

Exercise Sheet 2: Dense IR & RAG

Relevant code: RAG: CO5 repo [LINK](#)

This exercise focuses on building a **semantic search engine** to retrieve the most relevant passages from **SQuAD** given a user question. You will work through two packs of exercises:

1. Indexing and index evaluation
 2. Question Answering through RAG and RAG evaluation.
-

Overview

The Stanford Question Answering Dataset (SQuAD) is a collection of nearly 100K question-answer pairs with supporting passages curated by Mechanical Turk annotators. It is a popular benchmark for general question-answering systems.

In this exercise, we:

1. Preprocess the SQuAD dataset
 2. Create a dense index of the passages
 3. Test the accuracy of the index using SQuAD questions
 4. Construct a Retrieval-Augmented Generation (RAG) QA system
-

Step 1 — Preprocess the SQuAD dataset

Retrieve the **SQuAD Train dataset** (`train-00000-of-00001.parquet`) from the HuggingFace repo:
https://huggingface.co/datasetsrajpurkar/squad_v2/tree/main

Place it inside the **squad** subfolder.

From the dataset, extract:

1. A folder containing all passages (used for indexing).
2. A dictionary mapping each question to the passages using which it can be answered.
 - Keep only questions with more than two passages so that micro (per-question) F1 score can be computed.

The script `preprocess_dataset.py` (inside the `squad` folder) performs these tasks. Ensure you understand what the code does.

Step 2 — Index SQuAD Passages Using ChromaDB

ChromaDB is a vector database with an embedded retriever that supports seamless indexing and retrieval.

Before embedding SQuAD passages into vectors and loading them into ChromaDB, passages must be **loaded and chunked** into appropriate sizes depending on the embedding model.

Check the pretrained embedding leaderboard:

<https://huggingface.co/spaces/mteb/leaderboard>

to evaluate embedding attributes and choose one appropriate for your hardware.

You may begin with the default values in `I_constants.py`.

The script `II_index.py` performs the indexing. Ensure you understand what the code does.

Step 3 — Evaluate and Improve the Index Performance

Once indexing is complete, evaluate how well the index performs in practice.

Use the mapping file created earlier (`squad_multiple_contexts.json`) as test data:

1. For each question, query it to retrieve the top passage.
2. Compare the retrieved passage ID with the true passage IDs in the dictionary.
3. Compute F1 score as described in the lecture slides.

The script `II_index_test.py` provides baseline code.

You must write a script to compute the F1 score.

Experiment with Hyperparameters

Now that you have a quantitative metrics (F1 score) for performance, it is easy to check different settings and see if/how they improve the performance. You can check:

- **Chunk size**
- **Chunk overlap**
- **Embedding type**
- **K_source_chunks**

Other possible improvements:

- **Language detection** — skip documents in irrelevant languages (look for language detection libraries in GitHub; there are many)
 - **Multiprocessing** — speed up indexing for large datasets
 - **Hybrid + Dense indexing** (blog: <https://towardsdatascience.com/dance-between-dense-and-sparse-embeddings-enabling-hybrid-search-in-langchain-milvus-7c8de54dda24/>)
 - **Index more formats** (HTML, CSV, TXT, PDF, DOC, PPT, etc.)
 - **Incremental indexing** Begin where you end earlier when indexing
-

Step 4 — Question Answering with RAG

Use `III_RAG_UI.py` to define a RAG system. It includes:

1. An LLM model and its tokenizer
2. A HuggingFacePipeline for constructing a generator LLM
3. Loading the index and creating a retriever
4. A basic RAG prompt
5. Connecting the retriever and generator
6. A UI (optional)

Additional RAG Improvements

Try asking questions and subjectively assessing the quality of provided answers. Improve your RAG system by tuning:

- **Top-K vs Top-P and Temperature**
 - **K_source_chunks**
 - **Base LLM choice**
-

Is there a better way to assess the impact of tuning rather than subjectively inspecting the result? We discuss this tomorrow when we talk about generative evaluation.

Congratulations

You developed a cutting edge semantic search engine employing RAG methodology. Although RAG and its variants (e.g., graph RAG, agentic RAG, ...) systems are still pretty young technologies, due to their simplicity, they are highly popular both in research and business for many use cases.