Documentation (Final Report)

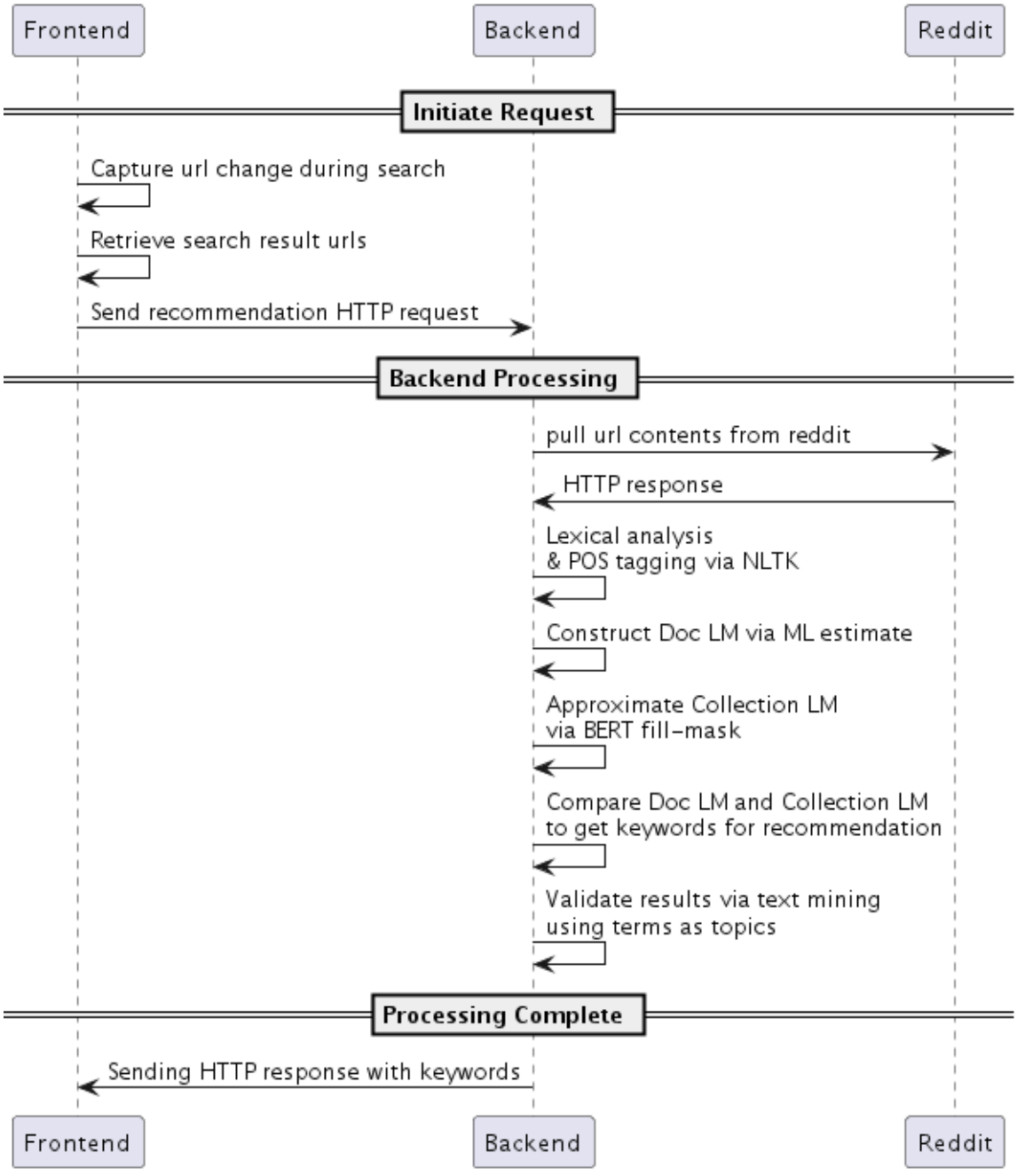
Team Wang

Overview

Our team worked on a query keyword recommendation system for Reddit searches. This project fits into the theme of intelligent browsing with the goal of accelerating the information retrieval process when non-exact keywords are used. For example, they might want to search for a specific type of cheese (e.g. Camembert) but not remember the exact name of the cheese. Then they might search for the keyword “cheese” and browse information from there. We have built a Google Chrome extension that automatically detects Reddit searches, communicates with our backend server to generate recommended keywords, and provides the suggestions in hyperlinks.

Implementation Details

Sequence Diagram



Frontend Details

The frontend is implemented with vanilla Javascript. *background.js* is the background worker for the chrome extension and contains an event listener on tab update events. If the URL of the tab changes (which happens with every search on Reddit), it triggers the code from *inject.js* on the tab.

inject.js first checks the current URL to see if it should be processed. We note down the subreddit (if applicable) and search query. The subreddit is used to construct the suggestion URLs to maintain the query context. Then we continuously pull the search result element that is of class `SQnoC3ObvgnGjWt90zD9Z _2INHSNB8V5eaWp4P0rY_mE`. Reddit uses dynamic loading, and the page load status is set to complete before search results are fully rendered. Because generating recommendations on the fly could be time-consuming, we feel that it is important to keep users informed on what is happening. As soon as search results are returned, we add a label to the DOM indicating that we are processing keyword recommendations. Then we retrieve the URLs from all the anchor elements in the results, and send the top 3 results to our backend server for processing. Once we get a response back from the server, we construct the URLs using the suggestions and subreddit information. The results are then added to the UI.

Backend Details

After getting the HTTP POST requests with the search result urls from the frontend, the backend processes the request as follows:

Text Access

To get the text data for further analysis, we leveraged BeautifulSoap's web scraping functionality for html documents parsing to retrieve the contents from the urls.

Text Retrieval

The core logic is to compare between document LM (Language Model) and collection LM to assign scores to the words from the result based on the frequency differences between the 2 LMs. In this way, the software can recommend the top words that occur in document LM more frequently than in collection LM to users for related searches.

Upon retrieving the contents of the search results, we first pre-process the data for further training. First, we used NLTK for lexical analysis and POS tagging. This is needed to distinguish noun words from the corpus for later training via BERT.

The next step is to construct the 2 LMs, while it is easy to construct the document LM via ML (Maximum Likelihood) estimate on TF, it's hard to get an existing collection LM that suited our use case. We have done some research on the state-of-the-art popular LMs including BERT, BLOOM, and GPT-3, but all of those are generative LMs, so there's no easy way to directly retrieve the word distribution (the mapping between vocabulary and its probability) without context. Thus we investigated and figured out a way to approximate a collection LM without explicit training. Upon getting a fairly large collection of corpus, we

mask the noun tokens based on POS tagging in the corpus and then leverage the Masked LM feature of BERT to recover those hidden tokens. For each masked token during the MLM process, BERT produces a list of possible vocabularies with corresponding probabilities, which is then used to aggregate and then approximate the collection LM.

After the LMs are constructed, we iterate over the tokens in document LM and compare those with collection LM and assign scores based on the TF differences. The top words with the highest TF differences are selected as the keywords to be recommended for the users.

Proof of Correctness

To validate the results given, the backend performs text mining by choosing the keywords (terms) as the topics. There are some similarities between this problem and topic mining problem - both analyze a list of documents and attempt to extract 'common ideas' from the list of docs. Given that the stop words have been filtered out during the text retrieval process, we no longer need to consider too frequent words or IDF weighting, so we just used TF (term frequency) as the metric for scoring.

Software Usage

To set up the backend server, a requirements.txt has been included. Users are able to start the backend server via:

1. *pip3 install -r requirements.txt* (first time only)
2. *python3 handler.py*

Note that the first time spinning up the server would take some time to download the corresponding NLTK libs and the BERT language model.

To install the Chrome extension, visit chrome://extensions/ from Google Chrome and enable Developer Mode. Then select "Load Unpacked" and select the entire cs410-extension folder.

Visit reddit.com and enter a search query of interest, and press enter to search. A text label will show up under the search bar to inform that the server is generating keyword suggestions. All training and processing are done on the fly, and it could take up to 30 seconds to generate recommendations. Once results are available, the UI will be updated to show the top 5 keyword recommendations. If you are searching within a subreddit, search context is kept when you click on a suggested keyword.

In our software tutorial, we demonstrated searching for "taylor swift ticketmaster" both in all of reddit and r/TaylorSwift.

Team Contribution

Both team members proposed and voted on project ideas. After we nailed down on a topic, we also did planning, scoping, and estimation together. Daocheng worked on the backend server design, implementation, and testing. The backend components involve web scraping, document LM construction, collection LM construction, logic to retrieve keywords for recommendation, and validation of results. Ziyue worked primarily on the Chrome extension, and made several changes to the backend server (such

as implementation of OPTIONs and addition of CORS cookies) in order to integrate it with the Chrome extension.