Project Report

Project “PriviSearch” - UIUC CS 410 Text Information Systems

Xiaohan Tian (xtian13@illinois.edu)

1. Overview of the Function

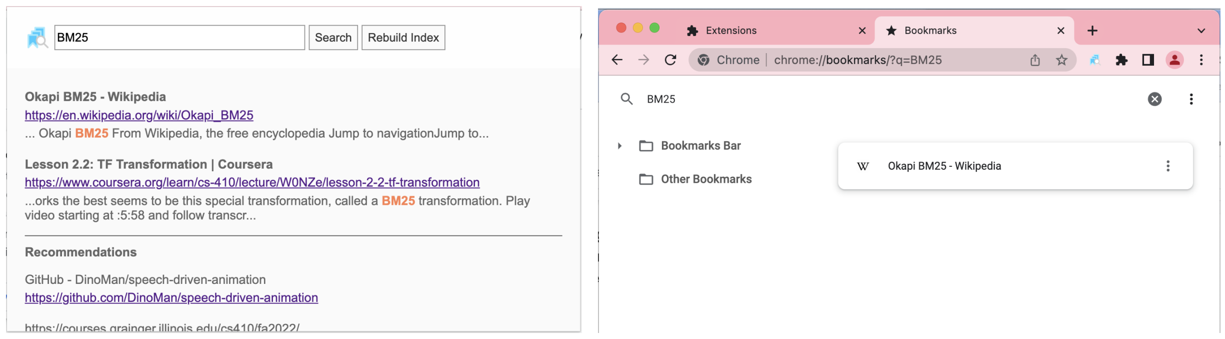
1.1 Motivation

The motivation for this project came from one of my personal challenges in real life. Currently, if users want to search among bookmarked pages, all three major browsers on the market today can only allow users to query from the page titles; it is very difficult to accurately find the information the user needs as the page titles generally can’t completely cover the page content, the page titles are merely an oversimplified version of the page.

To resolve the problem, I wish to leverage the knowledge I have adopted from this course to build a personal search engine that can index my own bookmarked pages and allow me to query from it easily, just like a personal “Google” stays inside of my own browser.

1.2 Features

Project “PriviSearch” is a personal search engine that can perform full-text searches against your bookmarked pages instead of Chrome’s default bookmark manager (which can only perform keyword searches against the page titles). It can also provide you with additional recommended pages from your bookmarks that might be related to your original query terms.



Okapi BM25 - Wikipedia
https://en.wikipedia.org/wiki/Okapi_BM25
... Okapi **BM25** From Wikipedia, the free encyclopedia Jump to navigationJump to...

Lesson 2.2: TF Transformation | Coursera
https://www.coursera.org/learn/cs-410/lecture/W0NZe/lesson-2-2-tf-transformation
...orks the best seems to be this special transformation, called a **BM25** transformation. Play video starting at :5:58 and follow transcr...

Recommendations

GitHub - DinoMan/speech-driven-animation
https://github.com/DinoMan/speech-driven-animation

https://courses.grainger.illinois.edu/cs410/fa2022/

(Figure 1. Comparison between this project with Chrome default bookmark manager. Notice that PriviSearch can find both two documents containing the keyword “BM25”, but the Chrome Bookmark Manager can only find one document, which is the one with “BM25” as part of the page title)

Here is a list of features PriviSearch can provide:

- PriviSearch work as a Chrome Extension can index all the bookmarked pages and allow users to perform full-text search.
- PriviSearch is based on Lunr.js with customizations and extensions.
- Result will be ranked by similarity.
- Potentially related pages will be provided to the user as recommendations.
- Automated indexing mechanism for all existing bookmarked pages and future new bookmarks.
- All the data will be stored in the users' local environment. Users can even perform the search offline. (Actual page can't be rendered without Internet connection except for local pages)

1.3 Tasks and Challenges

Here is the original task breakdown in the project proposal document:

Research suitable JS-based Text Processing Toolkit	2 hours.
Embedding and Enhancing the chosen toolkit. Or, write one from scratch.	3-10 hours
Build extension UI.	5 hours
Implement the text retrieval unit.	7 hours
Implement the text filtering unit.	7 hours
Testing and evaluation.	3 hours

The main task is to build the text procession core. It includes two units: the text retrieval unit will be in charge of indexing bookmarked pages and querying pages by keywords, and the text filtering unit will be in charge of providing related recommendations for the user's query results.

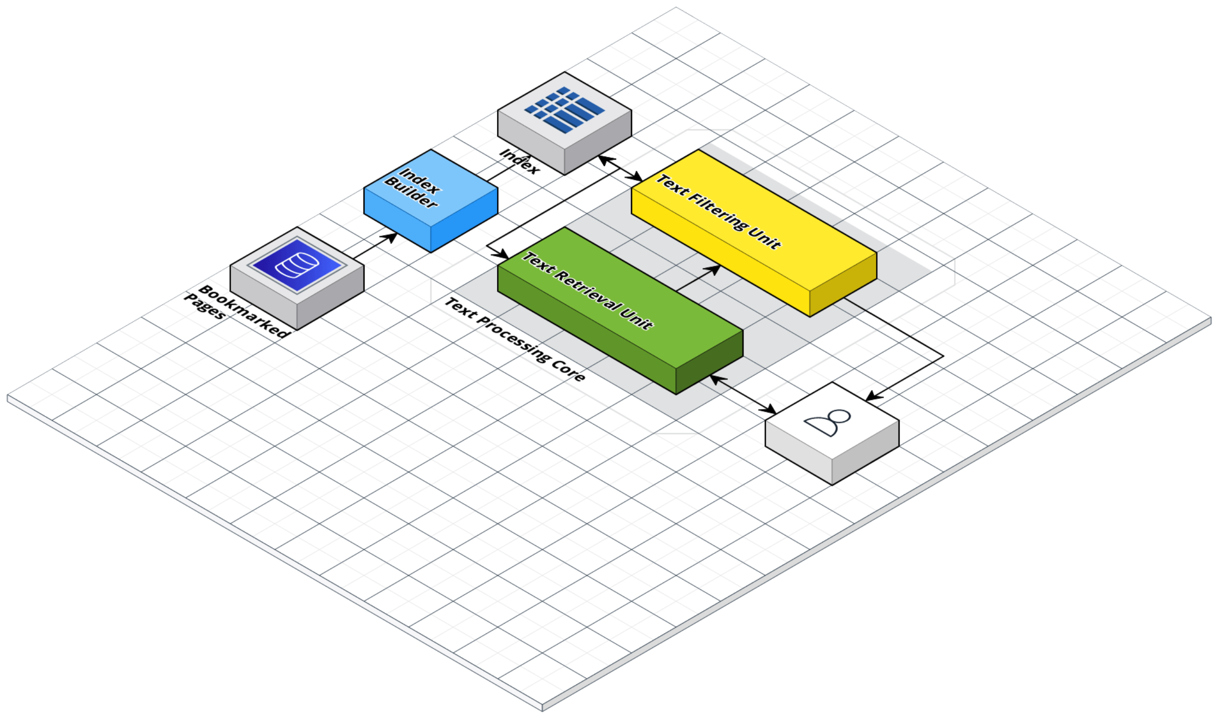
Next, for privacy concerns, some bookmarked pages might contain content that requires users to log in. Thus, I would like to keep everything on users' local machines. Thus, a proper browser-supported persistent mechanism and method need to be implemented.

Also, since this project will be working as a Chrome Extension, thus, HTML-based UI needs to be implemented.

There are two major challenges in this project: the first one is building the text processing core, which is highly time-consuming and important as it serves the core functions of the project; to avoid re-inventing the wheels, I prefer to build the text processing core based on an existing JavaScript-based text processing library. The second one is capturing the page content automatically. Since I need to use Chrome Extension APIs, the permission is very restricted, and the functionality is also limited; it took me quite some time to research and find a way to automatically iterate all the bookmarked pages and capture the page content one by one.

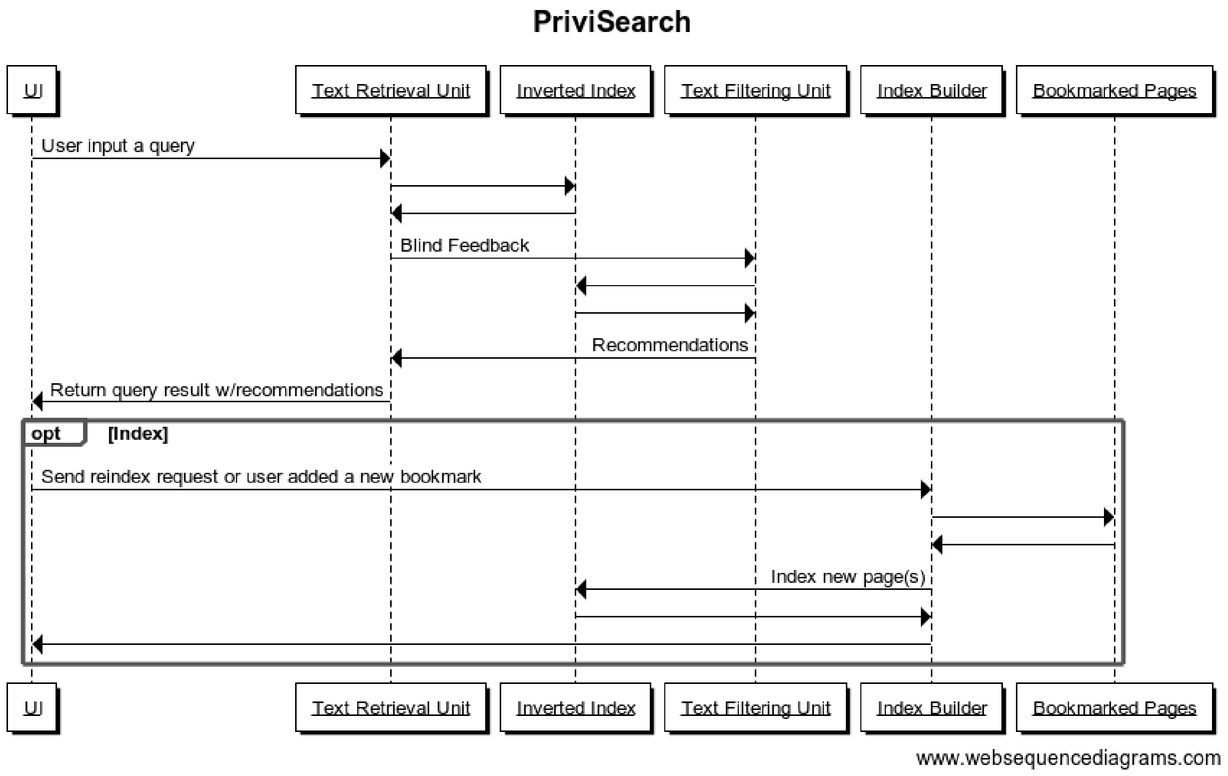
2. Code Overview

2.1 High-Level System Architecture



(Figure 2. High-Level System Architecture)

The system can be divided into three major components: Text Retrieval Unit, Text Filtering Unit, and Index Builder. The Text Retrieval Unit is in charge of performing queries from the inverted index and also provides top k results to the Text Filtering Unit as blinded feedback; The Text Filtering Unit will take blinded feedback results and organize updated query terms which will be used to find recommendations, the recommendations will be provided to the user after filtering out any duplicated records already exist in the original query results; The Index Builder is working independently, it will iterate every existing bookmark and capture the page content then convert it into the inverted index, it will also capture newly bookmarked pages by setting up a Chrome Bookmark event listener.



(Figure 3. Sequence Diagram)

2.2 Function of Code

Source Code	Description
`./popup.html`	Main popup window, it's in charge of taking users' queries and showing users the result and recommendations. It also provides users the option to reindex all the existing bookmarked pages.
`./popup.js`	UI Logic for `./popup.html`
`./bm_indexer.html`	Stand-alone page for rebuilding the index; it provides user guides and progress of indexing.
`./bm_indexer.js`	UI logic for `./bm_indexer.html`. It also contains the logic to iterate users' existing bookmarked pages and capture the page content one by one, the whole process is automatic, and no user intervention is required.
`./background.js`	The source file contains core logic; all three major components are inside this file. This file is set to be executed in the background, and it will be automatically invoked once the user starts using the PriviSearch extension.

`img/*`	Image files.
`lib/lunr.js`	A 3rd party javascript-based text processing library. This file was slightly modified by me under MIT Licence (https://github.com/olivernn/lunr.js/blob/master/LICENSE). This file contains the original Lunr.js and the Lunr extension “Lunr Mutable Indexes”. Please see the reference for detailed information.

3. Implementation

3.1 Technical Overview

The project is a purely JavaScript-based personal search engine with a recommendation system, and it’s living in the Chrome Extension ecosystem.

The project was coded upon a third-party JavaScript text processing library Lunr.js, this library uses an in-memory inverted index and VSM model-based query algorithm. Certain modifications and extensions were made to the library to support the persistent inverted index and the recommendation system.

The UI portion was purely based on vanilla HTML5/CSS/JavsScript (ECMA6), to keep the number of dependencies at a minimum, no other third-party libraries were being used.

3.2 Text Retrieval Unit

The main logic of the Text Retrieval Unit is inside the `query` method in `./background.js`.

The method will take the user’s input string and pass it into Lunr.js pre-processing pipeline. There are three components inside the pipeline, and they are connected sequentially. They are a stemmer (a standard Porter Stemmer), a stop word filter which uses a list of pre-defined stop words, and a trimmer.

After being tokenized, the result of the pre-processing pipeline will be sent to the `lunr.Index` to query the result. Here is the similarity score comparing formula which is being used in Lunr.js:

$$\text{score}_w = \frac{(k1+1)\ \text{IDF}(w)\ \text{TF}(w)}{k1\left(1-b+b\frac{|d|}{avdl.}\right)+\text{TF}(w)}$$

Where:

$$\mathrm{IDF}(w) = log\left(1 + \left|\frac{M - k + 0.5}{k + 0.5}\right|\right)$$

It’s a relatively standard JavsScript-based BM25 implementation. The final score will be computed as:

$$\mathrm{score} = \sum_{w \in V} \left(\mathrm{score}_w \cdot \mathrm{boost}_w\right)^{c(w,q)}$$

All the results will be provided to the user by comparing the similarity score in descending order. The top k results will be considered as blinded feedback and will be sent to the text filtering unit.

3.3 Text Filtering Unit

The main logic of the Text Retrieval Unit is inside the `recommend` method in `./background.js`.

This project will use the blinded feedback method to provide recommendations to the users. As mentioned earlier, after querying the results from the inverted-index, the top k results will be considered as blinded feedback. Those results will first be sent to the same pre-processing pipeline to get tokenized, and then they will be combined with the original query tokens to generate a new set of tokens to query again. The mixture will be computed as follows:

$$\vec{q}_{recommendation} = \alpha \vec{q}_{original} + (1 - \alpha)\,\vec{q}_{rel}$$

Note that this formula ignored the non-relative documents compared to the original formula since normally negative (non-relevant) examples are neglectable.

The queried results will be filtered to remove those duplicated records which already inside of the original results, and the rest of the records will be sent to the user as recommendations.

3.4 UI

The UI is implemented using standard HTML5/CSS3/ECMA6 JavaScript, no 3rd party libraries are being used.

Additionally, in order to reduce the number of total files, all the CSS is being embedded inside of the headers of HTML files accordingly (NOT at the tag level).

3.5 Browser Event Listener

There are two listeners that are being set when initializing the extension.

The first one is a general message listener, which is being used to communicate between `background.js` and frontend pages. This listener has been attached to the event `chrome.runtime.onMessage`.

The other one is a listener on the bookmark system. It will be automatically triggered when the user adds a new bookmark; when such a situation happens, the extension will automatically capture the page content and send it back to the Index Builder. This listener has been attached to the event `chrome.bookmarks.onCreated`.

4. Usage

4.1 Introduction

This project is in Chrome Extension form, users can follow the instructions below to install this Extension to their Chrome browser.

4.2 System Requirements

Extension Runtime Environment Requirement:

- Google Chrome 107+

Google Chrome System Requirements:

Windows:

- Windows 7+
- Intel Pentium 4+ w/SSE3 Support

Mac:

- macOS High Sierra 10.13 or later

Please refer to this page for detailed system requirements for Google Chrome:

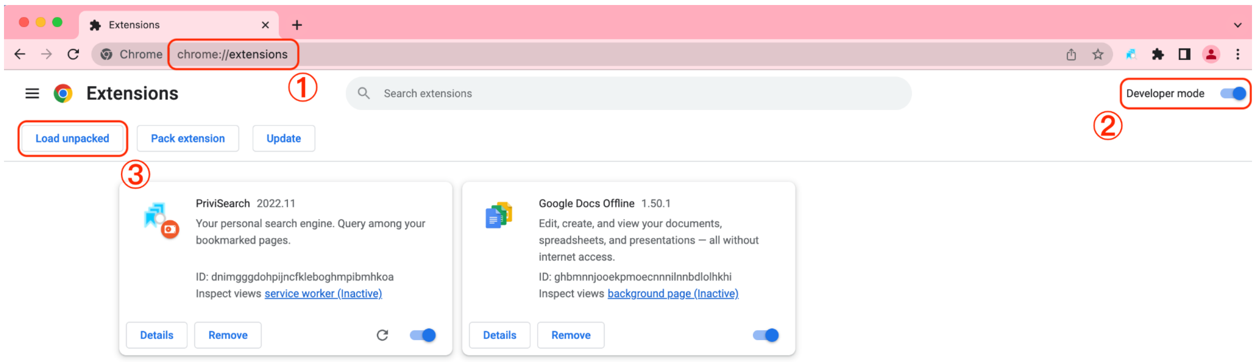
- https://support.google.com/chrome/a/answer/7100626?hl=en

4.3 Installation

Please put the unpacked extension to a folder with correct reading permissions.

1. Open Google Chrome, then click “⋮” -> Settings -> Extensions (or open a new tab and input `chrome://extensions/` on the URL bar)

2. Enable “Developer mode”.
3. Click “Load unpacked” and select the folder containing the unpacked extension.



PriviSearch 2022.11
Your personal search engine. Query among your bookmarked pages.
ID: dnimgggdohpijncfkleboghmpibmhkoa
Inspect views service worker (Inactive)

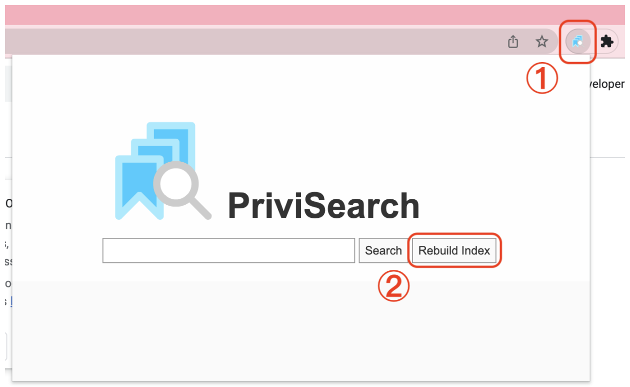
Google Docs Offline 1.50.1
Edit, create, and view your documents, spreadsheets, and presentations — all without internet access.
ID: ghbmnnjooekpmoecnnnilnnbdlolhkhi
Inspect views background page (Inactive)

(Figure 4. Installation)

4.4 Building the Index

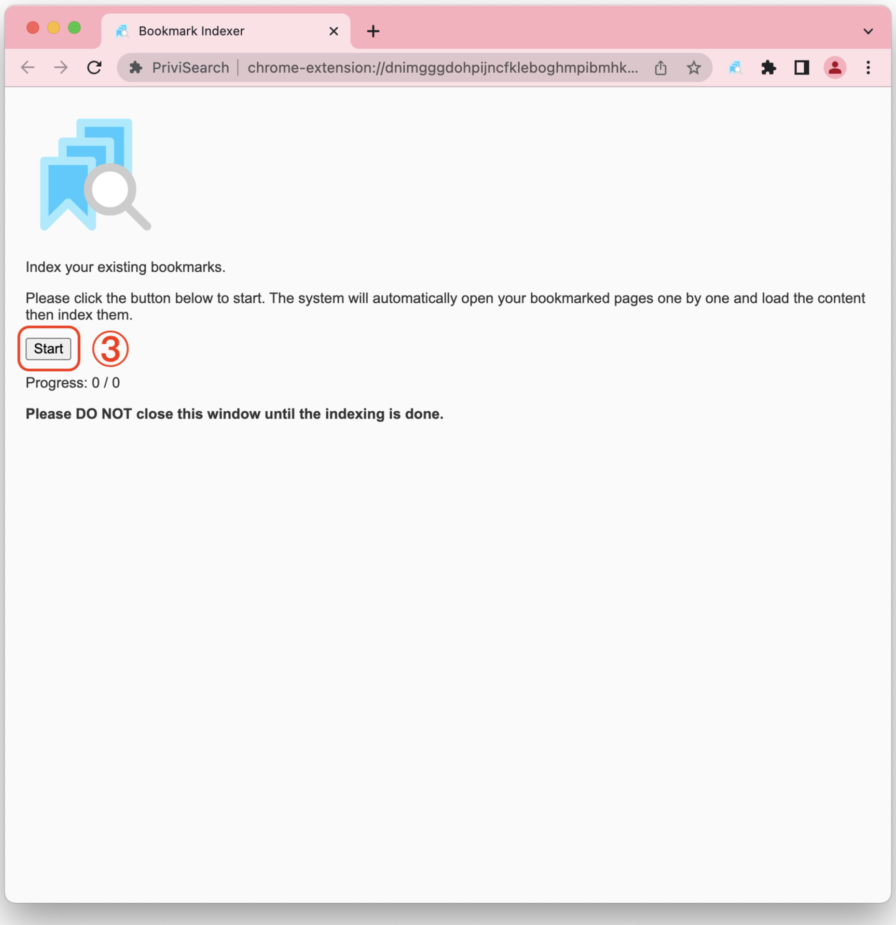
For the first time run, please build the index for your existing bookmarked pages. To build the index:

1. Click the extension icon
2. Click the “Rebuild Index” button



(Figure 5. Building the index, part 1)

3. On the newly opened window (Note in rare cases, Chrome will open this window in the background. If you can’t see this window after step 2, please try to switch between different windows, it might be hidden behind some other windows) and click “Start”


Bookmark Indexer
PriviSearch | chrome-extension://dnimgggdohpijncfkleboghmpibmhk...
Start
③

Index your existing bookmarks.

Please click the button below to start. The system will automatically open your bookmarked pages one by one and load the content then index them.

Progress: 0 / 0

Please DO NOT close this window until the indexing is done.

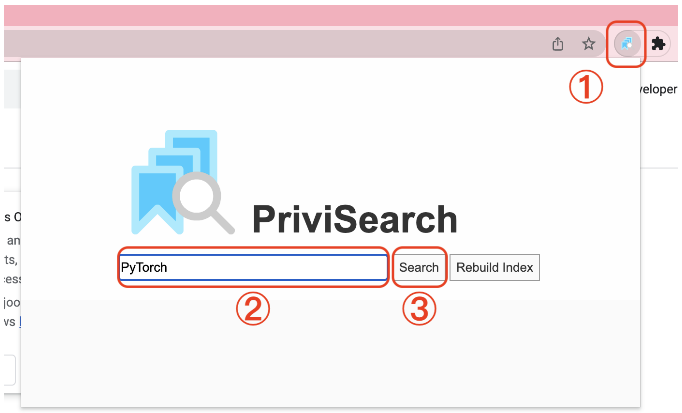
(Figure 6. Building the index, part 2)

4. Now the Extension will automatically iterate all your bookmarked pages and open them one by one, and it will capture the page content individually. This process will take some it depends on the number of bookmarked pages and the network speed. The “Progress” section will show the progress in “current/total” format. **Please DO NOT close this window until the indexing is done**.
5. After indexing is done, the page will show you a message that the window can be safely closed. Note for all the new bookmarked pages in the future. It is **NOT** required to rebuild the index as a bookmark creation listener will be automatically set up for you. You only need to perform this step for existing bookmarks. If you don’t have any existing bookmarks, this step can be skipped.
6. You can rebuild the index anytime by following the steps above (e.g. in case of the existing index is corrupted).

4.5 Query among Bookmarks

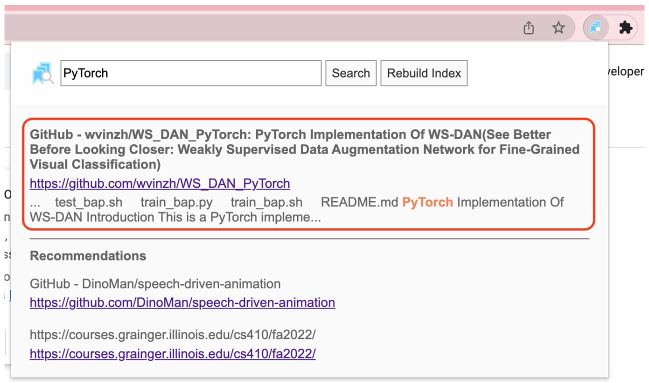
To perform a query:

1. Click the extension icon
2. Input the query terms in the search box
3. Press Enter (Windows) / Return (macOS) key, or click the “Search” button.



(Figure 7. Query)

4. The matched results will be displayed to the user with the links, clicking the link below the page title will open the bookmarked page in a new tab.



GitHub - wvinzh/WS_DAN_PyTorch: PyTorch Implementation Of WS-DAN(See Better Before Looking Closer: Weakly Supervised Data Augmentation Network for Fine-Grained Visual Classification)
https://github.com/wvinzh/WS_DAN_PyTorch
... test_bap.sh train_bap.py train_bap.sh README.md PyTorch Implementation Of WS-DAN Introduction This is a PyTorch impleme...

Recommendations

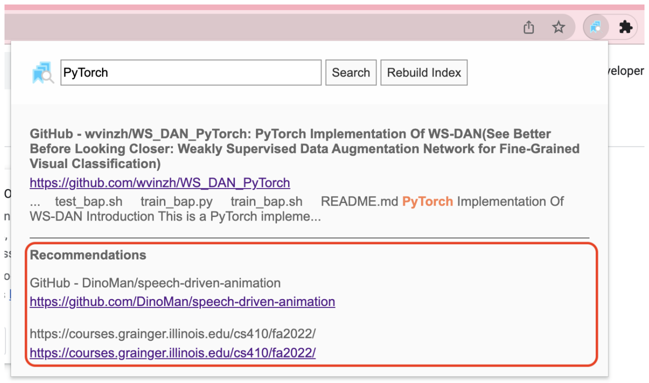
GitHub - DinoMan/speech-driven-animation
https://github.com/DinoMan/speech-driven-animation

https://courses.grainger.illinois.edu/cs410/fa2022/
https://courses.grainger.illinois.edu/cs410/fa2022/

(Figure 8. Query results)

4.6 Using Recommendations

On the query results page, below all the matching records, the “Recommendations” section will list some other bookmarked pages that might be related to the original query term. Clicking the link below the page title will open the page in a new tab.



GitHub - wvinzh/WS_DAN_PyTorch: PyTorch Implementation Of WS-DAN(See Better Before Looking Closer: Weakly Supervised Data Augmentation Network for Fine-Grained Visual Classification)
https://github.com/wvinzh/WS_DAN_PyTorch
... test_bap.sh train_bap.py train_bap.sh README.md PyTorch Implementation Of WS-DAN Introduction This is a PyTorch impleme...

Recommendations

GitHub - DinoMan/speech-driven-animation
https://github.com/DinoMan/speech-driven-animation

https://courses.grainger.illinois.edu/cs410/fa2022/
https://courses.grainger.illinois.edu/cs410/fa2022/

(Figure 9. Recommendations)

5. Contributions

I worked as a single-person team, all the codes, documentation, and presentations are created by me.

6. References

- Google System Requirements
  - https://support.google.com/chrome/a/answer/7100626?hl=en
- Google extension developer guide
  - https://developer.chrome.com/docs/extensions/
- CS 410 Course Materials
  - https://www.coursera.org/learn/cs-410/
- HTML/CSS Guides from W3School
  - https://www.w3schools.com/html/
- A simple implementation of an in-browser search engine (pure JS implementation)
  - https://javascript.plainenglish.io/building-a-simple-in-browser-search-engine-d87c86ac3261
- “Build a simple search engine with HTML, CSS and JS”
  - https://www.youtube.com/watch?v=o2aAaVsC1Oo
- “Create Product Search Engine/Bar/Filter in JavaScript”
  - https://www.youtube.com/watch?v=ZFUOC-y4i0s
- Lunr.js and Lunr Mutable Indexes extension
  - https://lunrjs.com/
  - https://www.npmjs.com/package/lunr-mutable-indexes