Team17-Engineering Copilot using LLMs augmented with Public and Proprietary Documentation

Aswin B[a,b], Karthik Subburaj[a,b], Sanuj Kulshrestha[a,b], Shivam Kumar[a,b] and Shreya Gupta[a,b]

[a]Engineer, Texas Instruments
[b]Dept. of EECS, Indian Institute of Science

Abstract. This paper presents a chat-bot application that is capable of answering engineering queries in the field of semiconductor devices. It is targeted as a replacement to traditional engineer-to-engineer interactions for addressing customer queries on specific Integrated Circuits (ICs) by referring to company proprietary documentation such as IC datasheets, application notes, and user manuals, as well as publicly available information. This application is built using publicly available Large Language Models (LLMs), software based on Retrieval Augmented Generation (RAG) and LangChain framework. Performance comparisons of multiple RAG approaches are published based on a test dataset of questions and answers.

1 Introduction

In semiconductor industries, queries related to products from the customers are typically answered directly by design engineers though there are sufficient documentations already present. The challenge here is for these simple queries we need not spend engineer's time. To sole this issue we propose a system where we use LLM to answer these queries based on the technical documentations (publicly available documents like datasheets, application notes, user manuals, and engineering guides and also internal documents like implementation documents*)

The system retrieves precise, contextually relevant answers from athese documents, thereby reducing manual effort and improving turnaround time.

This paper presents the implementation and evaluation of multiple Retrieval-Augmented Generation (RAG) strategies for such a system. We compare retrieval approaches, post-retrieval processing techniques—such as semantic re-ranking and contextual compression—and analyze their effect on LLM-only answer quality. Using a representative dataset of engineering queries, we evaluate output quality across configurations.

Such a solution is necessary now to address the growing challenge of technical support automation in the semiconductor domain.

2 RAG processing chain

The chatbot system is architected using the **LangChain framework**, which provides a modular and extensible pipeline for Retrieval-Augmented Generation (RAG). The system is designed to answer technical queries based on the 22 engineering documents, including datasheets, application notes, user manuals, and product guides. The entire RAG chain comprises multiple stages: document embedding, retrieval, post-retrieval processing, and LLM-based answer generation and creating an interactive user interface.

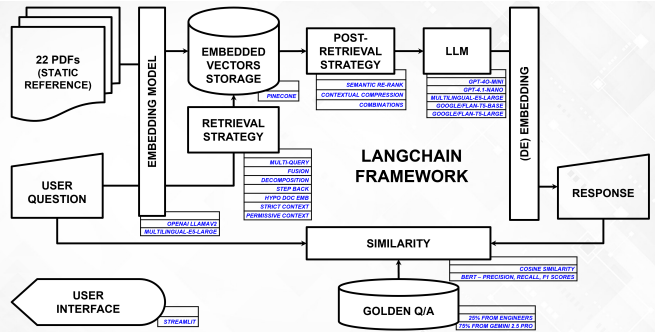

Figure 1. System architecture of the RAG-based engineering copilot

2.1 Document Embedding and Storage

The embedding process converts document chunks into dense vector representations using **LLaMA v2** and **multilingual-e5-large**. These vectors are indexed and stored using **Pinecone**, a high-performance vector database, allowing low-latency similarity-based retrieval across large document corpora.

2.2 Retrieval Strategies

We employed multiple Retrieval-Augmented Generation (RAG) strategies leveraging vectorized document storage in Pinecone to enhance the accuracy of user query responses. These strategies, implemented using LangChain's retriever modules, can be dynamically configured via the user interface. The evaluated retrieval methods include:

- **Simple RAG:** Performs direct vector similarity-based retrieval of relevant document chunks to ground answers in technical documentation.
- **Multi-query Retrieval:** Generates and retrieves multiple query variants to address phrasing differences, enhancing coverage.
- **RAG-Fusion:** Combines results from multiple sub-queries using Reciprocal Rank Fusion (RRF) for multi-faceted query resolution.
- **Query Decomposition:** Splits complex queries into atomic parts, retrieves context for each, and combines results for improved precision.
- **Step-back Prompting:** Reformulates queries at a higher abstraction level to retrieve broader and more general context.

- **HyDE:** Uses an LLM-generated hypothetical document for retrieval when the original phrasing lacks direct matches in the corpus.
- **LLM-only:** Skips retrieval entirely and generates answers based solely on the model's internal knowledge which is used as a baseline.

2.3 *Post-Retrieval Processing*

To enhance answer precision, we apply refinement techniques after the initial top-10 retrieval from Pinecone:

- **Semantic Re-Ranking**: Uses the cross-encoder or ms-marco-MiniLM-L-6-v2 to re-score and rank retrieved chunks, selecting the top-3 or 5 based on semantic relevance.
- **Contextual Compression**: Utilizes an LLM to extract only the most query-relevant sentences, reducing noise and providing appropriate information to the final LLM.
- **Semantic Re-Ranking + Contextual Compression**: Applies re-ranking followed by compression to maximize contextual quality passed to the LLM.

2.4 *Prompting Strategies for Answer Generation*

Answer generation is performed using LLMs such as **GPT-4.0-mini**, **GPT-4.1-nano**, and **Flan-T5**, guided by two prompting strategies:

- **Strict Context**: Restricts LLM to use *only* the retrieved documents for answering, ensuring factual accuracy and traceability.
- **Permissive Context**: Allows use of both retrieved context and the LLM's prior knowledge.

2.5 *User Interface and Integration*

The user interface is developed using **Streamlit**, a interactive Python framework enabling real-time interaction with the backend Retrieval-Augmented Generation (RAG) pipeline. This ensures smooth communication between user inputs and LLM-based responses.

The main panel of the application includes:

- A **dropdown menu** for selecting retrieval strategies such as Simple, Multi-query, RAG Fusion, and Decomposition.
- **Selectors** for post-retrieval processing (semantic re-ranking, contextual compression, or both).
- A **toggle** for choosing between strict and permissive prompting modes.

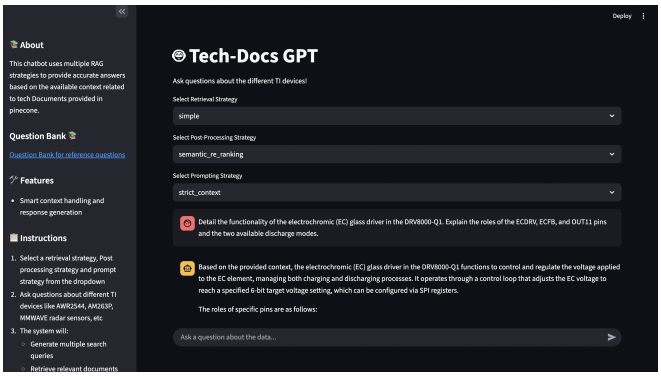

Figure 2. User interface of the engineering copilot (Click on image)

The sidebar provides detailed usage instructions and quick access to a curated bank of evaluation questions for user experimentation. All user interface selections are dynamically relayed to the backend orchestration layer, which invokes the relevant **LangChain components and Pinecone retrievers**. This modular and tightly coupled UI-backend architecture ensures fast, transparent, and domain-grounded response generation.

3 Evaluation and Inferences

To assess the performance and usability of the proposed chatbot, we conducted an evaluation using a curated test set of **100 golden questions** derived from **22 technical documents**. The questions were constructed with a balanced mix: **25% engineer-authored** and **75% LLM-generated** (via Gemini 2.5 pro), ensuring both human relevance and language model diversity.

3.1 *Retrieval Strategy Comparison*

Using `GPT-4.1-nano` as the answering LLM, we evaluated the effectiveness of various retrieval strategies on the evaluation dataset. Results in Table 1 and Figure 3 show that RAG-based methods achieve approximately a **3% improvement** in both **BERT F1** and **cosine similarity** scores over the non-retrieval (*LLM-only*) baseline, indicating a meaningful enhancement in answer relevance and grounding.

Among the retrieval strategies, *Decomposition* yields the highest BERT F1 score (0.8654) and cosine similarity (0.7903), followed closely by *HyDE*, *RAG Fusion*, and *Multi-query*, all of which demonstrate competitive performance.

RAG Strategy	BERT Precision	BERT Recall	BERT F1	Cosine Similarity
Decomposition	0.8546	0.8773	0.8654	0.7903
HyDE	0.8530	0.8760	0.8640	0.7812
LLM Only	0.8085	0.8584	0.8324	0.7584
Multi-query	0.8531	0.8747	0.8634	0.7765
RAG Fusion	0.8532	0.8752	0.8637	0.7754
Simple	0.8540	0.8739	0.8635	0.7721
Step-back	0.8520	0.8734	0.8622	0.7608

Table 1. Performance Metrics Across retrieval Strategies

In Figure 3, Performance across the different RAG strategies is observed to be fairly similar, with cosine similarity improving steadily as we progress towards decomposition RAG on right extreme of the graph. The standard deviation of scores remains practically same across RAG strategies in our dataset and experiments.

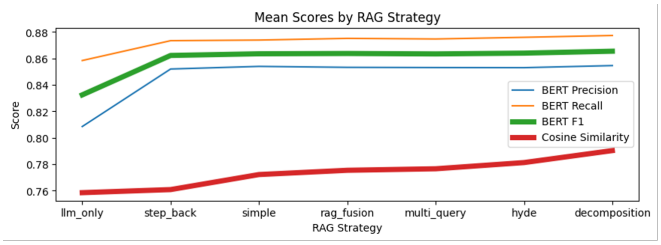

Figure 3. Comparison of avg scores of different RAG strategies (with gpt-4.1-nano)

The differences across strategies are subtle, suggesting that while RAG enhances factual grounding, the marginal gains between advanced strategies are incremental. Nonetheless, semantic-rich methods like Decomposition and HyDE are slightly better than simpler approaches, particularly in scenarios requiring multi-aspect reasoning.

3.2 *Post-Retrieval Processing Evaluation*

To enhance answer quality, we evaluated post-retrieval refinement techniques applied after an initial top-$k = 10$ vector retrieval from Pinecone.

- **Semantic Re-ranking** using the cross-encoder `ms-marco-MiniLM-L-6-v2` to re-score top-10 candidates and select top-k chunks.
- **Contextual Compression**, where only query-relevant sentences are retained using an LLM-based extractor(`gpt-4o-mini`).
- The **combined strategy** (re-ranking + compression), with `top_k = 10` and `reranking_top_n = 3 or 5`.

Post-Retrieval Strategy	BERT Precision	BERT Recall	BERT F1 Score	Cosine Similarity
Contextual Compression	0.8427	0.8751	0.8583	0.7799
Semantic Re-ranking	0.8496	0.8705	0.8595	0.7723
Semantic Re-ranking + Contextual Compression	0.8495	0.8662	0.8574	0.7523

Table 2. Comparison of average scores for post-retrieval processing strategies using `GPT-4.1-nano` as LLM.

Table 3.2 summarizes the results for contextual compression, semantic re-ranking, and a combined approach. In our datasets and experiments, no significant performance difference is observed across these post retrieval processing strategies.

3.3 *Prompt Strategy Analysis*

Strict context—which constrains the LLM to generate answers using only retrieved documents—improves cosine similarity and BERT precision by approximately 3-5% in our evaluations, promoting slightly higher factual accuracy by avoiding reliance on the model's prior knowledge (Table 3).

Prompt Strategy	BERT Precision	BERT Recall	BERT F1 Score	Cosine Sim.
Permissive Context	0.8343	0.8761	0.8545	0.7952
Strict Context	0.8602	0.8697	0.8646	0.7533

Table 3. Comparison of average scores of RAG prompt strategies using `GPT-4.1-nano`.

3.4 *Embedding model Comparison*

In our datasets and experiments, no significant performance difference is observed across different embedding models (`multilingual-e5-large` vs. `llamav2`), both of which are high-performing transformer encoders (Table 4).

Embedding Strategy	Precision	Recall	F1 Score	Cosine Sim.
multilingual-e5-large + decomposition	0.8499	0.8745	0.8617	0.7778
multilingual-e5-large + rag_fusion	0.8497	0.8722	0.8605	0.7626
llamav2 + decomposition	0.8546	0.8773	0.8654	0.7903
llamav2 + rag_fusion	0.8532	0.8752	0.8637	0.7754

Table 4. Comparison of average scores across different embedding models.

3.5 *Answering LLM Comparison*

All the above analysis is based on high end LLM - GPT-4.1-nano. To assess the influence of the language model on retrieval performance, we compared `GPT-4.1-nano` to `GPT-4o-mini` (>40B parameters) across two RAG configurations: *multi-query* and *simple retrieval* with top-k = 10 (Table 5 - indicates no significant differences among them in our datasets).

Additionally, we compared them with smaller language models (`google/flan-t5-base` (250M parameters) and `google/flan-t5-large` (750M parameters), smaller in terms of parameter) . We observed cosine similarity score to be significantly poorer (20% , refer to Figure 3 for Large Language Model and Figure 4 for Smaller Language Model), while BERT scores are comparable. Even after RAG, the absolute cosine similarity with golden answers remains relatively low with smaller language models, indicating limited semantic alignment due to model capacity constraints.

Configuration	Precision	Recall	F1 Score	Cosine Similarity
GPT-4.1-nano + Multi-query	0.8531	0.8747	0.8634	0.7765
GPT-4o-mini + Multi-query	0.8523	0.8711	0.8613	0.7715
GPT-4.1-nano + Simple	0.8540	0.8739	0.8635	0.7721
GPT-4o-mini + Simple	0.8575	0.8737	0.8652	0.7742

Table 5. Comparison of RAG strategies using `GPT-4.1-nano` vs. `GPT-4o-mini` as the LLM.

For smaller language models, the use of retrieval methods yields a modest but meaningful improvement of approximately 5–10% in both **BERT F1** and **cosine similarity** scores on our dataset (Figure 4) with top-k = 3.

Variation of top-k values were seen to provide very small differences for our dataset.

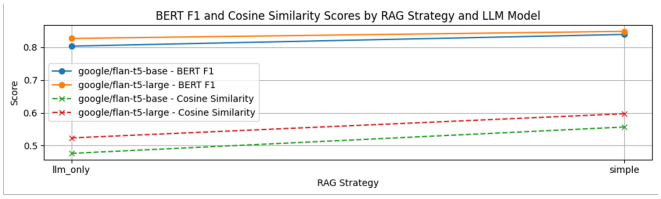

Figure 4. Score improvements due to RAG for smaller language models

4 Conclusions

We have implemented and evaluated aspects of retrieval augmentated chain and have published the similarity scores. In general, we have observed the cosine similarity to be a stricter comparison metric as compared to BERT which appears to be more optimistic. We have observed RAG-based strategies improve BERT F1 and cosine similarity scores by 3-5% over the LLM-only baseline. Among them, Decomposition and HyDE perform slightly better.

5 Contributions

Some common contributions include curating documents for creating embedding, creating golden question and answers. Although everyone was involved in all other aspects of the development, some lead individual contributions are as follows:

Aswin - Created embeddings of the documents based on llamav2, multilingual-e5-large using Pinecone, python script to extract images and tables in the document, used LLM (Claude 3.5 Sonnet v2) to describe the images and embedded to the db, so can be used with LLM (text only input models). Also contributed in final running of full chain

Karthik - Initial python code demonstrating basic RAG chain for the project's aim, framework for iterating over all LLMs and RAG strategies, adding low end LLMs, analysing and generating comparisons and inferences, and report making.

Sanuj - Development of retrieval strategies, infrastructure setup of pinecone vectorDB, custom retrieval class and langchain, generating bert and cosine scores in validation, backend integration to UI, and hosting application

Shivam - Developing python code using langchain framework for multiple retrieval strategies, development of different post-retrieval strategies and different prompting and setup of chain.

Shreya - Designed and implemented an interactive Streamlit-based user interface featuring dynamic RAG strategy selection, real-time query processing, and configurable post-retrieval options, backend integration of UI to work with the implemented chain, report making

6 Online application

The application UI is available in the following link:

https://techdocsgptiisc-8r3eb2algcfscgszk7rnnm.streamlit.app

This UI uses Gemini based free model to avoid API cost (as we have exhausted our paid model in evaluation).

7 Github Link

The link to github having code, reference documents and questionnaire is mentioned below -

TechDocs-GPT Github link