## Your names

Roee Ezer 312493281, Aviv Weidenfeld 315589937

## A description of your system and the choices you made. We should be able to understand how your system works in all the different parts, and especially the key choices and key points that are required to replicate it. If something worked particularly well, specify it. Structure this part with a dedicated section for each of the stages: pre-processing, indexing (sparse and dense), retrieval, RAG (unconstrained and budget-constrained).

## Our RAG system is composed of 4 components (each has its interface and implementations):

**pre\_proccessor** – process the html pages to WebTextUnit (interface).

Here we decided that each WebTextUnit will represent a paragraph or a “section” as defined in the script extract\_content.py and each section will contain the id of the doc and the id of the section (just the index of the section inside the doc) and the full index of the section will be:

self.doc\_id + "\_" + self.section\_id

This approach worked well for us and we didn’t try to change it except for a caching mechanism (pickle serialization) to avoid processing the html files in each run.

**index\_optimizers –** these optimizers manipulate the query or the documents or both to improve the indexing performance.

The changes made to the optimizers are relevant only to the indexing part. The “final\_answer\_retriever” for example will use the original query/documents.

We tried many kinds of optimizers as you can see in the folder components/IndexOptimizer. The one that worked the best was prefix\_suffix\_splitter\_optimizer.py which was our implementation of lemmatization with a few rules of thumb (splitting known Hebrew prefixes and suffixes) that achieved very fast and good results in the recall,mrr metrics with BM25 indexer.

**data\_indexer** – the **data indexer** processes WebTextUnit objects by organizing their content into a searchable structure. It allows efficient retrieval of relevant documents by matching query terms to the indexed data and returning the most relevant results.

The best indexer approach that worked well for us was combining two different indexers: BM25 (with prefix\_suffix\_splitter\_optimizer) and document embeddings similarity. This hybrid strategy allowed us to leverage both traditional keyword matching (BM25) and semantic understanding via sentence embeddings. By combining these two techniques, we could cover more ground and address cases where one method alone would fail. We observed that when a document retrieval went astray, the error types were different, and using both indexers allowed us to mitigate such discrepancies. This combination improved the robustness and recall, leading to more accurate retrieval and ultimately better overall results.

improved recall and MRR scores, ensuring fewer “no-rank” scenarios and more accurate retrieval of the correct documents.

**final\_answer\_retriever**

**This class has a one implementation that sends Gemini through the free tier API the query and the text to answer in the following pattern:**

**“**

**תקרא את השאלה הבאה:**

**{Query}**

**תענה בקצרה על השאלה לפי הטקסט הבא:**

**{AnswerSource}**

**“**

**We Assumed here that we need to answer the question only on based of the web pages data and never get an answer from the LLM without a source.**

## The evaluation results you obtained.

* 1. End-to-end evaluation, in which you go from query to final answer. This will have to be done manually. Choose 10-20 queries and manually evaluate their answers. Beyond accuracy, see if you can find common trends.

**We will split this question to 3:**

**Queries with low rank (1-3):**

**These queries’ final answer is usually good and sometimes more elaborative then the one in the gold document as expected – because it contains data from more documents.**

**Although there are some cases where the rank is misleading because we did retrieve the wrong section of the correct document like in this query "איזה תושבים זכאים לגמילה מסמים באשפוז?" in which we retrieved only section number 1 without section number 3 which seem more relevant.**

**Queries with high rank (10+):**

**These queries acted similarly to the queries with low rank. Most of the final answer were accurate.**

**Queries no rank (gold document was not retrieved):**

**The final answers here do not answer the question according to the specified text and sometimes returns that there is no answer in the text to the question.**

**In some cases, the reason the document is not found is because the query seem to be wrongly labeled like in here:**

**"gold\_doc\_id": "7cfab80d",**

**"query": "האם גם בן הזוג יכול לקבל חופשת לידה?",**

**In other cases it was because the question was very general and there were a lot of semi-relevant resources that included relevant to context words like here:**

**“חבר שלי סובל מליקוי ראייה ומתקשה בלימודים, יש לו אולי זכויות שניתן להקל עליו?”**

* 1. Additionally, see if you can identify cases where the retrieval found the correct document but the overall system produced a wrong answer, or the other way around: cases where the retrieval failed but the system overall produced an adequate answer.

**Yes, in this query which was probably wrongly labeled with the wrong gold document id, our rag system produced the correct answer according to the correct document 228fae0f**

**"gold\_doc\_id": "7cfab80d",**

**"query": "האם גם בן הזוג יכול לקבל חופשת לידה?",**

**"rank": null,**

**"final\_answer": "כן, בן הזוג יכול לקבל חופשת לידה בתנאים מסוימים, כגון: החלפת היולדת לאחר 6 שבועות, שבוע במקביל ליולדת (בהסכמתה בכתב), במקרה של לידה מרובת עוברים או אם היולדת אינה מסוגלת לטפל בילד.\n",**

* 1. Do things change from the unconstrained to the token-budget constrained cases?

**Yes, the 1000 tokens constraint didn’t affect most queries but it did affect some of the higher rank queries. For example this one:**

**"gold\_doc\_id": "7b3779b0",**

**"query": "האם מותר להעסיק אישה בהיריון בחודש השישי ביום שבת?",**

**"rank": 8,**

            "final\_answer": "אסור להעסיק אישה בהיריון בחודש השישי ביום המנוחה השבועי שלה (שבת במקרה זה), אלא אם כן קיימים תנאים מיוחדים: הסכמתה בכתב, אישור רפואי של רופא מומחה ביילוד ובגניקולוגיה שאין מניעה, והעסקה בהתאם לתנאים המפורטים באישור הרפואי.\n",

            "final\_answerconstrained": "הטקסט אינו מתייחס ישירות לשאלה האם מותר להעסיק אישה בהיריון בחודש השישי ביום שבת.  הוא מזכיר איסור אפליה על רקע היריון ומגבילות על עבודה בשבת (מנוחה שבועית), אך לא מספק תשובה ישירה לשאלה הספציפית.\n",

* 1. Retrieval Evaluation, here, we evaluate the system on its ability to find the correct page (ignoring the RAG part). Following part 1, we ask you to use two metrics: The first metric is recall@k which measures how many times the correct page was within the top-k retrieved documents (after reranking, if you do them). Use k=5 and k=20. The second metric is MRR.
     1. Dev-set

**"recall\_20": 0.6,**

**"recall\_5": 0.55,**

**"mmr": 0.34,**

* + 1. Test-set

**"recall\_20": 0.94,**

**"recall\_5": 0.81,**

**"mmr": 0.66,**

## Can you think of ways to perform non-manual end-to-end evaluation, given the data you created? Note that we asked you to also include an annotated text-span containing the answer. Can you use this for automatic evaluation of the end-to-end RAG system? How? Discuss this in your report.

## To perform a non-manual end-to-end evaluation, we can compare the predicted answer spans with the annotated answer spans in the dataset. By calculating metrics such as **Exact Match (EM)** and **F1-score**, we can automatically evaluate the accuracy of the retrieved answers. Additionally, the system’s recall and precision can be measured by checking how often the correct spans are retrieved. Using the annotated spans ensures consistent and objective assessment. This approach allows efficient evaluation of the entire RAG pipeline without requiring manual verification.

## If you experimented with different methods and ideas, it is good to describe not only what worked, but also what didn’t work.

\*In general we focused on the queries who didn’t get a rank (the gold document was not retrieved) and from the instructions suggestions to get inspirations for improvements.

We experimented several ideas with the following components:

**Optimizers**

1. lemmatize\_optim\_trankit.py This uses the Trankit pipeline to lemmatize Hebrew texts. It concatenates the input texts with a [SEP] marker, lemmatizes the combined text, and then splits it back based on the marker. It supports both CPU and GPU for efficient processing.
2. lemmatize\_optim\_bert.py This uses the DictaBERT model for semantic lemmatization. It tokenizes the input, processes it with dicta-il/dictabert-lex, and converts the output into lemmatized sentences. The script supports GPU acceleration for faster performance.
3. prefix\_suffix\_splitter\_optimizer.py – this is very simple try to achieve something that is similar to lemmatization using a few rules of thumb for and splitting common Hebrew prefixes and suffixes. For example if we see the word “הדרכים” we will add the following forms to the text “הדרכים דרכים דרך הדרך”. We did the transformation to both the query and the documents text
4. hyde\_indexing\_optimizer.py – this optimizer implements the suggestion from the instruction “use an LLM to generate an answer, and then use the answer as the query”. This optimizer made the recall,mrr metrics non-deterministic because the gemini LLM doesn’t return the same answer to the query each time. We tried both with and without the original query and it in most runs made our metrics worse
5. word\_filtering\_indexing\_optimizer.py – this filters a constant list of words that seemed irrelevant to the query searching like “אבל”, “מה”, “האם”. In some cases combined with prefix\_suffix\_splitter\_optimizer it increased a bit the recall and decreased a bit the MRR. Eventually we decided not to use it.
6. synonym\_encrichment\_optimizer.py – This optimizer add for each word a list of synonym words using alephbert-base model. The synonym words are not good in quality and this optimizer did not improve our metrics.

**Indexers**

The data indexing process was tested using two types of indexers:

* BM25s – A probabilistic ranking function based on term frequency (TF) and inverse document frequency (IDF).
* Sentence Embeddings – Encodes sentences into dense vector representations to capture semantic similarity.

In the final approach, we selected all sources but alternated the ranking: one result from BM25s, one from sentence embeddings, and so on.

## What would you have implemented if you had unlimited time and compute resources? What would be an ideal approach that you think would work very well? Be specific, e.g. don’t say general things like “I will train my own similarity model” but describe on what kind of data you will train it.

The main challenge of the rag system is the indexing and retrieve components. To improve this component we believe that good quality Lemmatization and synonym enrichment process can improve the indexing and retrieval significantly.

We could also train new retrieval following those steps

1. We would build a dedicated Hebrew QA dataset with tens of thousands of question–answer pairs, focusing on legal and social service domains.
2. We would develop advanced morphological analysis and lemmatization tools to handle Hebrew prefixes, suffixes, and inflections accurately.
3. We would fine-tune a large Hebrew generative model on these QA pairs, ensuring domain-specific coverage.
4. We would combine sparse (BM25) and dense (bi-encoder) retrieval, re-ranking via a cross-encoder, and feed the most relevant chunks into the generative model.
5. We would add a continuous improvement loop: whenever the system fails, new labeled data would be added to retrain both retrievers and the model.

By building such a pipeline—end to end and domain-aware—we would ensure minimal hallucinations, high retrieval precision, and robust, explainable answers.

## Any additional thoughts you had based on the assignment, or ideas you may have.

Usually looking and the queries with no rank (the golden document was not retrieved) was the best place to think about ideas to improve. We notice that in some cases special characters interfere with the index and retrieval process.

## A discussion of the challenges of part 2 compared to part 1, what are their causes, and what do you think is needed to overcome them.

**Hebrew Morphological Complexity**:  
Hebrew's rich morphology, with prefixes, suffixes, and root-based word forms, posed a challenge for the BM25s indexer, which relies on exact term matches and struggled with these variations. We overcame this by using lemmatization to normalize the words to their base forms and removing prefixes and suffixes. This allowed the BM25s indexer to handle the morphological complexity more effectively and improve retrieval performance.

**Lack of High-Quality Pretrained Models**:

There are fewer high-performance, pretrained models for Hebrew compared to English, affecting sentence embedding quality. To address this, we suggest fine-tuning on Hebrew question-answering datasets.

The approach involves embedding both questions and answers and maximizing their similarity during training. Negative sampling, by adding irrelevant pairs, can further improve the model's ability to differentiate relevant from irrelevant answer*s.*