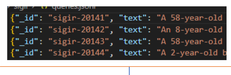
Raw Patient file (quires.jsonl) : json line format
{**_id**: "Patient id", **text**: " Containing patient case"}

Sample:



STEP 1 – KEYWORD GENERATION

Using gpt-4-turbo, user prompt given to generate upto 32 keywords from the patient raw case study.

Clinical trial raw file (trial_info.json) : json format, extracted and parsed through clinicaltrial.gov
{"NCT_ID": {**brief_title**: "The title of the study ",
phase: " phase the trial is in",
drugs: str(list()) (single string),
drugs_list: ["drug1", "drug2", ... , "drugn"]
diseases: str(list()) (single string)
diseases_list: ["disease1", "disease2", ... , "diseasen"]
enrollment: str(number of people enrolled)
inclusion_criteria: "inclusion criteria having \n\n as a separator"
exclusion_criteria: "exclusion criteria having \n\n as a separator"
brief_summary: "summary of the trial if given"

Query Patient file (id2quires.json) : json format
{"**Patient id**": {**raw**: "patient case study",
model_name:{
summary: "patient summary by model used",
conditions: ["C1", "C2",..., "Cn"] (list of keywords extracted from patient upto 32 keywords)
}
}
}

Clinical trial query file (corpus.jsonl) : json line format, processed trials information for efficient retrieval
{_id: "NCT_ID", title: "brief_title of the study", text: " brief_summary + inclusion + exclusion", metadata: "all info present in raw file" }

Sample:

RETRIVAL PROCESS (ALL TRIALS RELEVANT TO PATIENT)

Input: each keyword of the given patient.

Query conversion to vectors:
MedCPT-Query-Encoder model. Used for vectorization of smaller texts.
- Step 1 is to tokenize text using "AutoTokenizer"
- Step 2 is to convert the tokens into embeddings using "AutoModel"

Ouput: a pytorch tensor of length 256.

Input: The "text" field of the trials only.

Search algorithm: Hybrid search

Trial query conversion to sparse vectors:
BM25 ranking model - BM25Okapi, stored as tokenized json format.

Trial query conversion to dense vectors:
MedCPT-Article-Encoder model. Used for vectorization of large texts.
- Step 1 is to tokenize text using "AutoTokenizer"
- Step 2 is to convert the tokens into embeddings using "AutoModel"

Ouput: a pytorch tensor of length 512.

Storage : "Faiss" for MedCPT embeddings.

Input parameters :
K = int() "different k for fusion",
bm25_wt = int() "bm25 weight (the weight used here was 1),
medcpt_wt = int() "MedCPT weight (the weight used here was 1),
N = int() "top k matches to be retrieved for each query and at final aggregation".

BM25 search output: Top N list of NCT_ids for every key words for a patient based on ranking.
MedCPT search output: Top N list of NCT_ids for a every key words for a patient based on embedding match.

Calculation of BM25 and MedCPT scores:
Scoring function = (1/(rank+K))(1/(combination_index+1))

Combination of BM25 and MedCPT scores:
The score for a NCT_ID is 0 at start. Considering a keyword match with NCT1 in BM25 scoring 0.5 and the same keyword is matched with NCT1 in MedCPT retrieval giving the score 0.2. Thus, a final score for NCT1 given the keyword hence the patient is 0.7 due to maximization problem.

Output:
A json format, each patient having Top_N NCT_IDs based on scoring system in list: qid2nctids_results_{model_name}_{corpus}_{k}_{bm25_wt}_{medcpt_wt}_{N}.json
{"**P1**" : ["NCT1", "NCT2", ..., "NCT{N}"], "**P2**": ["NCT1", "NCT2",..., "NCT{N}"], ..., "**Px**": ["NCT1", "NCT2", ..., "NCT{N}"]}

MATCHING PROCESS (PATIENT - CRITERION LEVEL SCORES)

Input: Processing of the output file from retrieval step as below

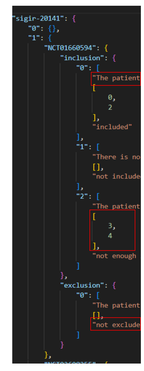
LEVERAGING LLM CAPABILITY (GPT-4):

System prompt:
- customized prompt with details based on inclusion/exclusion criteria given
- contains output details and format
- **output details:** for each criterion,
1. Reasoning process of the criterion w.r.t. patient.
2. List of indices of relevant sentences w.r.t to criteria found in patient case.
3. Label for the criterion based on the reasoning, {"not applicable", "not enough information", "included", "not included"}
- **output format:** dict{str(criterion_number): list[str(element_1_brief_reasoning), list[int(element_2_sentence_id)], str(element_3_eligibility_label)]}

Processing of the patient data:
Case study converted to a sentence_index format using sentence tokenizer and concatenated again to make whole of of patient study for input into the LLM.

User prompt:
The patient details : "Processed sentence indexed form"
Clinical trials: "trial criteria information based on the NCT_ID retrieved for the patient using the file in left"
Output: "how the output be represented, as a plain json"

Output: The predicted reasonings for patient-criterion level as json format.



RANKING PROCESS (PATIENT - TRIAL LEVEL SCORES)

LLM AGGREGATION SCORE (GPT-4):

System prompt:
- given the inputs, provide output of 2 scores.
1. Relevance Score (R)
2. Eligibility Score (E)

- **output details:** for each trial,
1. Provide an explanation for the relevance of the patient to the trial.
2. Provide a relevance score (R). R can be range from [0,100],
- If R = 0 the patient it totally irrelevant to the given trial
- If R = 100 the patient is exactly relevant to the given trial
3. Provide an explanation for the eligibility of the patient to the trial.
4. Provide a eligibility score (E). E can be range from [-R, R],
- If E = -R the patient is ineligible (not included by any inclusion criteria, or excluded by all exclusion criteria)
- If E = R the patient is is eligible (included by all inclusion criteria, and not excluded by any exclusion criteria)
- If E = 0 the patient is neutral (i.e., no relevant information for all inclusion and exclusion criteria)

-**output format:** Dict{"relevance_explanation": Str, "relevance_score_R": Float, "eligibility_explanation": Str, "eligibility_score_E": Float}

Processing of the patient data:
Case study converted to a sentence_index format using sentence tokenizer and concatenated again to make whole of of patient study for input into the LLM.

User prompt/inputs:
The patient details : "Processed sentence indexed form"
Clinical description: "trial tittle along with disease conditions and a brief summary of the trial"
Patient criteria predictions: "String format of the prediction json output generated in matching process"
Output: "how the output be represented, as a plain json"

LINEAR AGGRIGATION SCORE:

Matching score:
Based on labels of the predicted criterion the scores is being updated via a common match and penalty scoring system.

Aggregation score:
Aggregating the relevancy score and eligibility score to obtain a final aggregation score
aggregation_score = (R + E)/100

Final ranking of trial for a given patient:
Rank = match_score + aggregation_score

FINAL OUTPUT: