Medical Necessity Evaluation

Yash Agrawal, Doclens.ai Summer Intern 2025

Introduction

The Medical Necessity report is used to get quick insight into the differences between prescribed medical procedures (as presented in the Medical Summary that is generated based on information present in multiple documents of a Claim File) and the usual and customary medical procedures (Ground Truth) as an attempt to gauge the accuracy and necessity of procedures mentioned in the Medical Summary. Thus, the Medical Necessity report would clearly point out discrepancies and anomalies in the medical procedures specified in the documents corresponding to an insurance claim. For example, if someone has seasonal allergies, this report would point out if they decided to go for an x-ray of the lungs – a possibly excessive procedure as compared to going for the generic OTC & prescription medications.

This medical necessity report is designed to be only a suggestion of probable medical excessiveness and in no manner does it try to provide sound medical guidance, which almost always has to take into account prior medical history of patients as well as multiple different and equally valid treatment options, which might not be readily available either in the ground truth or the medical summary that are used to generate the medical necessity report.

The current method to generate this Medical Necessity report uses a plain LLM (Large Language Model) with no additional tools or access to the internet. LLMs are trained on a vast pool of publicly available data, including data available on the internet. MayoClinic is a trusted source of generic information about a large set of medical conditions and diseases and is often referred to by laymen to get a general understanding of treatment options of common medical conditions – *it is not meant to replace expert medical practitioners!* When required to generate the report, the LLM is prompted to use MayoClinic as the GT (ground truth) and base its inference and comparisons off of it. However, due to its non-deterministic nature, LLMs are prone to hallucinations leading to incorrect outputs. LLMs usually operate like a black box and it is practically impossible to understand its inner workings to figure out whether or not a response that it gives is grounded in factual data. The only way one can verify the outputs of an LLM is by testing a set of outputs, verifying them and hoping the similar trend continues. Additionally, most LLMs are accessed through APIs and the freshness and recency of the data on which the underlying LLM has been trained is not easy to ascertain. This can in turn cause LLMs to spit out responses based on outdated information.

Hence, we propose an experiment to figure out whether there is a better method to perform the task of generating a Medical Necessity report giving us an accurate medical necessity evaluation.

Hypothesis

Instead of asking the LLM directly about the usual and customary procedures for any medical condition, if the information is scraped directly from the source (such as MayoClinic) and that information is provided as context to an LLM. The LLM will be able to give responses that are more accurate and with practically no hallucination.

Note: For the sake of this experiment we consider the initial step of query generation and latter step of comparison and report generation to be error free, keeping it consistent throughout the different approaches.

Background

The ideal process of Medical Necessity report generation comprises obtaining:

1. MayoClinic's Treatment data for a disease - Relevant information from MayoClinic about the diseases
2. Medical Summary Document - Documents consisting of the medical actions taken by the persons involved. This is generated from the claim files that are uploaded.

These 2 are compared to generate a Medical Necessity Report which points out inaccurate/unnecessary diagnosis and treatment in the Medical Summary Document(2) considering MayoClinic's Treatment data(1) as the GT(ground truth).

The main process that this experiment aims to test is the one of retrieval of **relevant** information (treatment data of a disease) from the MayoClinic website. There are multiple ways this can be done:

1. As MayoClinic is a public repository of information, assume and hope that the big LLMs have been trained on this data and rely on its own knowledge base to get MayoClinic information
    a. This process completely relies on the LLMs ability to verify within itself whether the information it is providing is (i) MayoClinic's data (ii) Correct data within MayoClinic (iii) Latest information
    b. An LLM's training process and responses are information based, and not source(citation) based
    c. The LLM may paraphrase, misinterpret nuance and overlook details
    d. As there is no external check, there will be cases where the LLM provides some information presented as MayoClinic's information, but is actually influenced by its own knowledge base and other trained information
2. Give the LLM access to the internet via tools and allow it to search for the disease in MayoClinic's website for information
3. Give the LLM access to a local database with the MayoClinic information and use it as a knowledge base for the LLMs inference

4. Pass all the information directly in the context window of the LLMs prompt(currently impractical)
    a. Passing the entire MayoClinic's Database(~10MB) into the context window is as of now is not possible due to the strict context window limitations. It will be better to just pass the relevant information into the context.
    b. Even if we were to pass the entire dataset, LLMs have shown to lose context and hallucinate when provided with very big amounts of data. They either forget the task to accomplish, or forget the context they must operate under.
    c. Passing relevant information into the LLM and prompting the LLM to base its responses on this information is a good way to ground and regularise the information, minimising chances of hallucination and variability.

While option 1 may seem unreliable, effective prompting methods have proven to work thus far, producing appropriate results. However, this experiment is centered around option 3 where we extract all the information from MayoClinic website, store it locally and have the LLM interface with it. We propose 2 ways by which this interfacing can take place:

1. Embedding Vector Search
2. RAG(Retrieval-Augmented Generation)

Please refer to 'Proposed Approaches' for more details.

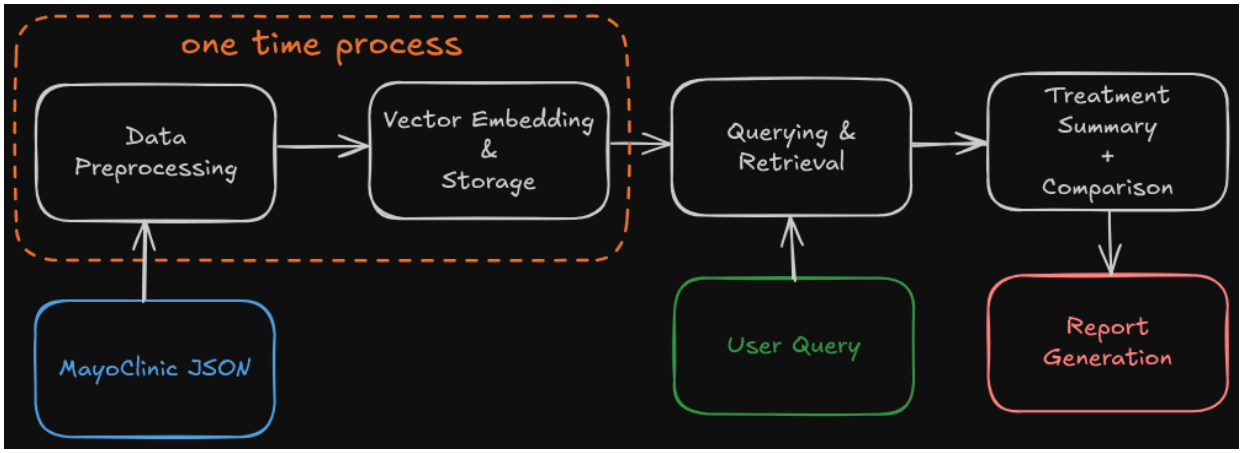
Once the retrieval of information is completed, we have with us 2 things that need to be compared:

1. LLMs response(relevant MayoClinic data) for the treatment of a disease
2. Medical Summary Document data

These 2 are compared and a Medical Necessity Report is produced which points out all the places in the Medical Summary Document(2) where there seems to be a mismatch with MayoClinic data(1). This report is the output of the experiment run.

To conduct this experiment, the first step is to scrape all the data from MayoClinic's website, which was completed successfully and stored in a JSON file. We will refer to this scraped data JSON as '*Mayo Clinic's Data*' in this report. Refer to 'Evaluation' for more details.

Proposed Approaches



High-level approach (Embedding vector search, RAG)

Approach 1: Semantic Vector Search

Vector embeddings are perfect to convert any form of data into numerical representations. For our use case, we will be embedding all the keys – the disease names (e.g., Diabetes, Abdominal Aortic…) and storing them in a vector database. This allows us to:

1. Capture semantic meaning, the number of dimensions in each embedding allows it to represent semantic meaning. While what each dimension represents is unknown, the specific position taken by a vector in this n-dimensional space will represent a unique meaning
2. Create a nuanced distribution of all the diseases, which means things like ‘Broken Toe’ will be closer to “Broken Limb” than “Diabetes” in the n-dimensional vector embedding space.
3. Perform similarity searches, which will let us arrive at keys that are exactly the same, or close to the meaning of the given query
4. Perform lexical searches, which will let us also perform lexical similarity searches, giving us word-to-word matches

This approach takes into account the semantic and lexical similarity when searching, yielding accurate results if the dataset consists of the data and probable results in cases where the dataset is lacking.

This approach consists of the following steps:

1. Data Preprocessing: Processing the information stored in the *Mayo Clinic’s Data* JSON, to a format such that the key is the disease name, and the returned data are the ‘overview’, ‘diagnosis’ and ‘treatment’ sections.
    a. The restructured data has the following schema:
    “disease_name”: {‘overview’: string, ‘diagnosis’: string, ‘treatment’: string}

e.g.,

```
{
   "A fib": {
       "overview": "Atrial fibrillation (AFib)...",
       "diagnosis": "You may not know you have...",
       "treatment": "The goals of atrial..."
   },
   "Abdominal aortic aneurysm": {
       "overview": "An abdominal aortic aneurysm ...",
       "diagnosis": "Abdominal aortic aneurysms ...",
       "treatment": "The goal of abdominal aortic..."
   },
```

2. Vector Embedding and Storage: The 'keys' or disease names are embedded and stored in a vector database.
3. Querying and Retrieval of Information: When an input/query (disease name or condition) is received, we embed this, and perform a hybrid similarity search in the vector database so we can get a semantically similar meaning *key*. This *key* is used to query and fetch information from *Mayo Clinic's Data* to get the exact and accurate MayoClinic information
4. This extracted information along with the medical summary text is passed to an LLM which is prompted to perform a detailed comparison. The key match, the information (treatment and diagnosis), the comparison output are all put into a report.

This same process is followed for any and all disease queries.

Approach 2: RAG Pipeline

This approach uses a RAG pipeline to get the relevant information and create the report. This consists of the following steps:

1. Data Preparation: To prepare the data for loading into the vector database, *Mayo Clinic's Data* is split, and made into *chunks*. Each chunk has prefixed to it, the disease names (along with aliases) and what information it is (treatment).

```
{

"https://www.mayoclinic.org/diseases-conditions/atrial-fibrillation/symptoms-causes/syc-20350624": {
       "names": [
           "A fib",
           "Atrial fibrillation"
       ],
       "overview": "Atrial fibrillation (AFib)...",
       "diagnosis": "You may not know you have ...",
       "treatment": "The goals of atrial fibrillati..."
   },
```

   Two variables require special attention here, the *chunk_size* and the *chunk_overlap*, which are responsible for the amount of information and the continuity of information respectively.

   Note: These 2 parameters can be further tuned to get better results.

e.g.,

```
all_docs[0]
[37]
Document(metadata={'disease': ['A fib', 'Atrial fibrillation'], 'section': 'diagnosis', 'chunk_id': 0}, page_content='A fib, Atrial
fibrillation - Diagnosis: You may not know you have atrial fibrillation (AF ib). The condition may be found when a health checkup is done
 for another reason. To diagnose AF ib, the health care provider examines you and asks questions about your medical history and symptoms.
 Tests may be done to look for conditions that can')

all_docs[1]
[38]
Document(metadata={'disease': ['A fib', 'Atrial fibrillation'], 'section': 'diagnosis', 'chunk_id': 1}, page_content='A fib, Atrial
fibrillation - Diagnosis: Tests may be done to look for conditions that can cause irregular heartbeats, such as heart disease or thyroid
disease. Tests Tests to diagnose atrial fibrillation (AF ib) may include: Blood tests. Blood tests are done to look for health conditions
 or substances that may affect the heart or')
```

Two documents showing how chunk overlap looks like

2. Vector Embedding and Storage: The chunks from step 1 are converted to LangChain Documents which are embedded and fed into the vector database.
3. Querying and Retrieval: A RAG pipeline is implemented which uses the vector database for similarity searches and fetches all the chunks. These chunks are passed to an LLM for summarisation and to get a final RAG pipeline output. The number of chunks retrieved can influence the output quality.
4. This RAG output along with the Medical Summary text is passed to an LLM which gives us the comparison, same as Approach 1. The RAG output and comparison response are all consolidated into a report.

The same process is followed for any and all diseases present in the medical summary.

Process

Tech Stack:

Scraper

BeautifulSoup4, requests(with time delay)

Embedding Search & RAG Pipeline

LangChain, Qdrant Cloud (Vector DB + Searching), ReportLab (PDF), scikit-learn
Embedding Models:

- Google Text Embedding model (text-embedding-004, used via Google AI Studio)
- AWS Titan Text Embeddings V2 (amazon.titan-embed-text-v2:0, via AWS Bedrock)
- Qdrant BM25(via Qdrant, huggingface)
- Hugging Face Embedding - PubMedBert (neuml/pubmedbert-base-embeddings, Local)
- Hugging Face Embedding - Bio_ClinicalBERT(emilyalsentzer/Bio_ClinicalBERT, Local)

LLM(Large Language Models):

- Google Gemini (models/gemini-2.5-flash, used via Google AI Studio)
- Amazon Nova Lite(amazon.nova-lite-v1:0, via AWS Bedrock)
- Mistral 7B(Local, via Ollama)

Scraper

MayoClinic Website structure:

1. Access Mayo Clinic's 'Diseases and Condition' page that is indexed by letter [A - Z]
    a. url: Medical Diseases & Conditions - Mayo Clinic
2. Every disease, is grouped by the first 2 letters here, and contains the following data:
    a. Disease name
    b. Alternative name(if any)
    c. Link to the website with all the data
    d. Sample url: https://www.mayoclinic.org/diseases-conditions/abdominal-aortic-aneurysm/symptoms-causes/syc-20350688
3. In some cases, you get redirected to an unexpected website, that's different from the normal, eg: https://www.mayoclinic.org/diseases-conditions/bicuspid-aortic-valve/cdc-20385577 from which we have to redirect ourselves to the 'Symptoms and Causes' page
4. At the 'Symptoms and Causes' page we can find the 'Overview' and also the link to the 'Diagnosis and Treatment' page
5. From the 'Diagnosis and Treatment' page we can get the 'Diagnosis' and 'Treatment'

Scraping Process:

1. Once at the 'Diseases and Condition', page we perform the following steps:
    a. Extract the URL of the 'Diagnosis and Treatment' webpage
    b. Extract a small portion of the Overview
2. Then we can crawl to the next URL with the 'Diagnosis and Treatment' where we perform the following steps:
    a. Extract the data on Diagnosis
    b. Extract the data on Treatment

The output is a JSON, e.g.,


```
{
   "A": {
       "link": "https://www.mayoclinic.org/diseases-conditions/index?letter=A",
       "diseases": {
           "A fib": {
               "link":
"https://www.mayoclinic.org/diseases-conditions/atrial-fibrillation/symptoms-causes/syc-20350624",
               "other_name": "Atrial fibrillation",
               "data": {
                   "overview": "Atrial fibrillation (AFib) is an irregular...",
                   "diagnosis": "You may not know you have atrial fibrillation (AFib)...",
                   "treatment": "The goals of atrial fibrillation treatment are..."
               }
           },
```

Note:

- The architecture of MayoClinic's data webpages have 2 different architectures, which are accounted for in the code.
- Few disease webpages don't include certain information in the diagnosis or treatment sections, because of which a few empty sections exist in the dataset. Missing Data:
    - 'Acne': ['diagnosis'],
    - 'Aneurysm, aortic': ['diagnosis', 'treatment'],
    - 'Aneurysms': ['diagnosis', 'treatment'],
    - 'Aortic aneurysm': ['diagnosis', 'treatment'],
    - 'Blackheads': ['diagnosis'],
    - 'Common variable immunodeficiency': ['diagnosis', 'treatment'],
    - 'Cramp, muscle': ['diagnosis'],
    - 'Crib death': ['diagnosis'],
    - 'CVID': ['diagnosis', 'treatment'],
    - 'Diaper rash': ['diagnosis'],
    - 'Jet lag disorder': ['diagnosis'],
    - 'Mosquito bites': ['treatment'],
    - 'Muscle cramp': ['diagnosis'],
    - 'Pimples': ['diagnosis'],
    - 'SIDS': ['diagnosis'],
    - 'Sudden infant death syndrome (SIDS)': ['diagnosis'],
    - 'Whiteheads': ['diagnosis']

Semantic Vector Search

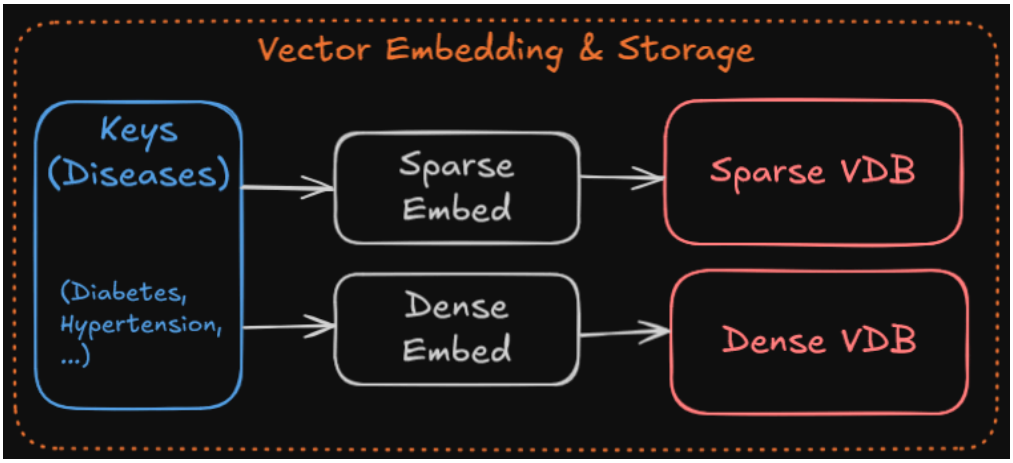
Choice of Embedding Model:

- Initial runs of the experiment were conducted using Google's text embedding model
- For ensuring compliance, an experiment was run with AWS Titan Text Embeddings V2, but the results failed to the match the accuracy of Google's embedding model
- A fine tuned model for medical terms was used(PubMedBert) on local machine which seemed to match Google's model's results
- Underlying concept:
    - We need a model that can correctly understand and embed medical terms
    - It should be able to keep semantically similar vectors close and the distance between these vectors should increase with reducing semantic similarity.

Process

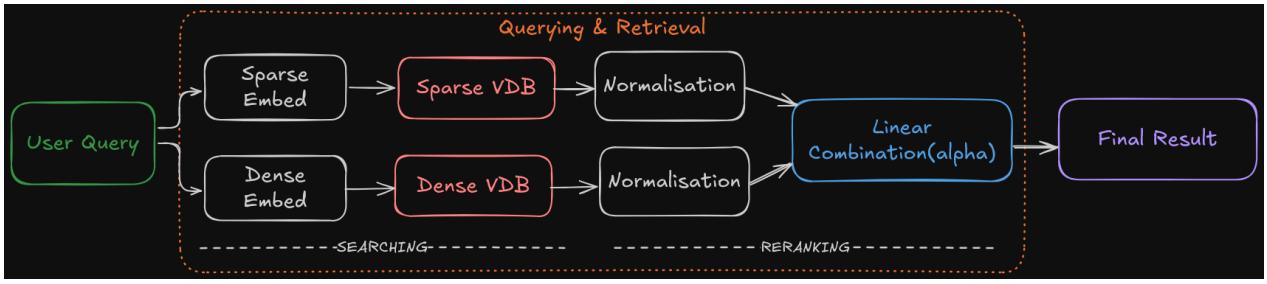
1. Key Embedding:
    a. Dense Embedding: Each key is encoded into a dense vector embedding. Performing similarity searches on dense embeddings gives us semantically similar search results

b. Sparse Embedding: Each key is also encoded into a sparse vector embedding (BM25 Qdrant). Performing similarity searches on sparse embeddings gives us keyword-level similar search results
c. Dense and Sparse Embedding:
    i. Dense Embeddings:
        - Short vectors with lesser dimensions(e.g., 384 or 768 dimensions)
        - Most values are non-zero
        - They capture the meaning or context of a word, sentence, or document.
    ii. Sparse Embeddings:
        - Long vectors with higher dimensions(e.g., 10,000+ dimensions).
        - Most values are zero.
        - They focus on exact words and how often they appear.



Embedding process for the keys(diseases)

2. The resulting vectors are stored in a vector database(Qdrant Vector Database). The dense and sparse vector embeddings have their own separate vector database collections
3. Query Processing and Retrieval:



Query processing for Embedded Vector Search

a. The user's input is embedded using the same dense and sparse embedding methods
b. A hybrid search is performed which involves performing 2 similarity searches:

        i. Sparse Similarity Search: The top k = 100 similar vectors are searched for within the sparse collection and returned along with their scores (usually returns less than 100 results)
        ii. Dense Similarity Search: The top k = 100 similar vectors are searched for within the dense collection and returned along with their scores
    c. We now have 2 sets of results (Sparse results and Dense results). The 2 sets of results from both the search results are normalised among themselves using z-score normalization
    d. We introduce a weighing variable alpha, ranging from 0-1 which is used give more or less weightage to the Sparse embedding search results, currently set at 0.9
        i. This alpha can be changed based on the embedding model used, the dataset, the query type
        ii. The alpha shows the weightage of sparse (lexical) embeddings on the final result
    e. Each disease that was found in both sets of results are then given a new score using the formula: *(alpha * (normalised Sparse score)) + ((1-alpha) * (normalised Dense score))*
    f. The scores are then sorted and the highest score match is returned. This is the key (disease name) that can be used to get other relevant information from *Mayo Clinic's Data.*

The reason behind settling for hybrid search is for cases when we can't find a good sparse match, we can get the most similar match in terms of meaning and use that as the ground truth. For cases where we do get a dense and sparse match, this specific result is boosted even more and given the highest priority. Using a hybrid search you can get more accurate, more relevant, and fewer missed results.

4. The relevant information is received from *Mayo Clinic's Data* and the 'treatment' information is passed to an LLM for summarisation and formatting into a readable format. This information is saved to the report under the treatment section of the disease
5. Comparison Segment:
    a. The treatment information(extracted from *Mayo Clinic's Data*, treated as ground truth) is compared to the Medical Summary Document and a comparison is performed
    b. This section is performed by an LLM and the output is a bunch of points of the type "x suggests/includes/mentions a, whereas/however/and y suggests/includes/mentions b".
    c. This is followed by a subsection with all the relevant information from the Summary Document that the LLM used for drawing comparisons

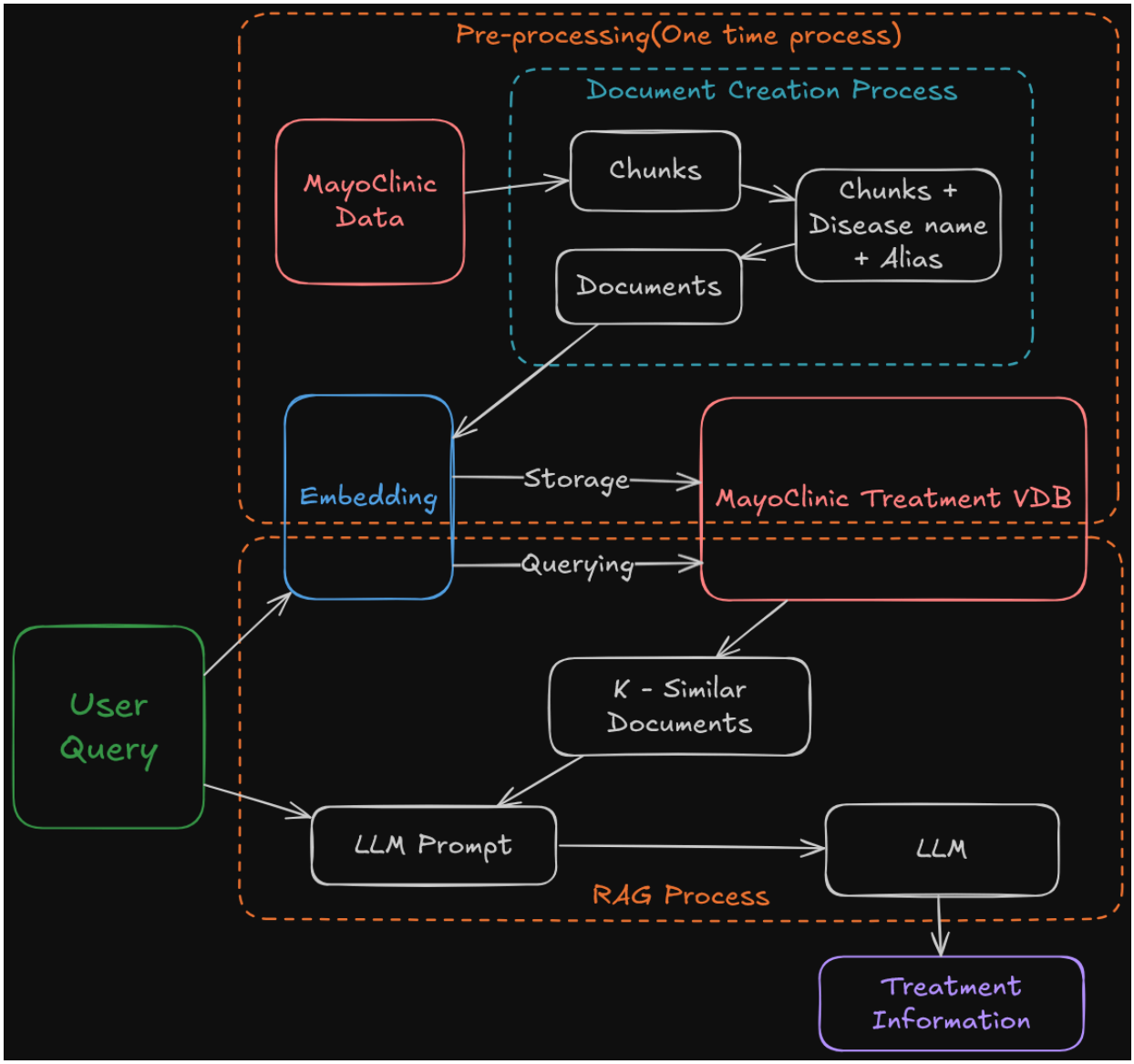
Problems to solve

- Embedding-based similarity search often fails to clarify in what way two items are similar. For example, "Broken Foot" may be close to "Broken Toe" (both fractures) and "Swollen Foot" (same body part), despite representing different types of similarity. This ambiguity can lead to misleading results

Sample report

[Peterson Transport - Vector Embedding Search Report.pdf]

RAG Pipeline



RAG Pipeline

Process

1. Data Preprocessing:
    a. Flatten and simplify the *Mayo Clinic's Data* to a format as 'disease_link': {names: [aliases], overview: "...", diagnosis: "...", treatment: "..."}
    b. The information is cleaned, formatted taking into account punctuation, capitalisation, taking into account unicode characters
2. Splitting and Chunking:
    a. The treatment information is split with 2 variables open for tuning, 'chunk_size'(currently at 300) and 'chunk_overlap'(currently at 50)
        i. Chunk size is the length of the information in each chunk
        ii. Chunk overlap is the length that 2 chunks overlap in page_content
    b. Each chunk has a prefix added to it. The prefix comprises 'diseases_name' and 'section_name'. diseases_name includes the names and the aliases for the diseases and section_name is 'treatment'.
3. Embedding: The chunks are embedded and stored in a vector database, quadrant vector database
    a. A retriever is initialised with a 'k' value, which is for the number of chunks the RAG pipeline will return(set to 15)
4. Searching and Retrieval:
    a. When a new user query is received, it is embedded and a similarity search is performed and the top k(metric for the retriever set in 3a) results are returned as 'context'
    b. The 'context' and the 'user query' are passed to an LLM that summarises and gives the final result. It is prompted to only use 'context' to answer 'question' and the output is returned, this is considered as the ground truth from *Mayo Clinic's Data*
    c. The data is put into the pdf in the treatment section
5. Comparison Segment: Similar to the Semantic Vector Search step 5, a comparison section is generated and included into the report

Notes

- The parameter chain_type when initialising the chain has been set to 'stuff' and can be changed to other like:
    - stuff: Concatenates all docs and sends them as one input to the LLM.
    - map_reduce: Answers each doc separately, then combines the answers.
    - refine: Starts with one doc's answer and improves it with others.
    - map_rerank: Scores each doc's answer and picks the best one.
- The parameter search_type when initialising the retriever has been set to 'similarity' and can be changed to other like:
    - similarity: Returns the top-k most similar documents based on vector similarity
    - mmr: Returns diverse and relevant documents by balancing similarity and uniqueness.

    - similarity_score_threshold: Returns only documents with similarity scores above a set threshold.
    - mmr_score_threshold: Like mmr, but filters out documents below a similarity threshold.

Sample Report

[Peterson Transport - RAG pipeline Report.pdf]

Integrating to Production

- Embedding Model: using the Hugging Face embedding model on AWS
    - WIll be required for:
        - Embedding the dataset: one time, heavy payload
        - Embedding the incoming queries(medical conditions): invoked for every query, light payload
- Vector Database: shifting the vector DB from Qdrant(current) to AWS
- LLMs: Already using the Amazon Nova models from bedrock, accessed via LangChain
- Integrating this into the pipeline so we don't have to depend on the Medical Necessity report for the generation of the Medical Necessity report
- This will improve accuracy of disease selection the Medical Necessity report covers

Evaluation

Disease	**MayoClinic Treatment(GT)**	**Medical Summary Data**	**Embedded Search Matches**	**RAG Pipeline Matches**
Dialysis	N/A - Diseases & Conditions Closest match - CKD(Chronic kidney disease) - Treatment for End-Stage Kidney Disease: Dialysis: Hemodialysis Peritoneal dialysis	The patient has a history of Renal Failure, Dialysis, back pain, fever, and bacteremia, requiring a CT L-Spine with IV contrast examination. The patient has a history of renal failure and has undergone dialysis treatment.	Match: CKD(Chronic Kidney Disease)(🟡)	Dialysis Options, Dialysis Adjustments, Alternatives and Complications, Additional Treatment Refer: Outputs

Diabetes	https://www.mayoclinic.org/diseases-conditions/diabetes/diagnosis-treatment/drc-20371451	Mary Collins, a female patient with a complex medical history including diabetes, hypertension, cancer, and … The patient's medical history includes a pre-existing condition of diabetes, as well as other comorbidities..	Match: Diabetes(✅)	Blood Sugar Monitoring Medication Diet and Lifestyle Insulin Therapy Transplant Options Refer: Outputs
Lumbar Fusion	N/A - Diseases & Conditions	Mary Collins, a female patient with a complex medical history including diabetes, hypertension, cancer, and a previous lumbar fusion Mary Collins suffered severe injuries, including brain and lumbar spine injuries, in an accident	Match: Spondylosis, cervical(🟡)	Short Explanation for Lumbar Fusion how it is related to diseases present in the database. Diseases where lumbar fusion is a correct pick Refer: Outputs
Left Shoulder Impingement	N/A - Diseases & Conditions Closest match - Frozen Shoulder	Additionally, the patient has a history of left shoulder derangement, impingement, and rotator cuff tear, which required previous surgical intervention	Match: Frozen Shoulder(🟡✅)	Mentions relevant treatments Refer: Outputs

		Additionally, the patient was experiencing left shoulder derangement, impingement, and a rotator cuff tear		
Slip and Fall Accident	N/A - Diseases & Conditions	The patient's medical history suggests that the condition originated from a slip and accident that occurred in March 2015 a 59-year-old male, has lumbar radiculopathy and a lumbar disc herniation at the L4/L5 level resulting from a slip and fall accident	Match: Foot Drop(❌)	Mentions possible reasons and their treatment Refer: Outputs
Soft Tissue Injuries	N/A - Diseases & Conditions Closest match - Sprains, Muscle Strains	The patient presented with soft tissue injuries, likely resulting from a vehicle accident. The medical evaluation suggests the need for further neurological assessment if the patient continues to experience persistent headaches. The recommendations include rest and the use of Tylenol to	Match: Sarcoma, soft tissue(❌)	Mentions possible treatment, generic, yet inclusive Refer: Outputs

		manage any discomfort associated with the soft tissue injuries.		

Conclusion

The methods performed in this experiment gave promising results, but definitely need refinement for it to be used in production.

The embedding-based search approach delivered specific and often impressive results. However, it lacked generalizability and inclusivity across a broader range of diseases. One key limitation is the tendency for lexical matches to outweigh semantic relevance, occasionally leading to inaccurate or misleading results. Moreover, this method demonstrated high dependency on the underlying dataset, making it vulnerable to gaps in disease coverage or incomplete treatment information.

The second approach using RAG performed well on the shortcomings of Embedding Vector search. While inclusivity was a problem in Embedding Vector search, RAG was able to give results for almost all diseases. It was able to fetch relevant chunks even if the chunks aren't directly related to the disease. While RAG does well in inclusivity, it lacks specificity and precision and often returns loosely related or overly broad information.

Both approaches discussed in this report rely heavily on 2-3 main factors:

1. Database: The outputs are bound by the datasets involved. While Mayo Clinic is a reputable public source, it lacks comprehensive coverage of all diseases, particularly rare or less-documented conditions. For example, if the database lacks treatment information for infections, a wide range of conditions will be underrepresented or entirely missed.
2. Embedding Quality: Both approaches rely heavily on semantic similarity searches, which in turn depend on the quality of the embeddings used. Accurately capturing the nuanced differences and similarities between medical terms is critical for meaningful results.

Next Steps:

Enhance the Dataset

Expand the database to include more comprehensive medical sources such as the Cleveland Clinic. A broader and more diverse dataset will improve disease coverage and increase the reliability of treatment suggestions.

Improve Embeddings

Leverage domain-specific, fine-tuned medical language models to generate higher-quality embeddings. These models can better capture medical context, increasing both the accuracy and consistency of results.

Workflow Integration

For production deployment, integrate the system into a pipeline that does not rely solely on a medical summary for disease and treatment extraction. This would allow more flexible and scalable use cases, such as automated triage, clinical decision support, or patient-facing applications.