Sure, here are some examples of available embedder models, chunkers, and Pinecone settings that can be used to improve the accuracy and relevance of retrieved chunks in the RAG paradigm:

Embedder models:

BERT

RoBERTa

GPT-2

T5

XLNet

Chunkers:

Spacy

NLTK

Stanford CoreNLP

Gensim

Pinecone settings:

Number of dimensions for vector space

Distance metric (cosine, euclidean, etc.)

Indexing strategy (flat, hierarchical, etc.)

Query parameters (filtering, boosting, etc.)

These are just a few examples, and there are many more options available. It's important to experiment with different combinations of these techniques to find the best configuration for your specific use case.  
  
  
I cannot perform tasks on my own. however, i can provide you with a general outline of how to build a retriever function and test it on the benchmark:

load the benchmark data: start by loading the benchmark data, which consists of text and relevant questions and answers.

preprocess the text: preprocess the text by tokenizing it, removing stop words, and stemming or lemmatizing the words.

create the vector space: create a vector space by embedding the text using an embedder model. this will convert the text into a high-dimensional vector space where semantically similar texts are close to each other.

define the query: define the query by embedding the question using the same embedder model as in step 3.

calculate the similarity score: calculate the similarity score between the query vector and the vectors of the chunks in the vector space. this can be done using cosine similarity or another distance metric.

retrieve the top k chunks: retrieve the top k chunks with the highest similarity scores.

evaluate the performance: evaluate the performance of the retriever function using the evaluating tool provided. this will give you precision@k and mrr scores, which can be used to compare the performance of different configurations.

iterate and improve: iterate over steps 3-7, trying different combinations of embedder models, chunkers, and pinecone settings to find the best configuration for your specific use case.

by following these steps, you can build a retriever function and test it on the benchmark to evaluate its performance.

**Practical approach**

Selecting appropriate sentence embedding models:

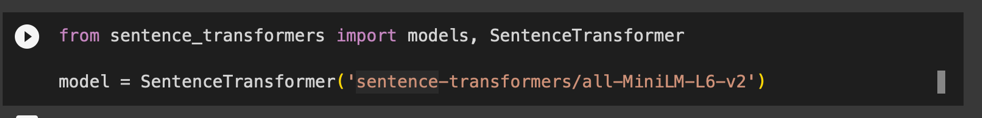
Model: all-mpnet-base-v2 with MRR@k : 0.786

A screen shot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

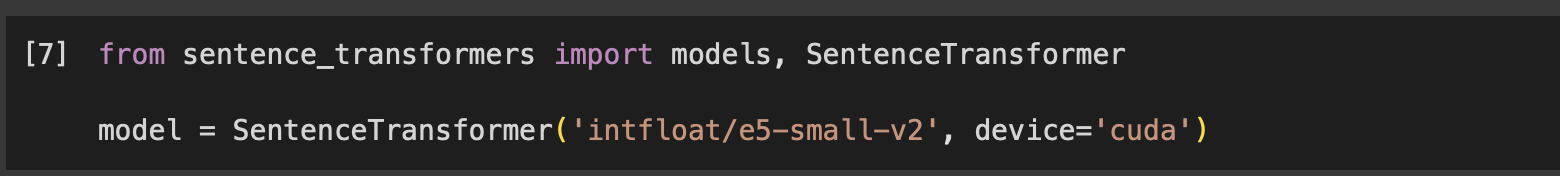
Model: all-MiniLM-L6-v2 with MRR@k : 0.77278



A screenshot of a computer code

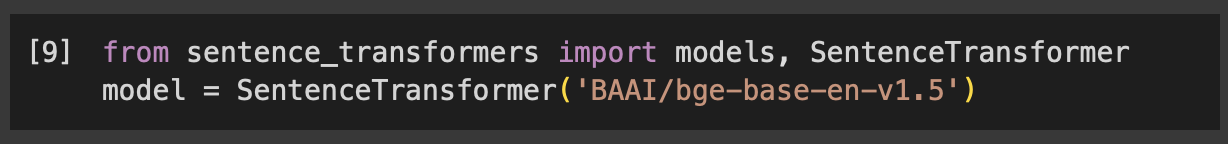
Description automatically generated

Model: e5-small-v2 MRR@k : 0.81243

A black rectangular object with white text

Description automatically generated

Model: BAAI/bge-base-en-v1.5 MRR@K : 0.792003



A screenshot of a computer

Description automatically generated

Model : all-distilroberta-v1 MRR@K: 0.7896441

A screen shot of a computer

Description automatically generated

A black and white text

Description automatically generated

Lets move forward with “e5-small-v2” as it performs much higher.

**Introduction:**

In the rapidly evolving landscape of information retrieval and natural language processing, the RAG (Retriever, Answerer, Generator) paradigm has emerged as a pivotal framework for bridging the gap between human language and machine understanding.

One of the fundamental challenges within this paradigm is the retrieval of relevant information—selecting the right chunks of data to feed into the language model (LLM), which acts as the answerer and generator. Traditionally, this retrieval process has heavily relied on cosine similarity as a means of identifying the most relevant text chunks, but this seemingly straightforward task is far from simple when we delve deeper into its intricacies.

In the conventional retriever approach, the process involves three key steps: first, embedding the text chunks into a vector space representation; second, embedding the user's query or question; and finally, ranking the text chunks based on their cosine similarity to the query vector, subsequently returning the top-K most similar chunks. However, the effectiveness of this method depends on a multitude of factors, each playing a crucial role in the overall success of the retrieval process.

This report delves into the intricacies of RAG-based information retrieval, exploring the challenges and opportunities associated with the retriever component. We investigate pivotal considerations such as the choice of text embedding methods, the strategy for chunking the text into manageable portions, and the evaluation of alternative metrics beyond cosine similarity. Moreover, we explore the concept of hybrid search, which combines traditional keyword-based search with semantic (vector-based) search, offering a more comprehensive approach to information retrieval.

**Objective of the POC**

The objective of this project is to optimize and enhance the retriever component within the RAG paradigm by conducting comprehensive research, experimentation, and performance evaluation. Specifically, the project aims to achieve the following goals:

1. Investigate Embedding Techniques: Explore a variety of open-source and paid embedding methods to represent text chunks effectively. Evaluate their performance in information retrieval tasks and identify the most suitable embedder for our retrieval system.
2. Chunking Strategy Development: Develop a robust text chunking strategy, which may include creating a customized chunking approach, to ensure that text segments are manageable and their sizes remain below 3,000 characters when tokenized with the Tiktoken tokenizer.
3. Hybrid Search Exploration: Investigate different configurations of hybrid search, which combines traditional keyword-based search with semantic (vector-based) search, to assess the potential advantages it offers in improving retrieval accuracy.
4. Retriever Function Implementation: Design and implement a retriever function that leverages the selected embedder, chunker, and hybrid search settings. The retriever should effectively retrieve relevant text chunks in response to user queries.
5. Connect to Evaluation Tool: Integrate the retriever with the provided evaluation tool, allowing the system to calculate precision@k and Mean Reciprocal Rank (MRR) metrics for performance assessment.
6. Optimization and Evaluation: Continuously iterate and fine-tune the retriever component by experimenting with various combinations of embedders, chunking methods, and hybrid search configurations. Evaluate the retriever's performance using the established evaluation metrics.
7. Utilize Pinecone for Vector Database: Use the Pinecone platform for managing the vector database, ensuring efficient storage and retrieval of vector representations. Register for a free Pinecone account to facilitate testing and evaluation.

Dataset we used to train the embedding models:

**SQUAD V2**: Stanford Question Answering Dataset (SQuAD) is a reading comprehension dataset, consisting of questions posed by crowdworkers on a set of Wikipedia articles,

**Data structure:**

{

"answers": {

"answer\_start": [94, 87, 94, 94],

"text": ["10th and 11th centuries", "in the 10th and 11th centuries", "10th and 11th centuries", "10th and 11th centuries"]

},

"context": "\"The Normans (Norman: Nourmands; French: Normands; Latin: Normanni) were the people who in the 10th and 11th centuries gave thei...",

"id": "56ddde6b9a695914005b9629",

"question": "When were the Normans in Normandy?",

"title": "Normans"

}

Train: Validation split=130319:11873

**1)Investigate Embedding Techniques**

**Objective**: Explore a variety of open-source and paid embedding methods to represent text chunks effectively. Evaluate their performance in information retrieval tasks and identify the most suitable embedder for our retrieval system.

**Details**: To effectively retrieve relevant information, the choice of embedding technique is paramount. Various embedding models, both open-source and paid, offer diverse capabilities for representing text in vector space. As part of this project, we will research and experiment with a range of these embedding methods, considering their suitability for the task. For this task we mainly preferred to use pretrained word embedding models that are available in Hugging Face. We were able to train

The types of text embedding models used:

1. **all-mpnet-base-v2 :** This is a sentence-transformers model: It maps sentences & paragraphs to a 768 dimensional dense vector space and can be used for tasks like clustering or semantic search. We trained this model on SQUADv2 train split and was able to benchmark a **mAP@K score of 0.786**, where @K is 100 by default.
2. **all-MiniLM-L6-v2**: This is a sentence-transformers model: It maps sentences & paragraphs to a 384 dimensional dense vector space and can be used for tasks like clustering or semantic search. We trained this model on SQUADv2 train split and was able to benchmark a **mAP@K score of 0.772**, where @K is 100 by default.
3. **E5 Small V2**: Text Embeddings by Weakly-Supervised Contrastive Pre-training.. We trained this model on SQUADv2 train split and was able to benchmark a **mAP@K score of 0.812**, where @K is 100 by default.
4. **bge-base-en-v1.5** : FlagEmbedding can map any text to a low-dimensional dense vector which can be used for tasks like retrieval, classification, clustering, or semantic search. And it also can be used in vector databases for LLMs. We trained this model on SQUADv2 train split and was able to benchmark a **mAP@K score of 0.792**, where @K is 100 by default.
5. **all-distilroberta-v1**: This is a sentence-transformers model: It maps sentences & paragraphs to a 768 dimensional dense vector space and can be used for tasks like clustering or semantic search. We trained this model on SQUADv2 train split and was able to benchmark a **mAP@K score of 0.789**, where @K is 100 by default.

**2)An overview of the model training:**  
  
The initial step in our process involves loading the SQUAD V2 dataset and selecting the E5 Small V2 model. The crux of model training revolves around encoding both the question and context into a shared vector space, with the dataset formatted to include questions and corresponding contexts.

An essential observation pertains to the nature of the dataset, which exclusively comprises positive questions and their corresponding contexts—indicating a high positive similarity between them, with no dissimilar pairs. Given this specific scenario, we opt for the Multiple Negative Rating loss function.

To mitigate the risk of overfitting, our model training is confined to a single epoch. Additionally, we incorporate a learning rate warm-up during training, set at 10%, as a higher value would predispose the model to overfitting.

With the training phase concluded, the subsequent step involves encoding the validation split of the dataset into embeddings and uskirting them to a vector database—utilizing Pinecone for this purpose. During this process, we exclusively consider non-duplicate values of the context component, encoding them into the database.

During the uskirting process, an index is established within our Pinecone account, and the corresponding API key is obtained. Leveraging this key, we proceed to uplink the model-embedded context into the newly created index.

Upon completion of these steps, we evaluate the performance of our retrieval model. To achieve this, we formulate a question, encode it using our trained model, and leverage this encoding to retrieve the pertinent context from the vector database. The accompanying screenshots provide a visual representation of the efficacy of our retrieval model.