

Technical Report

Executive Summary

In Assignment 2, I built a naive RAG pipeline and used it as a baseline. I explored prompts and key parameters, then moved to an advanced version and evaluated it with a stronger framework. The starter code gave me a clean path for data loading, embedding, storage, retrieval, and generation. I ran controlled tests on embedding dimensions and retrieval strategies. Results varied with settings, with some prompts performed better with 512-d embeddings and top-3 retrieval, while others worked better with 384-d or top-1. The key finding was about balance rather than a single best method.

I added grounded citations and context window optimization to the advanced RAG. While citations pushed the model to mark claims to sources, windowed chunks reduced information loss in long passages. EM and F1 rose significantly after these changes, while the RAGAs report showed that faithfulness increased, and context precision improved as well. On the other hand, context recall and answer relevancy went down. The system became more strict about evidence and less willing to pull wider context. This result shows the trade-off for advanced features. Tighter context and connection to answer succeeded in increasing accuracy, but also in reducing the coverage of the answer, thus affecting the information containment of the answer.

System Architecture

For Naive RAG, I developed my flow by adjusting an AI generated framework that could actually run, and I moved that flow into the starter code structure for a cleaner and more standard setup. I turned the passage column into a list of strings, so each passage matches one embedding. I used all-MiniLM-L6-v2 for embeddings as recommended and finished vectors for 3,200 texts. I tried FAISS for vector DB, but the schema design in the starter code felt more complete and helped me learn how schemas and indexes work. Therefore I switched to Milvus Lite.

For the LLM model, I first used the ChatGPT API as a starter. But the key ran into quota limits, so I decided not to rely on the OpenAI API and switched to Hugging Face's Flan-T5-small. It runs in Colab with a simple import and no extra setup or keys, which fits the project better.

I encountered several issues and fixed them by hand. A `KeyError: 0` came from misaligned DataFrame indexing. I printed counts and saw the mismatch, so I used `dropna()` and converted to a list before encoding. On top of that, Milvus raised "required argument is not a float" during search. I wrote an `as_float_list()` helper to force pure Python floats on both insert and query, and I checked the dimension is 384. At one point the collection doubled to 6,400 rows, caused by a duplicate insert. I then ran the `drop_collection` step and rebuilt the collection to keep it clean. Each fix helped me understand the flow and details of the pipeline.

Experimental Results

Through the experimentation phase I worked with two embedding dimensions, 384 and 512, and tested three retrieval strategies: top-1, top-3, and top-5, each with three prompt methods.

For the embedding dimension, my expectation was that 512 would have better performance than 384, since a larger dimension should preserve more semantic nuance and detail from both the queries and documents. The results partly confirmed this idea. With the instruction and persona prompts, 512 consistently gave a small advantage, usually about 0.1 higher in EM and F1 compared to 384 under the same settings. The CoT prompt, however, showed the opposite trend. In several cases, the lower dimensional embedding actually worked better. My interpretation is that for CoT prompts, the answers are already overly elaborate and complicated, therefore raising the dimension only amplified this complexity, producing outputs that drifted further from the references. For prompts that are more direct, such as introduction prompts, more complex dimensional space improved accuracy, but CoT appeared to suffer from the added semantic information.

The number of retrieved passages created a different pattern for different prompts, as each prompt seemed to favor a different k. Instruction prompts performed best with top-3, persona leaned toward top-1, and CoT achieved its best scores with top-5. To begin with, for CoT, top-5 remained optimal across both embedding dimensions. In my opinion, this is evidence that more passages allow CoT to have a better understanding of overall logic and entities, making its step by step reasoning stronger. Instruction behaved differently, with the smaller embedding dimension it preferred more passages, but with 512 dimensions it shifted toward fewer. This suggests a balance between semantic coverage and information load. When the space could not capture key meaning well enough from the queries and documents, larger k filled the gap, while higher-dimensional embeddings made additional passages less necessary. Persona, in comparison, tended to rely on fewer passages than instruction under the same conditions, perhaps because instruction prompts provide more explicit format for answers and thus benefits from richer supporting material.

Across all variations of parameters, instruction remained consistently better performance than both CoT and persona. This outcome shows that the way the prompt is written can shape performance more strongly than the fine-tuning of parameters.

Enhancement Analysis

The reasons I chose these features as enhancements are based on an initial prediction of how easy they would be to integrate with the naive RAG code. On top of that, I also wanted to select features from different stages of post-retrieve, pre-retrieve, and retrieve, instead of selecting the duplicate phase. Finally, I tried out some features that resulted poorly, so the two with the best results are the chosen ones.

In the selection of advanced features, I first choose grounded citations, and force LLM to output indexes in the design of prompt, which makes the answers more convincing. But in the process of implementation, I found that this enhancement has some limitations. This is because the model sometimes deviates from the format and sometimes outputs the index instead of the answer. My suggested reason is that the choice of the LLM models is open sourced, which leads to less output content, forcing the model to drop certain information in the answers. Another feature I added is context window optimization. I implemented sentence segmentation and sliding windows to cut the excessively long paragraph into small segments of 180 tokens to avoid losing key information due to truncation. The final number of embeddings went from 3200 to 11255 with higher retrieval coverage, but I also observed an increase in memory. Based on these, I experimented with query rewriting as well, finding that it increased the output time drastically and also caused a large drop in EM and F1. The reason could be because the expression of the rephrased question is more deviated from the reference, the model answer is rephrased accordingly, making less word overlap with the reference answer, resulting in a lower score.

From the results, first of all, RAGAs results show two enhancements and two indicators are weakened. Faithfulness (+50%) and context precision (+55%) are significant improvements. There are two possible reasons. First, the requirement of citation makes the answer must be related to the context, which inhibits free generation and illusion, thus promoting faithfulness. Second, windowing improves the usefulness of the retrieved context, because it makes the excessively long sentences reasonably segmented, resulting in the content retrieved back to be the key information. Meanwhile, context recall (-16.7%) and answer relevancy (-24%) decreased. The biggest reason is that window optimization makes the context range smaller, so the comprehension of questions and answers becomes worse. In addition, the decline of context recall may be due to the fact that citation prompt is set to answer more concise content, and only recall by citation is used, so recall appears to decrease.

Even more valuable is the qualitative change in EM/F1, with a 4-8 improvement in EM and a 3-8 improvement in F1, especially for CoT prompt. My explanation is that the citation format reduces redundant wording and makes the output more similar to the annotated text, while the window optimization makes the evidence more focused, so it is easier for CoT answers to improve, and it is easier to reduce noise, which together improve strict matching and word overlap.

Production Considerations

After I added context window optimization, passages grew from 3200 to 11255 chunks. The index, memory use and latency all increased. Even in this case, generation stayed fast on easy items, with three prompt styles across 1000 questions finished in around seven minutes. This suggests the system responds quickly on simple tasks even with more chunks. Many queries in the test set ask for yes or no, a date, or one short fact answer, which keeps decoding light. If the context becomes denser and the questions require more reasoning, the answer time will likely rise.

The assignment runs in a Colab Python environment. The stack is lightweight by design, with Milvus Lite and Flan-T5-small to keep costs and setup low. For higher traffic, it is recommended to move to Milvus distributed or FAISS on GPU. For harder contexts and queries, a stronger model like GPT-4o would perform better. The top-k setting and the prompt can also be tuned more finely, while the best mix may differ by model.

With grounded citations on, answers are traceable, ensuring following each claim back to a source. The format increases interpretability, yet outputs sometimes are too minimal. In a few cases the model returns mostly citations with little prose, which shows instability. In conclusion, the system is usable as a small experimental setup, but generalization is limited and the behavior skews toward simple, tightly grounded responses.

Evaluation of RAG

Naive RAG

This evaluation of Naive RAG is written during In the initial evaluation phase, which is based on the original results. Over the process, the evaluation results have minor changes, but the overall trends remain consistent.

During the first evaluation phase, I compared three prompting strategies, which are instruction prompting, chain-of-thought prompting, and persona prompting. For each question in the dataset, I passed the top-1 retrieved document and built a prompt according to the chosen strategy. The system then produced answers that were evaluated with the HuggingFace Squad metric, using EM and F1 score. While EM is a strict measure that requires the predicted answer to match the reference exactly at the string level, F1 is more flexible, since it allows partial word overlap and captures cases where the prediction is semantically close to the real answer even if it is not identical.

The aggregated results are presented below:

Strategy	EM	F1
Persona	40.09	46.59
Instruction	39.76	46.51
Chain-of-Thought	18.52	27.38

The results show that persona prompting achieved the best overall performance. It was slightly better than instruction prompting in both EM and F1, and it clearly outperformed CoT prompting. By framing the model as a historian who specializes in U.S. political history, the answers became closer in style to the references. This role setting may have reduced unnecessary expressions and also encouraged the model to reason in a way consistent with a historian,

which helped raise EM and F1 because the outputs had more word overlap with the reference answers.

Instruction prompting produced results that were very close to persona prompting. This indicates that direct task instructions can be just as effective in short-answer QA. However, compared with persona, its answers were a little weaker in style and precision. Without guidance about accuracy, the outputs sometimes included different wording or phrasing, which lowered EM.

In contrast, chain-of-thought prompting performed the worst. While CoT is often useful in arithmetic or multi-step reasoning tasks, most answers in this dataset were short factual answers. The extra reasoning steps created long explanations and paraphrases, which reduced the strictness of string matching and lowered word overlap with the answers. CoT also tended to produce responses with strong logical structure, often rephrasing the context in its own way, which further decreased overlap in wording and logic. In some cases, the length of the prompt and the output went beyond the model's limit, leading to poor answers and errors.

Overall, the best strategy was persona prompting. A likely reason is that by giving the model a domain-specific identity, it implicitly shaped the answer style, cut down on redundant language, and maintained accuracy. However, if the persona is set in the wrong direction, the performance of the model can drop sharply.

Advanced RAG

In the advanced evaluation, the experiment covered the full set of queries, which ensured both diversity and sample size. To stabilize citation output, I modified the prompt several times and finally embedded few-shot examples into a single unified template, combined with a strict rule that required citations. This adjustment raised the consistency of citation appearance. I also tested retrieval with k set to 1, 3, 5, and 10. The best results came with k=1. My view is that after windowing, the number of candidate chunks increased sharply, and Flan-T5-small became very sensitive to noise under citation constraints. More context only introduced contradictions or disrupted the answer format, which lowered EM and F1.

The advanced results are presented below:

	strategy	EM	F1
0	instruction	43.790850	49.658188
1	persona	43.028322	48.559629

2 cot 24.618736 32.762037

The results show that the gains in EM and F1 were substantial. Instruction remained the most stable strategy, moving from EM/F1 of 39.76/46.51 in the naive setting to 43.79/49.66 in the advanced setting. The increase was about 4.03 for EM and 3.15 for F1. Persona followed a similar pattern, improving from 38.89/45.16 to 43.03/48.56, with slightly larger gains than instruction. With n close to 1000, the 95% confidence interval for EM and H1 is about 3% on both sides, so the difference can be considered statistically significant. I believe the main driver for both strategies was window optimization. The shorter and more focused context reduced interference, which allowed the model to answer more directly and increased the overlap with the reference answers.

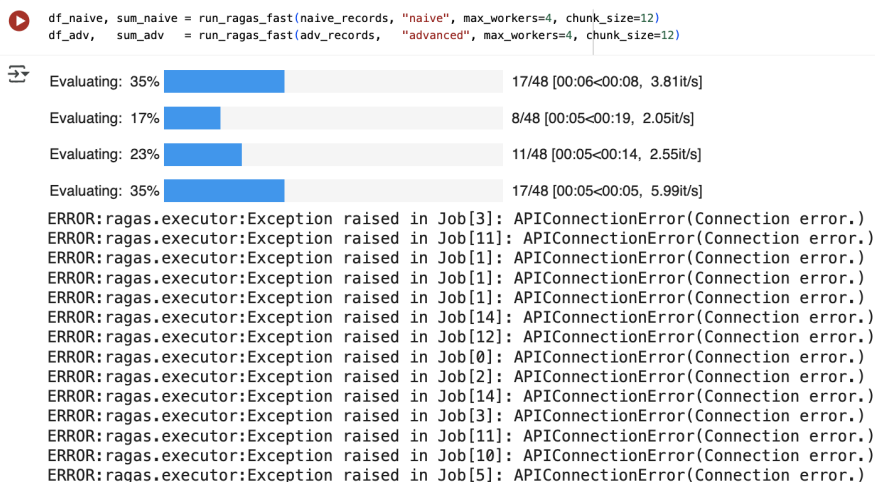
The most striking change came with CoT. EM rose from 16.12 to 24.62, and F1 from 24.73 to 32.76. The gains were about eight points, which is a clear improvement compared to the naive baseline, although the absolute level still lagged behind. In my view, this improvement came from the citation requirement, which constrained the long reasoning chains of CoT and forced the model to produce answers closer to the reference. Yet the complex structure of CoT still led to mismatches with the strict format, so EM and F1 remained the lowest.

From the error cases, I observed three common types. Some answers failed because the key information was not retrieved. Some were too narrow, producing only a citation number or a very short response. Others showed stylistic issues, such as long and unnecessary phrasing. I attempted to reduce these errors by refining the unified prompt, which helped in many cases. In general, advanced shows a clear improvement over naive across all prompts, but the magnitude varies from prompt to prompt.

Advanced Evaluation with RAGAs

When I used RAGAs for evaluation, I ran into constant errors with multi-processing due to API and compute limits, such as:

```
df_naive, sum_naive = run_ragas_fast(naive_records, "naive", max_workers=4, chunk_size=12)
df_adv, sum_adv = run_ragas_fast(adv_records, "advanced", max_workers=4, chunk_size=12)
```



The screenshot shows a Jupyter Notebook interface. At the top, there are two code cells defining `df_naive`, `sum_naive`, `df_adv`, and `sum_adv` using `run_ragas_fast`. Below the code, there is a progress bar showing four evaluation steps: 'Evaluating: 35%', 'Evaluating: 17%', 'Evaluating: 23%', and 'Evaluating: 35%'. Each step has a corresponding blue progress bar and a timestamp. Below the progress bar, there is a list of error messages, all of which are 'ERROR: ragas.executor:Exception raised in Job[...]: APIConnectionError(Connection error.)'.

Progress	Timestamp
Evaluating: 35%	17/48 [00:06<00:08, 3.81it/s]
Evaluating: 17%	8/48 [00:05<00:19, 2.05it/s]
Evaluating: 23%	11/48 [00:05<00:14, 2.55it/s]
Evaluating: 35%	17/48 [00:05<00:05, 5.99it/s]

ERROR: ragas.executor:Exception raised in Job[3]: APIConnectionError(Connection error.)
ERROR: ragas.executor:Exception raised in Job[11]: APIConnectionError(Connection error.)
ERROR: ragas.executor:Exception raised in Job[1]: APIConnectionError(Connection error.)
ERROR: ragas.executor:Exception raised in Job[1]: APIConnectionError(Connection error.)
ERROR: ragas.executor:Exception raised in Job[1]: APIConnectionError(Connection error.)
ERROR: ragas.executor:Exception raised in Job[14]: APIConnectionError(Connection error.)
ERROR: ragas.executor:Exception raised in Job[12]: APIConnectionError(Connection error.)
ERROR: ragas.executor:Exception raised in Job[0]: APIConnectionError(Connection error.)
ERROR: ragas.executor:Exception raised in Job[2]: APIConnectionError(Connection error.)
ERROR: ragas.executor:Exception raised in Job[14]: APIConnectionError(Connection error.)
ERROR: ragas.executor:Exception raised in Job[3]: APIConnectionError(Connection error.)
ERROR: ragas.executor:Exception raised in Job[11]: APIConnectionError(Connection error.)
ERROR: ragas.executor:Exception raised in Job[10]: APIConnectionError(Connection error.)
ERROR: ragas.executor:Exception raised in Job[5]: APIConnectionError(Connection error.)

```

ERROR:ragas.executor:Exception raised in Job[8]: TimeoutError()
ERROR:ragas.executor:Exception raised in Job[10]: TimeoutError()
ERROR:ragas.executor:Exception raised in Job[4]: TimeoutError()
ERROR:ragas.executor:Exception raised in Job[7]: TimeoutError()
ERROR:ragas.executor:Exception raised in Job[11]: TimeoutError()
ERROR:ragas.executor:Exception raised in Job[10]: TimeoutError()
ERROR:ragas.executor:Exception raised in Job[9]: TimeoutError()
ERROR:ragas.executor:Exception raised in Job[12]: TimeoutError()
ERROR:ragas.executor:Exception raised in Job[13]: TimeoutError()
ERROR:ragas.executor:Exception raised in Job[10]: TimeoutError()
ERROR:ragas.executor:Exception raised in Job[11]: TimeoutError()
ERROR:ragas.executor:Exception raised in Job[16]: APIConnectionError(Connection error.)
ERROR:ragas.executor:Exception raised in Job[6]: TimeoutError()
ERROR:ragas.executor:Exception raised in Job[5]: TimeoutError()
ERROR:ragas.executor:Exception raised in Job[7]: TimeoutError()
ERROR:ragas.executor:Exception raised in Job[8]: TimeoutError()
ERROR:ragas.executor:Exception raised in Job[34]: APIConnectionError(Connection error.)
ERROR:ragas.executor:Exception raised in Job[14]: TimeoutError()
ERROR:ragas.executor:Exception raised in Job[15]: TimeoutError()
ERROR:ragas.executor:Exception raised in Job[12]: TimeoutError()
ERROR:ragas.executor:Exception raised in Job[13]: TimeoutError()
ERROR:ragas.executor:Exception raised in Job[14]: TimeoutError()
ERROR:ragas.executor:Exception raised in Job[21]: APIConnectionError(Connection error.)
ERROR:ragas.executor:Exception raised in Job[9]: TimeoutError()

```

Fortunately, I ran samples of only 8 queries from the dataset before. The sample size is very small and the variance is high, but it still gave me a rough sense of the trends. The results are as follows:

	naive	advanced	delta
context_precision	0.250000	0.800000	0.550000
faithfulness	0.500000	1.000000	0.500000
context_recall	0.166667	0.000000	-0.166667
answer_relevancy	0.783891	0.544308	-0.239582

Faithfulness increased from 0.50 to 1.00, which I believe is closely related to the hard citation constraint I added to the prompt. The model was forced to link its answers to retrieved context, reducing hallucination and reaching nearly perfect scores. Context precision rose from 0.25 to 0.80. In my view, this came from window optimization, which made the retrieved information more focused, and from setting k to 1 so that only the most relevant passage was taken, which reduced noise. Context recall, however, dropped from 0.1667 to 0. This may partly reflect the very small sample size, but it is also connected to the narrower windows and the fact that k equal to 1 left many key pieces of evidence out, leading to poor recall. Answer relevancy fell from 0.78 to 0.54. At first this was surprising, but after checking the examples I saw that many answers were either extremely short or only returned citation indices. Therefore, it could be because since I pushed the prompt too hard toward citation stability, the model often chose to output only the citation reference instead of summarizing the content, which hurt relevancy scores.

Although these results are not ideal, they are consistent with what I observed in the larger EM/F1 experiments. The advanced system is more faithful and more precise, but when retrieval returns the wrong passage or when the model misinterprets the prompt, its performance suffers more. This shows that the advanced version has lower tolerance for errors.

AI Usage Logs

1.

Tool: ChatGPT (GPT-5, September 2025 version)

Purpose: Code optimization and technical documentation assistance for Step 1 dataset exploration.

Input:

“Can you help me write the initial code to document dataset structure, sample entries, and data quality observations for the text-corpus and question-answer subsets of the RAG Mini Wikipedia dataset?”

The codes for text-corpus and question-answer subsets are:

```
df_corpus =  
pd.read_parquet("hf://datasets/rag-datasets/rag-mini-wikipedia/data/passages.parquet/part.0.parquet")  
df_qa =  
pd.read_parquet("hf://datasets/rag-datasets/rag-mini-wikipedia/data/test.parquet/part.0.parquet")  
)
```

“----- AssertionError Traceback (most recent call last) /tmp/ipython-input-688615814.py in <cell line: 0>() 10 QA_Q, QA_A, QA_ID = "question", "answer", "id" 11 ---> 12 assert {CORPUS_TEXT, CORPUS_ID}.issubset(df_corpus.columns), "corpus columns mismatch" 13 assert {QA_Q, QA_A, QA_ID}.issubset(df_qa.columns), "qa columns mismatch" 14 AssertionError: corpus columns mismatch “

Output Usage: I adapted the generated Python code to run in Colab for Dataset EDA. On top of it, I modified it to fit the columns for datasets, and cleaned the code to make it more simple.

Verification: I executed the code myself, confirmed the printed statistics matched the dataset (df.head() samples).

2.

Tool: ChatGPT (GPT-5, Sept 2025)

Purpose: Clarify how to organize a minimal Naive RAG pipeline.

Input: “According to Step 2 file, help me write a simplest Naive RAG pipeline step by step. Ensure this is the neatest version!”

Output Usage: I used AI’s suggested skeleton as a reference to outline my own implementation. I then rewrote the codes following the starter code frame in Colab, simplified unnecessary modules from the original skeleton, and integrated with the starter code framework.

Verification: I compared AI's suggested flow with starter code procedures, and confirmed the steps needed. Then I ran the modified code from the starter code file to ensure embeddings, searching, and fetching worked correctly.

3.

Tool: ChatGPT (GPT-5, Sept 2025)

Purpose: Debugging data preprocessing and embedding alignment

Input: Error message: `KeyError: 0` when attempting to access corpus data from the DataFrame.

Output Usage: The AI suggested converting the passage column into a cleaned Python list before embedding, to avoid DataFrame index misalignment errors. I modified my preprocessing accordingly so that passages and embeddings align one-to-one.

Verification: I printed the length of passages and `embeddings.shape[0]` before insertion into Milvus and confirmed both matched to 3200.

4.

Tool: ChatGPT (GPT-5, Sept 2025)

Purpose: Fixing search errors with MilvusClient

Input: Error trace: `MilvusException: required argument is not a float`

Output Usage: The AI explained that the error came from numpy float types not being cast to pure Python floats. I implemented the suggested `as_float_list()` helper function to enforce type conversion during both insertion and querying.

Verification: Running `client.search()` again successfully returned Top-3 passages with no errors, confirming the issue was resolved.

5.

Tool: ChatGPT (GPT-5, Sept 2025)

Purpose: Model selection and conceptual understanding (which LLMs work in Colab without APIs)

Input: "My model options include Claude, ChatGPT-3.5, Gemini, Llama, Mistral, Qwen, and Flan-T5. Which ones don't need APIs and are easiest to add to my Colab code?"

Output Usage: Adopted the recommendation to use Hugging Face Transformers with `google/flan-t5-small` since it has no API key, lightweight, and is suitable for CPU.

Verification: After understanding about the model itself, I verified feasibility by loading the model in Colab and making sure it produced a reasonable result from the context of the question.

6.

Tool: ChatGPT (GPT-5, September 2025)

Purpose: Assisted in drafting initial versions of Instruction, CoT, and Persona prompts for evaluation.

Input: "Help me write prompts for Persona Prompting and Instruction Prompt according to their definitions"

Output Usage: Used AI outputs as a starting point. I independently refined the prompts, adjusted style and words to fit the examples and word limits. I also modified them to make sure they align with each other.

Verification: I validated each prompt strategy by generating sample outputs and confirming that the answers matched the dataset's answers where possible.

7.

Tool: ChatGPT (GPT-5, September 2025)

Purpose: Suggested modifications such as adding a tqdm progress bar and extracting the answer part for CoT.

Input: "Is it normal that it took more than 18 mins? What should I do" "My CoT performs the worst, is it possible that it's because my CoT contains reasoning?"

Output Usage: I used the AI's suggestions as a reference to add a progress bar for monitoring runtime and to implement a simple function that extracts the final answer from CoT outputs by splitting at Answer. I adapted the code to fit my existing evaluation loop and ensured that it worked with my dataset and prompt structure.

Verification: I ran the modified code to ensure the progress bar advanced correctly and that the extracted answers matched the expected format. I compared EM and F1 scores before and after applying the extraction to verify that performance improved as intended.

8.

Tool: ChatGPT (GPT-5, September 2025)

Purpose: Code debugging and troubleshooting for RAG pipeline components with different parameters

Input:

"NameError: name 'dim' is not defined" when defining Milvus schema

"ParamError: expected IndexParams but got dict" when creating index

"KeyError: 'answer'" when constructing SQuAD-format references

flan-t5-small pipeline freezing at 0/102 progress in evaluation loop

Output Usage:

AI pointed out the missing dim assignment and suggested using `emb.shape[1]`.

For Milvus index error, AI explained that IndexParams must be created via `client.prepare_index_params()` instead of a raw dict.

For the KeyError, AI recommended merging the answer column back after sampling or including it earlier in qdf.

For the frozen pipeline, AI proposed running a single-sample test, truncating long contexts with the tokenizer, and reducing `max_new_tokens`.

I selectively adopted the fixes, merged them into my workflow, and modified them to fit my dataset structure.

Verification:

After fixing dim, I re-ran collection creation and confirmed embeddings loaded correctly with the expected dimensions. After changing index creation, I re-inserted 3200 vectors and verified via `client.get_collection_stats` that search worked without index errors. After merging the answers, I printed `eval_df.head()` and confirmed references matched the Squad schema. After truncating contexts, I manually tested one query to confirm flan-t5 produced outputs, then validated the tqdm loop advanced from 0/102 to completion.

9.

Tool: ChatGPT (GPT-5, September 2025)

Purpose: Extend the original retrieval and evaluation pipeline to support multiple embedding dimensions 384d and 512d and multiple top-k retrieval strategies (k=1/3/5).

Input:

"Originally I had def retrieve_top1_context, but now I need three variants: top1, top3, and top5. I will have multiple cases since I use rag_mini_384d and rag_mini_512d, combined with k=1/3/5. My original loop looked like this: [code snippet]. What should I do next?"

Output Usage:

The AI suggested generalizing retrieve_top1_context into a flexible retrieve_topk_context function and introducing outer loops over embedding collections and top-k values. I integrated these modifications into my original code while keeping my tqdm progress bar, prompt-building logic, and rag_pipeline inference unchanged. With this adjustment, I generated predictions for all 6 combinations under each prompting strategy and evaluated them in a single pipeline.

Verification:

I checked the reported output dimension for each embedding model to confirm that the 384d and 512d encoders produced vectors of the expected size. I rebuilt the Milvus collections with the correct dim schema and verified the collection statistics to ensure embeddings matched the intended dimensions. Finally, I ran HuggingFace evaluate.load to compute EM/F1 scores for all combinations and verified that the number of predictions matched the number of references, ensuring evaluation integrity.

10.

Tool: ChatGPT (GPT-5)

Purpose: Technical documentation assistance for Assignment 2 report writing

Input: Prompts requesting help to translate analysis into academic English, rephrase sections of the report, and refine wording to fit university-level writing style while maintaining clarity.

"In advanced evaluation, I observed that EM and F1 increased slightly after adding grounded citations and context window optimization. This is my original analysis, can you help me rewrite this section in academic English, with clear logical flow and at a university-level writing style?"

Output Usage: AI suggestions were used to draft report sections, improve paragraph flow, and ensure the structure matched the required Technical Report format. Rewritten passages were selectively integrated into the final report after manual review.

Verification: Accuracy was confirmed by cross-checking AI-translated text with my own experimental results, metrics tables, and assignment requirements. I verified all the generated analysis and rewrote the logic, evidence, reasoning, etc. making sure all interpretations and translations remained consistent with my independent analysis.

11.

Tool: ChatGPT (GPT-5, October 2025)

Purpose: code optimization and debugging in advanced RAG pipeline.

Input: Example prompts included:

"My code is throwing a MilvusException: No index found in field [embedding]. What should I add or change to fix it?"

"This output only shows [1] instead of the final answer. How should I modify build_prompt or extract_final_answer to make it output a clean answer?"

"Can you rewrite my build_prompt to support Query Rewriting but still keep it simple?"

"Reranking will it run slow? Can it be integrated easily into my code?"

Output Usage: I adapted the AI's suggestions into my workflow. For example, I inserted index creation and load_collection() into my Milvus code, rewrote build_prompt with stricter formatting, and minimally patched extract_final_answer to strip citation markers while keeping my baseline logic.

Verification: I systematically verified each change. I ran preview_examples() to inspect raw results and references to confirmed fixes to the citation issue. I re-ran metric.compute() to measure EM/F1 differences and ensure improvements were meaningful and consistent.

12.

Tool: ChatGPT (GPT-5, October 2025)

Purpose: Code debugging and optimization suggestions during Step 6 (RAGAs evaluation)

Input: Example prompts I asked:

"My code is stuck at 0% evaluating, what should I do?"

"If I keep getting RateLimitError, how can I change my code?"

"Can you rewrite my run_ragas function to use ThreadPoolExecutor?"

"I am still getting APIConnectionError, can you give me a safer serial version with retries?"

These prompts were focused on diagnosing errors (TimeoutError, APIConnectionError, cannot pickle RLock) and exploring different execution strategies (multi-threading, multi-processing, serial execution with retries).

Output Usage: I integrated the AI suggested modifications with my own code and did manual experiments with each variation, including changing max_workers, adjusting chunk_size, and restarting Colab runtime to confirm network stability. I integrated the usable version into my pipeline, ensuring RAGAs could run the sample subset for evaluation.

Verification: I verified correctness by comparing outputs between models (naive vs advanced) to ensure metrics like faithfulness and context precision were computed. I ran smaller subsets to confirm the code logic to solve errors.

Implementation Instruction

Configuration

embedding_model: "all-MiniLM-L6-v2"

vector_db: "milvus-lite"

top_k: 1

index_params:

metric: "COSINE"

nlist: 128

generation_model: "google/flan-t5-small"

Tool Inventory

GenAI Models: HuggingFace Transformers (flan-t5-small), OpenAI GPT

Embedding Models: sentence-transformers/all-MiniLM-L6-v2,
distiluse-base-multilingual-cased-v1

Vector DBs: Milvus Lite

Libraries/APIs: pandas, numpy, tqdm, ragas, evaluate, sentence-transformers, pymilvus, torch, transformers (AutoTokenizer, AutoModelForSeq2SeqLM, AutoModelForCausalLM, pipeline), datasets (Dataset), sentence_transformers (SentenceTransformer), pymilvus (MilvusClient, FieldSchema, CollectionSchema, DataType), milvus-lite, ragas (evaluate, metrics: faithfulness, answer_relevancy, context_precision, context_recall), evaluate (HuggingFace metrics, e.g., squad), tiktoken (for token counting / chunking), tqdm.auto (progress bar), logging (system logging), os, sys, re, time, random

Technical Specifications

Language: Python 3.10+

Frameworks: HuggingFace Transformers, SentenceTransformers

Evaluation: HuggingFace evaluate (F1, EM), RAGAs (faithfulness, recall, precision, relevancy)

Hardware: CPU/GPU (Colab supported)

Github

https://github.com/AislynnLi/NLX_RAG