on CPU-only hardware. Cloud dataset is done on Snowflake-Arctic-Embed. After getting embeddings, all retrieval backends - Milvus, pgvector or FAISS (Johnson et al., 2017) use Approximate Nearest Neighbor (ANN) search to find top-k candidate passages. Retrieved results are eventually translated to the JSONL format required by SemEval in which each query is matched with a list of document identifiers and similarity scores ranked. This brings together the evaluation channel in all the three retrieval architectures.

5.1 Approach 1: Milvus ANN Retrieval

The former discusses a Milvus-based retrieval system that was applied to Subtask A. This architecture is depicted in Figure 1 as Milvus-rag-architecture, and it is based on a high-throughput text embedding database, and has the ability to construct ANN indices and scale similarity queries. The system has four Milvus-specific phases namely: (1) creation of the collection schema, (2) batched addition of vectors, (3) ANN index building and (4) search at query time.

5.1.1 Milvus Collection Schema and Construction

We create a Milvus collection named `clapnq_passages` with a schema that stores each passage ID, its dense embedding, and the raw text:

- **id**: VARCHAR primary key

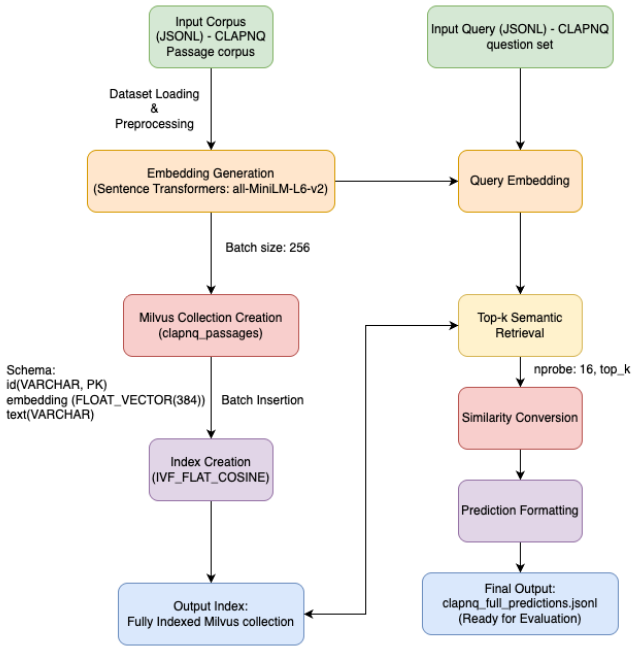

Figure 1: Overall architecture of our Milvus based Retrieval-Augmented Generation (RAG) system

- **embedding**: FLOAT_VECTOR(384)
- **text**: VARCHAR

Any existing collection with the same name is dropped to ensure a clean and consistent execution environment. The new collection is then instantiated using the PyMilvus API, which manages schema validation and memory allocation on the Milvus server.

5.1.2 Batch Insertion of Embeddings into Milvus

We then insert the vector data into Milvus after schema creation. So as to avoid memory bottlenecks and ensure transactional consistency, we insert embeddings in batches that reflect how the embeddings were batched during embedding creation. Each insertion call provides Milvus with a structured list:

`[passage_idsBatch, embeddingsBatch, passage_textsBatch]`

Milvus has automatic storage segment assignment and internal optimization to prepare the vectors to be indexed and queried. Because each of the vectors is bound to a distinct passage ID, the results obtained can still be understood and traced throughout the assessment.

5.1.3 Index Construction: IVF_FLAT with COSINE Similarity

Once all the embeddings are inserted we construct an ANN index, with IVF_FLAT, cosine similarity, and `nlist = 2048`. IVF_FLAT breaks the embedding space into several clusters and stores raw vectors without any compression, hence, providing a reasonable compromise between the retrieval speed and the accuracy. The reason why cosine similarity is used here is that it fits well with MiniLM embeddings. To be able to query the index, the index is loaded into memory with the command `collection.load()`

5.1.4 Query Embedding and Retrieval

We load the set of questions of ClapNQ in the evaluation directory each with a query text and a special task identifier. Each query text is encoded with the same MiniLM encoder to make the query and passage vectors compatible.

For each query embedding, we conduct a top-k similarity search:

```
collection.search(
    data=query_embedding,
    anns_field="embedding",
    param={"metric_type": "COSINE",
           "params": {"nprobe": 16}},
    limit=top_k,
    output_fields=["text"]
)
```

The parameter `nprobe=16` controls how many IVF clusters are probed during search. Higher values increase recall at the cost of latency, while lower values speed up search but risk missing relevant neighbors. Our implementation selects a moderate in order to balance accuracy and efficiency.

Cosine returns distances in the range $[0, 2]$ in Milvus. We convert these to similarity scores using:

$$\text{similarity}(d) = 1 - d,$$

which was in agreement with the expectation for high scorers to indicate stronger relevance.

Each of the retrieved neighbours is represented as a pair $(\text{id}, \text{similarity})$ and appended to the ranked context list for the query.

5.2 Approach 2: pgvector + Snowflake-Arctic Retrieval with MMR

Another vector space-based approach uses `pgvector` and `langchain_postgres` for the storage and retrieval of dense embeddings within a PostgreSQL database. This is coupled with a hefty embedding model, *Snowflake-Arctic-Embed-L-v2.0*, for semantic representation. The system allows for native SQL filtering, distributed index management, and complex strategies in retrieval, such as MMR scoring.

5.2.1 Architecture Overview

The three key stages in the pgvector system architecture are:

1. **Embedding Generation & Storage**: All passages from MTRAG are encoded with the Snowflake-Arctic-Embed-L-v2.0 model into 1024 dimensional embeddings. The embeddings and metadata of the passages (doc ID, Title, and URL) are stored in a PostgreSQL table supporting pgvector.

2. **Index Management**: pgvector creates an Approximate Nearest Neighbor index (usually IVF or HNSW) upon the embedding column for accelerating similarity search. The index is automatically maintained by PostgreSQL.

3. **Query-Time Retrieval with MMR**: At the time of retrieval, queries coming in are embedded using the same Snowflake-Arctic model. Instead of simple cosine similarity, relevance and diversity in the result set being retrieved are balanced with the use of Maximum Marginal Relevance (MMR).

5.2.2 Snowflake-Arctic-Embed-L-v2.0 Embeddings

The Snowflake-Arctic-Embed-L-v2.0 model is a large-scale, instruction-tuned dense embedding model designed for high-quality retrieval. Key characteristics include:

- **Instruction Tuning**: The model is fine-tuned to respond to retrieval instructions, making it particularly effective for conversational and domain-specific queries.

- **Multi-lingual Support**: Due to its large dimension size, it supports multi-lingual embeddings and was primarily trained on corpora of different languages.

- **Computational Cost**: More computation is needed to generate the embedding, but the quality of the retrieval is improved downstream. This is especially useful in multi-turn conversational settings where accuracy is crucial.

5.2.3 Maximum Marginal Relevance (MMR) Retrieval

By penalising redundancy among the top-k results, MMR strikes a balance between relevance and diversity, in contrast to standard cosine similarity retrieval, which selects the top-k results based only on cosine relevance. With a query q and a collection of potential passages D, MMR chooses the document d that maximises:

$$\text{MMR}(d) = \lambda \cdot \text{sim}(q, d) - (1 - \lambda) \cdot \max_{d' \in R} \text{sim}(d, d'),$$

where:

- $\text{sim}(q, d)$ is the cosine similarity between the query and document embeddings.

- $\max_{d' \in R} \text{sim}(d, d')$ is the maximum similarity between the candidate and any already-selected document d' in the result set R.

- $\lambda \in [0, 1]$ is the diversity parameter (we use $\lambda = 0.7$ to prioritize relevance while maintaining diversity).

The algorithm operates in an iterative way: we retrieve an initial candidate set of size $K \cdot f$ (e.g., $100 \times 2 = 200$ candidates) using cosine similarity from pgvector to ensure high recall. We then

apply MMR over these candidates to reorder them with the goal of maximizing the above objective and select the top-K documents ($K = 100$ for evaluation).

5.2.4 Advantages of pgvector + MMR

This approach has a number of advantages:

1. **Diversity**: MMR intrinsically lowers redundancy within the result set, something especially useful for multi-turn conversational scenarios when the user might well ask follow-up questions about different subtopics.

2. **Scalability**: PostgreSQL and pgvector can efficiently manage millions of embeddings and support distributed indices.

3. **SQL Integration**: Standard SQL WHERE clauses can be used either before or after semantic search to filter passages based on metadata (such as domain or time range).

4. **Embedding Quality**: Snowflake-Arctic embeddings are larger and more expressive than the ones produced by MiniLM, yielding superior semantic matching on domain-specific corpora.

5.3 Approach 3: MiniLM + Bge-reranker

This approach works as the retrieval implementation in the full RAG pipeline.

5.3.1 First-Stage Dense Retrieval

In the first stage, we employ MiniLM to embed queries and passages into a shared vector space. ANN search is performed using FAISS (Wang et al., 2021) to efficiently retrieve a small set of candidate passages. This stage mainly focuses on retrieval efficiency, serving as a filtering step that reduces the re-rank (Xiao et al., 2024) workload and maintaining low latency on large document collections.

5.3.2 Second-Stage Cross-Encoder Reranking

To improve ranking precision beyond dense retrieval, we apply a second-stage reranking step using the `bge-reranker-v2-m3` cross-encoder. After the MiniLM+FAISS retrieval returns an initial pool of $K_{\text{search}} = K \times r$ candidate passages, we score each query–passage pair (q, d) using the cross-encoder, which performs full cross-attention over the concatenated input:

$$s(q, d) = f_{\text{cross}}([q; d]).$$

Unlike the bi-encoder used in the first stage, the cross-encoder jointly encodes both sequences, capturing token-level interactions that allow it to detect fine-grained semantic relevance.

All candidates are rescored using $s(q, d)$ and then sorted in descending order. The top-K reranked passages form the final retrieval set used for both Subtask A evaluation and for Subtask C generation. This two-stage (dense $\rightarrow$ cross-encoder) structure follows established RAG practice, where the first stage maximizes recall and the second maximizes precision.

6 Generation

6.1 Conversation Encoding

For every task corresponding to turn k in a dialogue, we concatenate all previous user–assistant turns and append the current user question. This step includes role tags like `[USER]` and `[ASSISTANT]`.

6.2 Answerability Prediction

A considerable portion of queries in MTRAG are unanswerable because the relevant information may not appear in the corpus. To deal with answerability issue, we introduce an answerability prediction step. The model examines the re-ranked (Xiao et al., 2024) passages and determines whether they provide sufficient evidence to answer the question, using a short classification-style prompt such as: `Decide if the provided context contains any evidence that directly helps answer the question. Only answer yes/no.` If the model predicts `no`, the system directly returns a response: *"I do not have enough information to answer this question."*

6.3 Grounded Answer Generation

If the query is answerable, we continue to final answer generation. We construct a RAG prompt that includes: (1) the multi-turn dialogue context, (2) the top-R passages labeled as `Document 1`, `Document 2`, etc., and (3) instructions requiring evidence-grounded responses and discouraging hallucination. Generation is performed using an instruction-tuned LLM (here we use `Qwen2-1.5B-Instruct`).

6.4 Multi-turn Memory Mechanism

To handle long conversations, we use a hybrid memory mechanism combining recent raw turns with a

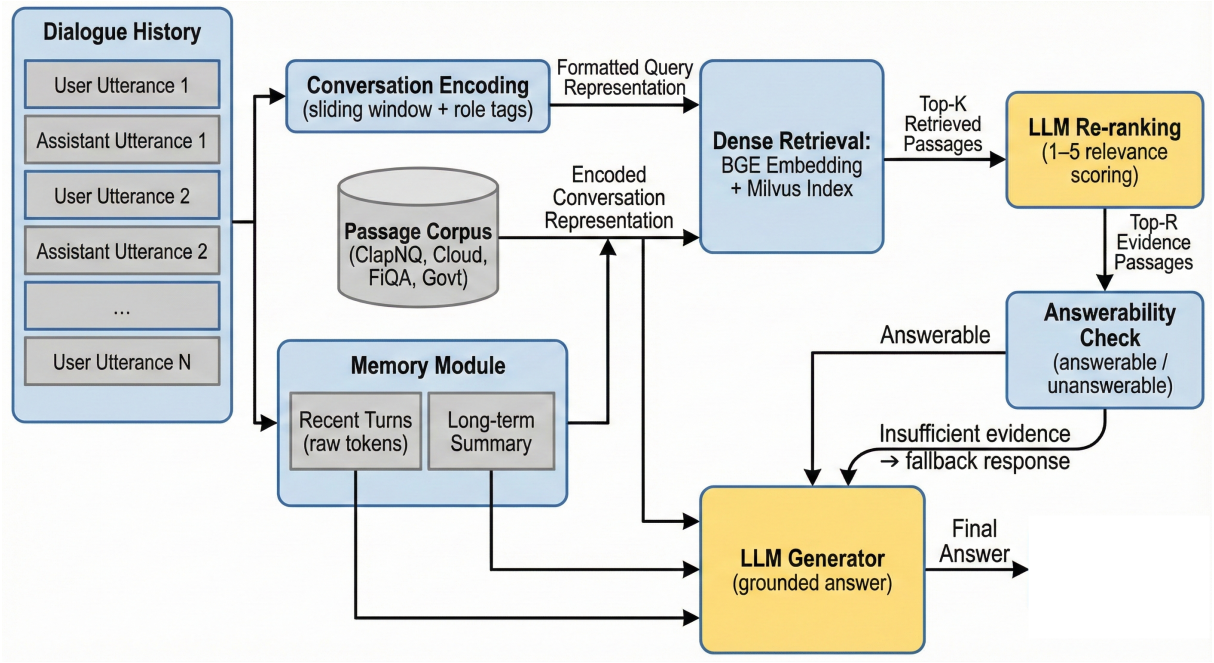

Figure 2: Overall architecture of our multi-turn Retrieval-Augmented Generation (RAG) system, including dense retrieval, LLM-based re-ranking, answerability prediction, and grounded generation with a hybrid memory mechanism.

long-term summary. The system maintains a memory buffer containing: (i) the last N dialogue turns (sliding window), and (ii) a concise summary of earlier content if out of context limit. The summary is periodically refreshed using a short instruction such as: `Summarize earlier turns.`

7 Evaluation

We evaluate our retrieval and generation output using the officially provided script:

```
python3 run_retrieval_eval.py \
    --input_file predictions.jsonl \
    --output_file eval_results.jsonl
```

The evaluator computes metrics such as Recall@k and aggregates performance across the full query set and generated predictions. It follows an automatic evaluation paradigm and provides standardized metrics for assessing answer correctness, grounding fidelity, and answerability detection. All system outputs are serialized as JSONL files in the exact format required by the benchmark's scoring script. By running the script, we programmatically ensures reproducibility and alignment with benchmark results used in the shared task.

7.1 Evaluation Protocol

We adopt the evaluation settings defined by the official benchmark:

- **Retrieval**: the results of retrieval
- **Reference-only Generation**: the generation results using human-chosen reference passages.
- **Full RAG**: the system retrieves passages and generates answers based on the retrieved evidence and multi-turn context.

For each setting, we run our system over specific corpus and produce a JSONL file containing the corresponding attributes. We then run the official scoring script provided in the benchmark repository.

The script produces a summary report after running all metrics.

7.2 Retrieval Metrics

nDCG@5 and Recall@K ($K = 1, 3, 5$), which capture complementary aspects of ranking quality and retrieval coverage, are used by the benchmark to assess retrieval performance.

nDCG@5 (Normalized Discounted Cumulative Gain at 5). By considering both the relevance of the retrieved documents and their ranking positions, nDCG@5 evaluates the quality of the ranked retrieval results. Because of logarithmic discounting, relevant documents that appear at higher ranks contribute more to the score. The ideal ranking is used to normalize the metric, producing values

between 0 and 1. Better ranking quality among the top retrieved passages is indicated by a higher nDCG@5.

Recall@K (R@1, R@3, R@5). Recall@K quantifies the percentage of queries that yield at least one pertinent document in the top-K results. In particular, R@3 and R@5 determine whether a relevant document appears in the top three and top five results, respectively, whereas R@1 determines whether the top-ranked document is relevant. Recall places more emphasis on retrieval coverage than on the precise order of outcomes.

7.3 Generation Metrics

RB_{alg} is a reference-based metric that compares the model response (*MR*) with the reference answer (*RA*) using several algorithmic similarity measures. It combines semantic recall, precision with respect to the retrieved passages *P*, and phrase-level overlap to approximate how complete, grounded, and appropriate the response is.

Recall. Recall measures the proportion of reference information that is successfully reflected in the generated response. It evaluates whether key facts present in the ground-truth references are covered by the model output, focusing on content completeness rather than surface-level similarity.

ROUGE-L. ROUGE-L computes the longest common subsequence between the generated response and the reference text. It captures sequence-level overlap and rewards responses that preserve the structural ordering of the reference content.

BERTScore Precision (BERT-P) and Recall (BERT-R). BERTScore evaluates semantic similarity using contextualized token embeddings. BERT-P measures the precision of semantic content in the generated response relative to the reference, while BERT-R measures recall, indicating how much reference content is semantically covered by the generation. Together, they provide a soft semantic matching signal beyond exact lexical overlap.

Length. Length reports the average number of tokens in the generated responses. This metric provides diagnostic insight into generation verbosity and helps contextualize other metrics, particularly in long-form answer settings.

7.4 Evaluation Workflow

Our evaluation process proceeds as follows:

1. Run the system on all MTRAG test tasks under the desired evaluation setting (Full-RAG, Reference-only, etc.).
2. Serialize predictions into JSONL in the exact format specified by the benchmark.
3. Execute the provided evaluation script to obtain automatic metrics.
4. Analyze scores across domains (Wikipedia, Cloud, Finance, Government), conversation depth (early vs. late turns), and answerability categories.

This ensures that results produced by our system are fully comparable to the baselines and reference systems reported in the benchmark.

8 Results

8.1 Retrieval

Table 2 reports retrieval performance on the ClapNQ and Cloud datasets using nDCG@5 and Recall@K. Overall, the results highlight substantial differences in retrieval effectiveness across models and demonstrate the importance of reranking for improving ranking quality.

The MiniLM-L6-v2 dense retriever's performance on the ClapNQ dataset is almost zero across all metrics. This demonstrates that the fine-grained relevance information that this dataset may contain is not significantly captured by coarse nearest-neighbor retrieval, which is based solely on lightweight bi-encoders.

Conversely, the snowflake-arctic-embed-l-v2.0 model performs well with the Cloud data with considerably high nDCG@5 and Recall@K. This observation indicates that bigger embedding models with higher representational capacity can be effective on data sets with no particularly ambiguous semantic matching, or where the retrieval geometries are well accommodated by the embedding geometry.

However, the robust BGE-reranker-v2-m3 model and the addition of MiniLM-L6-v2 to the benchmark's cross-encoder reranking stage resulted in appreciable gains in all ClapNQ evaluation metrics. Despite these improvements, the highest scores of 0.4862 nDCG@5 and 0.5255 Recall@5 were still below the performance of a number of baselines that only use bi-encoder retrieval. This disparity draws attention to the incremental benefits of cross-encoder reranking as well as the intrinsic drawbacks of mainly using embedding-similarity

Approach	Dataset	Dimension	Retrieval Metrics			
			nDCG@5	R@1	R@3	R@5
MiniLM-L6-v2	ClapNQ	384	0.00073	0.0	0.0	0.0012
Snowflake-Arctic-Embed-l-v2.0	Cloud	1024	0.31574	0.1484	0.28138	0.34911
MiniLM-L6-v2+bge-reranker-v2-m3	ClapNQ	384	0.4862	0.2012	0.4314	0.5255

Table 2: Retrieval performance on ClapNQ and Cloud datasets.

models to capture intricate semantic relationships between queries and passages.

Taken together, these findings confirm the theory that dense retrieval is trainable to achieve an effective candidate retrieval mechanism but that reranking by the cross-encoder is required to achieve high precision retrieval. Retrieve-then-rank design is much more superior in the quality of ranking, especially on the data sets that require more complex semantic reasoning.

8.2 Generation performance

Table 3 presents aggregated generation performance on MT-RAG Task B and Task C using multiple metrics. Overall, the results suggest clear tradeoffs between grounding quality, content coverage, and generation length under different reference settings.

For the Cloud dataset, we evaluate Qwen-0.5B under both reference-access and no-reference conditions. With reference access, the model achieves an RB_{agg} score of 0.2636, slightly higher than the 0.2612 obtained without references. This small gap may suggest that the retriever used yielded very similar passages compared to the human-annotated "gold passages". The reference-enabled setting yields higher Recall and ROUGE-L values, reflecting better coverage of the provided passages; however, these gains come with substantially longer outputs, indicating that the model often responds verbosely without proportionate improvement in factual grounding.

The Qwen-1.5B model tested on ClapNQ (no references) achieves the highest RB_{agg} score (0.3053). This result reflects stronger internal knowledge and improved reasoning capacity relative to the 0.5B model. Its shorter outputs may further limit hallucination opportunities, contributing to improved grounding. As expected, ROUGE-L and BERTScore remain low across all settings due to the difficulty of matching long-form target answers at the lexical or embedding level.

Across all settings, ROUGE-L and BERTScore values remain very low due to the inherent difficulty of exact lexical or semantic alignment in long-form generation tasks. These observations stress the importance of RB_{agg} as the primary metric of evaluation for Task B and Task C, as it more directly captures factual grounding and support from references than surface-level similarity measures.

Overall, the results show that access to references alone does not guarantee improved grounding quality, particularly for smaller models. Instead, model scale, verbosity, and content selection behavior have substantial influence on factual consistency, underscoring the importance of grounding-aware generation strategies in multi-turn RAG systems.

9 Future Work

There are several directions that follow naturally from our current findings and system limitations.

First, we will incorporate stronger reranking models. Our similarity-based retrieval often surfaces passages that are topically related but not answer-bearing; cross-encoder rerankers or

Model	Dataset	Reference	Generation Metrics					
			RB_{agg}	Recall	Rouge-L	BERT-P	BERT-R	Length
Qwen-0.5B	Cloud	✓	0.2636	**0.4290**	0.1584	-0.0920	0.1030	1958.23
Qwen-0.5B	Cloud		0.2612	**0.4078**	0.1383	-0.1178	0.0836	1895.33
Qwen-1.5B	ClapNQ		**0.3053**	0.3107	0.1896	0.0125	0.1152	762.5

Table 3: **Aggregated generation performance on MT-RAG Task B and C.** All metrics are averaged over the evaluation set. RB_{agg} is the primary metric, while the remaining metrics provide diagnostic insights.

lightweight LLM scoring may improve evidence quality.

Then, we would like to refine the multi-turn memory mechanism: the current buffer simply stores history and does not summarize or filter for relevance. More sophisticated memory mechanisms (e.g., hierarchical summaries or salience-based retrieval) may yield improved coherence and reduce drift in multi-turn generation.

This is followed by the evaluation of all four MTRAG domains and component-wise ablations, quantifying the contribution of retrieval, reranking, memory, and answerability detection. Such an analysis will help to establish which modules provide the highest value for future grounding and overall RAG robustness improvements.

References

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. 2024. Retrieval-augmented generation for large language models: A survey. *Preprint*, arXiv:2312.10997.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2017. Billion-scale similarity search with gpus. *Preprint*, arXiv:1702.08734.

Yannis Katsis, Sara Rosenthal, Kshitij Fadnis, Chulaka Gunasekara, Young-Suk Lee, Lucian Popa, Vraj Shah, Huaiyu Zhu, Danish Contractor, and Marina Danilevsky. 2025. Mtrag: A multi-turn conversational benchmark for evaluating retrieval-augmented generation systems. *Preprint*, arXiv:2501.03468.

Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, Kun Yu, Yuxing Yuan, Yinghao Zou, Jiquan Long, Yudong Cai, Zhenxiang Li, Zhifeng Zhang, Yihua Mo, Jun Gu, and 3 others. 2021. Milvus: A purpose-built vector data management system. In *Proceedings of the 2021 International Conference on Management of Data*, SIGMOD '21, page 2614–2627, New York, NY, USA. Association for Computing Machinery.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. 2020. Minilm: deep self-attention distillation for task-agnostic compression of pre-trained transformers. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA. Curran Associates Inc.

Shitao Xiao, Zheng Liu, Peitian Zhang, Niklas Muennighoff, Defu Lian, and Jian-Yun Nie. 2024. C-pack: Packed resources for general chinese embeddings. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '24, page 641–649, New York, NY, USA. Association for Computing Machinery.