RAG For BGU Report

Eyal Stolov

1 Design Choices

1.1 Open-Source Generative Model

For the open-source model, I decided to use the state of the art model ”Deepseek-R1” with 32B parameters, which was released just last month, and is the best open-source model by far right now. It has reasoning capabilities, and, if we look at the benchmarks on the right, we can see its performance is on par with OpenAI-o1-mini on most tasks.

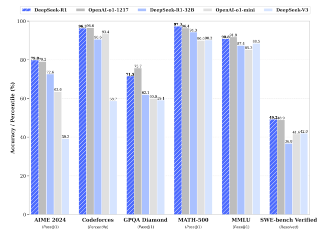

Figure 1: Deepseek-R1 Benchmarks.

1.2 Vector Store

I decided to use ChromaDB, because, ChromaDB is ideal for building RAG **fast**, it automatically handles document storage, metadata, and embeddings, eliminating the need for external databases. It comes with built-in persistence, cosine similarity search, and filtering, making it easy to retrieve relevant context without extra setup, which was perfect for this time-limited task.

1.3 Embedding Model Comparison

To choose an embedding model, I did a dense retrieval model analysis between 3 different embeddings models:

1. sentence-transformers/all-mpnet-base-v2 - A widely used embedding model, size - 109M params.
2. FacebookAI/xlm-roberta-large - One of the most downloaded embedding models in Hugging Face, this model was trained on more than 100 languages, including Hebrew, making it a good candidate, size - 561M params.
3. dicta-il/dictabert - An embedding model specifically crafted for Hebrew, size - 184M params.

To choose a model, I started by generating 127 questions using Chat-GPT O1 reasoning model (NOTE - in an ideal world, this task would have been done by a human, but because of time constraints, I let a LLM do it), each question has a corresponding relevant pdf which the question was generated upon. I then used 3 different metrics to assess all the models:

1. Top_k_indexes - I retrieve the top 10 documents for each question, and count how much times out of the 127 questions the model found at least 1 document that came from the source document the question was about (a bit similar to Recall@K).
2. Average Top_k_indexes - I retrieve the top 10 documents for each question, and count, on average, how many documents overall across all questions were found, where the document is of the source document the question was about (average Recall@K acroos all questions), see figure 2.

3. Success_amount - I retrieve the top 10 documents and let Chat-GPT O1 answer each question based on the context. I then count up how many times Chat-GPT was able to produce an answer (NOTE - in an ideal world, this task would have been done by a human, but because of time constraints, I let a LLM do it).

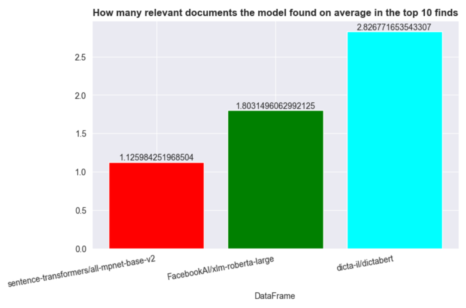

Figure 2: Average Recall@K

I choose those metrics because, the first and second metrics help us understand if the search is helping us retrieve relevant documents, and last metric helps us understand if the documents have sufficient information for the model to answer the question (To see the full graphs go to retrieval_performence_report.ipynb). Acroos all benchmarks, it was clear that **dict-il/dictabert** is the best embedding model.

1.4 Retrival Methods

To create a hybrid retrieval method, I first did an analysis on a dense retrival method using cosine similarity (best for semantic meaning), and found that this retrieval method is finding relevant information 109/127 times for the questions (see figure 2). I then did analysis on a lexical retrieval method called BM25, and saw that out of 13 pdfs, BM25 ranks the relevant pdf in the top 7, for 96% of questions. With this information, I decided to create a new hybrid retrieval method, where I first use BM25 to search relevant pdfs, I then filter the top 7, and use dense retrieval only on those pdfs. The result was almost the same for top_k_indexes, but for average top_k_indexes there was a 10% increase (see retrieval_performence_report.ipynb for more information).

2 System Improvements Suggestions

1. If I had access to more computing power, I would use a parameter-wise bigger generative model, like deepseek-R1 671B parameters version.
2. As mentioned in the overall performance section, I would move focus on the retrieval part, and try more comparing more retrieval methods.
3. Try using a tokenizer crafted specifically for Hebrew, like the following one: https://github.com/amir-zeldes/RFTokenizer
4. If I had access to more computing power, I would try a parameter-wise bigger embedding model, there are some good options provided here: https://huggingface.co/spaces/mteb/leaderboard

5. Most generative models & embedding models are better in English, so I would try translating the pdfs to English, and only then embedding them. On infrence I would translate the questions to english using a LLM for better retrieval. See the following link for more information - https://towardsdatascience.com/exploring-rag-applications-across-languages-conversing-with-the-mishnah-16615c30f780/