RAG: Retrieval Augmented Generation

Introduction

Retrieval Augmented Generation is a technique for enhancing the accuracy and reliability of generative AI models with information fetched from specific and relevant data sources. RAG uses vector database technology to store up-to-date information that's retrieved using semantic searches and added to the context window of the prompt, along with other helpful information, to enable the Large Language Model (LLM) to formulate the best possible, up-to-date response.

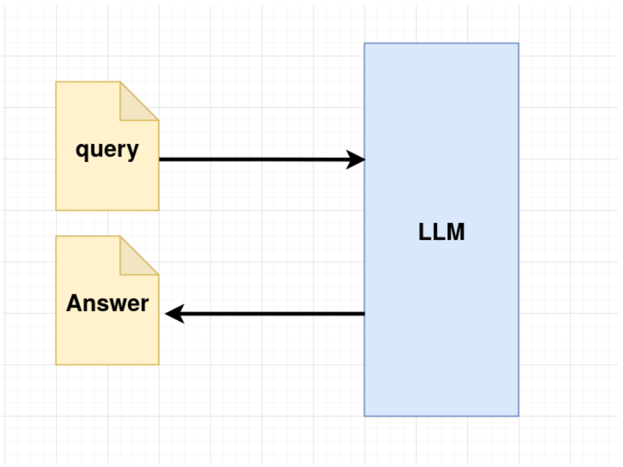
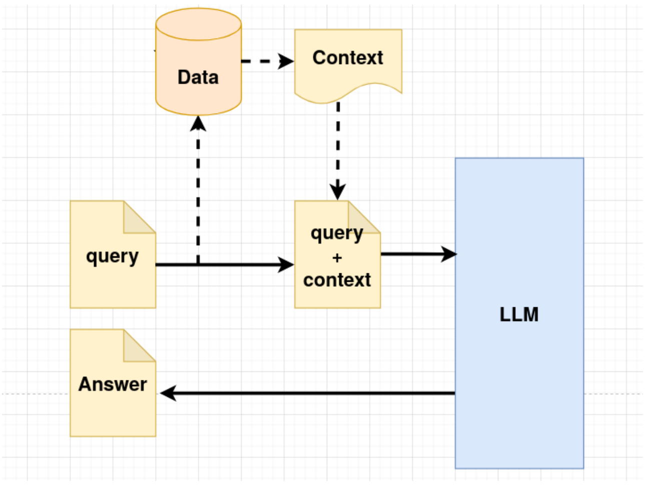


Figure 1: Just LLM



Figuer 2: RAG Integration

We provide query and LLM answers that query. Large Language Models (LLMs) showcase impressive capabilities but encounter challenges like:

- **Hallucination**: LLMs sometimes generate false or misleading information with confidence due to their reliance on statistical patterns rather than factual verification.
- **Outdated Knowledge**: Since LLMs are trained on static datasets, they lack real-time awareness and may provide obsolete information.
- **Non-Transparent, Untraceable Reasoning**: LLMs function as "black boxes," making it difficult to understand or verify how they arrive at their conclusions.

Retrieval-Augmented Generation (RAG) has emerged as a promising solution by incorporating knowledge from external databases. We process a query by first retrieving relevant context from external databases, then combining the original query with the retrieved context to form a prompt for the LLM. This approach enables the LLM to generate more accurate and contextually relevant answers.

Downstream Task Of RAG

The downstream tasks of Retrieval-Augmented Generation (RAG) refer to the specific applications or tasks where RAG can be applied to enhance the model's performance by combining retrieved context with query processing.

1. **Question Answering (QA)**: RAG improves open-domain question answering by retrieving relevant documents and generating accurate, contextually rich answers.
2. **Summarization**: RAG enhances document summarization by retrieving key content and generating concise summaries that are contextually relevant.
3. **Dialogue Generation**: RAG boosts conversational agents by retrieving prior context or external knowledge to generate informative, engaging responses.
4. **Fact-Checking**: RAG aids fact-checking by retrieving supporting or contradictory evidence to verify the accuracy of statements.
5. **Document Retrieval**: RAG improves document retrieval by combining relevant content with query context to generate more precise and contextual results.

RAG Paradigms

1. **Naïve RAG**: Retrieves relevant documents and directly appends them to the query before passing it to the LLM without additional processing.

2. **Advanced RAG**: Enhances retrieval with ranking, filtering, and summarization techniques to improve context relevance before generating a response.

3. **Modular RAG**: Uses a structured pipeline with separate modules for retrieval, preprocessing, augmentation, and generation, allowing flexibility and optimization at each stage.

To understand **Retrieval-Augmented Generation (RAG)**, we will start with the **Naïve RAG** approach, which serves as the simplest implementation of this technique.

Naïve RAG

Naïve RAG operates by directly retrieving relevant documents or context from an external knowledge source and appending them to the user's query before passing it to a Large Language Model (LLM). The LLM then generates a response based on this combined input.

Characteristics of Naïve RAG

- **Simple Implementation**: It does not involve complex ranking, filtering, or summarization techniques, making it easy to set up.
- **Limited Optimization**: Since all retrieved documents are used as it is, there is a risk of including irrelevant or redundant information, which may lead to suboptimal responses.
- **Baseline for Improvement**: It provides a fundamental understanding of RAG before moving on to more advanced optimizations, such as context ranking, summarization, or modular pipelines.

RAG Architecture

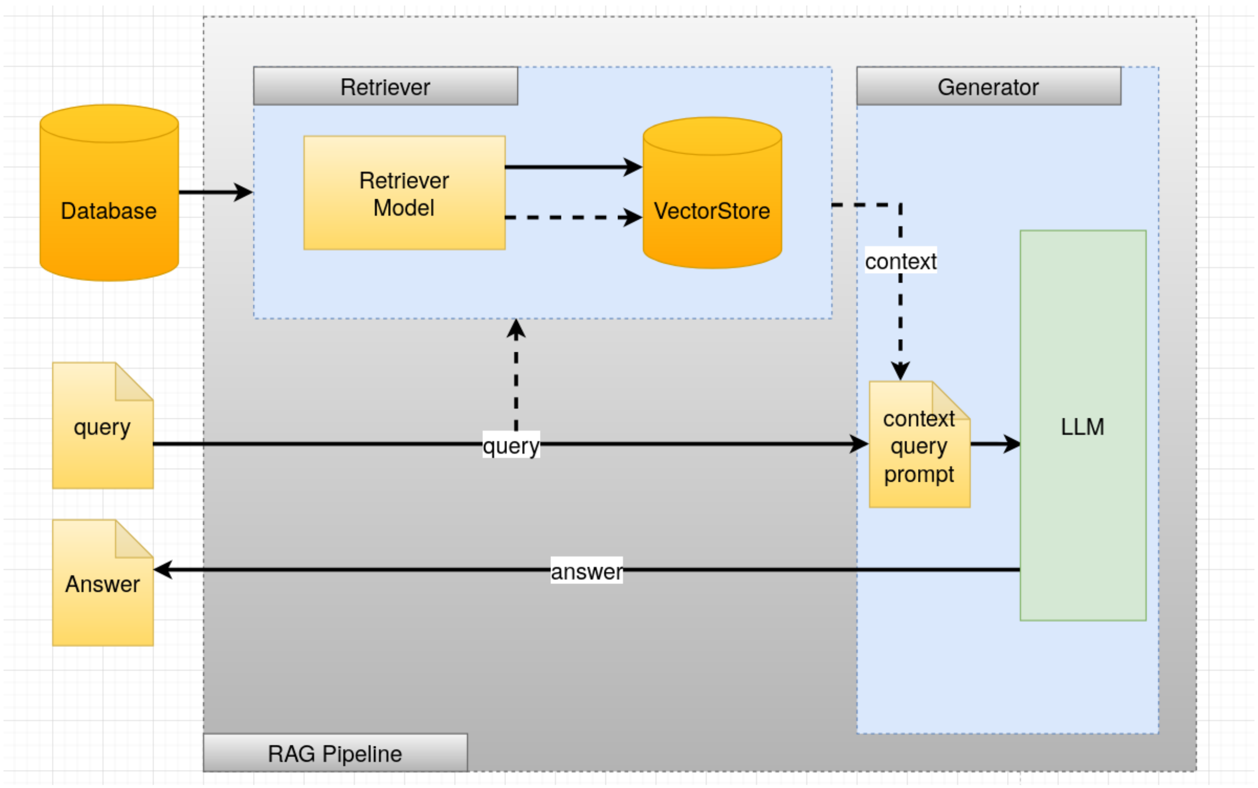


Figure 4: RAG Architecture

Working of this Architecture

1. **User Query**: The user submits a question or prompt.
2. **Context Retrieval**: The system searches an external database (such as a vector store, document repository, or knowledge base) to find relevant documents based on the query.
3. **Context Augmentation**: The retrieved documents are **directly** added to the user's original query without additional filtering or ranking.
4. **Prompt Formation**: The combined query and retrieved context are formatted into a single prompt.
5. **LLM Response Generation: The LLM processes the prompt and generates a response based on both the query and the retrieved contex.**

RAG Pipeline

The **RAG pipeline** consists of two main components: **Retrieval** and **Generation**, ensuring that an LLM provides more accurate and context-aware answers.

Retriever

1. Document Processing & Storage

- **Chunking**: Large documents are split into smaller text chunks for efficient retrieval.
  Document:
  This is just 1 page pdf file that talks about pink cat.

```
tom, the pink cat, was unlike any other feline in the neighborhood. with his soft, pastel-pink fur that shimmere
d under the sunlight, he often became the center of attention. despite his unusual color, tom was as playful and
 mischievous as any other cat, always chasing after butterflies and rolling around in the grass. he had bright g
reen eyes that sparkled with curiosity, making him look even more enchanting. tom loved to sit by the window, wa
tching the world go by, occasionally letting out a soft meow to call for attention. his favorite pastime was sne
aking into the kitchen and trying to steal a bite of fish from the table. the children in the neighborhood adore
d him and would often gather around to pet his silky fur. even the other cats seemed to accept him despite his u
nique appearance. tom, with his gentle purrs and affectionate nature, brought joy to everyone around him. he was
 not just a cat, he was a magical presence in the lives of those who knew him.
```

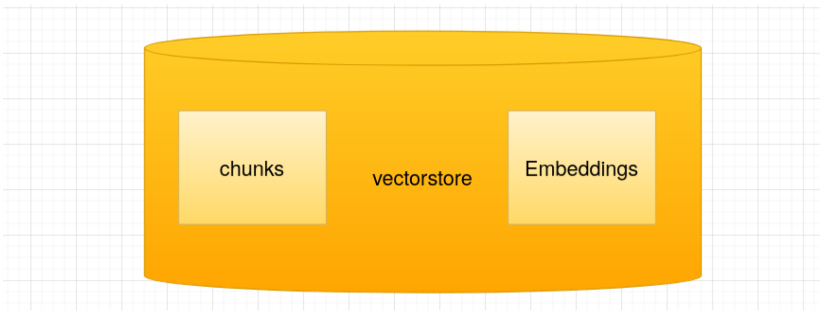
  Chunks:
  The text content is divided into chunks of length 100 characters each.

```
['tom, the pink cat, was unlike any other feline in the neighborhood. with his soft, pastel-pink fur t', 'hat sh
immered under the sunlight, he often became the center of attention. despite his unusual color', ', tom was as p
layful and mischievous as any other cat, always chasing after butterflies and rolling ', 'around in the grass. h
e had bright green eyes that sparkled with curiosity, making him look even mor', 'e enchanting. tom loved to sit
 by the window, watching the world go by, occasionally letting out a s', 'oft meow to call for attention. his fa
vorite pastime was sneaking into the kitchen and trying to ste', 'al a bite of fish from the table. the children
 in the neighborhood adored him and would often gather', ' around to pet his silky fur. even the other cats seem
ed to accept him despite his unique appearance', '. tom, with his gentle purrs and affectionate nature, brought
joy to everyone around him. he was not', ' just a cat, he was a magical presence in the lives of those who knew
him.']
```

- **Embedding Generation**: Each chunk is converted into a vector representation using an embedding model (e.g., Sentence-BERT, OpenAI Embeddings).

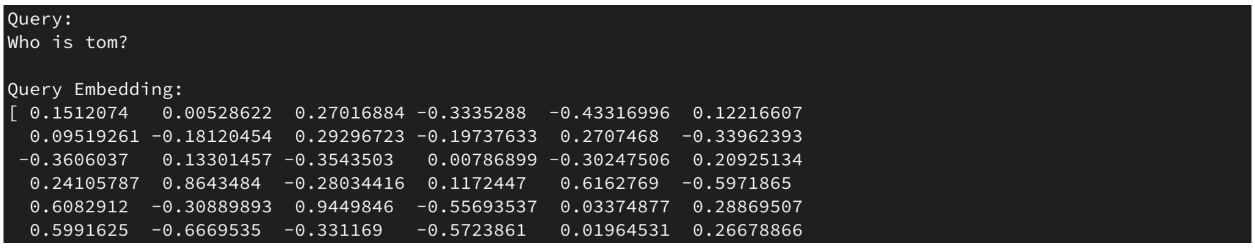
```
Embeddings:
[[ 0.16743788 -0.5457768   0.5253833  ... -0.0880075   0.4896534
   0.3612664 ]
 [ 0.24067646  0.74029577 -0.00524869 ...  0.01187833 -0.13988823
   0.07723967]
 [ 0.04531208  0.01084917  0.23031802 ...  0.08132657  0.30727002
   0.7121252 ]
 ...
 [ 0.20039086  0.19444197  0.10238229 ... -0.18903373  0.385372
   0.59252703]
 [ 0.19985904  0.5895685   0.53143597 ...  0.14515807  0.2363665
   0.5828338 ]
 [-0.06113426  0.1784846  -0.30699706 ...  0.24032931  0.16707872
   0.33649245]]
```

- **Vector Storage**: The embeddings and corresponding text chunks are stored in a **vector database** (e.g., FAISS, Pinecone, Milvus).



2. Query Processing & Retrieval

- **Query Embedding**: The user query is transformed into an embedding using the same model.

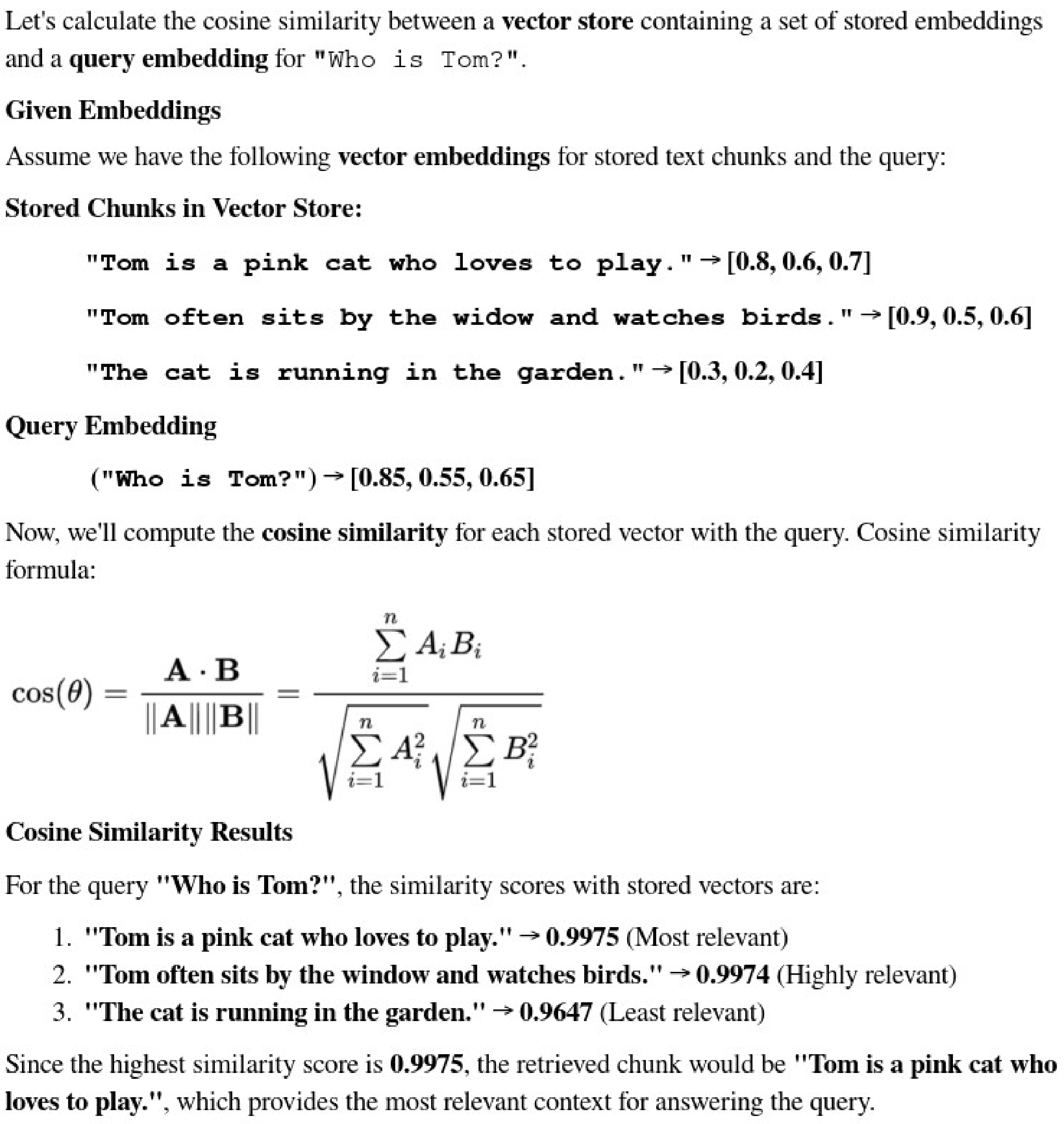


```
Query:
Who is tom?

Query Embedding:
[ 0.1512074   0.00528622  0.27016884 -0.3335288  -0.43316996  0.12216607
  0.09519261 -0.18120454  0.29296723 -0.19737633  0.2707468  -0.33962393
 -0.3606037   0.13301457 -0.3543503   0.00786899 -0.30247506  0.20925134
  0.24105787  0.8643484  -0.28034416  0.1172447   0.6162769  -0.5971865
  0.6082912  -0.30889893  0.9449846  -0.55693537  0.03374877  0.28869507
  0.5991625  -0.6669535  -0.331169   -0.5723861   0.01964531  0.26678866
```

- **Similarity Search**: The query embedding is compared against stored embeddings using **cosine similarity** or other distance metrics.

  Example:



Let's calculate the cosine similarity between a **vector store** containing a set of stored embeddings and a **query embedding** for `"Who is Tom?"`.

Given Embeddings

Assume we have the following **vector embeddings** for stored text chunks and the query:

Stored Chunks in Vector Store:

`"Tom is a pink cat who loves to play."` → **[0.8, 0.6, 0.7]**

`"Tom often sits by the widow and watches birds."` → **[0.9, 0.5, 0.6]**

`"The cat is running in the garden."` → **[0.3, 0.2, 0.4]**

Query Embedding

`("Who is Tom?")` → **[0.85, 0.55, 0.65]**

Now, we'll compute the **cosine similarity** for each stored vector with the query. Cosine similarity formula:

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

Cosine Similarity Results

For the query **"Who is Tom?"**, the similarity scores with stored vectors are:

1. **"Tom is a pink cat who loves to play."** → **0.9975** (Most relevant)
2. **"Tom often sits by the window and watches birds."** → **0.9974** (Highly relevant)
3. **"The cat is running in the garden."** → **0.9647** (Least relevant)

Since the highest similarity score is **0.9975**, the retrieved chunk would be **"Tom is a pink cat who loves to play."**, which provides the most relevant context for answering the query.

- **Top-k Retrieval**: The most relevant **top-k** chunks are retrieved from the vector database. Here top 3 relevant context to “who is tom?” is retrieved.

```
Context:
['. tom, with his gentle purrs and affectionate nature, brought joy to everyone around him. he was not', 'e ench
anting. tom loved to sit by the window, watching the world go by, occasionally letting out a s', 'oft meow to ca
ll for attention. his favorite pastime was sneaking into the kitchen and trying to ste']
```

Generator

1. Context Augmentation & Generation

- **Input Construction**: The retrieved chunks are combined with the original query to form a structured prompt.
  Prompt = Context + Query

```
Prompt:
Context:
['. tom, with his gentle purrs and affectionate nature, brought joy to everyone around him. he was not just a ca
t, he was a magical presence in the lives of those who knew him.', 'e enchanting. tom loved to sit by the window
, watching the world go by, occasionally letting out a soft meow to call for attention. his favorite pastime was
 sneaking into the kitchen and trying to ste', ', tom was as playful and mischievous as any other cat, always ch
asing after butterflies and rolling around in the grass. he had bright green eyes that sparkled with curiosity,
making him look even mor', 'tom, the pink cat, was unlike any other feline in the neighborhood. with his soft, p
astel-pink fur that shimmered under the sunlight, he often became the center of attention. despite his unusual c
olor', 'al a bite of fish from the table. the children in the neighborhood adored him and would often gather aro
und to pet his silky fur. even the other cats seemed to accept him despite his unique appearance']
Query:
Who is tom?
Answer:
```

- **Feeding to LLM**: The constructed prompt is passed to a language model (e.g., Gemini, GPT-4, T5, LLaMA).

- **Generating Response**: The LLM produces a final, context-aware response based on the provided information.

```

Response: Tom is a pink cat with soft, pastel-pink fur that shimmered in the sunlight. He was playful, mischievo
us, affectionate, and had bright green eyes.  He was loved by the neighborhood children and even other cats, des
pite his unusual coloring.
```

Key Issues With RAG

1. **What to Retrieve**: Deciding which documents or context to retrieve is challenging, as irrelevant or low-quality content can negatively impact the model’s performance.
2. **When to Retrieve**: Determining the optimal timing for retrieval is tricky, as retrieving too early or too late can affect the model's ability to generate coherent and contextually appropriate responses.
3. **How to Use Retrieval**: Effectively integrating retrieved content with the query requires careful handling to ensure the model can utilize the context meaningfully without overwhelming or confusing the response generation.

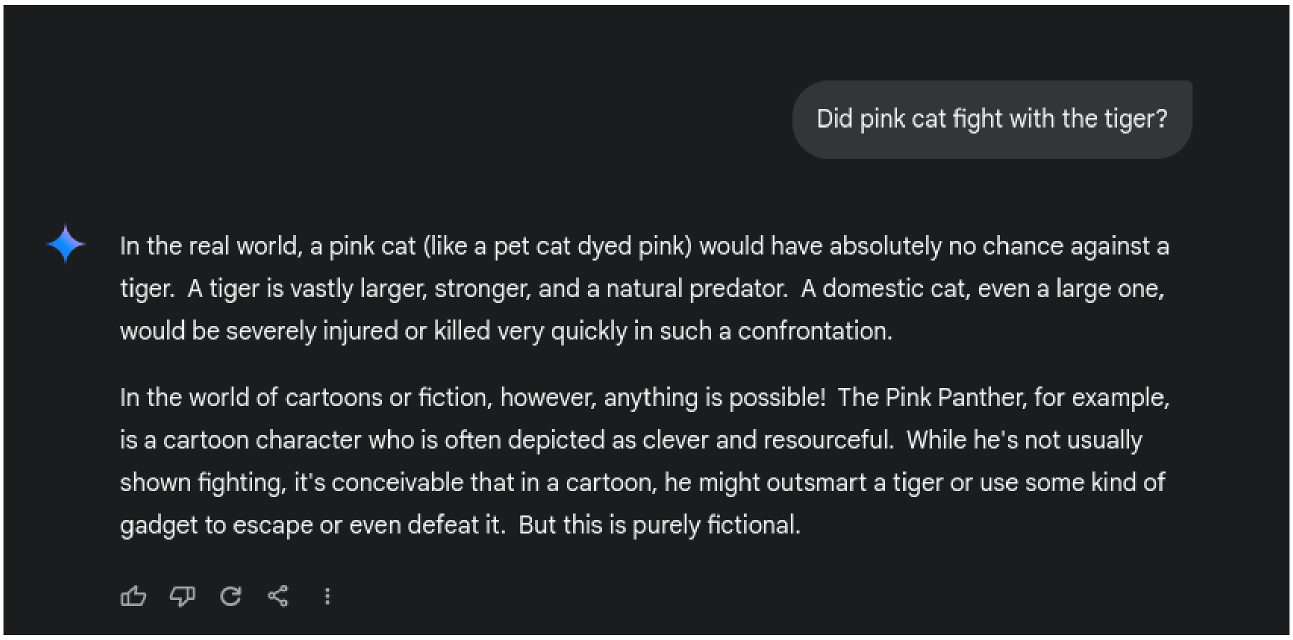
Techniques For Better RAG

1. **Chunk Optimization**: Breaking documents into smaller, meaningful chunks helps improve retrieval efficiency and relevance, ensuring more focused context for the model.
2. **Query Transformation**: Rewriting or expanding the original query before retrieval improves the chances of fetching the most relevant context.
3. **Context Selection**: Choosing the most relevant context from retrieved documents ensures the generated response is accurate and focused on the query.
4. **Iterative Retrieval**: Continuously refining the retrieved context by repeating the retrieval process allows for a more precise and detailed response generation.
5. **Recursive Retrieval**: Using the model's output to refine the context retrieval iteratively helps ensure that each new query iteration improves the relevance of the retrieved documents.
6. **Adaptive Retrieval**: Adjusting the retrieval strategy dynamically based on query types or evolving context allows for better accuracy and responsiveness.
7. **Retrieval Fine-Tuning**: Fine-tuning the retrieval mechanism on specific tasks or domains improves the accuracy and relevance of the retrieved context.
8. **Generator Fine-Tuning**: Fine-tuning the generation model to better use retrieved context leads to more coherent and contextually accurate responses.
9. **Dual Fine-Tuning**: Fine-tuning both the retrieval and generation models together ensures that the context retrieval and response generation are aligned and optimized for each task.

Comparison

This simple architecture, constructed using paraphrase-MiniLM-L6-v2 from SentenceTransformers for embedding, FAISS for creating a vector store, and Gemini 1.5 Flash (Free Tier) as the generator LLM, is compared against Gemini Flash 2.0.

Gemini Flash 2.0


Did pink cat fight with the tiger?

In the real world, a pink cat (like a pet cat dyed pink) would have absolutely no chance against a tiger. A tiger is vastly larger, stronger, and a natural predator. A domestic cat, even a large one, would be severely injured or killed very quickly in such a confrontation.

In the world of cartoons or fiction, however, anything is possible! The Pink Panther, for example, is a cartoon character who is often depicted as clever and resourceful. While he's not usually shown fighting, it's conceivable that in a cartoon, he might outsmart a tiger or use some kind of gadget to escape or even defeat it. But this is purely fictional.

RAG

```
Query:
Did pink cat fight with the tiger?
```

```
Response: The provided text describes a pink cat named Tom, but makes no
mention of a tiger or any fights.  Therefore, the answer is no.
```

Gemini Flash 2.0 LLM Response: Gave a general real-world explanation about a cat vs. a tiger and then drifted into fiction, mentioning The Pink Panther, which was not part of the query.

RAG Response: Analyzed retrieved text, determined that no fight was mentioned, and directly answered **"No."**

Conclusion

Retrieval-Augmented Generation (RAG) enhances the accuracy and contextual relevance of LLM-generated responses by retrieving relevant information, augmenting it, and generating answers based on that context. Even a simple RAG implementation can produce more precise and relevant responses compared to an LLM operating alone. However, this is just a basic implementation with limited capabilities. There is significant room for improvement, as RAG has a broad scope for optimization and refinement. Hopefully, this provides a general understanding of how RAG works.