Report on semantic search of the Phap Dien website

My Duong

This is a technical report of my work on semantic search with data taken from the Phap Dien website.

1 Workflow

The descriptions given for the 3 tasks are quite clear and easy to follow, so I first devised a workflow, then worked on the three main components: web scraping, chunking and embedding, and vector database search deployment in parallel. Overall, the component that consumes the most amount of time was web scraping, due to the overwhelming number of websites extracted from the `demuc` folder (around 5956 hyperlinks in total). I found that using Hugging Face Space augmented with computing hardware turned out to be very efficient and easy to carry out, which is something that I definitely will make use of in future projects.

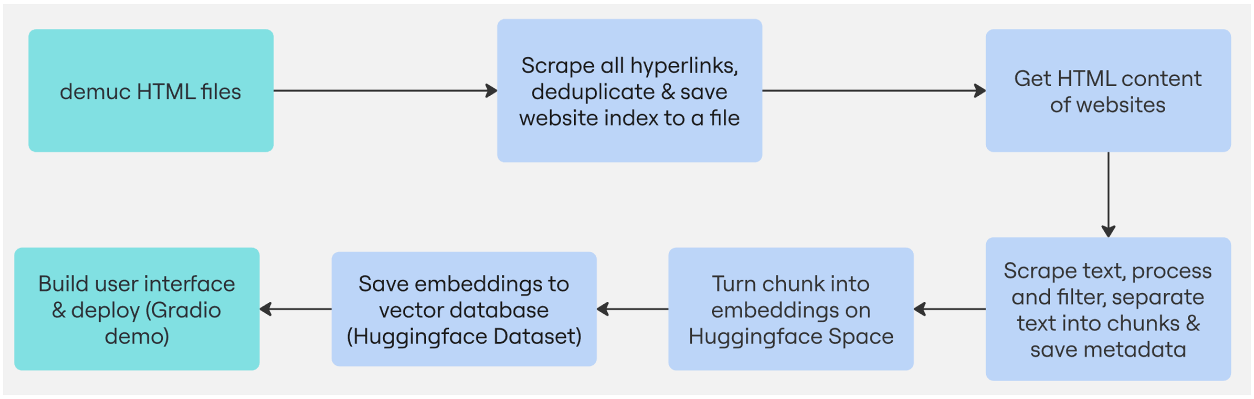
demuc HTML files
Scrape all hyperlinks, deduplicate & save website index to a file
Get HTML content of websites
Scrape text, process and filter, separate text into chunks & save metadata
Turn chunk into embeddings on Huggingface Space
Save embeddings to vector database (Huggingface Dataset)
Build user interface & deploy (Gradio demo)

Figure 1: Overall workflow for Phap Dien semantic search

Algorithm 1 Detailed Webscraping Workflow

```
 1: Input: A collection demuc of HTML files
 2: Output: Web pages saved in respective folders (vbpl, related, history, property, pdf)
 3: $index\_list$ = []
 4: $base\_url\_list$ = [vbpl, related, history, property, pdf]
 5: for all $html\_file \in$ demuc do
 6:     Scrape URLs from $html\_file$
 7:     Append found indices to $index\_list$
 8: end for
 9: deduplicate $index\_list$
10: for all $index \in index\_list$ do
11:     if base_url $\neq$ pdf then
12:         Scrape base_url + index
13:         Save content to folders: vbpl, related, history, property
14:     else
15:         Scrape embed_link in base_url + index
16:         Save content to folder: pdf
17:     end if
18: end for
```

Due to computational constraints, the embedding code was run on the Hugging Face (HF) Space with four L4 GPUs and taking around 2 hours to finish. The vector database was saved to the `camiellia/phapdien_demo` Hugging Face Dataset, and the retrieval process then unzips and pulls from this database. The interface is deployed with the Gradio demo platform and runs on the free HF Space CPU.

Algorithm 2 Vector Database & Retrieval

```
1: Input: Gradio text input (text_input), integer top-$k$
2: $document$ = ””
3: Unzip vbpl folder
4: for all $html\_file \in$ vbpl do
5:     Scrape and clean text from html_file
6:     $document = document +$ extracted text
7: end for
8: $chunk$ = chunk($document$, chunk_size=2000, chunk_overlap=20)
9: Remove chunks with large spacing (menu text, HTML warnings, headers, etc.)
10: Embed each chunk and save along with metadata to vector_db
11: Unzip vector_db from HF Datasets
12: text_output = ””
13: Retrieve top-$k$ chunks based on text_input
14: text_output = text_output + top-$k$ chunks
15: Output: text_output
16: Launch demo
```

2 Notes

2.1 Web scraping

Interestingly, the PDFs are embedded in the `van-ban-goc` page, so the scraping logic for this particular item type is a bit different. I also notice that there are cases where hyperlinks are redirected [1]to other sections of the index page, most of the time from the target page `vbpl.vn/TW/Pages/vbpq-toanvan.aspx?ItemID=` to `vbpl.vn/tw/Pages/vbpq-van-ban-goc.aspx?ItemID=`, most likely due to the fact that the target page does not exist. This problem also persists when scraping related, history and property pages. While my code was designed to deal with problems like status code 404 or redirection, this can lead to potential loss of information. Besides, I could have performed this scraping process more efficiently using an online server like what I did with chunking and embedding due to the large amount of data and long running process.

2.2 Chunking, Indexing and Embedding

While the chunking configuration in the task description was static (`chunk_size=2000`, `chunk_overlap=20`), I realized there are a few parameters like text separators and text length threshold for irrelevancy cutoff, and modules we can tweak, either by defining our own splitting heuristics, or using different built-in modules like `CharacterTextSplitter` and `RecursiveCharacterTextSplitter`. I found that using the common newline split `"\n"` creates very large chunk sizes, while the space character `" "` splits the chunks too finely. Thus, I decided to apply `RecursiveCharacterTextSplitter`, since it allows me to dynamically split documents using multiple types of text separators. I chose `".  "`, `"\n"`, `"; "` and `"\t"` since each law statement in the text extracted from the HTML files are often proceeded by these characters. When processing these chunks, I noticed that irrelevant chunks like menu text, HTML warnings, headers and so on contain very large spacing areas, whereas the core information does not have this characteristic, which prompted me to remove these instances.

The procedure generates nearly 700,000 text chunks. Using my local device, I was not able to complete the embedding component due to limited RAM. I then tried experimenting with the Hugging Face computing units, including CPU Upgrade, one NVIDIA L4 GPU and four NVIDIA L4 GPUs with a maximum runtime of 5 hours (due to the HF `startup_duration_timeout` setting), and found that the four L4 GPUs was able to finish the text embedding process in approximately 2 hours.

2.3 Semantic search

I personally find semantic searching to presents the most intriguing problems out of the three tasks. I used the simple `similarity_search` function, which is based on the embeddings generated by the base model `bkai-foundation-models/vietnamese-bi-encoder`. I would like to find out if more advanced methods like topic modeling with LDA, embedding clustering, information deduplication with aggregate embeddings can further optimize this search process, but due to time constraint, I was not able to carry out these experiments.

[1]Since I was not aware of webpage redirection, I created a few HTML files that contain the information of redirected pages instead of the main page, which is why I had a block of code for deleting these files.